AI Driven System of Trading Systems

Submitted to the Faculty
Of
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

By

Alan Fernandez

Remy Kaldawy

Roberto Esquivel

Zoraver Kang

January 25, 2019

Project Advisors
Professor Hossein Hakim
Professor Reinhold Ludwig
Professor Michael Radzicki

Abstract

The purpose of this Interactive Qualifying Project (IQP) was to explore the development of a system of trading systems in combination with machine learning techniques. This project covers the entire development process from the base strategies that compose the individual trading systems to the grouping of those strategies into a system of trading systems.

Among the most notable of the report's findings are that it is possible for retail traders to develop strategies with positive returns. The report also proves that it is possible to develop and implement an artificial intelligence-driven system of trading systems which is predictive with several algorithms. Neural networks in particular are useful for the detection of market patterns in trading systems.

Acknowledgements

The authors of the report would like to thank Professors Hossein Hakim and Michael J. Radzicki for their continuous guidance and support in the writing of this report.

We also thank TradeStation for granting us use of their platform for our research.

Table of Contents

List of Figures

List of Tables

1. Overview and Introduction

1.1 Motivation to Invest

Investment is a key tool to generate and maintain wealth. Investing provides an alternate source of income that can help fund expenses, cover education costs or most commonly reduce your retirement age. In the United States, according to the Economic Policy Institute, the average savings for retirement of the 50th percentile is just $5,000 [1]. Assuming that the monthly living expenses of a single person in Worcester is approximately $1,250 such low savings would allow a person to survive for four months out of the 15-20 years of retirement. It is thus advisable to invest early to begin building wealth that can cover the costs of living during retirement.

Whereas the returns of individual trading strategies will fluctuate based on market conditions and the validity of the market edges (systematic ways to beat the market) that they exploit, a system of trading systems should produce more constant returns throughout any type of market. While some of the strategies in the system of systems may produce losses, in a well-balanced system, the remaining strategies should produce balancing profits. The goal of the system of systems is to reduce the likelihood of producing overall losses. The possibility of a more stable stream of profit justifies the slightly larger time investment demanded by the development of systems of trading systems.

There is a large supply of financial data available to help develop and execute trading strategies. However, a human or group of humans would need an absurd amount of time to process and to arrive at conclusions from such a large body of data. Machine learning allows efficient analysis that derives conclusions from these large volumes of data. These findings can then be leveraged to construct strategies and algorithms that achieve a higher level of performance than those built solely by humans. Another advantage of machine learning for trading is that it decreases the human bias in decision making, which can be influenced by a trader's mood or personality. By eliminating the potential mistakes caused by these biases, machine learning can help construct strategies that yield better results.

This report covers the basics of investing and introduces the reader to all the necessary background knowledge to begin to comfortably develop his own trading strategies. Sample strategies are also presented and thoroughly analyzed to give the reader an example of how to approach the development. The report finally covers the basics of machine learning and the implementation of a complex system of trading systems with machine learning algorithms.

1.2 Our Project

Our project centers on the development of trading strategies and their execution via a system of systems with an overall goal of systematically beating the market. There is value in the development of algorithmic trading because it allows individuals to participate in markets typically dominated by large institutional players. Furthermore, artificial intelligence is an emerging technology whose applications to finance have not been fully realized. We determined that the development of trading algorithms under an AI-driven system of systems would provide novel work to the field of finance and potentially influence positive externalities in society.

To execute our goals, we first gathered detailed knowledge on both trading system development and neural networks as a technology. We explored market patterns, types of trading, and optimization and analysis techniques to inform our own strategy design process. To build the system of systems, we examined artificial intelligence as a whole, focusing our study on the architecture and implementation techniques for neural networks. We also did a review of financial technology to determine how to implement the supersystem.

The remainder of this section presents an introduction to trading systems. We also provide a set of motivations explaining why investment is a useful tool to accrue wealth.

1.3 An Overview of the Financial Markets

Before the nature of investing and trading is introduced to the reader, it is necessary to know about financial markets and what each type of financial market is composed of.

A financial market is any activity where there is an entity that supplies funds and an entity that acquires the funds. Wherever transactions exist, there are financial instruments. The types of financial markets are capital markets, money markets, cash markets, derivative markets, and currency and interbank markets [2].

The capital market is where individuals and institutions trade securities. It can be divided into the stock market and the bond market. The stock market allows investors to buy and sell shares of publicly traded companies and derivatives. The primary stock market is where initial public offerings of stocks are first offered. The secondary market is where the buying and selling of shares occurs between the investors rather than the entities that initially issue the shares in the primary market. There are many stock exchanges around the world. The largest are the New York Stock Exchange (NYSE), the National Association of Securities Dealers Automated Quotations (NASDAQ), the Tokyo Stock Exchange, the London Stock Exchange, the Shanghai Stock Exchange and the Euronext [2].

The bond market allows investments in which an investor loans money to an entity for a defined period and interest rate. Bonds are issued by states, federal governments, corporations, and municipalities mainly to raise funds and fund projects [2].

The cash market is a market in which financial instruments are sold for cash and are delivered immediately. This is different from other markets because the price is determined at the spot and if the market is very liquid (high volume of transactions) the price can vary from second to second [2].

The money market deals with short-term loans. It trades certificates of deposit, banker's acceptances, certain bills, notes and commercial papers. It is mainly used for short-term borrowing and lending, ranging from days up to a year [2].

The derivatives market is a financial market that trades entities that get their value from an underlying asset or assets. It is mainly used to formulate risk management strategies. Derivatives markets are highly complex and thus not recommended for inexperienced traders [2].

The interbank market is a financial market where banks and financial institutions trade amongst themselves and where currencies are traded. The most prominent currency market is the Foreign exchange market (FOREX). It is the largest market in the world and anyone can participate in it. It is open 24 hours a day, 5 days a week [2].

Now that background knowledge of financial markets has been introduced, we can talk about our project.

2 Trading and Investing

The previous section gave a high-level overview of financial markets and provided a set of motivations for why people should invest. There are clear social benefits when people are financially independent, and investing in financial markets is a way for anyone to reliably build their wealth. What is more, investment and trading spurns economic growth, which benefits society as a whole. There is an onus for market experience to be adopted as common knowledge. This section presents methods of investment and trading in financial markets, which is requisite knowledge for anyone who wants to operate in those markets.

2.1 Retail Trading and Investing

Retail traders are typically individuals who trade securities for personal gain rather than for an institution's gain. Traders who trade for an institution are known as Institutional Traders [3]. Retail traders often trade smaller accounts and are limited in the complexity of the trades in which they may engage. Institutional traders, for example, may have access to IPOs while retail traders do not. Retail traders, however, can trade most securities available in the exchanges: stocks, bonds, options and currencies. Given that they are limited in the size of their account, retail traders will often engage in small volume trades that have a minimal impact on the market price. While institutional traders may control the market when placing their positions, retail traders will follow the market. Retail traders are also more likely to trade low cost securities given their limited capital.

One of the key disadvantages to retail traders is that they are at the mercy of their broker in the speed at which their trade is executed. This means that retail traders will suffer more from slippage than institutional traders as well as from higher transaction costs. Retail traders may find that trading often and scalping are both rather unprofitable strategies due to slow order execution and high transaction costs.

2.1.1 The Trading Personalities

In *The Art and Science of Technical Analysis,* Adam Grimes writes about the importance for the individual discretionary trader to understand and to adapt his trading strategies to his personality type. Individuals who are extremely risk averse may only trade during the day when they have complete control of their position and never enter overnight trades. Traders who are less risk averse may enter overnight gap positions before the market closes. The trader's personality, as well as his trading style, will partially determine the trader's success in the market. A trader who uses a strategy that is not suitable for his personality, is more likely to fail to follow through with strategy's rules, and thus has a higher likelihood of failure.

The personality of the trader will also determine the length of the positions that he enters and thus the type of market participant that he is. A trader, by definition, is someone who is looking for short term gains while an investor is someone who places long term bets on the market. The trader opens and closes positions with a higher frequency than the investor. The length of the positions of the trader may last anywhere from sub second to months. It is imperative for every individual to engage in honest reflection about his personality and his ability to resist the temptation to make abrupt decisions. Only then will the individual have the ability to make an informed decision about his approach to the market.

These abrupt decisions, can also be minimized with the use of machine learning, which the paper will review extensively further on. There is more background information on trading and investing within financial markets that needs to be addressed before approaching the idea of developing strategies and machine learning.

2.2 The Four Asset Classes

2.2.1 Currencies and the Foreign Exchange Market

According to the tenets of Modern Monetary Theory, sovereign currencies are the unit of account of sovereign nations. A sovereign currency is one which is issued by a single country, for example, the US Dollar (USD) and the Japanese Yen (JPY). A sovereign nation has absolute control over its currency and, as the sole issuer of the currency, the government is infinitely wealthy in its own currency [4].

Modern Monetary Theory suggests that since fiat, or government-issued paper currencies, do not have any material backing, that is they are not exchangeable for anything, the taxes levied by the government on its people are the drivers of the currency. The reason why US citizens accept the US Dollar in the United States is that they have to pay taxes in US Dollars to the US Government [2]. Since taxes are paid in dollars and the government is the issuer of the dollar then the government has to spend dollars before it can tax. The following is thus concluded: The sovereign government does not need to collect taxes in order to spend. Taxes are collected in order to force the inhabitants of the nation to use the currency. Now with the existence of a widely accepted unit of account, the nation can develop complex economic systems where people may trade goods and services.

The Foreign Exchange exists as a result of the need to exchange one currency for another in order to purchase goods and services from a different country. Since exporters pay taxes in their country's currency, they prefer to accept their country's currency as a form of payment. Thus, the buyer must first obtain the seller's currency, in the Foreign Exchange market, in order to trade. The value of one currency against another is set by the typical market forces of supply and demand. The USD stands out among currencies in that it is widely accepted in exchange for goods and services around the world given its status as the most stable currency in the world. Sellers worldwide will accept US Dollars over their own currency in exchange for goods and services [4].

2.2.2 Bonds

A bond is a form of debt issued by an entity to raise money to finance its operations. Bonds are issued by governments, municipalities, states, and corporations. Similar to a loan obtained from a bank, the issuer of the bond agrees on an interest rate paid over the duration of the loan. The full face value of the bond is returned to the debtholder upon the maturity of the bond [5].

Should the bondholder need to recover the face value of the bond before it reaches its maturity, he may trade the bond in an exchange. In doing so, the bondholder forfeits the remaining interest earnings on the bond, but keeps any that were accumulated during the period that it was held. As a result of selling the bond past its release date, the reseller may have to forfeit part of the face value of the bond as well since the new holder will no longer receive the full interest that the bond accrued since its creation.

Lastly, it is important to consider the credit rating of the bond because it represents the likelihood that the debtor will pay the loan when the bond matures. The most creditworthy bonds are those issued by the United States Treasury. Treasury Bills are considered default free because they are backed by the Federal Reserve of the United States. Since they are considered risk free, T-Bonds typically carry low yields compared to bonds issued by companies. Bonds are rated by credit rating agencies, such as Moody's and Standard and Poor's, on a scale where the most trustworthy are rated "AAA" and the least trustworthy are rated "D" [6].

2.2.3 Equities

Equities, otherwise known as stocks, represent part ownership of a company and may grant access to a small portion of the company's profit through dividend payments. Preferred stock, in contrast to common stock, grant voting rights as well as preference during the liquidation of a company's assets following bankruptcy. Common stock carry no rights aside from dividend payments if the company offers them. Investors will typically trade equities through the stock market, but they may be traded over the counter as well. Over the counter trades carry higher risk because the stocks traded in these markets do not meet the strict financial requirements of the stock exchanges. These companies are typically young and volatile and may cease to exist without notice [7].

2.2.4 Commodities

Commodities are goods used for the creation of other goods and services. These are the four categories of commodities traded in the Commodity Exchanges:

- **Metals** (Gold, Silver, Palladium…)
- **Energy** (Crude oil, heating oil, natural gas…)
- **Agricultural** (Corn, Coffee, Sugar)
- **Livestock and Meat** (Live cattle, hog, pork bellies…)

Most traders do not trade commodities because their trade is not as straightforward as trading bonds or equities. Commodities are typically traded using Futures Contracts. Futures Contracts are agreements to buy the commodity at a later date at an agreed upon price. Futures Contracts allow companies to plan their expenses ahead of time by securing a price regardless of the future actions of the market. For example, should the Organization of the Petroleum Exporting Countries (OPEC) decide to cut the supply of oil, the current price of oil may rise to $70 per barrel. A company that holds a Futures Contract stating a price of $50 per barrel would not have to pay the current market price of $70 per barrel even though the final transaction takes place when the price is $70 per barrel. Another way to engage in the commodities market is through ETFs (Exchange Traded Funds) that invest in commodities. Since ETFs trade in the stock exchange, the investor can get exposure to a commodity without having to deal with the mechanics of futures contracts [8].

2.2.5 Derivatives

Derivatives are financial instruments that represent groups securities. The value of the derivative is dependent on the value of the underlying securities. Common derivatives include futures contracts, forward contracts, swaps, options, mortgage backed securities and credit derivatives.

Futures contracts are used by producers of raw materials and by those who purchase them to protect themselves against fluctuations in prices of raw materials due to the natural market forces. When a bread producer enters a futures contract for wheat, he will purchase a set amount of wheat with a set delivery date at a set price. If the price of wheat goes up, then the bread producer gets a bargain for the wheat and if the price of wheat goes down then he is overpaying for the wheat. In either case, the producer knows exactly how much he will pay and how much he will get. This allows the company to plan months or years in advance without worrying about fluctuating prices. Speculators enter the market by purchasing futures contracts when they believe that the market consensus for the behavior of the price is incorrect. If, for example, a speculator believes that the price of oil will rise rather than fall he may buy a futures contract which will be worth a lot when the price of oil does rise. The speculator is not interested in receiving the barrels of oil rather he is interested in reselling the futures contract at a profit before it expires because the contract locked a set amount of oil at a lower price than the current market price.

Forward contracts, unlike futures contracts, are only traded over the counter (OTC). This is as a result of their greater customizability allowing for any date, commodity and amount. They incur a higher default risk because they are traded OTC [9].

Swaps are commonly used to allow for two parties to trade loan obligations. For example, two individuals may trade a variable interest rate loan and a fixed interest rate loan if they are better off with the other. The loans are still written to the name of the original owner, but the swap contract states that each party must pay for each other's loan at an agreed rate. Swaps are risky because if one party defaults, the party that is left is forced back into the original loan [9].

Options are similar to futures contracts in that they are an agreement to trade in the future at a predetermined price. The key difference, however, is that in an options contract the buyer is not obligated to execute the trade if he does not desire to do so. In a futures contract both parties have the obligation to fulfil the agreement. The buyer of the options contract has the option to execute the contract [9].

Mortgage Backed Securities (MBS) are a way for the banks to free up their capital and to provide a way for investors to buy into mortgages. When an investor purchases a MBS, he receives the principal payments as well as the interest payments made by the holder of the loan towards the mortgage. MBS provide a semi constant stream of money since the MBS is made up of hundreds, if not thousands, of mortgages. If a mortgage defaults, there are many others that will not and will continue to make their payments [9].

Credit Derivatives (CDs) allow two parties to transfer credit risk [10]. For example, a bank who wishes to reduce exposure to credit risk may buy Credit Default Swaps (CDS) from an institutional investor who is willing to take on the risk of default. Should the debtor default on the payment, the seller of the CDS has the obligation to compensate the bank that bought the CDS for the full amount of the defaulted loan. If the debtor does not default then the seller of the CDS keeps the interest payments made by the buyer of the CDS in exchange for transferring the credit risk. In other words, if the bank thinks that the debtor will default, then the bank will seek to buy a CDS to ensure that it gets paid the full amount one way or another. The seller of the CDS is betting that the bank is misjudging the risk of default. If the seller is correct, he will receive interest payments from the bank for taking on the bank's credit risk [11].

Now that all of the background for financial markets, their components and their relationships has been introduced, the reader is ready to start learning how to analyze that information to start developing strategies.

2.3 Intermarket Analysis

Intermarket analysis refers to the analysis of the correlations between asset classes in order to predict the movements of the markets. Investors typically look at the relationships between the four main asset classes: equities, currencies, bonds and commodities.

According to John Murphy, the father of intermarket analysis, intermarket relationships are dependent on the inflationary environment [12]. During an inflationary period stocks and bonds have a positive correlation, while bonds and commodities as well as the US Dollar and commodities have an inverse correlation. In a deflationary environment these relationships are inverted. Stocks and commodities also develop a positive correlation because deflation is negative for both stocks and commodities, but positive for some types of bonds. US Treasury bonds are more attractive during deflation because the prospect of a fixed payment is more attractive than the dwindling value of the stocks. Corporate bonds may suffer, however, since companies find it harder to pay their loans as their profits decrease with the drop in prices [13].

Although intermarket relationships often last for long periods of time, there are periods during which these relationships break down. It is important to verify these assumptions while performing intermarket analysis.

2.4 Equity Sector Rotation and the Economic Cycle

Investors in the equity markets engage in sector rotation because different sectors perform better at different stages of the economic cycle. The economic cycle resembles a sine wave and is composed of four phases: Market Bottom, Bull Market, Market Top and Bear Market.

Before delving into the different cycles, it is important to understand the concept of the yield curve since it is one of the most important tools for investors to determine the stage of the economy. The yield curve is the plot of the bond interest rate vs time to maturity.

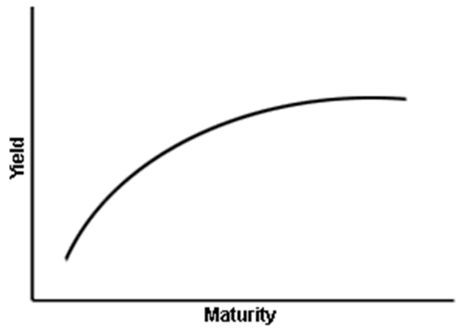


Fig. 1 - Yield Curve

The interest rate has a direct correlation to investors' fear of default. The shape of the yield curve is a reflection of the market's confidence in the fulfillment of debt obligations. During a bull market, the yield curve will slope upwards as investors feel confident about debt repayments in the short term. As the market begins to approach a recession investor confidence in repayment in the short term decreases and the yield curve begins to flatten. Naturally, the yield curve inverts during a recession since rising unemployment and falling profits increase the difficulty of the fulfillment of debt obligations; investors are not confident of debtors' ability to repay [14].

At the Market Bottom, the economy is considered to be at full recession. During this time businesses are struggling and unemployment rates are high. This means that it is difficult for businesses and individuals to fulfill their debt obligations and so the yield curve is flat or possibly inverted since all loans carry a high risk of default. The three sectors that have performed best during this stage are Cyclicals and Transport, Technology and Industrials.

During the Bull Market stage, the market begins to recover, production picks up, and unemployment decreases. Since the economy is in recovery, investors begin to trust debtor's ability to fulfill their debt obligations and short-term loans begin to carry less risk. The yield curve now begins to slope upwards. The best performing sectors during this period are Industrials, Basic Materials, and Energy.

During the Market Top, interest rates rise rapidly as investors begin to fear a recession, the yield curve flattens and so does industrial production. The sectors that perform best during this stage are Energy, Staples, and Services.

Finally, during the Bear Market, the economy enters the recession and interest rates peak, and industrial production begins to fall. At this stage, the yield curve inverts as investors lose confidence in the ability of debtors to fulfill their debt obligations. Investors begin to demand higher interest rates to account for the increased risk of default. The Services, Utilities and Cyclicals sectors perform best during this period.

Since different sectors perform best during each period of the economic cycle, investors rotate sectors in their quest for the highest possible returns. In order to gauge the stage of the economy investors will typically watch the yield curve with care as it is considered one of the best indicators for the stage of the economy [14].

2.5 Breadth of the Market

The Breadth of the Market refers to the direction of the market as a whole. Is the market in the hands of the bears or in the hands of the bulls? Traders use a series of Market Breadth indicators in order to gauge how much control each side has on the market at any given time. With this information traders are able to determine how to best enter their next position. The market is considered to be in the hands of the bulls when the ratio of securities increasing in price compared to those decreasing in price is greater than one. A bear market occurs when the opposite is true [15].

Traders have access to a variety of indicators and techniques, which predict the direction of major financial markets, in order to determine the breadth of the market. Some of the most common indicators are volume and price based indicators. Volume-based indicators look at the volume stock trades in each direction. If the volume up is greater than the volume down then the market is in the hands of the bulls and vice versa. Price based indicators use moving averages to determine the percentage of stocks making new record highs. Typical record highs include the 52 week average and the 200 day average. The SPXA200R index shows the percentage of stocks in the SP500 that are trading above their 200 day average. This gives traders an idea of the strength of the market as well as a way to determine overbought and oversold conditions based on the extreme readings of the indicator. Table 1 lists a few indexes/indicators for determining the breadth of the market [15].

Table 1 – Volume and Price Based Indicators

Volume Based Indicators	**Price Based Indicators**
Cumulative Volume Index	Advance Decline Index
On Balance Volume	New Highs-Lows Index
Up/Down Volume Spread	SPXA200R
Up/Down Volume Ratio	Arms Index (TRIN)
Force Index	McClellan Oscillator

2.6 Systematically Beating the Market

As mentioned in section 1, the purpose of the research presented in this paper is to develop a system of trading systems to systematically beat the market. The alternative, more suitable for traders who are unwilling or unable to dedicate a large amount of time to trading, is to ride the market with an investment in an Exchange Traded Fund (ETF). ETFs exist in many forms to suit most investment strategies. They typically hold a number of positions in multiple assets which sometimes span multiple markets. Their nature allows investors to easily diversify without the need to monitor multiple positions. Unlike mutual funds, ETFs tend to be passively managed meaning that they do not adjust to the changing market conditions and may thus perform poorly in changing market conditions [16]. This approach, is commonly referenced as the “buy and hold” approach.

Systematically beating the market, as opposed to riding the market, is more attractive given the possibility of higher returns. It does, however, require active development of new trading strategies as well as active monitoring of the market. The time commitment required to systematically beat the market is the main barrier to entry. The key to systematically beating the market is to develop a number of trading systems. Although individual trading systems will not necessarily outperform the market, when the trading systems are combined into a system of trading systems their compounded individual success will create the possibility of outperforming the market. (Radzicki.)

2.7 Manual and Algorithmic Trading

Individual traders typically engage in manual trading given the lower barrier to entry in comparison to algorithmic trading for which the trader must learn how to program. The greatest advantage of algorithmic trading is the removal of the human aspect from trading. As discussed in Section 2.1.1 each trader must develop a strategy suited to his personality and must adhere to his trading rules. A trader who has difficulties following his own rules can delegate the management of the trades to an algorithm that will follow the rules without exception.
Since trading algorithms run continuously in real time, they grant the trader access to the entire trading session rather than the portions of the day where he is free from other responsibilities. This is especially helpful for non-professional traders who may be otherwise unable to trade outside of a very limited timeframe.

In contrast, manual trading gives the trader complete control over the trades that are executed. On the one hand, while the trading system adheres to the rules, it may not consider external sources of information that could provide an indication that the rules no longer apply. On the other hand, the trader would immediately recognize these conditions and cease to apply the strategy. A properly designed trading system should recognize when the strategies that it follows no longer carry an edge.

2.8 Fundamental vs Technical Trading

Trading properly requires research in order to determine whether to trade and when to trade. The two prominent trading analysis techniques are fundamental and technical trading. Fundamental trading encompasses the research of what is known as the fundamentals of a company. Fundamental traders are typically divided into two camps: Value and Growth investors. Value investors will research the financials of a company in order to determine whether it is fairly priced by the market. The indicators used by value investors include Price to Earnings Ratio (PE Ratio), Dividend Payouts, Corporate History, Research and Development, and the state of the market. Growth investors also look at the fundamentals of the company, but they focus on the rate of earnings and revenue growth. They look for “rockets ready for takeoff.” These are companies that are on the verge of dramatically increasing their share price [17].

Technical analysis focuses solely on reading the charts and price movement indicators in order to recognize patterns that predict the future movements of the price. In theory, a technical trader does not need to know the name of the company in order to successfully trade its stock. The chart and its indicators is all that a technical trader needs in order to successfully trade a stock. Typical indicators applied to technical analysis include moving averages and ranges.

In reality, most traders use a combination of fundamental and technical analysis in order to maximize the potential to win. Fundamental analysis will tell the trader which stocks to trade and technical analysis will tell him when to trade [17].

Before we list various well-known trading strategies and the ones we created as part of the project, it is necessary to introduce the tools which can used to develop those strategies. Furthermore, we give a brief introduction to the trading platforms which enable a system to place its trades. The following section provides this introduction, and specifically enumerates the platforms and tools we used to build our own strategies.

3 Trading Systems

There is a lot trading systems-specific knowledge and methodology which must be understood before implementing such a system. This section will introduce the reader to the aspects of trading systems which are requisite to their development. Section 3.1 provides a summary of trading accounts, and Section 3.2 briefly reviews the capabilities of various trading sources. Sections 3.3 and 3.4 study several trading techniques and time periods respectively which are the foundation of any successful trading strategy. Finally, Section 3.5 introduces the different parts of a trading strategy which we took into consideration with our own designs.

3.1 Trading Platforms and Brokerage Accounts

A brokerage is a company that acts as a middleman between buyers and sellers of financial assets. Brokerage companies demand a compensation for providing the service to connect both parties and for offering a platform on which to perform the trade. The trading platform is the software on which the buyers and sellers connect. This report covers the development of systems of trading systems on one of the most popular online brokerages: TradeStation. Other popular brokerages and trading platforms include Interactive Brokers, E*trade and Robinhood. Robinhood is increasingly popular since, at the time of writing, it allows traders to trade with no transaction costs. The company instead makes money from the customer's cash balance and through margin lending.

3.2 Data Sources

Traders may obtain market data through several online sources. Below, we list several popular data sources, some of which were consulted for the purpose of this research.

Table 2 – Data Sources

Data Source	**Assets**	**Free**
Oanda	Forex	Yes
Quandl	All	No
TradeStation	All	No
Yahoo! Finance	Equity	Yes
AlphaVantage	FX, Equity, Crypto	Yes
Google Finance	Equity	Yes
Quantopian	Equity	Yes
SimFin	Equity	Yes

3.3 Trading Styles

Trend trading involves the use of technical analysis to identify temporary trends in price data and to exploit those trends for profit. Many trend trading strategies make use of different averaging schemes and analyze the behavior of those averages to identify signals for trading.

Support and Resistance is another form of technical trading where temporary floors and ceilings are identified in the price data. These floors and ceilings may be the result of the average trader finding comfort in those prices. For example, the average trader has a preference for round numbers and so it is likely that many trailing stops will be placed at an even price slightly lower than the current price. As the price approaches that floor the triggering of the trailing stops pushes the price back up giving the impression that a band of support exists at that price. Ceilings result from traders taking profits and short sellers buying to cover [18].

The validity of support and resistance is often questioned because any random price was a support or resistance at some point in the history of the instrument. If any random point can appear to be a support/resistance at some point in time then how can a trader be sure that a particular price is actually a support/resistance level and not just a random event?

Institutional traders sometimes sweep the stops meaning that they push the price down until the trailing stops placed by traders are triggered thus giving the impression of a new runup in price and creating a new opportunity to profit for the institutional trader.

Gap Trading describes the trading of the naturally occurring gaps in price over time. There exist multiple types of gaps in the stock market:

1. Breakaway Gaps: Signal the beginning of a new trend.
2. Exhaustion Gaps: Signal that a price pattern is coming to an end.
3. Continuation Gaps: Occur when traders are rushing to take advantage of price pattern.
4. Common Gaps: Do not form part of a price pattern. These gaps include the overnight gap and small gaps throughout the trading day outside a price pattern.

Before approaching gap trading, it is important to understand why gaps occur in the market. Breakaway gaps signal the beginning of a new trend. They are the result of a sudden surge in demand following an event such as the publication of a strong earnings report. In a breakaway gap, the price breaks through an area of support/resistance. The move is often followed by increased trading volume resulting from a combination of traders on the wrong side of the move seeking to cover or to sell their position and from enthusiastic traders attempting to ride the new trend.

Breakaway gaps are followed by runaway gaps which take place in the middle of a trend. Runaway gaps are created by a new surge in traders who were initially wary about the formation of the trend but are not convinced that the trend is strong and safe to ride.

Exhaustion gaps occur near the end of a trend. The price begins to slow down and traders who cannot get enough of the stock see this as another opportunity to profit from the trend. The new sudden increase in demand creates a gap in the price and soon after the stock finds itself at an unsustainable valuation where all demand has dried up. A quick drop in price follows and a new trend is formed on the downside. Since both runaway gaps and exhaustion gaps are so similar it is difficult to differentiate them as they happen. It is often easier to identify them in hindsight. Examples of the four types of gaps are available in the appendix.

3.4 Time Frames

Defining the time frame of a trade, both before, during, and after execution, is vital for a successful strategy.

While performing technical analysis traders have access to price data at multiple time frames depending on the trading platform used. TradeStation offers data ranging from monthly down to five-minute ticks. A trader may benefit from observing the price data at different time frames because larger trends are not visible in the smaller time frames. Even though the market may look like it is in a downtrend in the 15-minute candlestick bars, the daily candlestick bars may reveal that the price is in an uptrend. Since the price will typically align with the larger trend, the trader may decide to wait for the price to start climbing before entering a long position. Typical time frames include 5 Min, 15 Min, 1 Hour, 4 Hour, Daily, Weekly, and Monthly.

Short time frame trading styles include swing trading and scalping. Scalping is a trading style characterized for its extremely short duration. Scalpers may hold a position from seconds up to a few minutes. These trades are typically large in size since scalpers attempt to profit from many small price movements. Scalping requires access to near real-time data and constant monitoring of the price.

In day trading all positions are closed by the end of the day. Traders who engage in day trading may wish to avoid the volatility of overnight positions. Unlike day trading, swing trading involves positions that do not necessarily close overnight and may last up to several months but typically last a few days. Swing traders utilize technical analysis to identify and take advantage of short-term momentum in stocks.

3.5 Components of the Trading System

An effective trading system must possess certain components in order to maximize the possibility of having a positive expectancy. Properly designed and optimized entry and exit rules monitor the market for the optimal conditions in which to trade a particular strategy. Properly designed rules should never miss a signal for which they were designed to detect. Trading system designers should be aware that entry rules are the least important part of the trading system and should not dedicate a disproportionate amount of time into their development. Position sizing and exit rules are comparatively more important.

Position Sizing and Position Management tools determine what percentage of the pot of money available to the strategy should be risked in a particular trade. Position sizing rules may alter the amount of money to bet following the Kelly Criterion, for example. The Kelly Criterion was developed as a method for gamblers to optimize their bets in order to attempt to guarantee a profitable session. The Kelly Criterion alters the position size based on the past success of the strategy. Should the strategy encounter a period of low success the impact on the account would be minimized with the progressively decreasing bets. The opposite is true when a period of high success is encountered. Position sizing tools are extremely important as they minimize the incurred risk.

System Monitoring Rules monitor the performance of the system as a whole in order to determine whether the system is apt to continue trading.

Asset Allocation rules determine what portion of the total available money supply to assign to each strategy within the system. The most successful strategies would receive a larger percentage of the pot as the least effective ones get phased out.

A successful trading strategy is defined not only by its components but also by how it is optimized and tested. It follows that at least a cursory knowledge of trading system analysis must be applied to a successful trading system. The next section will expand on these methods and provide explanations on how to implement them. This knowledge was enough for us to develop the individual strategies that comprise our system.

4 Optimizing and Analyzing Trading Systems

Successful trading strategies must be optimized and tested properly to sufficiently fit the markets they are designed for. A good trading strategy which has not been properly will perform as poorly as a bad trading strategy, if not worse. Of course, it is necessary to analyze a trading system to see when it must be updated, and to gauge its performance in a larger portfolio. Thus, we present a review of optimization and analysis techniques for trading systems. Section 4.1 studies optimization techniques for trading systems, with Sections 4.2 and 4.3 describing specific techniques in greater detail. Section 4.4 shows how those optimization techniques can be accessed in TradeStation. Finally, Sections 4.5, 4.6, and 4.7 describe analysis techniques to assess the performance of a strategy.

4.1 Optimization

Trading systems often implement mathematical calculations with variables whose optimal values are uncertain and thus require iterating through many possibilities, far too many to try manually. The TradeStation Optimizer performs this process automatically. The designer specifies the range of values over which to iterate and over what steps and the optimizer finds the optimal combination that yields the highest value for the selected objective function. The objective function describes the goal of the optimization. For example, find the highest profit, the lowest intraday drawdown or the highest expectancy score.

TradeStation offers two forms of optimization: Standard and Walk Forward Optimization. Standard optimization will iterate through all of the possible combinations to find the values that best fit the objective function. In this form of optimization, the values that are found are those that yield the best results over the entire dataset. This assumes that the future price data will behave very similarly to past price data. This is not necessarily the case meaning that a set of values that was optimal for one period of time is not necessarily optimal for the next. The Walk Forward optimizer separates the data into periods to find multiple sets of values that best fit each period. This type of optimization often yields better results.

4.2 Walk Forward Optimization

There exist two approaches to Walk Forward Optimization within TradeStation: Rolling Walk Forward and Anchored Walk Forward Optimization. In a rolling walk forward optimization the period over which the optimization occurs is selected starting from the most recent available bar and back until enough bars are included. The anchored optimization begins at a fixed bar and moves forward until the most recent bar. This means that each period progressively includes more data although the data is older in comparison to that in the rolling optimization.

4.3 Overfitting and Choosing Data for Optimization

Optimizing can result in overfitting meaning that the selected values perfectly fit the data in the given period, but hold no predictive power over future price data. Overfitting will typically occur when the entire dataset is included and a large set of variables are available for optimization. In order to reduce the possibility of overfitting, a portion of the dataset is excluded from the optimization. Another way to reduce the chance of overfitting is to include a large enough dataset relative to the number of parameters available for optimization.

In order to determine whether the optimization resulted in overfitting, the dataset may be split into two sets one for training and one for validation. The results of the training are applied to the verification dataset. If the performance with the verification dataset is comparable to that of the validation set then the strategy is not overfitted. If the performance decreases significantly then it is likely that the strategy was overfitted to the training dataset.

4.4 TradeStation Performance Report

The TradeStation Performance Report provides an overview of the performance of a strategy over the selected trading period. The report consists of seven sections, which provide data on the execution of the trades as well as on the overall performance of the strategy. A summary of the most important sections for the purposes of this research is provided below:

Performance Summary:

Total Net Profit, Gross Profit, Gross Loss, Profit Factor, Total Number of Trades, Percent Profitable, Ratio Avg Win:Avg Loss, Annual Rate of Return, Sharpe Ratio, K Ratio, Trading Period, Maximum Drawdown (Intra-Day) and Max Trade Drawdown.

Trade Analysis:

Time Averages, Outliers, Efficiency Analysis.

Performance Graphs:

Equity Curve Line

Trade Graphs:

Entry and Exit Efficiency, Maximum Adverse/Favorable Excursion Percent.

4.5 Expectancy, Expectunity and System Quality

Expectancy, as described by Van Tharp, is simply how much a trader can expect to make on average over many trades. The best way to visualize the expectancy of a system is to look at the expected earnings per dollar risked.

$$Expectancy = \frac{\sum_{i=1}^{n} (WP_i * WF_i) - \sum_{i=1}^{n} (LP_i * LF_i)}{(Average\ Loss)} \qquad (1)$$

$$WP_i = Winning\ Probability$$
$$WF_i = Winning\ Payoff$$
$$LP_i = Losing\ Probability$$
$$LF_i = Losing\ Payoff$$
$$n = Number\ of\ Trades$$

Expectunity is the result of the combination of expectancy and opportunity of a system. Opportunity is simply the frequency with which the system trades. The expectunity is defined below. Note that Opportunities is the number of trades over the time period for which the expectunity is calculated.

$$Expectunity\ = \frac{\sum_{i=1}^{n}\ (WP_i * WF_i) - \sum_{i=1}^{n}\ (LP_i * LF_i)}{(Average\ Loss)} * Opportunities \qquad (2)$$

It is extremely important to consider the expectunity of a trading system in order to ensure that the system is truly profitable and that the trader is not falling for the trap of prediction. Consider a system that is accurate 90% of the time, has an Average Winning Trade of $275, 10% Losing Trades and Average Losing Trade of $2700.

Expectancy = -$0.008 per Dollar Risked

The result of the expectancy calculation reveals that over time this system will result in negative profits even though the system is correct 90% of the time.

The System Quality describes the total profit/loss per dollar risked relatively to the total variability of the profit/loss per dollar risked. The system quality may be used to compare the performance of trading systems.

$$System\ Quality\ = \frac{Expectancy}{\sigma_{Profit/Loss}} * \sqrt{\#\ of\ Trades} \qquad (3)$$

$\sigma_{Profit/Loss}$ = Standard Deviation of the Profit/Loss

4.6 Monte Carlo Analysis

Monte Carlo Analysis creates a statistical distribution of possible outcomes using past data. When applied to trading, Monte Carlo analysis may be used to predict the likelihood of obtaining a particular trade sequence. The analysis takes historical trade data for a strategy, randomizes the order of the trades, analyzes the sequence and finally sorts its outcome by likelihood with a confidence level. Following a Monte Carlo analysis, although the trader does not know how the price will move in the future, the trader gains some certainty over the performance of the strategy in the future. For example, the trader may be confident that the strategy will result in overall profits over $10K with 85% confidence.

At the same time, he may find that a system that traded spectacularly in the past will result in negligible profits with 99% confidence meaning that the current results are extremely unlikely to continue.

4.7 Market System Analyzer (MSA)

For the purpose of this report, Market System Analyzer (MSA) was used to conduct the Monte Carlo Analysis of the developed strategies. MSA accepts TradeStation trade data and quickly performs Monte Carlo analysis on the given data. Sample results of using MSA are available for each strategy presented in the report.

All of the information necessary to develop a traditional system has been introduced. However, as we explain in our objectives, we also intended to use artificial intelligence, and specifically neural networks, to drive our strategies. Therefore, we provide an in-depth review of artificial intelligence as a discipline and its application to financial markets. The next section will introduce the reader to the basics of machine learning and will explain their applications to trading systems.

5 Artificial Intelligence and Machine Learning

We aimed to not only implement traditional strategies, but to also leverage artificial intelligence techniques to develop strategies and a system of systems. As a requisite, we needed to investigate the field of artificial intelligence as a whole and review how it as a discipline has intersected with financial research in the past. This section details our background research on artificial intelligence, and specifically the kinds of neural network techniques that guide our system of systems. Section 5.1 provides basic definitions on artificial intelligence, machine learning, and neural networks. Section 5.2 further clarifies the different types of machine learning, with specific time spent on tree search algorithms in Section 5.5. Sections 5.3 and 5.4 analyze the techniques to tune and test machine learning problems, with specific emphasis on feature selection. The remaining sections focus on neural networks. Section 5.6 introduces neural networks as a concept, while Section 5.7 features a discussion on sources of network error. Section 5.8 describes in depth how to counteract that error. Finally, Section 5.9 discusses different implementations of the basic neural network architecture, and Section 5.10 provides a small literature review on the financial applications of neural networks in the past.

5.1 Machine Learning Overview

Many applications in prior literature have proven the potential benefits of applying machine learning strategies to financial data. Machine learning as a discipline is “a field of computer science that uses statistical techniques to give computer systems the ability to "learn" data, without being explicitly programmed." [19] In this case, “learning” describes the computer’s ability to find predictive patterns within the data. [19] Since the first papers on machine learning were released in the late 1950s, an abundance of successful projects have proven the validity of machine learning. For example, machine learning has been used to determine the numbers represented in the MINST database, a set of handwritten digits from American high school students, released in 1998. [20] In 2011, IBM’s “Watson” computer won the TV game Jeopardy using machine learning techniques. [21] Within the last year, academia and the industry have reported major advancements in natural language processing, computer vision, audio generation, and autonomous vehicles using machine learning techniques. [22] These accomplishments speak to two points: first, that machine learning is one of the fastest growing fields in computer science, with many developments still in store over the coming years, and second, that machine learning has applications in a diverse set of fields where it may be useful to detect patterns in data. This widespread success speaks to the applicability of machine learning techniques with financial data.

5.2 Machine Learning Categories

Machine learning can be subdivided into two categories: Supervised and unsupervised learning. An algorithm performs supervised learning when it is trained on a set of X attributes or features, and a set of Y responses over N observations or samples. The goal for the model is to learn the underlying relationship between the X predictors and the Y response variable over N training observations and then be able to predict the value of the response Y when given new, unseen observations of the X predictors. An example of when these types of algorithms can be used for trading is when a trader has data on X indicators for a set of stocks and is interested in predicting the price Y of these stocks over a period of time when given new observations of the X attributes [23].

Supervised learning can be further divided into two types: Regression and Classification. Regression models are models whose Y response variable is a continuous or ordered variable. For example, an algorithm that tries to predict the price Y of a stock is a regression algorithm. On the other hand, Classification models are models whose Y variable is a categorical variable and their goal is to correctly assign new observations to defined classes Y. An example of this type of model would be a model that assigns each stock to either the class of stocks that "go up" or to the class of stocks that "go down". In this algorithm, there are two classes: the class that goes up, and the class that goes down. So rather than predicting the exact price of a stock, the algorithm is only predicting the direction of the stock's price in the future [23].

A model performs unsupervised learning when it trains only on a set of X features over N observations. There is no response variable Y. The goal of this type of algorithm is to find relationships between the X features, not predictions. It is typically used to find ways to visualize data and to discover groups among the data. An example of when a trader might use these types of algorithms is when the trader has a set of stocks and a set of X attributes for those stocks over N observations and is interested in discovering groups of "similar" stocks that have similar values for their X attributes [23].

The machine learning algorithms that are used to form strategies in this project are: Tree Based Classification Algorithms, Principal Components Analysis, and Neural Networks. Before algorithms are explained, an explanation of how to correctly implement these algorithms needs to be made.

5.3 Training and Testing

Training and testing an algorithm generally involves separating the dataset into a training set and a testing set where the algorithm is allowed to see the response variable Y during the training set and to "learn" the relationship it has with the X factors. After the algorithm has trained over the training data, it is then given the X factor data in the testing set and the algorithm uses the X factor test data to predict the Y response variable. To test the algorithm's performance, the algorithm's predicted response variable is compared to the actual response values in the testing set, which are the ones that the algorithm did not see.

The metric used to test a regression algorithm is the test mean-squared error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2, \qquad (4)$$

Where $\hat{f}(x_i)$ is the prediction that the model outputs, y_i is the actual value, and n is the total number of test observations. The goal is to minimize this quantity [24].

On the other hand, classification algorithms are tested using a variety of different metrics. The most common one is the test error rate:

$$Ave(I(y_0 \neq \hat{y}_0)), \qquad (5)$$

Where $\hat{y}_0$ is the predicted class label. We look for the model with the lowest test error rate [25].

However, the best way to evaluate a classification algorithm is by using a confusion matrix, as shown in Fig. 2. The confusion matrix is a table with 4 different combinations of predicted values. This method is particularly effective because it gives insight on more than just "right" predictions. It gives insight on the model's performance for predicting specific classes. It does so by displaying the true positives, true negatives, false positives, and false negatives.

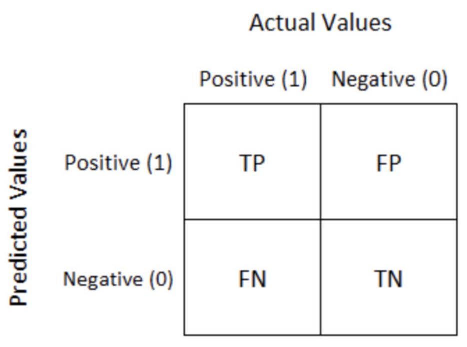


Fig. 2 Confusion Matrix

It is easier to understand true positives, negatives, and false positives and negatives if an example is used to do it. The example used here is the context of a classification algorithm that assigns stocks to the class that goes “up” and the class that goes “down” in the next 5 days. A true positive (TP) would be the instance when the model predicted that the stock would go up, and the stock went up. A true negative (TN) would be the instance in which the model predicted that the stock would go down, and the stock went down. A false positive (FP) would be an instance where the model predicted that the stock would go up, but in reality, it went down. Lastly, a false negative (FN) would be an instance when the algorithm predicted that the stock would go down, and it went up [26].

Based on these calculations, a couple of good performance measures can be derived. These measures are accuracy, misclassification rate, true positive rate, false positive rate, specificity, and precision. The explanations and the formulas for these measures are:

- **Accuracy** - a measure of how often the classifier is correct.
  (TP + TN)/total observations
- **Misclassification rate** -how often the classifier is wrong.
  (FP + FN)/total
- **True positive rate** -how often the model predicts stock that goes up when it actually goes up.
  TP/actual “up”
- **False positive ra**te -how often the model predicts that a stock goes up and it actually does down.
  FP/actual “down”
- **Specificity -** how often the classifier predicts that a stock goes down, and it actually goes down.
  TN/actual “no”
- **Precision** - how often the model is correct whenever it predicts that a stock goes up.
  TP/predicted “up”

These measures are valuable to assess the behavior of the model and adjust parameters accordingly [26].

5.3.1 Training and Testing Splits

There are many ways in which a dataset set can be split for training and testing. However, the best way to be certain about an algorithm's performance is to have multiple training and testing instances which are independent of each other. Having each testing and training observation set be independent of each other is important because it ensures the robustness of the algorithm and avoids overfitting. There are a couple of techniques that achieve both the aspect of having multiple tests and having the tests be independent of each other. These techniques are cross-validation and bootstrap [27].

There are various types of cross-validation. *Leave-one-out cross-validation (LOOCV)* works by splitting the dataset into two parts. It uses a single observation as the test set and the rest of the observations are the training set. We iteratively train and test the data always picking a random observation to be the test set, until *all* of the observations have been tested. Drawbacks with this approach are that it is expensive and time-consuming to implement if the dataset is very large, and if the observations are in chronological order, future observations can be used to feed information to the algorithm and hence "cheat" its way to accurate predictions. A way to work around this issue is to make sure that each random observation is only trained with data in the past [27].

Another type of cross validation is *k-fold cross validation.* This method randomly divides the data set into k groups of approximately equal size. The first group is treated as a test set, then the rest of the groups are used to train the data. This is done iteratively until all groups have been the test set. This method is far less expensive from a computational power standpoint and consistently yields better estimates of error than LOOCV. Recalling from what is mentioned in the bias-variance tradeoff, LOOCV has a very high variance compared to *k-fold* because it averages the outcome of models trained on an almost identical dataset. This causes high variance because it means that the average of each of the predictions is highly correlated, therefore a minor change in the data can skew the predictions. On the other hand, *k-fold* cross-validation averages the predictions of datasets that are less correlated with each other because the training sets are not as similar. This means that the test set prediction estimates that *k-fold* outputs are more consistent than the ones output by LOOCV. In addition to all of these implications, it is recommended to use *k-fold* using 5 to 10 partitions because these splits are the ones that have been shown, empirically, to yield the best test error rate [27].

Bootstrap is a very powerful method that can be applied in many areas. It is useful because it allows us to use computing power to emulate the process of obtaining new sample datasets, so that we can obtain more robust estimates without the need for more data. Instead of obtaining different, independent datasets, we get distinct datasets by iteratively sampling observations from the original dataset. This sampling is done with replacement, which means that the same observation may appear more than once in the bootstrap data set. The illustration below helps to visualize the bootstrap approach. [27]

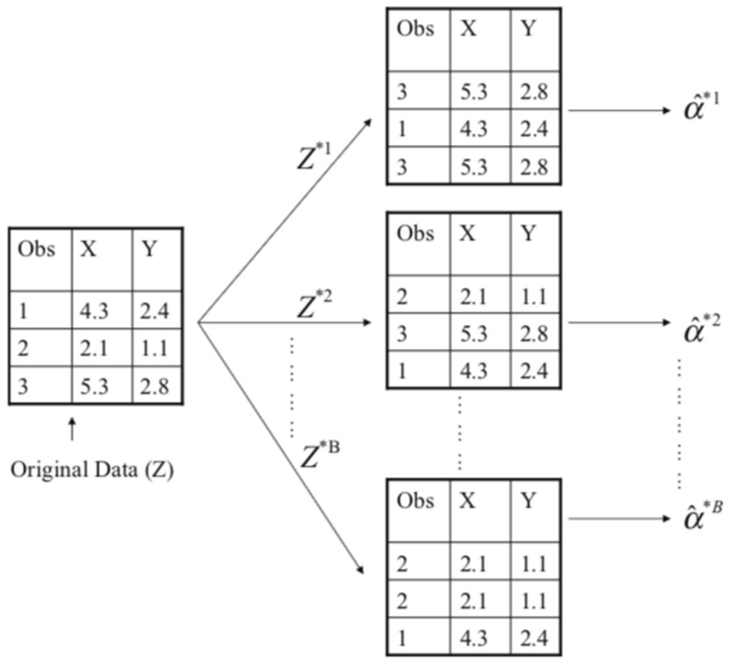


Fig. 3 - Bootstrap

In the illustration above, bootstrap is performed on a dataset containing 3 observations. Each bootstrap dataset contains the same amount of observations in the original dataset, sampled with replacement.

5.3.1 Training and Testing in Time Series Data

Financial data is overwhelmingly time series data. This means that the observations are in chronological order. The implications this has on training and testing algorithms are one of the reasons for which machine learning methods in finance are quite challenging to both develop and to test. Methods like bootstrapping and cross-validation do not work because they ignore the fact that the observations are in chronological order, so a portion of future data can be used to train and "predict" an observation in the past. This is problematic not only because it can give the algorithm a false measure of good accuracy, but because the predictive power of features can change over time, and thus different splits can yield very different measures of accuracy just because the most predictive predictor in some of the observations holds absolutely no predictive power in another set of the observations.

There are ways to work around this predicament. It requires the algorithm creator to be conscious of what parts of the algorithm, the training and the testing process are sensitive to the time index. For example, it is wise to be sure that whenever creating testing and training splits, the training set is always composed of data that is prior to the observations in the test set. That way the algorithm is only being tested with “future” data.

5.4 Feature Selection and Feature Engineering

Feature Selection is the action of selecting the most predictive predictors or features (x) from the set of predictors and to use those select predictors for the training and testing process of the algorithm. The amount of features used in an algorithm is important because it is directly associated with the performance of the algorithm.

With enough data, there should not be a limit on how many features can be used. However, there is not an infinite supply of data, and thus having too many features can hinder the performance of the model because they feed unnecessary noise to the data that the model has to be trained on. When this happens, the model finds insignificant patterns and overfits the data. This causes the model to have poor performance when it is tested on unseen data. There are various ways to deal with this.

Some machine learning algorithms are already designed to perform feature selection by themselves. Some algorithms, however, require that only the most predictive features be used to train them and require that the algorithm designer perform feature selection. The feature selection methods that were used in this project are recurrent neural networks, principal components analysis, random forest classification, and boosting tree classification.

Feature engineering is the action of doing computations or calculations with the already existing predictors to “create” more predictors or features (x). The goal is to create features that hold more predictive power than the original predictors and improve the performance of the algorithm on testing data. It is recommended to perform feature selection before performing feature engineering.

This process of feature selection and engineering requires domain knowledge, creativity, and is often the most important step to ensure that the algorithm can actually predict unseen observations. An example of when a trader might use feature selection and engineering is when the trader is looking for indicators (RSI, MACD etc) or creating new indicators to increase predictive power on the movement of a stock price.

5.5 Tree Based Classification Algorithms

5.5.1 Decision Trees

Decision trees can be applied to both regression and classification problems. A decision tree involves segmenting the predictor space into a number of regions. The segmenting rules are easily represented as nodes in a tree where each node separates the regions in different branches. In the example shown in Fig. 4, a decision tree is being used for classification. Each node represents a segmenting rule and the outcome of each rule is a branch that either goes to another segmenting rule or to the ending region or class. These trees are typically drawn upside down, in the sense that the leaves or segmented regions are at the bottom of the tree [28].

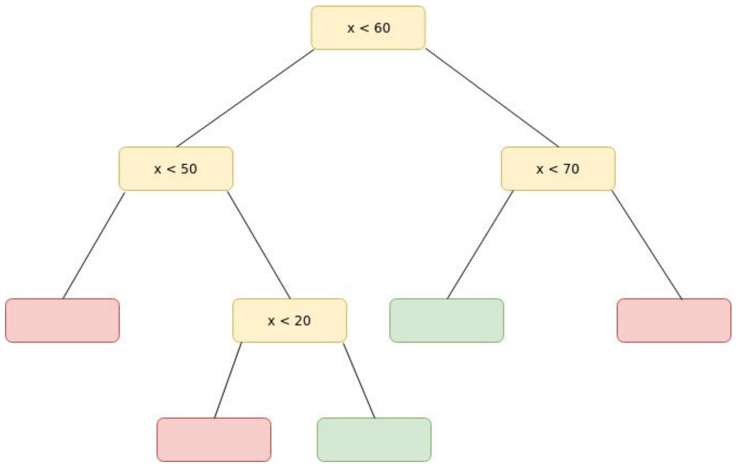


Fig. 4 - Decision Tree

5.5.2 Classification Trees

In the classification setting, the trees use segmenting rules to assign data to different classes. So instead of predicting a quantitative response, it predicts a qualitative one. A common example of this is when a trader uses a classification tree to attempt to classify stocks whose future returns are either the class that goes "up" or the class that goes "down". The image below shows the splits for a classification tree where each split assigns the points to either the points in the red class or the points in the green class [28]. Note that axes X1 and X2 are arbitrary, and could signify any two features.

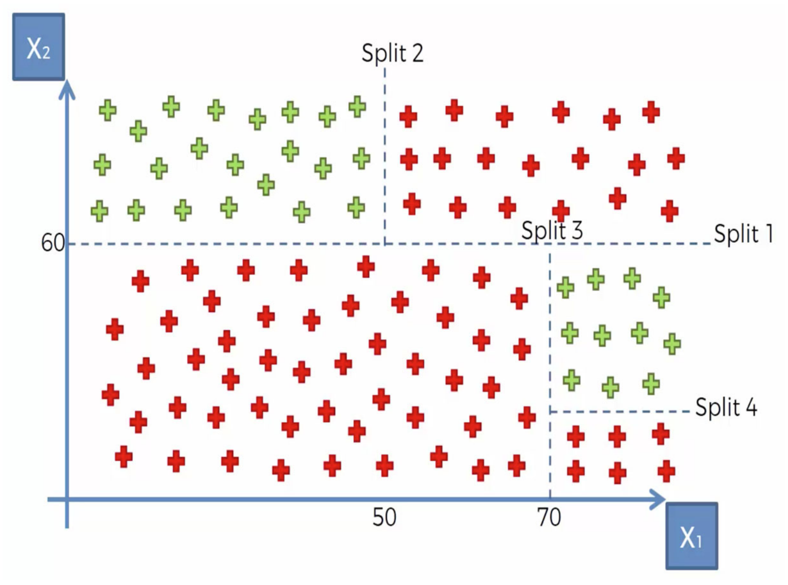


Fig. 5 - Split Classification Tree

5.5.3 Advantages and Disadvantages of Trees

In general, it is easy to visualize and interpret the output of a decision tree algorithm, and it can also handle qualitative predictors without the need to convert them to numbers. However, trees tend to overfit predictions too much. As such, a small change in the data can cause a significant change in the model's behavior. This is because the tree only goes through the data once, so it will pick up any random patterns along the way to help it make its splits. Another disadvantage of tree-based methods is that compared to other methods, their level of predictive accuracy is lower [28].

However, when many trees are combined and used for predictions in a single algorithm, the prediction accuracy improves. These practices of combining many "weak" algorithms to form more a more accurate algorithm are called ensemble methods. For tree-based methods, ensemble methods are bagging, random forests, and boosting. Although these ensemble methods can be applied to other algorithms, they are particularly useful for decision trees because they combine many "overfit" trees and combine their results to average out the underlying pattern in the data [28].

Bagging works by taking multiple samples of the dataset and training a decision tree on each sample. After that, the average of the predictions is taken as the prediction output of the algorithm. For the classification setting, a "majority vote" among the predictions and the most commonly occurring prediction is taken as the prediction output [28].

The random forest algorithm consists of randomly sampling a number of observations, a number of n predictors from the full set of predictors and making a tree go through that data set. At each sampling, a new set of predictors is chosen. This is the main difference between this algorithm and bagging. This allows the algorithm to be more robust because it forces the algorithm to train itself on predictors that are different, most of the time, and decorrelates the predictions that each tree has and thus the average of their predictions is more reliable [28].

In boosting, each tree is grown sequentially by using information from previously grown trees. Boosting does not involve getting multiple random samples of the data. Instead, it fits every tree on a modified version of the original dataset. Another thing that distinguishes this approach from the rest is that it fits each tree to the residuals (actual y response value minus predicted y response value), instead of the response Y. Each of these trees has a small number of nodes. Iteratively fitting these small trees to the residuals from the previous tree will improve the underlying function in areas where it is not performing accurately. One potential drawback of this ensemble method is that if the number of trees fitted is too large, the model can overfit. The best way of avoiding this is by using cross-validation [28].

5.5.4 Principal Components Analysis

Principal Components Analysis (PCA) is an unsupervised learning method that aims to create linear combinations of the predictors to summarize the data set with a smaller set of predictors that together explain most of the variability in the data set. This method is often used to visualize data or used in conjunction with a supervised learning model where the predictors are the principal components calculated in PCA. If the latter approach is used, it is important to exclude the target variable from the set of predictors that PCA can use to make the linear combinations and to exclude the principal components that do not attribute any variability to the data set as this will only make the algorithm more complex and defeat the purpose of reducing the number of predictors. These components can be screened out by computing their Proportion of Variance Explained (PVE). Only the components with the highest PVE should be selected [29].

5.6 Neural Networks

Artificial neural networks (ANNs) are a specific implementation of supervised learning that have seen extensive use in the realm of financial data [30]. Neural networks are designed to mimic the acquisition and organization of knowledge in the human brain [30]. In doing so, the design of an ANN reflects the assembly of neurons found in the brain which function as both processing and memory units [31]. The basic architecture of an ANN thus consists of layers of artificial neurons. Each artificial neuron takes a series of weighted input signals, combines them, and applies a non-linear “activation function” to the resulting signal [31]. Fig. 6 provides an illustration of this design:

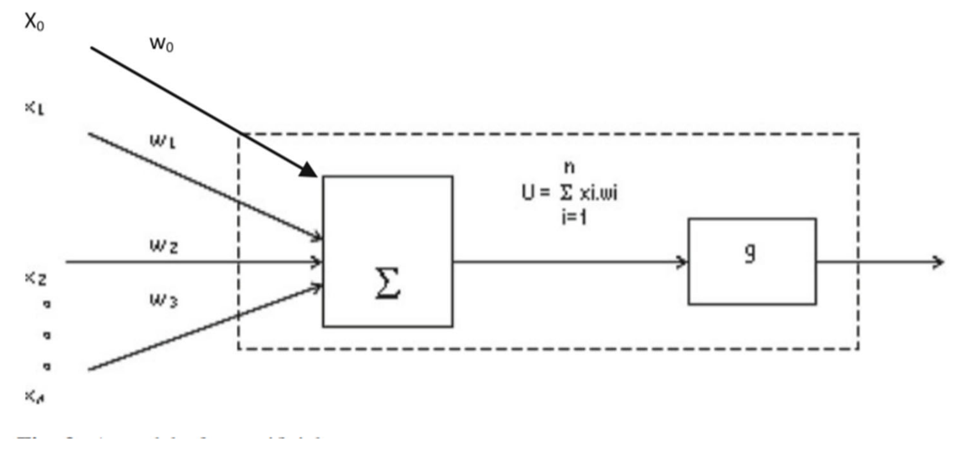


Fig. 6 - Activation Function

Note that X_0 is a bias value which does not originate from a neuron. The nonlinear activation function is essential to mimic the learning capabilities of the brain; otherwise, a chain of these neurons would be equivalent to a single linear transformation [31].

Neural networks are most popular in cases where data with many features needs to be classified; for example, in the case of the MINST database where input images composed of hundreds of pixels needed to be classified into numbers [20]. Applications in finance ranging from “pattern matching, classification, and prediction, such as bankruptcy prediction, loan evaluation, credit scoring, and bond rating” are potential candidates for neural network technology [30].

The activation functions typically map negative inputs to -1 and positive inputs to 1 [31]. This is not always the case; often, activation functions can scale to infinity, or between 0 and 1. The pictures below show some sample activation functions used in ANNs:

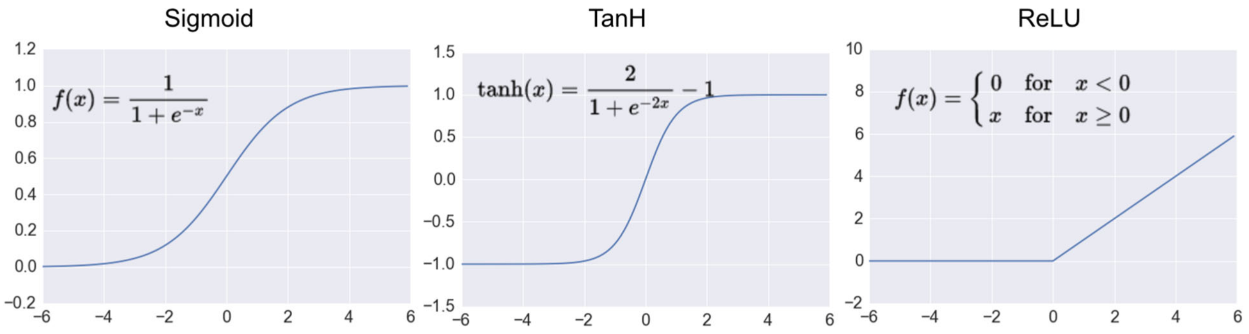


Fig. 7 - Activation Function Examples - Examples of the sigmoid, tanh, and ReLU activation functions. Note that each function outputs values between -1 and 1; these functions can be considered as filters that accept large positive and negative values and reject values near 0. **(18)**

Since each neuron takes a bundle of input signals and produces a single output, artificial neurons can be chained together in layers, where a previous layer of neurons provides the input signals for all the neurons in the next layer [31]. The figure below demonstrates this principle:

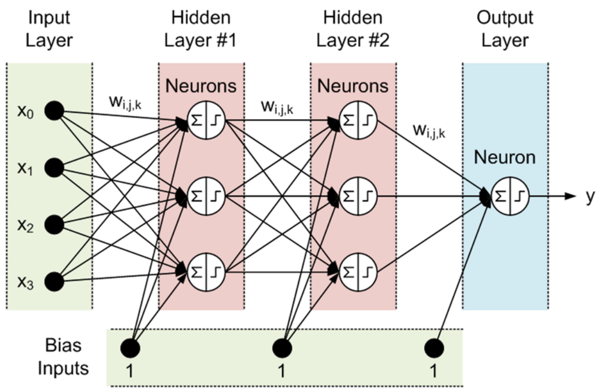


Fig. 8 - Illustration of a neural network with multiple hidden layers. Each node sums the weighted inputs of the nodes from the previous layer and applies an activation function. [21]

By combining these artificial neurons into layers, and connecting the layers together, a neural network can be trained to mimic any decision function. As seen in the diagram, the input layer here would consist of predictive features from which the network would drive patterns, and the output layer would be a representation of the predictive output, whether a binary classification or a numerical regression [31].

The training procedure of a neural network adjusts two elements of each neuron: the input weights and the activation bias. The input weights are the weights described previously which are applied to each input entering a given neuron. The activation bias pertains to the activation function described previously. The bias applies a horizontal translation to a neuron's activation function, setting a shifted range for what is classified in the -1 to 1 range. In adjusting these two features, a neural network may adapt itself to any decision boundary [31].

For a given neural network to learn a decision boundary, the network must train itself with data that has been previously classified. This process, thus requires the network designer to have previous examples of the data of interest that have been classified [31]. For example, in the case of the MINST set, the neural network would train with example images that have already been tagged with the proper digit [20]. Training data is released in batches to the neural network, which incrementally updates its input weights and activation bias to minimize the error between its predicted classification and the pre-marked classification of the training data. Training stops when error improvements are marginally small, resulting in a trained neural network [31].

5.7 Sources of Error in Algorithms

When designing and training a neural network, many potential sources of error must be accounted for to achieve satisfactory accuracy. A substantial portion of the error in neural networks is caused by the network “overfitting” and “underfitting” its training data. For a model M and a set of data L, we say that M overfits L when M learns the model it trained with too well, and cannot generalize to examples in L. While it is essential for a neural network to find patterns in the training data which can be extrapolated to the data set as a whole, if given enough time and power, the network will only memorize each of the training examples, and neglect to find general trends in the data itself. A typical sign of overfitting is if small changes in L cause large changes in the accuracy of M [32].

Given the same model M and dataset L, we say that M underfits L if there is “too much behavior” that is accepted by M that is not in L. Whereas in the case of overfitting, where M learned the training data too well, in underfitting, M learned the training data too generally, i.e. without finding enough patterns in the training data [32]. Fig. 9 contains graphical representations of an overfit dataset and an underfit dataset:

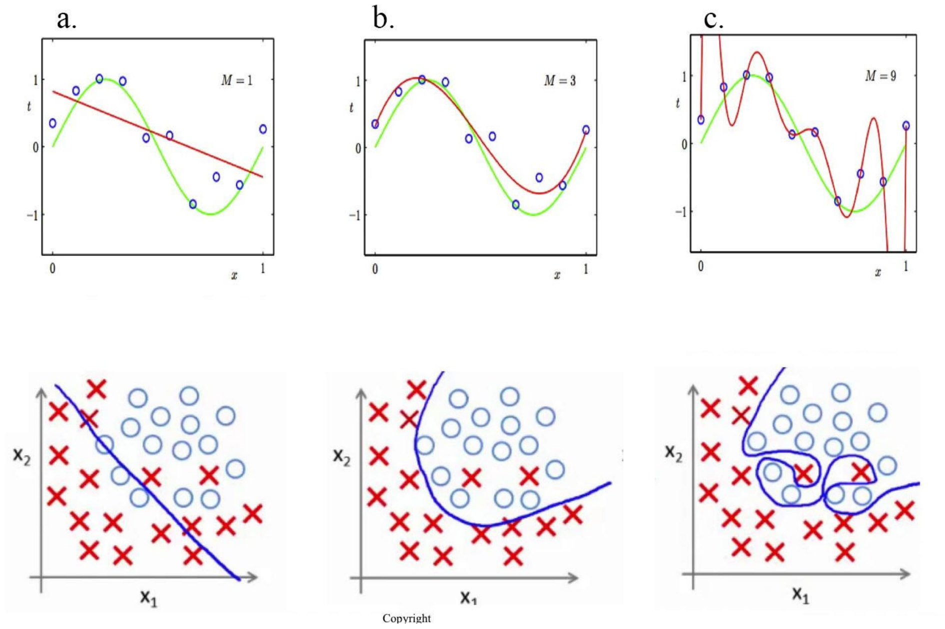


Fig. 9 - Dataset Fit - a. An underfit dataset. b. A well-fit dataset. c. An overfit dataset. **(11)**

As seen in Fig. 9, the underfit model failed to find a general pattern in the training data, while the overfit data oversampled individual examples in the training data. An important task of neural network engineers is to balance between overfitting and underfitting to achieve a well-fit example [32]. This phenomenon of balancing between overfitting and underfitting to achieve a well-fit model is referred to as the “Bias Variance Tradeoff”.

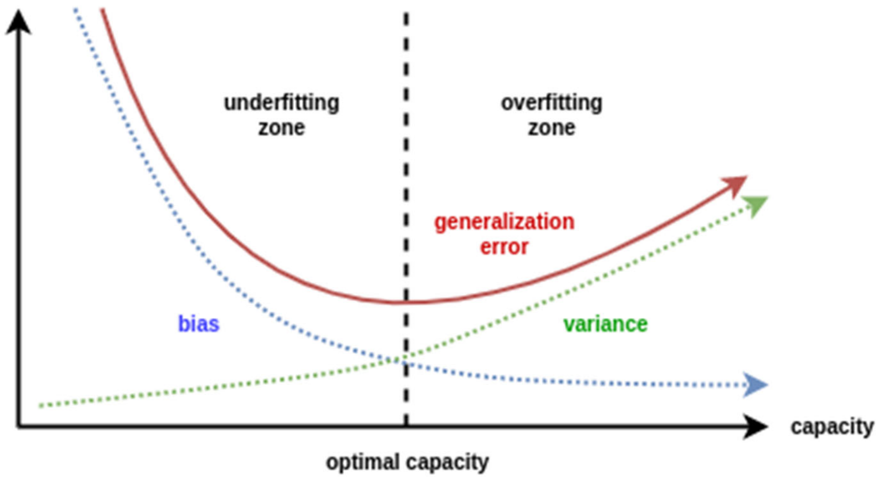


Fig. 10 - Model Error Relationship to Bias and Variance

The graph above gives us a representation of the relationship between the error in a model’s predictions due to bias and the error due to variance. When a model has a high bias, it is said to underfit, have low complexity, and its predictions are consistently misrepresenting the real world. When a model has high variance, it is overfitting, more complex than necessary, and its predictions are not consistent every time the model is replicated because it is learning random patterns in the data rather than learning the underlying information. This makes the model’s predictions exhibit high variance. The last source of error is called “irreducible error”. This error, as its name suggests, cannot be reduced because it is caused by the inherent randomness of the real world. The rule of thumb is that as a model’s complexity increases, its bias decreases and its variance increases. The optimal model is the one with a level of complexity that achieves the lowest combination of error due to bias and variance. Despite there being an “optimal model”, there will always be error in the model.

5.8. Solutions to Overfitting and Underfitting

Many different factors may contribute to both bias and variance. For example, a decision tree with too many nodes will have a lot of variance, however a decision tree with too few nodes will have too much bias. One such factor is the size of the neural network, i.e. the number of nodes and layers in the network [33]. Typically, a neural network with too many layers and nodes will overfit a dataset, while a neural network with too few nodes and layers will underfit a dataset [33]. The effects of network size on overfitting is especially pronounced in cases where the number of weights exceeds the data samples in the dataset [34]. Some subsets of the dataset however may be more tolerant to overfitting than others, especially in cases where backpropagation is the error minimizing method [34]. Since the architecture of the network is set upon implementation, manual tweaking of the architecture can reduce overfitting and underfitting. In cases of overfitting, the number of hidden layers and nodes per hidden layer can be reduced; in cases of underfitting, they can be increased. Additionally, techniques can be applied to a given architecture to reduce its overfitting via penalizing the weights of the network [35], [36]. One common method involves L1 and L2 regularization of both the input weights and the bias of each node [35]. In this technique, inhibitive terms are added to the error minimizing functions for each node, limiting the effectiveness of error reduction at each round of network training. By preventing patterns from being learned "too well", overfitting is reduced [35]. Other variations of regularization exist; in soft-weight sharing, for example, weights in a given layer are given an averaging term based on the weights of neighboring nodes [35]**.** In some cases, dropout may even be implemented, where nodes deemed excessive or unnecessary by an evaluation algorithm will be disconnected from the network [36].

Yet another cause for overfitting comes from a high ratio between input features and data samples. If there are not enough data samples to cover each input feature, a model may learn to discriminate patterns based on frequencies in different dimensions, rather than finding a pattern between all dimensions [37]. For example, in the case where there are N samples and N features, where each sample is uniquely present in a given feature, the model could learn to make binary classifications based on merely the presence those features or lack thereof, since there is not enough data for the model to find a pattern amongst all dimensions [37]. It is important that each input feature adds relevant, linearly independent data to the model, and that enough data exists to provide a non-trivial classification for the feature. Furthermore, it is important that the dataset is representative of the system being modeled [37].

While there is no simple remedy for high dimensionality of input features, there are techniques which serve as dimensionality heuristics [38]. Once such technique is Principal Component Analysis (PCA), which finds the directions of greatest variance over the input dimensions and plots each data point by its coordinates along those directions. By iteratively modifying the dimensionality of the set and applying PCA, a balance between dimensionality and variance can be achieved [38]. Various techniques exist which implement PCA; we used

Multi-Dimensional Scaling, ISO Map, and t-SNE throughout the project [39]. Other more advanced techniques exist which reduce dimensionality in a deterministic way; adversarial networks can be trained, for example, to scale a feature set between high dimensionality and low dimensionality. These are called autoencoders [39].

Finally, there are techniques beyond the design of the neural network itself which provide a holistic representation of the model's accuracy. Since the input weights and activation bias of each node are randomly initialized (typically over some distribution), the local error minima they reach during training may vary, even when the dataset training the network is the same. The performance of a single instance of a model is not enough to assess the accuracy of the modeling as a whole. Cross validation methods are typically employed to avoid these cases where abnormal training performance dictates the overall accuracy of the model [40]. In k-fold cross validation, for example, the dataset is split into a number of "folds", which are evenly sized buckets of training data. For N folds, N networks are trained; one fold is used to test the accuracy of each model, while the rest of the folds are used as training data. With N unique training and test set combinations, assessments about the model accuracy as a whole are reasonable [40].

5.9 Neural Network Implementations

While the architecture detailed above presents the fundamental design of neurons in a neural network, most implementations will add other features to the underlying design. Below, two implementations of neural networks are presented; both are used to implement the system of systems.

5.9.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) perform well with image processing and categorization problems. The central principle of the CNN is that through encoding assumptions about the locality and density of objects in image classification, training with very large datasets can avoid being prohibitively expensive while exhibiting only slightly worse performance than the regular feed-forward network [41]. CNNs have two additional types of layers beyond the neuron layers described above (called fully-connected layers in the context of the CNN): convolutional layers and pooling layers [41]. Convolutional layers apply "convolutions" to tiled areas of an image to extract features; given an N by N pixel tile, a convolution can be considered a filter that generates a feature from those pixels [42]. Pooling layers reduce dimensionality by aggregating values among neighboring pixels into a single node; for example, a pooling layer could reduce 2x2 pixel chunks to a single value [42]. Finally, the manipulated layer is passed to "fully-connected layers", serving as the feed-forward neural network described above.

Convolutional neural networks are of particular interest because their convolutional and pooling layers, which have great applicability to images, could be used to derive features from long periods of raw time data. This is especially relevant because typical feed-forward networks are not explicitly designed to handle time-series data; the best that can be done is mapping each time-series element to a feature [43]. The convolutional and pooling layers which enable CNNs to extract locality-independent features from pixels could be used to find time-independent features from time-series data [44].

5.9.2 Long-Term Short-Term Networks

Some neural network architectures are explicitly designed to handle time-series data. One broad class of these architectures are recurrent neural networks, or RNNs. Whereas in a feed-forward network, the inputs for each round of training include only the input features of the data, in an RNN, the outputs of the hidden layers from one round of training are fed back as additional inputs for subsequent rounds of learning [26].

5.10 Advances in Financial Applications of Machine Learning

Machine learning, and particularly neural networks have been the vehicle for substantial advancements in financial technology. Countless pieces of literature have been written on the financial applications of machine learning, and AI-driven finance is shaping the entire financial services industry today.

Financial applications on neural networks can be broken down into categories. In some cases, the applications involve sentiment or risk analysis for banking or risk forecasting [30]. In other cases, the applications involve the selection of assets through the prediction of their future values [30]. The latter category best describes the research in this report. There has been an evident lack of neural networks with the purpose of strategic planning since networks blindly find patterns and cannot “explain” their results. In general, most applications operate at the level of pattern detection whose results supplement a holistic financial system [30].

Several major advancements define the progress of neural networks for stock market prediction. Kimto, et al. (1990) found that modular neural networks could predict the pricing of the Tokyo Stock Exchange using a variety of different learning algorithms. Kryzanowski, et al. (1993) proved that neural networks could determine which stocks would outperform the market. Refenes et al (1995) used artificial intelligence with pricing models to rank the performance of different stocks. Han, et al. (2014) found that genetic algorithms could be used to reduce the complexity of the input feature space of neural network algorithms. These advancements, along with countless others, speak to the predictive capabilities of artificial intelligence when applied to financial times. Recently, research has focused on applying large-scale data to enhance the predictive capabilities of these networks. As explained in Walczak (2001), the quantity of data available for training is a deciding factor for the accuracy of neural networks working with time series data. For example, Bollen et. al (2011) has explored farming Twitter data to generate sentiment analysis features to apply to neural networks. A key challenge facing AI engineers working with financial datasets involves balancing the requirement for greater amounts of data with the risks of error like overfitting and underfitting.

While an overwhelming amount of research has been done on the financial applications of neural networks, much of that work has been relegated to immediate stock or fund price prediction. Artificial intelligence configured this way generates a single trading strategy. Less has been done in academia to develop a neural network system of systems, i.e. an AI-driven system that would manage a set of discrete trading algorithms and strategies. This motivates the research described in this paper, as we believe it is novel in this field. Given the context of AI-driven research in the past, we were motivated to pursue machine learning as a technique of both our individual strategies and our eventual system of trading systems, described in the following sections.

6 Overview of Work Achieved

Sections 2 through 5 of this report detail the research we did during first half of this IQP. While we knew from the outset that we wanted to design a set of trading algorithms and balance them with a system of systems, we had no clear direction about how we would achieve those goals, other than a hobbyist's intuition that machine learning would prove effective. Our background research informed us of both the individual techniques that would yield our algorithmic trading and the data science techniques which would likewise yield the system of systems. We thus felt adequately equipped to pursue our goal.

Our tasks naturally split into two major components; some of us would have to develop trading algorithms themselves, while others would need to build the system of systems to balance and trade them. The algorithm design methods would be similar to the algorithm design done by both researchers and industry. As a result, it is heavily informed by the background sections detailed above. For example, the general principles of the market described in Section 2 would inform the design of the algorithms themselves, and the TradeStation testing methods in Section 4 would be directly applied to test those algorithms. Alan and Roberto led the effort to design individual trading algorithms. In doing so, they followed the process of designing and implementing trading strategies with entry and exit rules, build position sizes with various assets, and testing those algorithms with Monte Carlo analysis. Their work is described in Section 7.

The second component is the creation of a system of systems which brings all of the strategies together and allocates funds to each strategy based on performance. The effort to develop this system was led by Remy and Zoraver. This component of our work is very code intensive, as the entire system of systems framework had to be written from scratch. Construction of the system of systems also provided an exercise in neural network architecture and optimization. While time did not permit us to implement the strategies from Section 7 into the system, we still succeeded in proving that the system is predictive with several networks. The work to develop the system of systems, load it with strategies, and test its performance is in Section 8.

The individual strategies and the AI-driven system of systems together comprise the work we completed during this IQP. Both components utilized the information presented in Sections 2 through 5 to produce systems which navigate the market. The next section provides details on the individual strategies we implemented.

7 Trading Systems Developed

7.1 Roberto Esquivel's Trading Systems

Long-Short Ensemble Machine Learning System Versions 1 - 4

Description

These systems use an ensemble learning algorithm to predict the direction of each stock in a large sample (up to 1500) and ranks the stocks based on the random forest's assigned probability of each stock to either go "up" or "down" in the next 5 days. Once the stocks are ranked, the algorithm shorts the bottom 20% of the stocks and buys the top 20% of the probabilities.

Finding Alpha

The goal of this system is to maximize alpha by using a machine learning algorithm to predict future returns and to allocate funds to the equities that the algorithm believes are either going to have the least (short) or most (long) profitability.

Before writing trade execution and risk allocation for the algorithm, there had to be an iterative research process to design the machine learning pipeline and attempt to find the actual predictive features.

Data Exploration

A dataset indexed by days and containing a high, low, open, close, volume, and 5-day returns value for each stock on each day was explored and tinkered with to attempt to use machine learning methods to find a trading signal within it.

Machine Learning Model Creation

Stock price data sets have a characteristic known as non-stationarity. It means that over time, the mean and variance of the data set is shifted [54]. In order for supervised learning to work, a dataset must have stationary features [54]. The rule of thumb is to make the data stationary by engineering features that proportionally normalize the information in the data. These feature engineering methods range from calculating percent changes to less orthodox ways of quantifying information for each observation.

A problem with this approach is that a lot of the information in the data is eliminated by the process of making the data stationary. This is because there is a trade-off between stationarity and memory. Memory is the combination of the historical trends that shift the data's mean and variance [54]. The features that are created remove all of the memory from the data. However, that memory is the basis for a model's predictive power because it contains the raw information on the trends that have shaped the data in the past. To deal with this dilemma, the feature engineering process must do the bare minimum to make the data stationary in order to preserve as much memory as possible.

The approach taken in the feature engineering process for this algorithm is an approach proposed by Dr. Marcos López de Prado in his book "Advances in Financial Machine Learning". He proposes that a method he called "fractional differentiation" be used to adjust the features in order to make them stationary by differentiating only by the bare minimum in only to retain maximize memory preservation [54].

Aside from being non-stationary, the market itself has different conditions that change over time. This means that certain predictors can be predictive only for a specific time-frames and lose their edge on other timeframes. This algorithm can try to adjust for this in various ways. One is to attempt to find an "all-powerful" predictive feature or way of uncovering that feature every time the algorithm is retrained when the portfolio rebalancing occurs. The other way is to feed the algorithm all of the features and to use an algorithm that does feature selection by itself.

The last characteristic of most financial data that violates the assumptions of machine learning-ready datasets is that the time-series formatted data for equities is undersampling information from highly active trading periods. First, markets process information at a constant time interval. This is problematic because there are time periods which contain more information than others, however all time periods are treated as though they all provide the same amount of information. Some examples are that, the hour following the open is much more active than the hour around noon (or the hour around midnight in the case of futures); or that there are days where there is a lot more trading volume than the rest. A solution to this problem is to use data indexed by dollar-changes as opposed to a constant time interval. This solution was also proposed by Dr. Marcos López de Prado. Unfortunately, the platform in which this algorithm was created (Quantopian) does not offer data that is indexed by dollar changes and thus the algorithms will always be susceptible to this undersampling for periods with higher trading activity.

The feature creation rationale was developed by observing that the market operates in cycles of momentum and mean-reversion. Because of this, the main features created are mostly momentum-based. The thought is that if the market is in a momentum phase, then the features will be predictive of stock trends; and if the market is in a mean-reversion phase, then the momentum features will also be predictive of stock trends because the data will specify a negative correlation between these and the target variable.

The features created are can be divided into fundamental factors and into technical ones.
The technical factors:

1. Returns for the past 2, 5, 10, 15, 20, and 30 days
    a. These let the algorithm know the context of the momentum of the returns
2. Rate of change for the money-flow index for 2, 5, 10, 20 days
    a. Aside from letting the algorithm know the context of price changes, these indicators adjust for volume as well.
    b. This indicator is calculated by multiplying price and volume over a couple of days, comparing those amounts for consecutive time periods, and computing a ratio of the highest amount to the lowest amount of price time volume over the selected time periods.
3. Rate of change of the RSI for 2, 5, 10, 15, 20 days
    a. This factor can allow the algorithm to learn the relationship between the changes in buying and selling pressure and the future returns of a stock.
    b. The calculation of the RSI is broken down into the calculation of its four components: the average gain, average loss, first rs and smoothed rs.
    c. The average gain or loss is computed by summing all of the gains or losses during a specified time period and dividing them by the number of time periods.
    d. The first rs is calculated by dividing the average gain by the average loss; the rest of the rs calculations use the previous period's average gain and average loss to smooth the data.
    e. After that, a single RSI data point can be calculated by subtracting 100/(1+rs) from 100.
    f. The rate of change of these data points is calculated by getting the rsi unit change per time period of choice.
4. Percent of positive return days to total days in the last 5, 10, 15, 20 and 30 days
    a. These were also designed because returns and momentum information can be influenced by a larger price swing in a short period of time. The percent of positive return days offer a better picture of the overall trend of the price.
    b. These indicator is calculated by counting the days where the closing price was higher than the opening price in the specified time period and dividing that number by the total number of days in the time period.

These let the algorithm know the relative degree to which the buying and selling pressure of the stock change over time and were designed because returns and momentum information can be influenced by a larger price swing in a short period of time. For example, if the 10-day returns of a stock are being calculated, there could be a day where the stock price rises 5 percent, but every subsequent day, the stock has negative returns. If the negative returns do not outweigh the initial 5 percent price upswing, the calculated returns will still be positive, which is not an accurate representation of momentum. These features overcome this by allowing the change in buying and selling pressure of the stock to be recorded.

The purpose of the algorithm is to use the change in the value of this indicator, not the value itself, as an indicator of future price direction.

The fundamental factors:
Quantopian provides a dataset for the following measures:

1. EBITDA Yield
    a. This number is calculated by dividing a company's earnings before interest, taxes, depreciation and amortization (EBITDA) by the company's enterprise value. EBITDA describes profitability before interest, taxes, etc. are factored in.
    b. There are various ways in which this number can be calculated. Generally, the goal is to get a ratio of profits to capital that the company allocates under its own judgement.
2. Net income margin
    a. This number is the ratio of revenue compared to all of the expenses that a company incurs
3. Operating Cashflows to assets
    a. This is a measure of a company's operations efficiency. It is a ratio of the return on investment for each asset compared to the cost of the assets.
4. Asset Growth
    a. This is a measure of the rate at which a company's assets grow in size.
5. Asset to Equity Ratio
    a. This is a good measure to quantify the debt that a company has used to finance its operations. It is calculated by computing a ratio of how much of the company's money is used in each asset to the amount of money in each asset that is coming from shareholders (equity)

These let the algorithm know the relative change in the inner workings of a company through time and allow the algorithm to learn the relationship between them and the company's change in stock price.

There are various ways of assessing the predictability of a feature. One of them is by using an algorithm to filter the features that add more noise than predictive value. Another way is to graphically display a scatter plot of a feature and the target value and check for patterns. For example, Pearson correlation coefficient-based elimination can be done by only selecting the highest scoring features. The last way is by using a machine learning model that can let the algorithm designer know what features are used the most for prediction. The latter approach does not necessarily mean that the feature is especially predictive, just that it holds more information relative to the others and thus the algorithm uses it the most to assign predictions. The most predictive features for the fundamental algorithm are EBITDA yield and return on invested capital. The most predictive features for technical factors are a trendline of past returns and the money flow indexes.

Once all features are chosen, there needs to be an assessment of how much the features are correlated with each other. If features exhibit a high correlation to each other, that means that one of them adds no value to the information learned by the algorithm and is unnecessary so one of the correlated features needs to be removed.

Trade Execution

The algorithm will execute trades every Monday as soon as the markets open. Before the market opens, the algorithm runs the machine learning algorithm on six months of data to output a ranking of stocks based on their calculated probability of having positive five-day returns. This way, the output will be ready for the algorithm to use in its trading as soon as the market opens. After that, the algorithm does not execute any trades for the rest of the week.

Risk Allocation/Portfolio Construction

The platform used to develop this algorithm offers an optimization feature that allows the algorithm author to specify a high-level objective, constraints, and the universe of equities that the algorithm will trade and letting the optimization feature do all of the dirty work. This feature will automatically calculate the set of portfolio weights that optimize the objective while still respecting constraints. After that, it computes the difference between the optimal calculated portfolio weights and the actual weights and places orders to move the current portfolio's state to the optimal state.

A portfolio optimization problem is a mathematical problem of the following form:

"Given an objective function, F, and a list of inequality constraints, $C_i \leq h_i$, find a vector w of portfolio weights that maximizes F while satisfying each of the constraints."

This problem can be expressed as follows:

$$
\begin{array}{lll}
\underset{\boldsymbol{w}\in\mathbb{R}^n}{\text{maximize}} & F(\boldsymbol{w}) & \\
\text{subject to} & & \\
 & C_i(\boldsymbol{w}) \le h_i, & 0 \le i \le m \\
\text{where} & & \\
 & F\colon \mathbb{R}^n \to \mathbb{R} & \\
 & C_i\colon \mathbb{R}^n \to \mathbb{R} & \\
 & h_i \in \mathbb{R} &
\end{array}
$$

The parameters for the optimization are:
Objective: To maximize alpha, or returns which beat the standard deviations of the market.
Constraints:

1. The algorithm must constrain gross leverage to 1.0 or less. This means that the absolute value of our long and short positions should not exceed the value of the portfolio. This is to avoid the risk of overdrafting and losing capital that we do not have.
2. Any given position cannot exceed over 2% of total portfolio value
    a. This constraint will allow risk of non-diversification of assets to be minimized.
3. The algorithm must be market neutral: Long and Short positions must compose equal parts of the portfolio value
4. The algorithm must be sector neutral: All positions must be weighted equally among all market sectors

Universe

The stock universe that the algorithm will trade on is any stock that is on the Fortune 500 group. The rationale behind picking this universe is that this universe minimizes risk factors in areas of lack of liquidity and excessive volatility.

Position Sizing

Handled by the optimization features.

Entry Rules

Aside from the rebalancing calculations done by the optimization, the main way in which the algorithm decides whether to short or long a stock is based on the output of the machine learning algorithm. The algorithm will output a ranking for the top and bottom 20% of the equities where the higher the ranking, the higher the calculated probability of the stock having positive returns in 5 trading days. This way, the algorithm will go long on the top 20% ranked equities and short on the bottom 20% of the equities and rebalance accordingly every time the algorithm is run.

Exit Rules

Other than the rebalancing done by the optimization feature, the main ways in which the algorithm decides when to exit a position are when the equity does not show up in its ranking group, or if the equity moved to another ranking group. This makes the algorithm vulnerable to losing profits from high transaction costs if there is a lot of turnover among the weekly rankings of equities. However, the turnover among the weekly rankings is directly linked to the robustness of the machine learning algorithm, so if the machine learning algorithm is robust, this should not be a problem. Another way to limit this, is to elongate the holding period of the stocks and widen the prediction range of the machine learning algorithm.

Versions and Variations

Each different version of the algorithm is a different approach in terms of the type of ensemble method used and the factors that each of those methods used to predict. The two types of ensemble methods that were tried are a random forest and a boosting method. The factors used for each of those methods to predict were either a combination of fundamental factors, or a combination of technical factors.

Analysis of the Results of each Version

Ensemble methods Prediction Performance on 5-day Future Returns

Before analyzing the performance of the trading algorithm that utilizes each of the ensemble methods, it is helpful and necessary to assess the prediction performance of each of the methods to have a better idea of how they contribute to the profitability of the actual trading algorithm.

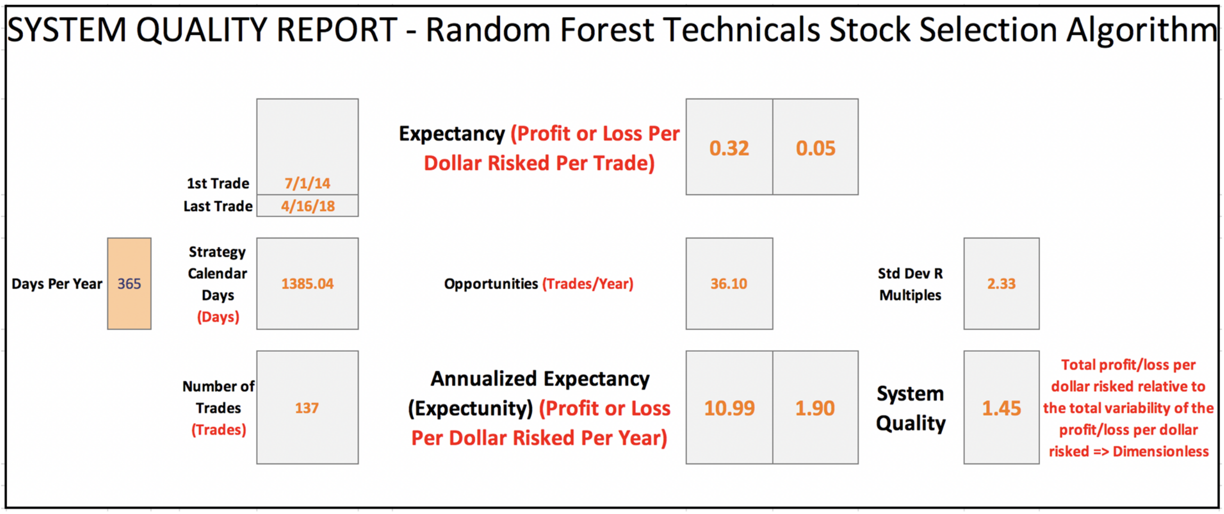
Random Forest Classifier with technical analysis factors
Prediction Performance of Random Forest Classifier with technical analysis factors

- 58% prediction accuracy

Performance Report Analysis

The algorithm has returns of 10% over the 2 year back test period and fails to outperform the market (the market had 35% returns). The Sharpe ratio is low at 0.80 and the max drawdown is 19%. The returns attributed to common risk factors of the market (the returns that are not specific to the actual stock) are 45%, which means that the algorithm by itself would actually have negative returns.

System Quality


SYSTEM QUALITY REPORT - Random Forest Technicals Stock Selection Algorithm

1st Trade	7/1/14
Last Trade	4/16/18
Days Per Year	365
Strategy Calendar Days (Days)	1385.04
Number of Trades (Trades)	137
Expectancy (Profit or Loss Per Dollar Risked Per Trade)	0.32 / 0.05
Opportunities (Trades/Year)	36.10
Annualized Expectancy (Expectunity) (Profit or Loss Per Dollar Risked Per Year)	10.99 / 1.90
Std Dev R Multiples	2.33
System Quality	1.45

Total profit/loss per dollar risked relative to the total variability of the profit/loss per dollar risked => Dimensionless

Fig. 11 - System Quality Report - Random Forest Technicals Stock Selection Algorithm

The system's expectunity and expectancy of 0.32 and 0.05 and annualized values of 10.99 and 1.90 indicate that the system is not worth implementing. These low measures and a high standard deviation of profit over average losses (2.33) tell us that the system's profit is not enough to offset the volatility and the risk is too high for the expected profits.

Monte Carlo Analysis

The 95% confidence interval spread is very large because the predictive performance of the model is close to chance (58%). The returns over two years for every scenario range between -75% and 100%.

Random Forest Classifier with Fundamental analysis factors

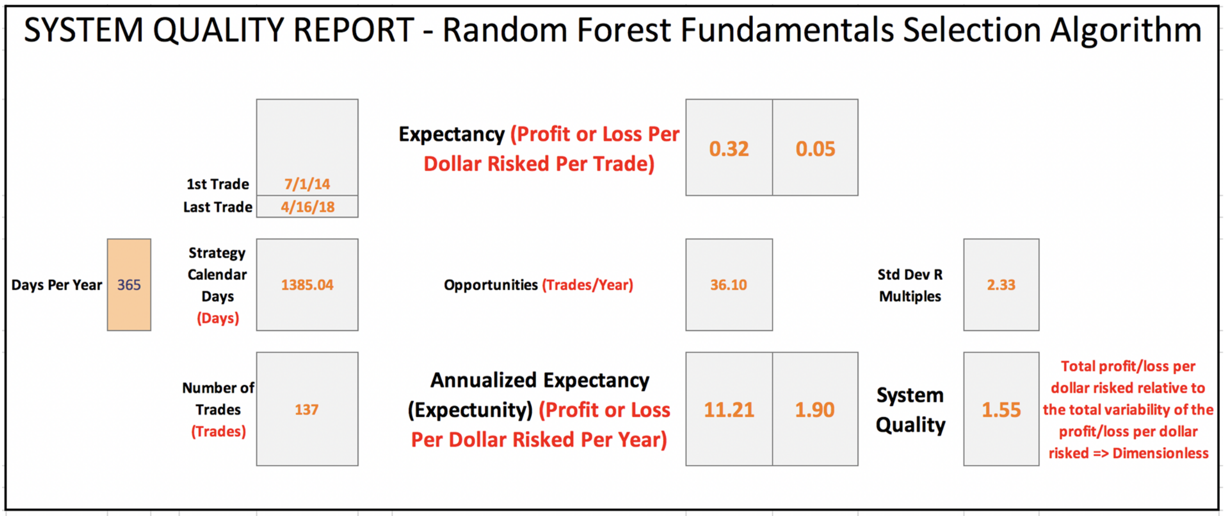
Prediction Performance of Random Forest Classifier with Fundamental analysis factors

- 55% prediction accuracy

Performance Report Analysis

This model is not worth implementing because despite its positive returns, it fails to beat the market over the past 2 years. It only returns 10%, compared to the 35% of the whole market. The Sharpe ratio is quite low at 0.25 and it has a maximum drawdown of 20% of the portfolio. The last metric that provides insight into the performance of this strategy is that its common returns, or returns that are attributable to common risk factors, account for 44% of the returns of the system. This means that close to half of the algorithm's results are due to chance, which makes sense given that the prediction performance of the random forest is only 55%, which is very close to chance.

System Quality



SYSTEM QUALITY REPORT - Random Forest Fundamentals Selection Algorithm

		1st Trade	7/1/14	Expectancy (Profit or Loss Per Dollar Risked Per Trade)	0.32	0.05			
		Last Trade	4/16/18						
Days Per Year	365	Strategy Calendar Days (Days)	1385.04	Opportunities (Trades/Year)	36.10		Std Dev R Multiples	2.33	
		Number of Trades (Trades)	137	Annualized Expectancy (Expectunity) (Profit or Loss Per Dollar Risked Per Year)	11.21	1.90	System Quality	1.55	Total profit/loss per dollar risked relative to the total variability of the profit/loss per dollar risked => Dimensionless

Fig. 12 - System Quality Report - Random Forest Fundamentals Selection Algorithm

The system quality report shows the expectancy and expectunity of the trading system. The report shows that this system is not ready being in the market. The expected return per dollar risked is extremely low 0.32-0.05. It is better to invest in the market as a whole than to trade with this system. The annualized expectancy is similarly low at 11.21 per dollar risked.

Monte Carlo Analysis

Due to the fact that the random forest used to make buying and selling decisions has an accuracy that is close to chance, the Monte Carlo analysis has a large spread and the 95% confidence interval consists of returns ranging between -75% to 100%. This further shows that the algorithm is not ready for deployment because it does not have consistent performance.

ADA Boost Classifier with technical analysis factors

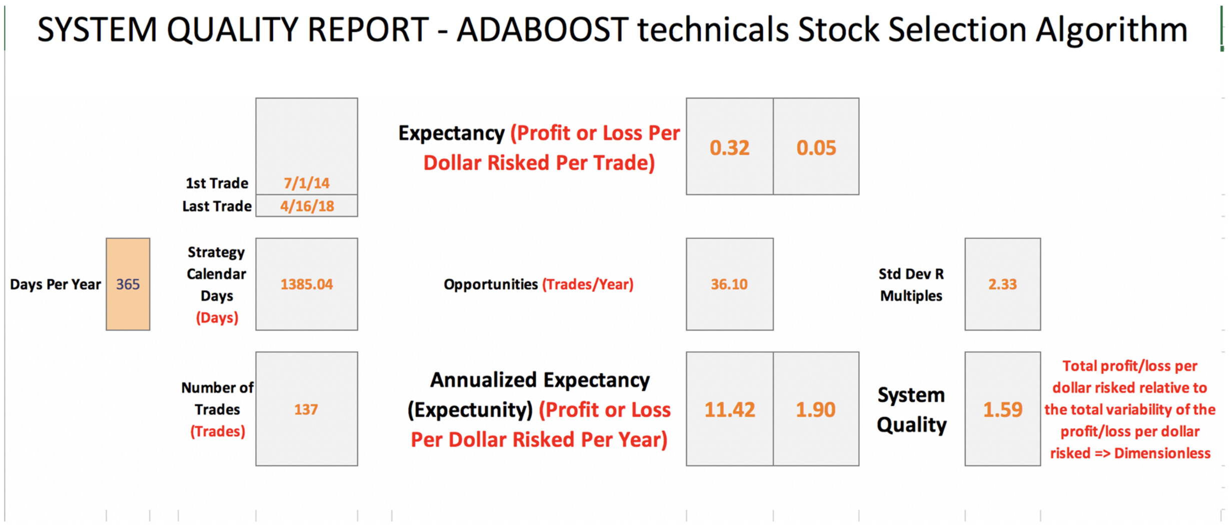
Prediction Performance of ADA Boost Classifier with technical analysis factors

- 61% prediction accuracy

Performance Report Analysis

This system, despite having better prediction accuracy, does not improve returns compared to the last systems. It still only manages to have 10% returns, has a bigger maximum drawdown of 20%. The frequency of winning trades is 50%.

System Quality



SYSTEM QUALITY REPORT - ADABOOST technicals Stock Selection Algorithm

		1st Trade	7/1/14	Expectancy (Profit or Loss Per Dollar Risked Per Trade)	0.32	0.05			
		Last Trade	4/16/18						
Days Per Year	365	Strategy Calendar Days (Days)	1385.04	Opportunities (Trades/Year)	36.10		Std Dev R Multiples	2.33	
		Number of Trades (Trades)	137	Annualized Expectancy (Expectunity) (Profit or Loss Per Dollar Risked Per Year)	11.42	1.90	System Quality	1.59	Total profit/loss per dollar risked relative to the total variability of the profit/loss per dollar risked => Dimensionless

Fig. 13 - System Quality Report - ADABOOST Technicals Stock Selection Algorithm

This system is a low 1.59 because the profit or loss per dollar risked per trade ranges only from 0.32 to 0.05. The standard deviation of the profit over average losses is high at 2.33 given the average potential profit per trade ($0.05). These metrics further show that the system is not ready for implementation**.** The Sharpe ratio is a low 1.04.

Monte Carlo Analysis

The spread of the potential scenarios under suitable confidence intervals like 70% or 95% is equally large as the other system's Monte Carlo's analysis, despite the fact that the predictive model's performance is a little higher at 61%.

ADA Boost Classifier with Fundamental analysis factors

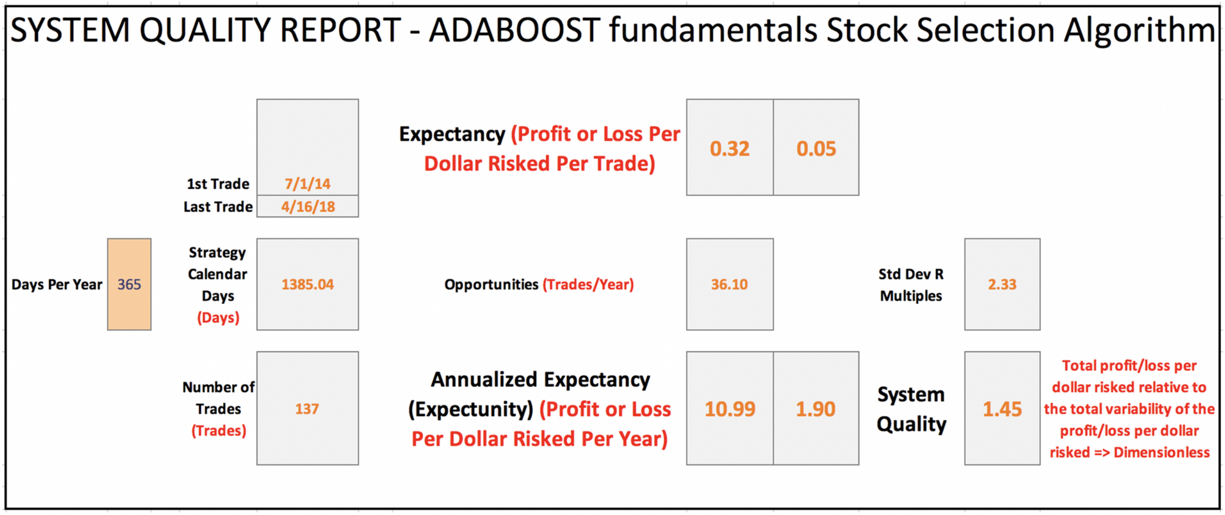
Prediction Performance of ADA Boost Classifier with Fundamental analysis factors

- 55% prediction accuracy

Performance Report Analysis

This system did not beat the market either and had returns of 8% over the two year backtest period. Its max drawdown was less than the rest of 18% percent and it had a higher frequency of winning trades at 57%.

System Quality



SYSTEM QUALITY REPORT - ADABOOST fundamentals Stock Selection Algorithm

		1st Trade	7/1/14	Expectancy (Profit or Loss Per Dollar Risked Per Trade)	0.32	0.05		
		Last Trade	4/16/18					
Days Per Year	365	Strategy Calendar Days (Days)	1385.04	Opportunities (Trades/Year)	36.10		Std Dev R Multiples	2.33
		Number of Trades (Trades)	137	Annualized Expectancy (Expectunity) (Profit or Loss Per Dollar Risked Per Year)	10.99	1.90	System Quality	1.45

Total profit/loss per dollar risked relative to the total variability of the profit/loss per dollar risked => Dimensionless

Fig. 14 - System Quality Report - ADABOOST Fundamentals Stock Selection Algorithm

The system quality is very similar to the system quality of the random forest with fundamental analysis factors. The only difference in between these systems is that the annualized expectancy of this algorithm is a little lower at 10.99 compared to the 11.21 of the random forest indicator.

Monte Carlo Analysis

The spread of the confidence intervals for the Monte Carlo analysis of this system is equally large as the spread for the confidence intervals of the other versions of the system. The main reason is still the fact that the predictive algorithm's accuracy is close to 50%.

Compiling all of the strategies and Simulating over Time

The following figure is a plot compiling the Monte Carlo simulation's spread of results for all of the ensemble algorithms. It can be seen that the spread of positive returns and negative returns is very large for the 95% confidence interval. A couple of useful metrics computed from the analysis are that 40% of the scenarios show positive returns, the worse and best case scenarios range from returns of -80% to 150%, and that the median returns for all scenarios are -8%.

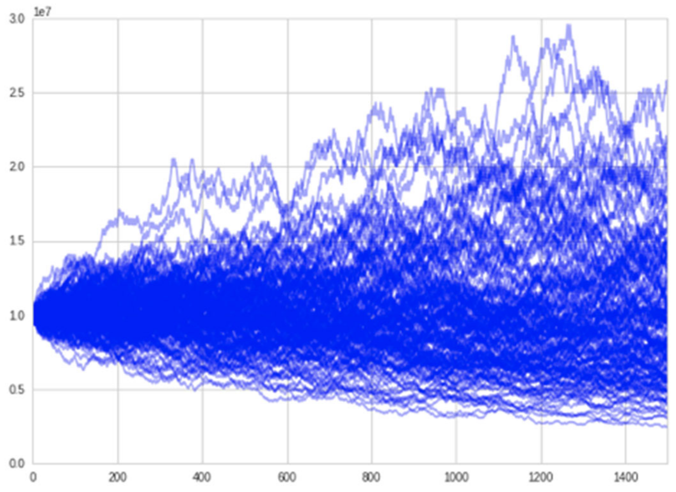


Fig. 15 - Monte Carlo Simulation Spread

Conclusions and Future Improvements

The most influential component of a system that uses machine learning to make trading decisions is the predictive performance of the underlying machine learning algorithm in the trading system. It is key to ensure that the machine learning algorithm has satisfactory performance before designing the trading algorithm around it. Otherwise, it does not matter how well-designed or optimized the trading algorithm is because it will be making trading decisions based on a faulty machine learning algorithm.

The possible reasons as to why both the random forest and the adaboost algorithms used to predict future stock price directions fail to have a good performance are because the features fed to the algorithm are not predictive enough, there was not enough data to train the algorithm given the scope of the stock universe, and feature engineering was not done correctly. The similarity of the performance of the algorithms shows that performance depends more on the features used for prediction than the algorithms used.

The features of the algorithm are not predictive enough because they fail to provide a clear class distinction of the stocks that go “up” versus the stocks that go “down”. This was clear after using principal component analysis to see if any linear combination of the features could map the data points to different clusters; because there was no linear combination of the features that could separate the two stock classes. The proportion of explained variance by the principal components was only 25%, while a satisfactory set of features should explain at least 60%. A good set of features would also allow the plot to display two distinct groups with different colors identifying their class. However, as seen below, there is no clear distinction.

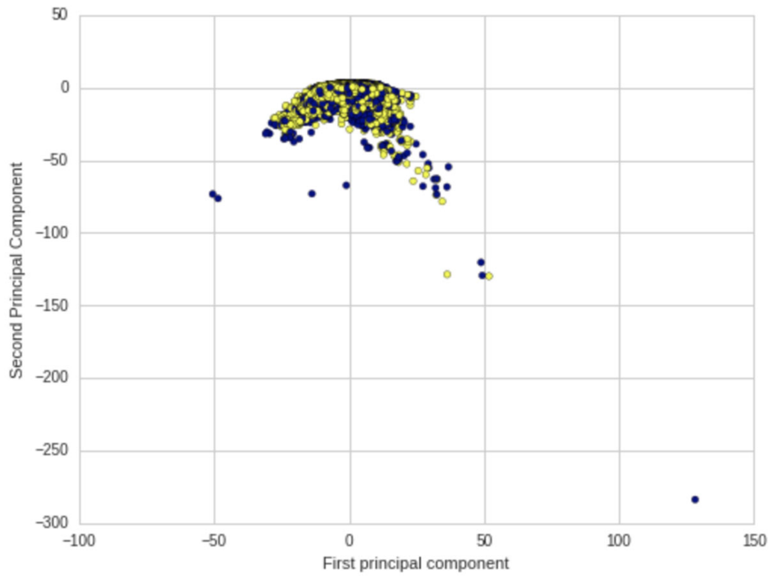


Fig. 16 - PCA Analysis

There was not enough data to train the algorithm given the scope of the stock universe for two reasons. Trying to generalize a set of features that can predict movements for every type of stock requires a vast amount of data that not only samples open, low, high, and close for each stock, it needs to sample data every time that there is a price movement; similar to what Dr. Lopez de Prado suggests. The problem is that quantopian does not support getting data in this format and the only platform we could find that supports it does not support machine learning packages (TradeStation). The other constraint is that there is a computational limit to the amount of data that quantopian can stream and thus including too many stocks leads to having a lot of data but very little data for each individual stock. Something that can be done to accommodate for this is to reduce the scope of the stocks that we are trying to predict to one common sector or cyclicality.

The last reason for which the random forest and boosting algorithms failed to have good predictive accuracy is because feature engineering was not done properly. Rather than finding semi-predictive features and then moving on to building a trading system around them, there should have been more time spent finding the best possible features and only until the features can ensure predictive performance should the trading system be designed around them. Feature engineering is the most difficult and time consuming part of the process and requires creativity and a high degree of domain knowledge in both finance and machine learning. Aside from spending more time creating features and acquiring domain knowledge, something that can be tried in the future is automated feature engineering.

In addition, other approaches to using machine learning for trading systems can be explored. Some ideas can be clustering and unsupervised learning to identify groups of stock that move together based on a variety of criteria and formulate pairs trading strategies around them, using anomaly detection algorithms to detect and learn features of big stock price swings, and using the machine learning algorithms for the trading execution part of the system rather than the alpha detection phase. The last thing can be to find a way to complement fundamental and technical factors in the algorithm, even if they have different time periods.

7.2 Alan Fernandez' Trading Systems

7.2.1 Forex Correlation Divergence

Description

The Forex Correlation Divergence Trading Strategy attempts to predict the direction of a particular currency pair, in this case, the USDJPY, and to trade divergences between the prediction and the current direction of the price. The direction is predicted using a set of securities that have an established relationship to the currency pair.

Selected Securities

USDJPY - US Dollar Japanese Yen

The USDJPY pair was chosen because it is considered a safe haven currency meaning that there are multiple securities that move with the pair. Most of the chosen securities are also considered safe haven assets or indicators of investors moving towards safe haven assets. Since in times of financial turmoil investors shift their assets to safe havens, a rise in the demand for any safe haven asset should correlate to similar increases in other safe havens.

HKDJPY

According to the Observatory for Economic Complexity (OEC), Hong Kong is one of the main trading partners of Japan. Thus, the exchange rate between the currencies of the two countries should reflect the strength of the respective economies. If the Hong Kong dollar strengthens in comparison to the Yen, then there will be an increase in the demand for Japanese products followed by a strengthening of the Japanese economy and a strengthening of the Yen.

USDCHF

The Swiss Franc is considered the default safe haven asset. Switzerland adheres to its self-imposed conflict neutrality policies adding to the perceived stability of its government and economy. According to the Heritage Foundation's Index of Economic Freedom, Switzerland's extremely developed economy is the fourth freest in the world. Since the Japanese Yen is also a safe haven asset, it is expected that following a surge in the strength of the Swiss Franc relative to the US Dollar, a similar surge should follow in the Japanese Yen. The strategy is naive in that it assumes that any surge in the demand for CHF is the result of investors seeking a safe haven for their assets. This is partially accounted for by looking at the moving average of the correlation between the two currencies.

DXJ

In order to obtain a better sense of the economy of Japan, the strategy follows the WisdomTree Japan Hedged Equity Fund. Japan is the world's largest exporter of vehicles.

According to the OEC, the automotive industry represents 26% of Japan's worldwide exports. DXJ's majority holdings are in Japan's major automotive manufacturers: Toyota, Mitsubishi, Nissan and Honda. The rest of the holdings are spread out across all sectors of the Japanese economy. The performance of these major automotive manufacturers has a direct impact in the health of the Japanese economy, which is to an extent represented by the performance of this hedge fund. It is acknowledged that the hedge fund could simply rebalance its portfolio should these manufacturers start to underperform. This would make the hedge fund less representative of the weakening economy.

SPY

Since the base pair traded is the US Dollar, the strategy attempts to gauge the performance of the US economy by looking at the S&P 500 index. If the index is on a downturn, it is expected that the Japanese Yen will strengthen as investors tend to flee in times of economic distress. During periods of growth, the relationship between the two is not clearly defined and left to the direction of the correlation between the two. Positive growth in the US economy could correlate to strengthening Yen as demand for Japanese products increases. It could also translate to a weakening Yen as investors who were previously escaping a falling market exit their Yen positions and return to stocks.

PowerShares US Dollar Bear and Bull

These two indexes are used in combination as a replacement for the Dollar Index, which is not freely offered in TradeStation. The Dollar Index measures the strength of the US Dollar against a basket of major currencies.

SPDR Gold Shares

Building on the idea that safe haven assets should move together when investors flee insecurity, rising gold prices should reflect rising CHF and JPY.

$SPGSCLTR

Japan is located in a small island which means that it must import the majority of the energy that is required to power its industries. This makes Japan's economy extremely dependent on the price of Oil. According to Statista, Japan receives approximately 71% of oil from Saudi Arabia. Europe, Africa and the Middle East use the Brent benchmark for the oil prices thus the strategy uses an index for Brent Oil. As the price of Brent rises, it is expected that the Japanese economy slow down and the price of its exports will increase. This should then lead to lower demand for its exports and lower demand for the Japanese Yen. An increase in the price of Brent then results in a decrease in the purchasing power of the Japanese Yen.

Position Sizing

The strategy uses a volatility adjusted position sizing approach similar to that used by the Turtle Traders. The volatility of the underlying security is calculated using the Average True Range over a certain number of bars.

```
N = AvgTrueRange(N_ATR_L);
DV = N*BigPointValue;

accountSize = STARTING_ACCOUNT_SIZE + NetProfit;

dollarRisk = (accountSize) * .05;
Cts = IntPortion(dollarRisk/DV);
```

This approach risks a maximum of 5% of the total account size in any trade. The total number of units purchased is then dependent on both the volatility of the security and the account size. The Turtle Traders used a notional account size in their position sizing strategy. This meant that the size of their account did not change until the end of the month when it was updated based on their performance. Unlike the Turtle Traders, this strategy does not use a notional account size and always risks a maximum of 5% of the current size of the account. Should it use a notional account size then it could risk more than 5% of the real account if it were to enter a losing streak.

Entry Rules

The strategy enters a position when it identifies a divergence between the predicted direction of the price with the actual direction of the price. The predicted direction is calculated in the following manner:

$$predictedDirection = W1 * P1 * Correl\_1 + \ldots + Wn * Pn * Correl\_n \qquad (6)$$

The result of the sum is a number whose value is not as important as its sign. A negative number indicates that the price is falling and a positive number indicates the price will rise. This direction is compared to a fast moving average of the actual price of the pair. If they do not coincide then the strategy opens a position against the current movement of the price and in favor of the prediction.

Exit Rules

Trailing Stops

Once a profit floor is crossed, the strategy begins to place trailing stops to maximize the profits of the position. The Average True Range is used to adjust the placement of the trailing stops such that they are not tripped by normal price volatility.

It is important to note that the strategy uses two different trailing stops depending on whether the position was increased as a result of the optimization through Maximum Favorable Excursion. The larger trailing stops allow the position to profit from the increase in the position size.

Maximum Adverse Excursion

The optimal position of the stops for the strategy is obtained through Maximum Adverse Excursion (MAE) analysis. This analysis is done following the optimization of the strategy. MAE compares percentage drawdown to percentage profit/loss. This allows the trader to identify the amount of drawdown after which the trade is most likely a losing trade. A stop loss placed at this level of drawdown reduces the maximum drawdown of the strategy. It is evident from the following MAE graph that positions that incur a drawdown greater than ~0.3% are more likely to result in overall loses.

These positions are eliminated with the addition of the following conditions to the strategy:

```
//For Long Positions
If (MarketPosition = 1) and (C <= EntryPrice*(1.003)) then begin
...
//For Short Positions
If (MarketPosition = -1) and (C >= EntryPrice*(0.997)) then begin
...
```

With the addition of these stop orders the maximum drawdown of the strategy was reduced from $465 to $197. This decrease in drawdown however, resulted in a sacrifice of the overall profits since some winning positions were eliminated. The choice between higher profits and lower drawdown depends on the trader and his goals. Given that clients typically scare away from large drawdowns, a fund manager may choose to sacrifice part of the profits to keep his clients calm.

Position Timeout

Following an analysis of the trades it was determined that positions that lasted more than a certain number of bars usually resulted in losses. As a result, a condition was added to close these positions to avoid further loses.

Position Adjustment

Position Re-Evaluation and Re-Entry

The nature of the strategy is to enter long trades that may last several days. In order to quickly exit losing positions, the strategy evaluates the OpenPositionProfit as well as the direction of the price. If it finds that the predicted direction of the price has reversed and that the current position is a loser, the strategy then reverses the position. The strategy allows the trade to develop for a day before evaluating the accuracy of the position.

Maximum Favorable Excursion

Using Maximum Favorable Excursion Analysis, it is possible to improve the strategy by determining the type of trades that have a high likelihood of profit. These trades are identified by their run-up. As seen on the Maximum Favorable Excursion Graph, the trades with a run-up above 0.4% are mostly profitable. Since there are also some large losing positions past this percentage of run-up, the trader may wish to exclude increasing the positions based on the MAE as these losses will also increase. In a more ideal scenario, the losses would be concentrated towards the origin such that there are no losing positions past a certain percentage of runup. The trader could then increase the size of these positions to increase profits with little added risk.

The increase of the position is achieved as follows:

```
//For Long Positions
If Close >= 1.004*EntryPrice and pIncreased = false then
...
//For Short Positions
If Close <= 0.996*EntryPrice and pIncreased = false then
...
```

Bad Entries

The strategy will quickly exit any position where within the first 24 hours after entry the actual direction of the pair and the predicted direction match. Positions where this is true are most likely improper entries given the length of the exponential moving averages used to determine the direction.

Optimization

The optimization of the strategy was performed using the TradeStation backtesting optimizer. The optimized values for the inputs of the strategies were obtained using the genetic optimizer given the extremely high number of combinations to test.

Performance Report Analysis

It is evident from the Strategy Performance Report that this particular strategy is not great; an informed trader would not make use of this strategy. Over the 6 year trading period over which the strategy was optimized, it only managed to generate an annual return rate of 0.50% Furthermore, the Profit Factor and the Win/Loss Ratio of the strategy are extremely low at a mere 1.22 and 1.00 respectively. Even though the strategy wins more often that it loses, the size of the winners is practically the same as that of the losers. This results in low returns since only 54.2% of the trades are profitable.

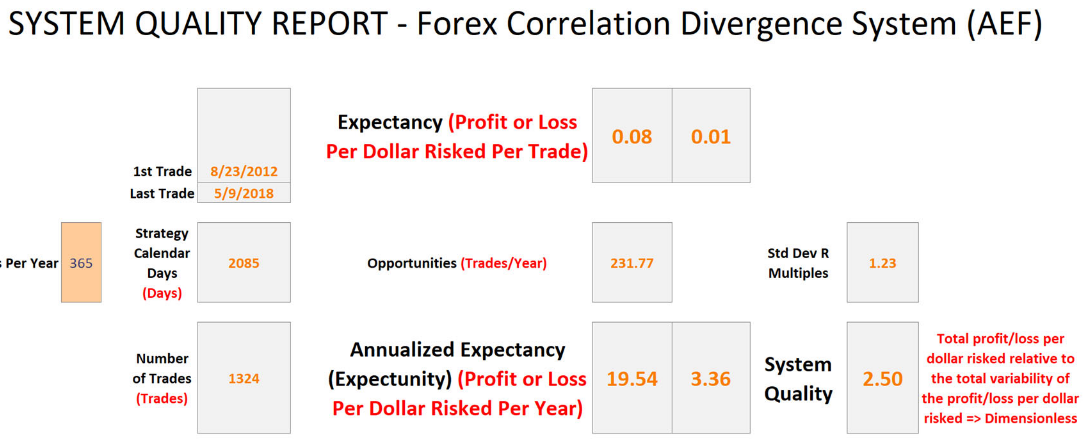
The Equity Curve shows that the strategy made most of its profits in the last two years. Prior to this period the strategy did not generate constant returns. It is likely that a fund manager would have disabled this system long before 2016 given that it did not produce significant profits for the first four years.

On a positive note, the strategy has an entry and exit efficiency slightly higher than 50%. Given that more than 50% of the entries are efficient, it may be deduced that the strategy has an edge that could be exploited with a better exit strategy.

Out of the 1,323 trades performed by the strategy over the 6 years during which it traded, only $115.55 resulted from outlier trades. This means that the bulk of the profits is obtained from the average trade rather than from unlikely trades.

The average time spent within trades was reduced to 1 hour as a result of the tight stops placed by the MAE analysis. This means that some of the exit conditions such as the position timeout will never be met. This is not a problem, however, because the addition of the MAE stops improved the strategy.

System Quality Report



SYSTEM QUALITY REPORT - Forex Correlation Divergence System (AEF)

1st Trade	8/23/2012	Expectancy (Profit or Loss Per Dollar Risked Per Trade)	0.08	0.01	
Last Trade	5/9/2018				
Days Per Year 365 / Strategy Calendar Days (Days)	2085	Opportunities (Trades/Year)	231.77	Std Dev R Multiples	1.23
Number of Trades (Trades)	1324	Annualized Expectancy (Expectunity) (Profit or Loss Per Dollar Risked Per Year)	19.54	3.36	System Quality 2.50

Total profit/loss per dollar risked relative to the total variability of the profit/loss per dollar risked => Dimensionless

Fig. 17 - Forex Correlation Divergence Strategy

The System Quality Report provides the Expectancy for the system, which is the expected return per dollar risked. The Expectancy of this particular strategy is between $0.02 and $0.09 per dollar risked. With such low expected returns it would be extremely difficult to convince any trader to trade this system. The Expectunity tells a similar story with expected annual returns between $3.56 and $21.09 per year per dollar risked. Based on these results the system is not yet ready for trading; further development is required.

Monte Carlo Analysis

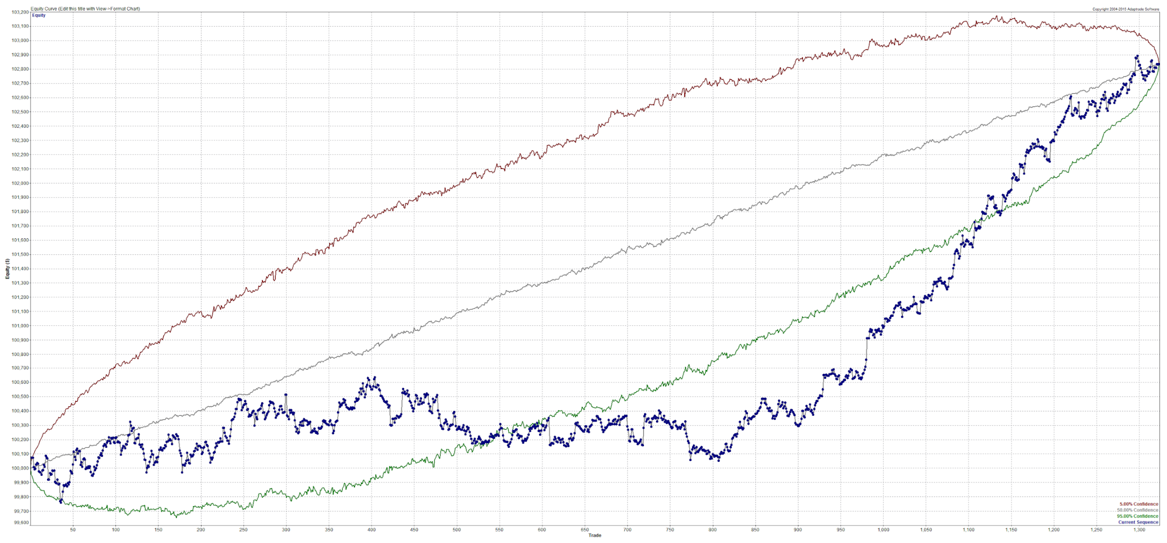


Fig. 18 - Monte Carlo Analysis - Forex Correlation Divergence Strategy

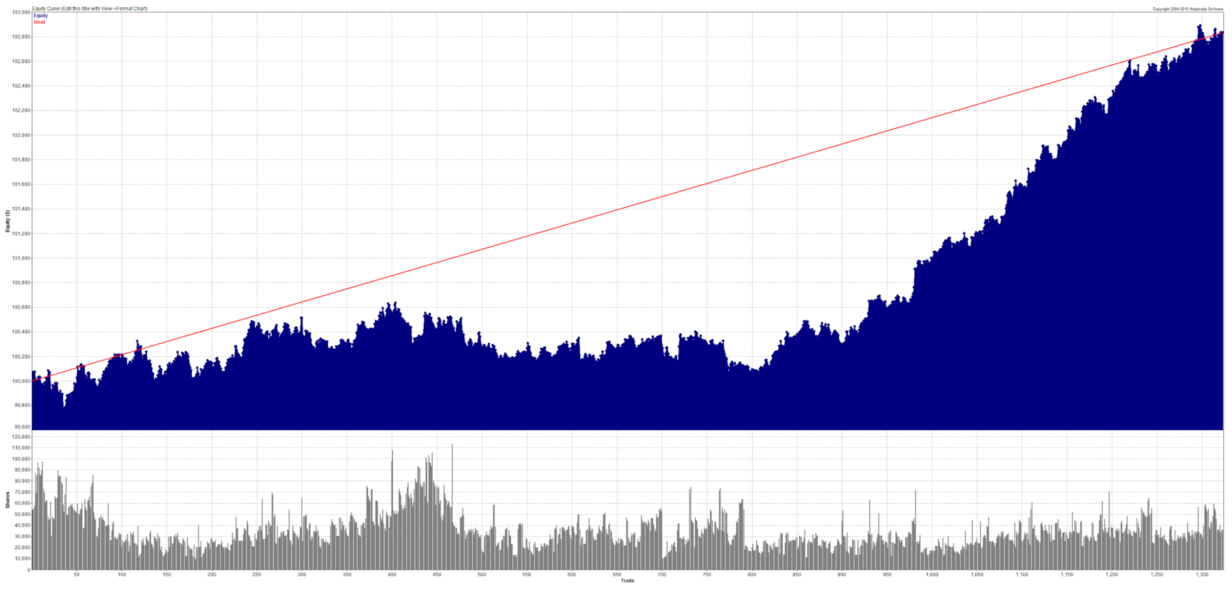


Fig. 19 - Trade Profit - Forex Correlation Divergence Strategy

The Monte Carlo analysis graph plots the 5% confidence and 95% confidence equity curves. It is clear from the graph that the analysis is not very accurate since the current sequence falls outside of the range outlined by the two lines. A larger sample of trades is required in order to generate a more accurate prediction. The analysis presents the following conclusion: The trader may assume with 95% confidence that this strategy will generate profit. This does not mean that the strategy will not fail. At a 95% confidence interval the remaining 5% represents the possibility of the prediction being incorrect.

A prediction for the equity curve over the last 134 trades is also provided and is based on a Monte Carlo analysis as well. It is observed from this prediction that the current sequence is a better outcome than the expected results with a 95% confidence interval. This suggests that the results of the last 134 trades were unlikely to occur.

Finally, Market System Analyzer allows the trader to randomize the order of the trades in order to generate other possible equity curves should the same trading opportunities had occurred in a different order. Two of the most extreme alternate equity curves are shown.

7.2.2 Overnight Gap Trading

Description

The overnight gap trading strategy attempts to exploit the overnight gaps in the stock market. On average, using a trend following approach it should be possible to predict the direction of the overnight gap. Since the overnight gap tends to be significant in some stocks, accurate prediction of the movement could be highly profitable. The strategy was inspired by studies such as those conducted by Cliff et al which show that the “US equity premium over the last decade is solely due to overnight returns." The overnight returns are consistently higher than those during the day. In their study, [55] conclude that the traditional explanations for this phenomenon such as risk, overnight earnings announcements and liquidity are not enough to explain the extent of the gap. They suggest that algorithmic trading might have a sizeable contribution to the overnight gap. Regardless of the reason behind the gaps, overnight trading is significantly more profitable than day trading.

Position Sizing

Kelly Criterion

The initial position sizing technique utilized in this strategy was the Kelly Criterion. The Kelly Criterion allows the trader to modify the size of the next bet based on the success of previous bets. The Kelly Criterion was originally used by gamblers to determine the percentage of their pot of money to risk on the next bet.

$$Kelly\ Criterion = (maxRisk) * \left(\left(\frac{PercentProfit}{100} \right) - \frac{1 - \left(\frac{PercentProfit}{100} \right)}{\frac{GrossProfit}{GrossLoss}} \right) \tag{7}$$

The Kelly Criterion is a percentage, which is multiplied by the size of the account to obtain the amount to bet. This amount is then divided by the price of the security to determine how many units to buy. For a highly successful trading system, the Kelly Criterion could suggest to bet an extremely large percentage of the account. In order to eliminate the possibility of betting the entire account, the criterion percentage is multiplied by the maximum risk that the trader is willing to take. This caps the criterion percentage to the trader's maximum risk. When this position sizing strategy was implemented, the trading system ceased to trade within the first few days because of the low ratio of winning to losing trades.

It was found that the strategy performed best when a notional account size was used. This approach is similar to that used by the Turtle Traders whose account size was updated, according to their performance, once a month. This method allows the strategy to risk a larger percentage of the account than the trader may be comfortable with.

A trader with a notional account size of $10,000 can place a $300 bet that represents 3% of the account (the actual account size is also $10,000). Following a losing streak, the actual account may be reduced to $5,000, but the notional account remains at $10,000. The next $300 bet still represents 3% of the notional account, but it represents 6% of the actual account. The bet is now much larger than what the trader may be willing to risk per trade. Although this approach may lead to larger trade risk, it allows the strategy to recover from losing streaks by increasing the bet size. If the strategy used a fractional approach, the bet size would decrease in tandem with the account size. Eventually the bets may be very small and the account would not recover from the small profitable trades. By maintaining the size of the trades the potential profit from winning trades remains high and the account could more easily recover. This is an extremely dangerous approach since a large enough losing streak would easily wipe out the account because the potential losses also increase in size.

Entry

The entries are determined using a sum of indicators multiplied by a weight that is assigned through optimization. The developed strategy weighed the following:

1. The direction of the price in the last hour
2. The direction of the price in the last day
3. The direction of the price in the last week
4. The direction of the price in the last six hours
5. Direction of the first bar of the day
6. Direction of the related commodity in the last hour
7. Value of the TRIN (Arms Index)

Since the sign of each indicator represents the direction of the price, the sign of the sum is assumed to represent the most likely direction of the price. If the sum is negative then it is likely that the price would gap down and vice versa. This prediction is cross checked with the exponential moving average crossover of the price of the traded security.

Exit

The strategy will exit the position as soon as the market opens if the position is not profitable. If the position is profitable then it will attempt to capture extra profits using a trailing stop that is adjusted for the price volatility using the Average True Range. Should the trailing stop not get triggered during the trading day, the strategy closes the position at 3PM before deciding in which direction to enter the market for the following gap.

Position Adjustment

Since the strategy predicts the direction of the overnight gap there is no place for position adjustment because the market is closed. The position size could be increased if the position was profitable, however the strategy only predicts the direction of the overnight gap, not the direction of the price throughout the day.

Optimization

The weights as well as the lengths of the exponential moving averages were optimized using the TradeStation backtesting optimizer.

Performance Report Analysis

The Strategy Performance report reveals that this particular strategy, although seemingly profitable, is not good enough to use for real trading. Both the profit factor and the Win/Loss ratio are very close to one meaning that overall the strategy will generate very small profits. This is due to the fact that it loses almost as often as it wins and the size of the wins are almost the same as the losses.

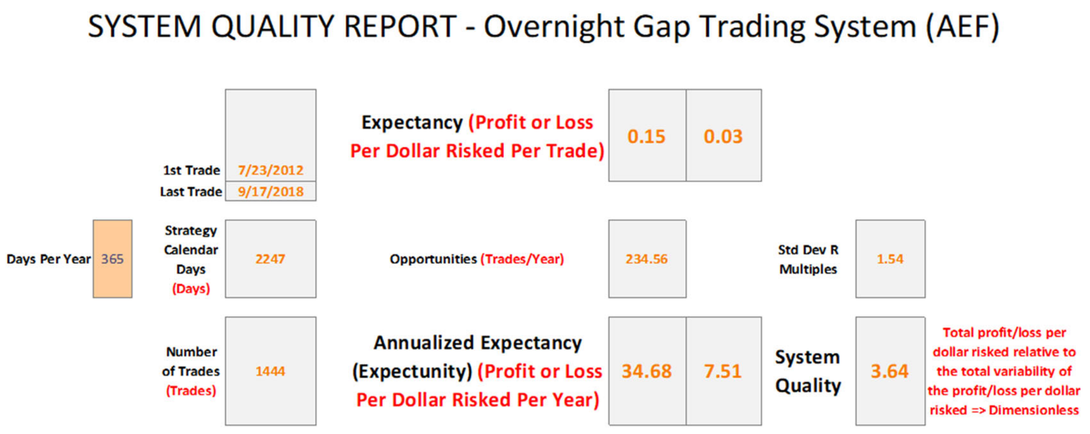
The Equity Curve shows that the returns of the strategy are somewhat consistent, with the best performance taking place in the last three years of trading. Interestingly, the bulk of the profits is made between the months of March and September suggesting that a possible step towards increasing the success of the strategy may be limiting trading to this part of the year. A quick look at the Underwater Equity Curve, which shows the monthly drawdown, paints a scary picture for a hedge fund manager. Between 2013 and 2015 the strategy topped 20% drawdown, a figure that would scare away most investors. Had this strategy traded between those years, a manager would have disabled it soon after it began to hit 10% drawdown in 2013.

The entry and exit efficiency graphs reveal that both entry and exit are slightly above 50% efficiency. Immediate improvements could be made to the exit strategy to increase the efficiency slightly and closer to that of the entries.

Analysis of the trades reveals that only a small percentage of the overall profit is the result of outlier trades. This result is positive in that it shows that the returns from the strategy are consistent between trades. Consistency is key for a hedge fund manager since investors tend to be risk averse and appreciative of consistent returns. Having large unpredictable losses is not an attractive quality for most investors.

The maximum drawdown of the strategy is quite large compared to the average size of the winning trade as well as compared to the overall profit over the trading period. Although the intraday drawdown is an unrealized loss, the possibility of having large losses many times bigger than the average profit is a good indicator that the strategy needs further development. Overall, even though the annual rate of return is 17.72%, the low profit factor, win/loss ratio and large drawdown make the strategy unsuitable for trading.

System Quality Report



SYSTEM QUALITY REPORT - Overnight Gap Trading System (AEF)

	1st Trade	7/23/2012	Expectancy (Profit or Loss Per Dollar Risked Per Trade)	0.15	0.03		
	Last Trade	9/17/2018					
Days Per Year 365	Strategy Calendar Days (Days)	2247	Opportunities (Trades/Year)	234.56		Std Dev R Multiples	1.54
	Number of Trades (Trades)	1444	Annualized Expectancy (Expectunity) (Profit or Loss Per Dollar Risked Per Year)	34.68	7.51	System Quality	3.64

Total profit/loss per dollar risked relative to the total variability of the profit/loss per dollar risked => Dimensionless

Fig. 20 - Overnight Gap Trading Strategy

The system quality report shows the expectancy and expectunity of the trading system. The report confirms that this particular trading system is not ready for deployment in the market. The expected return per dollar risked is extremely low $0.15-$0.03. With such a low expected return the investor is better off collecting interest payments from deposits in a savings account. The annualized expectancy is similarly low at $34.68 per dollar risked.

Monte Carlo Analysis

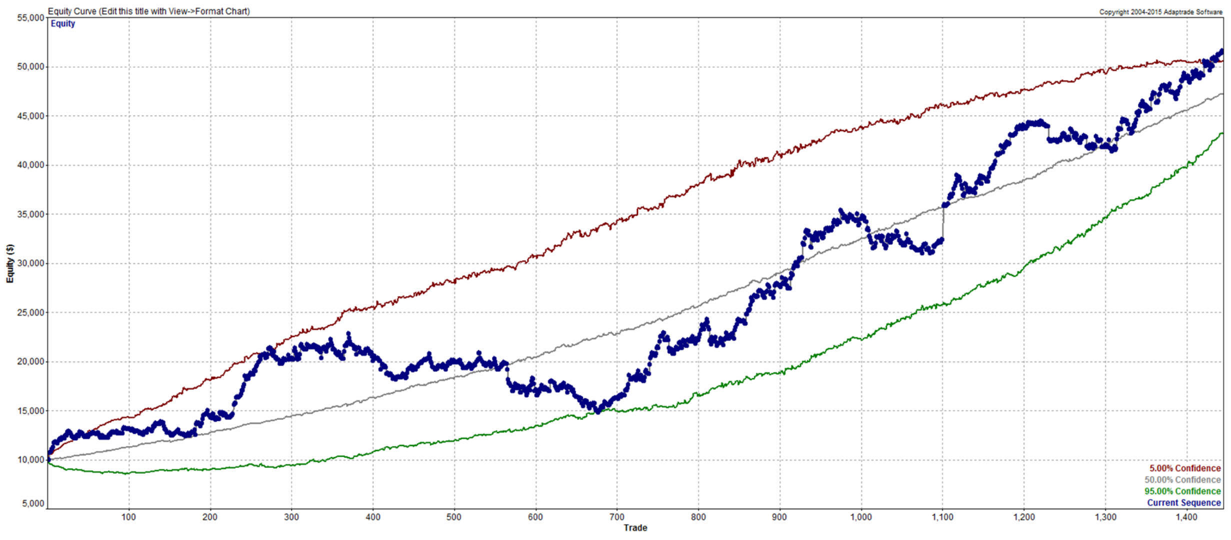


Fig. 21 - Monte Carlo Analysis - Overnight Gap Trading Strategy

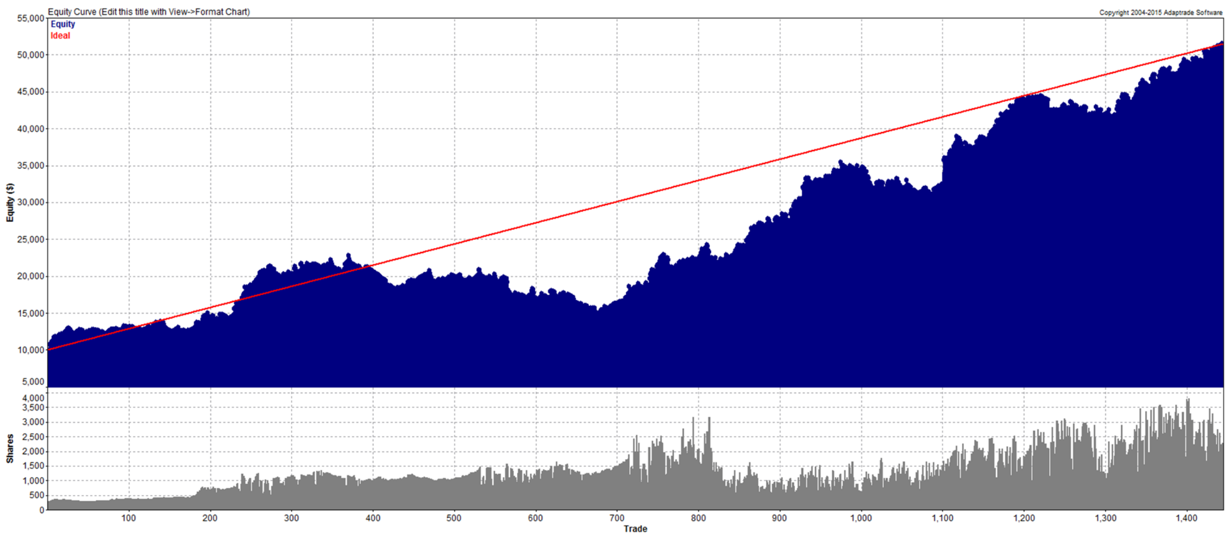


Fig. 22 - Equity Curve - Overnight Gap Trading Strategy

The Monte Carlo report produced by Market System Analyzer suggests that toward the end of the trading period the strategy began trading better than expected. This is not concerning, however, because at the 95% confidence level the strategy is still expected to produce a profit and the equity is expected to continue rising. A prediction over the last 60 trades shows that at the 95% confidence level the strategy was expected to perform worse than it did. This signals that it is possible that if the strategy continues trading it may produce losses.

7.2.3 Forex and Stock Market Breadth

Description

It is expected that a correlation exists between the market breadth indicators and the safe haven currencies in large timeframes greater than a day. This strategy, however, attempts to determine if there exists a correlation in a much smaller time frame: five minutes. The motivation behind exploring such a small time frame is the possibility that given the use of high speed trading algorithms by the major players in the market, there may exist the possibility that these algorithms are able to exploit even small movements in these timeframes and to trade between markets.

A strategy developed to determine whether a short term relationship between a set of market breadth indicators and safe haven currencies exists. The Market Breadth indicators used in the study are TRIN and TRINQ while the safe haven currency pairs used are USDJPY and USDCHF.

Position Sizing

The strategy uses a volatility adjusted position sizing approach similar to that used by the Turtle Traders. The volatility of the underlying security is calculated using the Average True Range over a certain number of bars. Refer to the Forex Correlation Divergence Strategy for a more detailed description of this position sizing strategy.

Entry

The strategy makes use of moving average crossovers to determine that the market conditions are ideal for a long entry. The inclusion of a market choppiness indicator stops the placement of trades when the market is choppy. Since the strategy looks to exploit short term trends, it is essential to avoid the placement of trades during a choppy market. Trend following strategies perform their worst during choppy markets.

Exit

The strategy will exit the trade when the exponential crossover indicates that the quote currency is depreciating. Once again trades are only executed when the market is not choppy. The strategy will not exit the trade until the market is not choppy.
Losing positions are eliminated after an optimized number of bars.

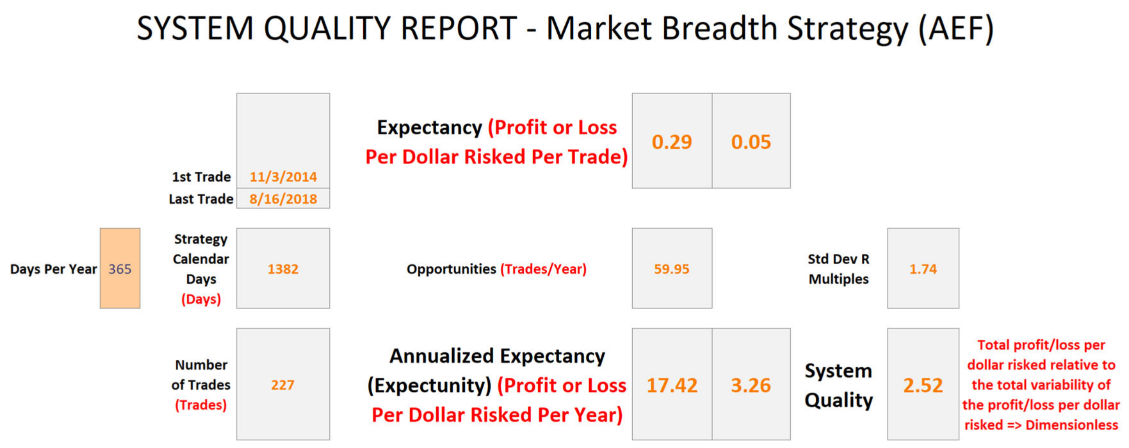
Position Adjustment

Positions are not adjusted given the short timeframe in which this strategy trades.

Optimization

Optimization was achieved using the TradeStation backtesting optimizer.

Performance Report Analysis



SYSTEM QUALITY REPORT - Market Breadth Strategy (AEF)

	1st Trade	11/3/2014	Expectancy (Profit or Loss Per Dollar Risked Per Trade)	0.29	0.05			
	Last Trade	8/16/2018						
Days Per Year 365	Strategy Calendar Days (Days)	1382	Opportunities (Trades/Year)	59.95		Std Dev R Multiples	1.74	
	Number of Trades (Trades)	227	Annualized Expectancy (Expectunity) (Profit or Loss Per Dollar Risked Per Year)	17.42	3.26	System Quality	2.52	Total profit/loss per dollar risked relative to the total variability of the profit/loss per dollar risked => Dimensionless

Fig. 23 - System Quality Report - Market Breadth Strategy

As shown by the performance report, this strategy fails to produce a profit. One concerning aspect of this strategy is the extremely low number of trades executed over four years. For a strategy that trades using a small timeframe it was expected that there would be a larger number of trades over such an extensive trading period. Perhaps the conditions specified for trading are too stringent apart from not optimal.

In contrast to the failure for the strategy at the five minute time frame, the same strategy optimized over a daily time frame resulted in profit. The Monte Carlo Analysis and System Quality Reports were not carried out for the strategy using the five minute timeframe given the failure of the strategy to produce a profit. The results presented in the appendix are for the strategy using the daily time frame.

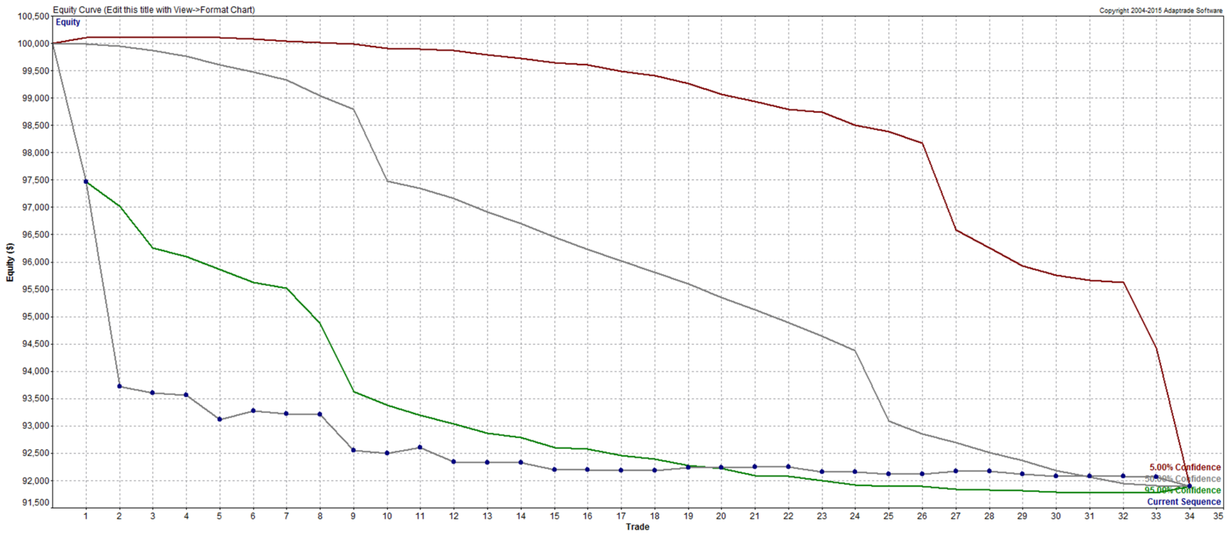


Fig. 24 - Monte Carlo Analysis - Market Breadth Strategy (5 Minutes)

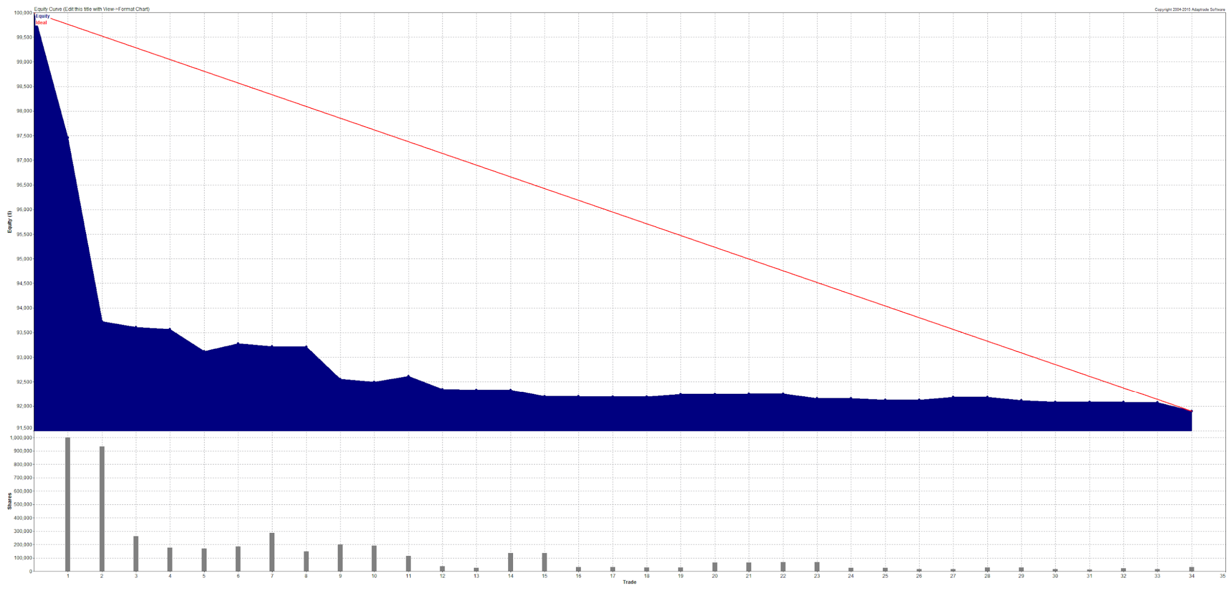


Fig. 25 - Trade Profit - Market Breadth Strategy (5 Minutes)

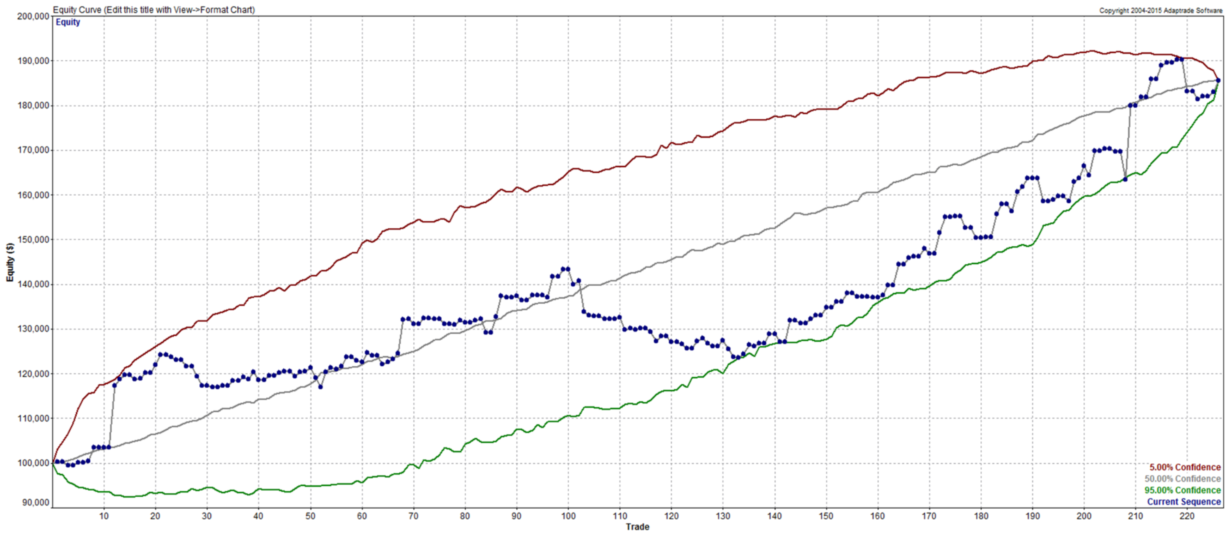


Fig. 26 - Monte Carlo Analysis - Market Breadth Strategy (Daily) X-Axis: Trades. Y-Axis: Equity ($)

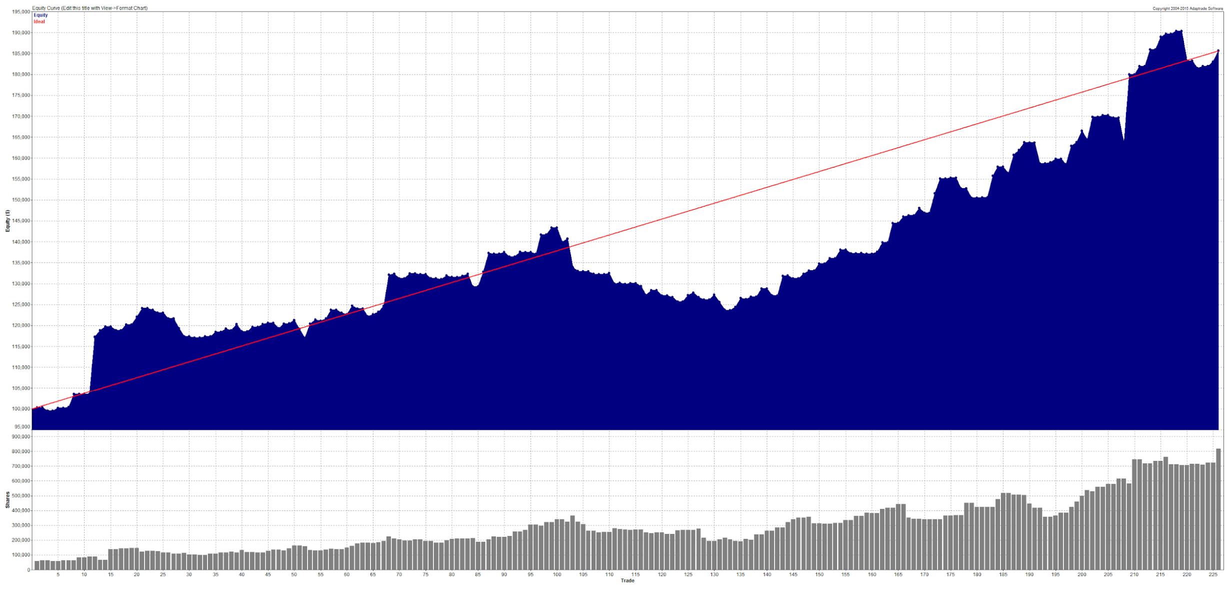


Fig. 27 - Trade Profit - Market Breadth Strategy (Daily). X-Axis: Trades. Y-Axis: Equity ($)

As evidenced by these reports, the market breadth strategy applied to a daily time frame has a positive expectunity higher than one and comparable to the success of the Forex Correlation Divergence Strategy. However, the expectunity is not good enough to merit real trading with the strategy. Trade analysis further reveals that the strategy is only successful in 40.7% of trades in average. Immediate improvements may be achieved by removing the long trades as only 24.32% of these are successful while short trades are successful 56.52% of the time. Furthermore, the maximum drawdown is exceedingly high at $7,573. Drawdown this high do not elicit a sense of confidence in traders, especially in combination with the low success rate of the strategy. The unrealized max drawdown is worse still at $19,768 given that this represents approximately 23% of the profit over the four year trading period. In summary neither market breadth strategy is ready for real trading. The strategy using daily bars requires improvements to reduce the drawdown, which may be achieved by either correcting the long trades or removing them completely.

7.2.4 Market Sentiment Indicator

Description

A Python script was developed to scrape headlines from Investing.com, and to perform sentiment analysis on the headlines. The use of news sentiment analysis is a well-documented approach to trading. In this sentiment indicator however, the headlines are analyzed rather than the body of the article. The motivation behind this approach is that the headlines are a short summary of the body of the article and may thus contain enough information to make an accurate prediction regarding the sentiment of the article. The site Investing.com was chosen because it is an aggregator of news articles from multiple sources and it provides no restrictions regarding scraping of the site unlike WSJ and Bloomberg.

The Python script also generated bigrams from the database of headlines scraped from Investing.com. Bigrams are every combination of word pairs in a sentence. For the purpose of this analysis, the collected bigrams are those that occur with higher frequency than usual. The NLTK library was used for the generation of the bigrams. Refer to the appendix for a list of bigrams collected from the headlines. The bigrams were classified as either negative or positive by the author. Given that the classification was done by the author it is possible that some of the bigrams may be misclassified due to confirmation bias. Of all the bigrams produced by the NLTK library only those that did not contain words present in either of the Loughran & McDonald (L&M) lists were selected. This is because the sentiment of a word is assumed to be constant except under the presence of the negator. None of the bigrams contained the negator, so the inclusion of a bigram with a word already present in the L&M list would only result in the double counting of that word and thus introducing more noise into the analysis. The sentiment indicator searches for the word ‘not’ in order to account for the negation of words. For example, the bigram ‘not good’ does not have a positive sentiment as suggested by the lexicon rather it has a negative value due to the presence of ‘not’. The algorithm will assign the opposite sentiment to any word included in the lexicon that is preceded by the word ‘not’. The overall sentiment of a headline is calculated by subtracting the count of positive words with the count of negative words in the headline. The resulting sentiment corresponds to the sign of the result.

A simple trading strategy was developed to test the ability of the sentiment indicator to predict the movement of the price. The strategy uses exponential moving average crossovers to trigger trades. The strategy was also optimized using maximum adverse excursion analysis to reduce the maximum drawdown. The performance of the strategy was compared over multiple forex pairs and indexes, for each of which a database of headlines was built by scraping Investing.com. Since some of the symbols were less popular and generated less news articles, the sizes of the databases were not identical.

Indicator Analysis

As evidenced by the pie charts included in the appendix, more headlines were classified as negative when the bigrams and negations were taken into account. The results were not consistent throughout forex pair, however, as evidenced by the analysis of EURUSD where there was no change in the number of headlines per category. The accuracy of the prediction was tested by comparing the prediction to the actual movement of the market for the same day. A table summarizing the results is available in the appendix. The results varied and the top predictions were USDJPY and DXJ while the worst predictions were USDCHF and EURUSD. These results were reflected by the performance of the trading strategy using the corresponding assets. The strategy performed best with USDJPY and worst with USDCHF. Naturally the assets for which the database of headlines was larger resulted in a larger number of trades. This is evidenced by comparing USDCHF, for which there was few headlines, with USDJPY or EURUSD which had thousands more. It is recommended that the indicator is used only with assets for which there are many headlines since a larger number of headlines per day will result in a higher likelihood of accurate prediction and more trades.

Position Sizing

To simplify the strategy and to facilitate the comparison of the performance of the strategy with different forex pairs, the position size was fixed to one mini lot.

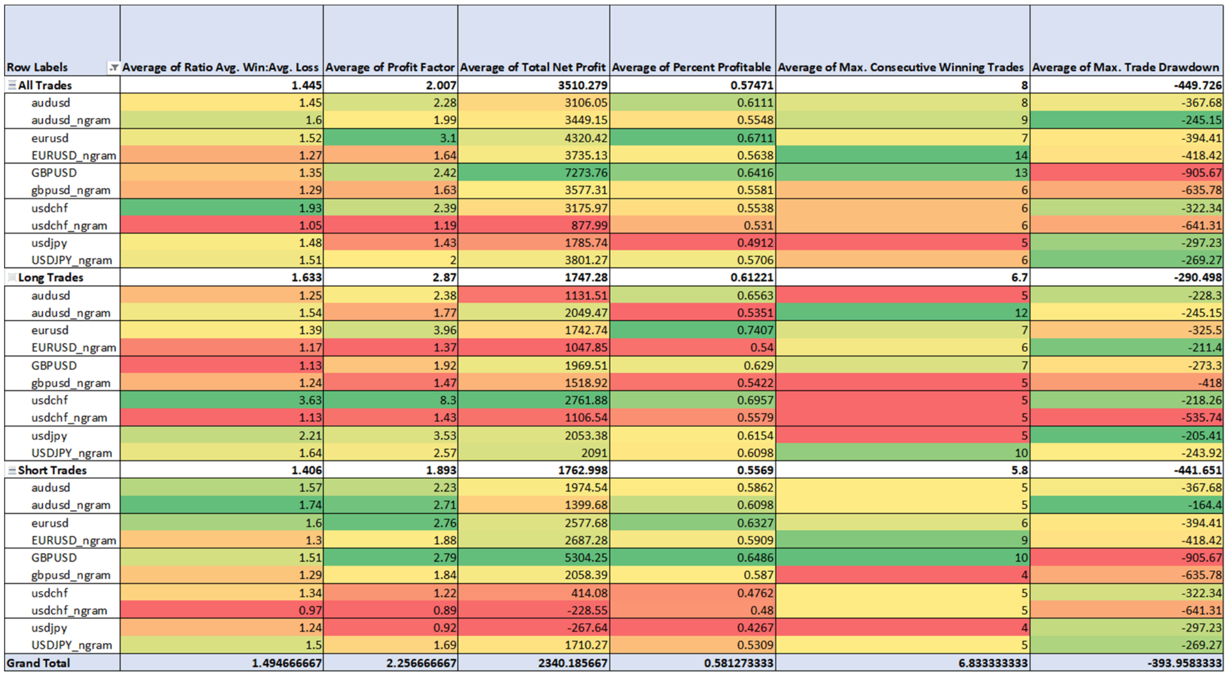
Entry & Exit

The entry strategy uses moving average crossovers to signal entry and exit conditions.

Optimization

Each strategy was optimized using the TradeStation backtesting optimizer.

Performance Report Analysis

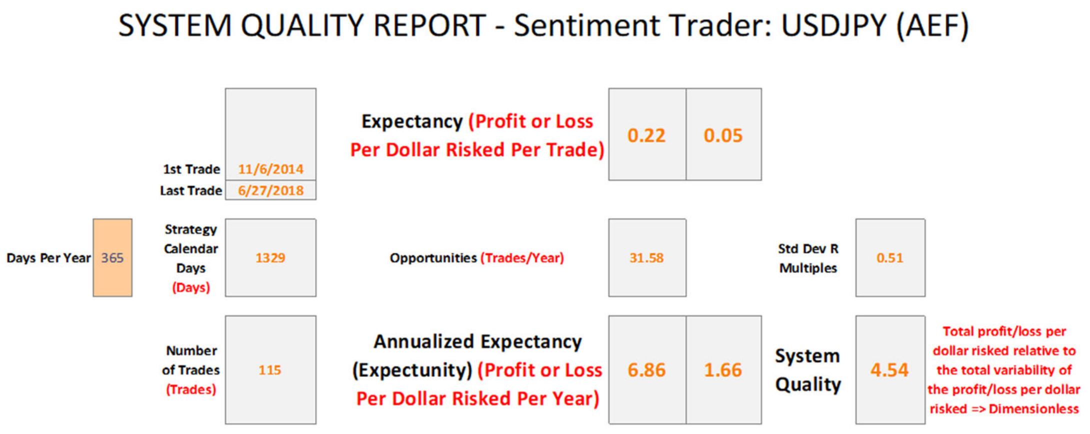


Row Labels	Average of Ratio Avg. Win:Avg. Loss	Average of Profit Factor	Average of Total Net Profit	Average of Percent Profitable	Average of Max. Consecutive Winning Trades	Average of Max. Trade Drawdown
All Trades	**1.445**	**2.007**	**3510.279**	**0.57471**	**8**	**-449.726**
audusd	1.45	2.28	3106.05	0.6111	8	-367.68
audusd_ngram	1.6	1.99	3449.15	0.5548	9	-245.15
eurusd	1.52	3.1	4320.42	0.6711	7	-394.41
EURUSD_ngram	1.27	1.64	3735.13	0.5638	14	-418.42
GBPUSD	1.35	2.42	7273.76	0.6416	13	-905.67
gbpusd_ngram	1.29	1.63	3577.31	0.5581	6	-635.78
usdchf	1.93	2.39	3175.97	0.5538	6	-322.34
usdchf_ngram	1.05	1.19	877.99	0.531	6	-641.31
usdjpy	1.48	1.43	1785.74	0.4912	5	-297.23
USDJPY_ngram	1.51	2	3801.27	0.5706	6	-269.27
Long Trades	**1.633**	**2.87**	**1747.28**	**0.61221**	**6.7**	**-290.498**
audusd	1.25	2.38	1131.51	0.6563	5	-228.3
audusd_ngram	1.54	1.77	2049.47	0.5351	12	-245.15
eurusd	1.39	3.96	1742.74	0.7407	7	-325.5
EURUSD_ngram	1.17	1.37	1047.85	0.54	6	-211.4
GBPUSD	1.13	1.92	1969.51	0.629	7	-273.3
gbpusd_ngram	1.24	1.47	1518.92	0.5422	5	-418
usdchf	3.63	8.3	2761.88	0.6957	5	-218.26
usdchf_ngram	1.13	1.43	1106.54	0.5579	5	-535.74
usdjpy	2.21	3.53	2053.38	0.6154	5	-205.41
USDJPY_ngram	1.64	2.57	2091	0.6098	10	-243.92
Short Trades	**1.406**	**1.893**	**1762.998**	**0.5569**	**5.8**	**-441.651**
audusd	1.57	2.23	1974.54	0.5862	5	-367.68
audusd_ngram	1.74	2.71	1399.68	0.6098	5	-164.4
eurusd	1.6	2.76	2577.68	0.6327	6	-394.41
EURUSD_ngram	1.3	1.88	2687.28	0.5909	9	-418.42
GBPUSD	1.51	2.79	5304.25	0.6486	10	-905.67
gbpusd_ngram	1.29	1.84	2058.39	0.587	4	-635.78
usdchf	1.34	1.22	414.08	0.4762	5	-322.34
usdchf_ngram	0.97	0.89	-228.55	0.48	5	-641.31
usdjpy	1.24	0.92	-267.64	0.4267	4	-297.23
USDJPY_ngram	1.5	1.69	1710.27	0.5309	5	-269.27
Grand Total	**1.494666667**	**2.256666667**	**2340.185667**	**0.581273333**	**6.833333333**	**-393.9583333**

Fig. 28 - Sentiment Indicator - Selected Measures Comparison

Analysis of the performance reports revealed that the strategy performed best when it was fed with the indicator that did not account for ngrams and negation. This is especially evident for USDCHF where the strategy with ngram produced a total profit of $877 while without ngrams it produced $3,175. A comparison of the profit factors again confirms that the use of the ngram decreases the success of the strategy.

System Quality Analysis

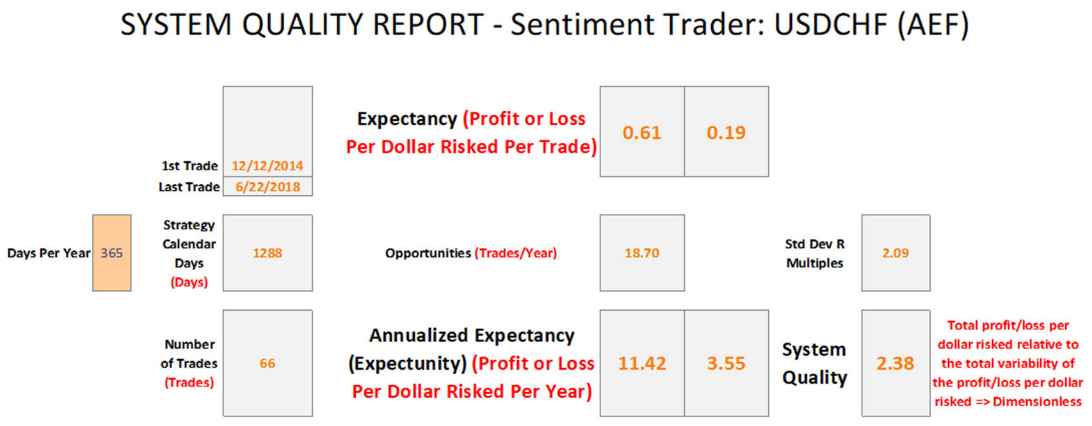


SYSTEM QUALITY REPORT - Sentiment Trader: USDJPY (AEF)

		1st Trade	11/6/2014	Expectancy (Profit or Loss Per Dollar Risked Per Trade)	0.22	0.05		
		Last Trade	6/27/2018					
Days Per Year	365	Strategy Calendar Days (Days)	1329	Opportunities (Trades/Year)	31.58		Std Dev R Multiples	0.51
		Number of Trades (Trades)	115	Annualized Expectancy (Expectunity) (Profit or Loss Per Dollar Risked Per Year)	6.86	1.66	System Quality	4.54

Total profit/loss per dollar risked relative to the total variability of the profit/loss per dollar risked => Dimensionless

Fig. 29 - Sentiment Trader: USDJPY



SYSTEM QUALITY REPORT - Sentiment Trader: USDCHF (AEF)

		1st Trade	12/12/2014	Expectancy (Profit or Loss Per Dollar Risked Per Trade)	0.61	0.19		
		Last Trade	6/22/2018					
Days Per Year	365	Strategy Calendar Days (Days)	1288	Opportunities (Trades/Year)	18.70		Std Dev R Multiples	2.09
		Number of Trades (Trades)	66	Annualized Expectancy (Expectunity) (Profit or Loss Per Dollar Risked Per Year)	11.42	3.55	System Quality	2.38

Total profit/loss per dollar risked relative to the total variability of the profit/loss per dollar risked => Dimensionless

Fig. 30 - Sentiment Trader: USDCHF

The quality of the strategy trading USDJPY and USDCHF was compared since these were two contrasting cases in terms of other success metrics. While USDCHF reports a higher expectunity, the system quality is higher for USDJPY given the lower variability of earnings in trades. Compare the standard deviation of 0.51 for USDJPY with 2.09 for USDCHF. Neither expectunity is high enough to merit real trading, but it may be concluded that the system works better with USDCHF. The win/loss ratio and average profit factor are also both also higher for USDCHF.

Monte Carlo Analysis

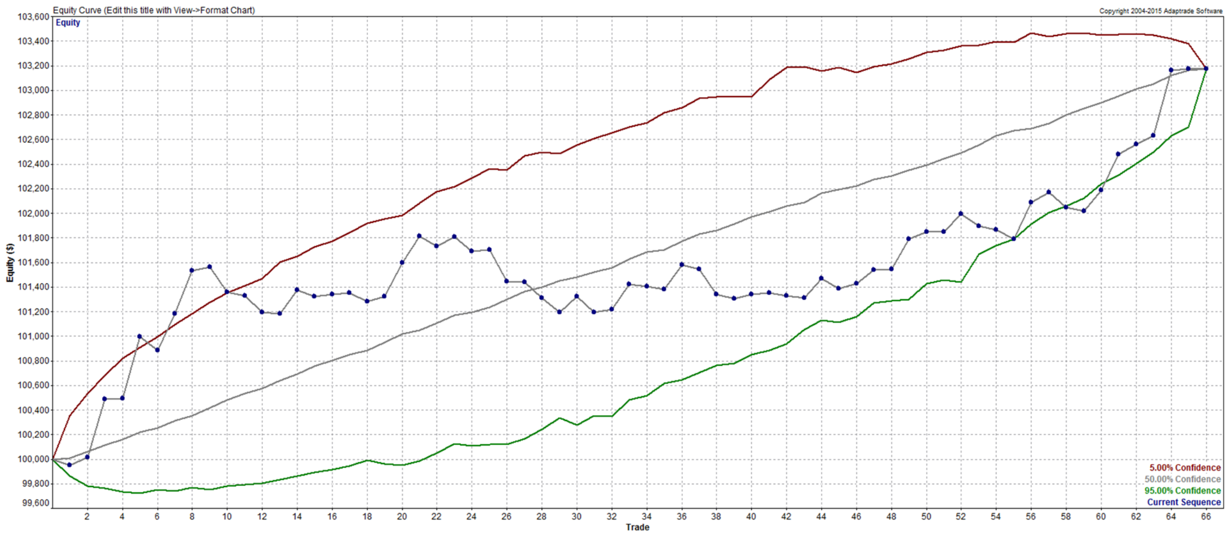


Fig. 31 - Monte Carlo Analysis - Sentiment Trader Strategy: USDCHF

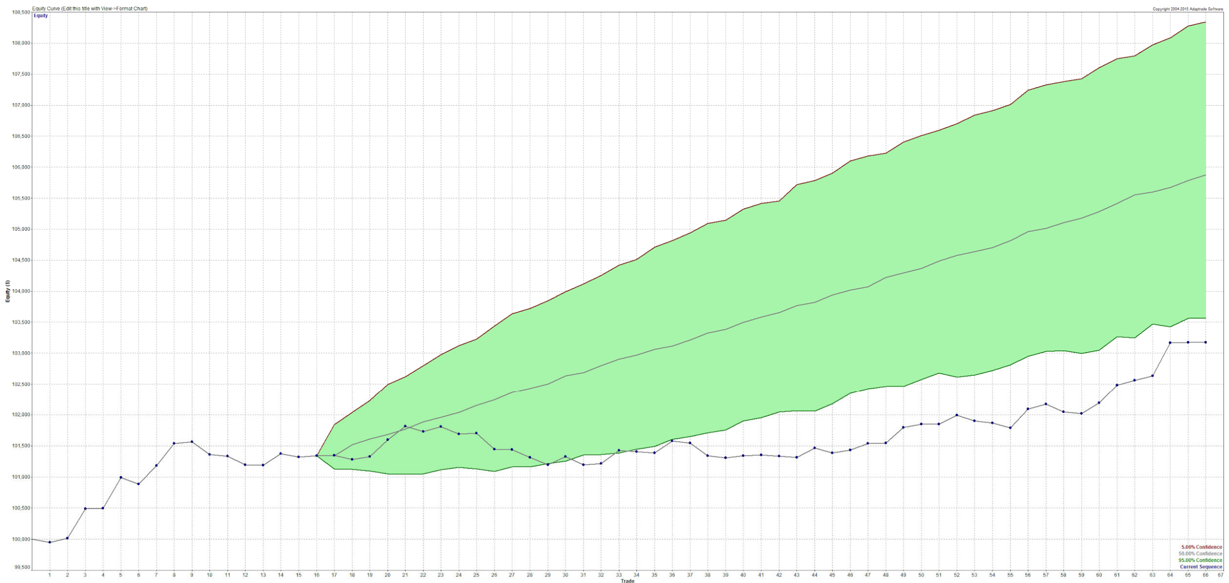


Fig. 32 - Monte Carlo Prediction - Sentiment Trader Strategy: USDCHF

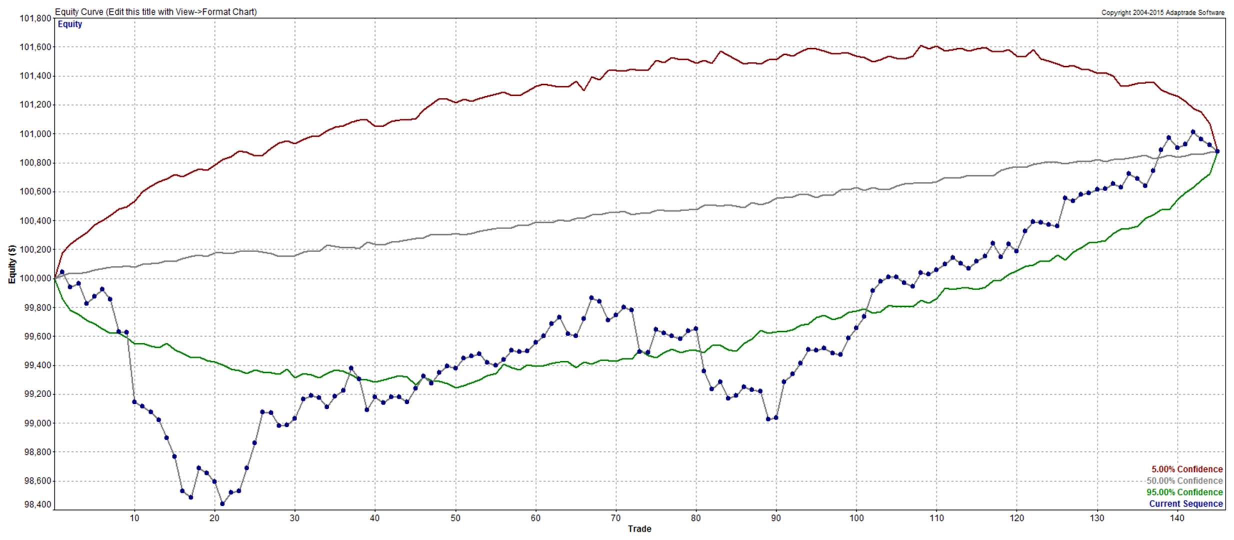
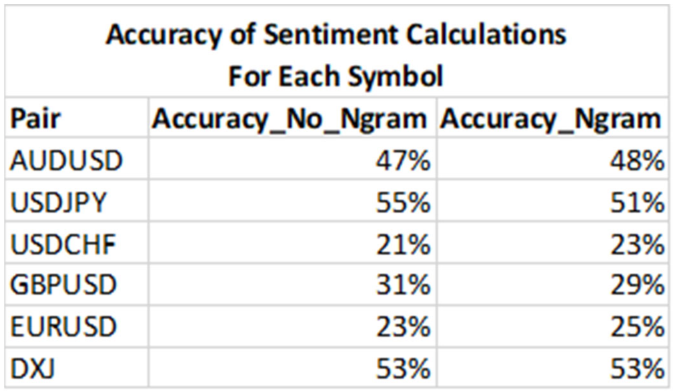


Fig. 33 - Monte Carlo Analysis - Sentiment Trader Strategy: USDCHF (ngram)

The majority of the Monte Carlo Simulations showed that the trade data extended beyond the 5% confidence boundary. This suggests that there was not enough data for the simulation to provide accurate results. Regardless, all but one of the simulations suggested that the strategy would not continue to deliver some level of profit in the near future. USDCHF with ngram was the only simulation where it was obvious that the results would remain slightly flat. Furthermore, in predictions using the last 50 trades where the actual data did not cross de prediction boundaries, most predictions suggested that the strategy would produce flat results or losses with a 95% confidence. The Monte Carlo analysis graphs are available in the appendix.

Strategy Analysis

The trading strategy devised to test the indicator was applied to all forex pairs twice: once accounting for bigrams and negation and once without accounting for these two characteristics. The results of the strategy under each condition are listed in the appendix. A comparison of the results of the application of the strategy for the past four years (2014-2018) shows that the strategy performs best when it does not account for the bigrams. The deterioration of performance with the addition of bigrams is likely due either to the misclassification of the bigrams or to the incorrect assumption that the bigrams contain a sentimental value. Bigrams that were incorrectly included in the analysis would have resulted in the addition of noise into the prediction. Interestingly enough, the addition of the ngram to the analysis results in a larger number of trades in comparison to the strategy that uses the indicator without the ngrams.



Accuracy of Sentiment Calculations For Each Symbol		
Pair	Accuracy_No_Ngram	Accuracy_Ngram
AUDUSD	47%	48%
USDJPY	55%	51%
USDCHF	21%	23%
GBPUSD	31%	29%
EURUSD	23%	25%
DXJ	53%	53%

Fig. 34 - Sentiment Indicator - Percentage Accuracy Comparison

8. System of Trading Systems

8.1. Rationalization

As explained in the overview of this report, one of the major goals of this research is to produce an AI-driven “system of systems” which manages trading with the algorithms described above. The development of this system is driven by the central idea that both manually generated trading algorithms and AI-based pattern detection have unique strengths and weaknesses.

On the one hand, manual trading algorithms can take into account various trade-by-trade intricacies, including stop losses, margin trading rules, commission costs, and profit targets. Manual trading algorithms can be contoured to consistently make money when specific patterns are present. Their weakness, however, is designing suitable entry and exit conditions that properly identify and fully profit from the patterns they are designed to exploit. Manually generated algorithms can run into cases of false positives where their target pattern is misidentified, upon which there is a high risk that money will be lost.

On the other hand, artificial intelligence excels at finding patterns in data. A neural network’s layered perceptrons allow for the system to extract non-trivial predictive relationships. As described in the literature above, abundant research has been done to demonstrate the ability of neural networks to predict the future value of stocks. It is plausible that neural networks have the predictive capacity to discover the sorts of patterns that manual algorithms could trade well. However, as explained above, there is no way to directly input the rules inherent to individual trades, like stop losses, margin trading rules, etc. Neural networks can take a set of features and produce predictions; this is a far cry from a robust trading algorithm that manages each trade.

Work has been done to reduce the weaknesses of both of these implementations. In the case of manually-generated trading algorithms, each algorithm can be manually bundled into a system that minimizes variance of return, thus insulating against cases where one algorithm's mispredictions dominate the portfolio. In the case of AI-driven indicators, excess work must be done to translate raw predictions into a workable trading system, and often the raw predictions of the indicator itself are rendered meaningless when factors like commission cost and slippage are taken into account.

The solution presented by the system of systems approach is to allow both a manually-generated component and an AI-driven component control complementary parts of the system. In this way, each component may buffer the other's weaknesses, leading to a robust and consistent trading system. At its fundamental level, the system of systems uses manually-generated algorithms to generate the actual trades of the system. They can best handle the minutiae of each trade and ensure that the system makes money on a profitable pattern. For each algorithm, the system generates a neural network that decides how heavily to rely on that algorithm for a given trading period. The network thus has two functions: first, it implicitly discovers the patterns for which the underlying algorithm performs best, and second, it tries to predict future market patterns to determine how well the algorithm will perform. The critical feature of this design is that the neural network component only has to focus on pattern prediction, and the manually generated algorithms can be designed to maximize trade by trade profit on specific patterns. This is a system of systems, because each manually-generated sub-system is being managed by a central AI-generated super-system.

It is under this central design principle that we rationalized the construction of the system of systems to manage our trades. By feeding the algorithms listed above into the system, and generating a network for each, the system may optimize performance for the algorithms as a whole.

The following sections will describe the implementation of the system of systems and present results on its performance. Section 8.2 provides a high-level overview of the design decisions taken at the outset of implementation. Section 8.3 through 8.5 describe the implementation of each component of the system. Finally, Section 8.6 analyzes the performance of the system of systems, and Section 8.7 presents conclusions and offers some reflections.

8.2 Design Overview

8.2.1 High-Level Breakdown of Design Goals

Since the individual manually traded algorithms are borrowed directly from the algorithms described above, the rest of this section will discuss the development of the AI supersystem that manages these algorithms. Supersystem development can be broken down into three major components:

1. A backtest data generator, which farms and organizes data to generate examples for supervised learning.
2. A neural network framework which can train various network architectures and visualize predictive accuracy.
3. A trading platform which actuates trades based on predictions from the trained neural network and displays basic performance metrics of the system as a whole.

This design structure makes sense since it accommodates the general features of a successful AI-driven application. First, raw data must be accumulated and organized into a training set for the neural network. The platform that generates the training set will make decisions on the format of each training example. This includes things like input data size and the format of the network's predictions. Most importantly, this component will decide on the kinds of data that are represented in the input features; it is therefore responsible for transforming raw data into each category of input feature. An API would be used here to simulate trading with historical datasets. This component should return generated training sets with fully-calculated training examples.

These examples must be ingested by the second component to build and train actual neural network instances. This component must process the examples into training, testing, and validation sets; it should also provide preliminary metrics on data variance and predictability. It then needs to interface with any number of neural network APIs to construct the networks themselves. Finally, the component should house a set of rigorous testing applications and data visualization tools to accurately assess the predictability of the networks. This component produces a fully generated set of weights and biases which hold the state of a trained network. In a sentence, this is the generic neural network training set that almost any AI project must have.

Finally, the third component uses the generated neural network to balance trading algorithms and conduct trading. An API should be employed to generate trading strategies and simulate a trading session. A lot of code is therefore shared between the first and third components, since both need to simulate trading sessions with different algorithms. Just like in the second section, a set of testing applications should be employed to test profitability of the system, along with variance, expectancy, etc. This system should be able to both trade using historical data and trade in real time.

Our implementation of the supersystem follows the design paradigm described above. We focused on building a working version of each component before moving on to the next one, since each component relies on outputs from the last one. As we were building the system, our design perspective was to sequentially transform raw financial data into an informed trading platform; this helped guide the order in which we implemented code. As described later in this report, we understood from the beginning that we reasonable only had enough to present a "Proof of Concept" for the supersystem design; therefore, we only did as much as we needed to finish each component and generate presentable results. In any time that remained, we returned to each section to add additional features, error checking, and unit tests.

8.2.2 Implementation-Wide Coding Decisions

Language Choice: Python

From the outset of the project, we were behooved to make decisions on the tools we would use to implement the system. The first and most fundamental choice we made was on the programming language of our implementation. As important as this choice is, it was not a difficult one to make; we knew almost immediately that Python would be our language of choice. Python is very well known for its neural network libraries, including Tensorflow and its API wrapper, Keras. We understood that we could quickly generate trained neural networks using these libraries because they abstract away a lot of the technicality themselves. Python also has many data analysis and visualization tools, like matplotlib and scikit, which would ease our burden of reporting the predictive capabilities of our networks. On top of this, many APIs exist in Python that can run trading simulations with historical financial data. We understood from the start that this project would be infeasible if we needed to make an algorithm backtesting API from scratch; this made the API options afforded by Python very attractive. In general, we found that Python eased the requirement for us to build the large programs that are foundational to our system, allowing us to focus more on the system itself.

Besides the API support Python offers, Python itself is a very good language for data manipulation. The language itself is designed to be simple and readable, so it is easy to connect together sections of code that implement different parts of the supersystem. Tools like numpy also make it easy to generate and manipulate training data. There is very little overhead to apply different APIs to the data. This allowed us to provide a robust and visual analysis of the system's results. On top of all of this, Python has an active community of users, especially in the field of artificial intelligence. Community support is always useful when implementing a project that spans many different areas of the language.

Financial API Choice: Zipline

While it was not difficult to choose a programming language for this project, the correct choice of finance API was not immediately obvious. Before continuing, the term “finance API” should be well defined. A finance API at its core is able to represent different trading algorithms and “backtest” those algorithms with historical financial data. It should operate on data sets from different sources and on different time scales. It must be adaptable enough to accurately represent the strategies described above, and must be detailed enough to accommodate for trading peculiarities like stop losses, profit targets, etc. There were other metrics we evaluated beyond the functionality of the finance API itself based on how easily we could integrate the API into our larger system. While we were not completely opposed to paying money for an API, we were heavily biased toward finding a free solution for the system. We also wanted to find an API that would separate itself from a Web service as much as possible. The reason for this was twofold- first, we wanted to make the system of systems as independent of other services as possible, and second, we knew that a Web-based API like Quantopian limits the number of libraries that can be used alongside the API. We were willing to accept the extra time and effort necessary to work with an open-source API.

Bearing the above parameters in mind, we decided that Zipline would be a suitable finance API for the system of systems. Zipline is an offline implementation of the Quantopian Web API. As a result, a lot of functionality and documentation is similar between the two. The Zipline project repository is regularly maintained, which minimizes the risk of us relying on discontinued or deprecated code. Not only is Zipline free, but it also accepts datasets from arbitrary sources provided that they fit a specific data format. This is a very large benefit for us because Quantopian’s base data is insufficient for algorithms at small timescales. Zipline’s use of a main state object provides a centralized means to retrieve any data we need for backtest generation. We believe that the benefits outlined above make Zipline the best candidate for our application.

Of course, Zipline does not come without its faults. Since Zipline is open source, it relies on free APIs to acquire basic information it needs to run simulations. It relied on the Google Finance API until that library became defunct in 2018; after that, we needed to find a substitute API (in our case, Yahoo Finance) and manually substitute the API in the Zipline source code. Also, Zipline’s documentation does not cover all that the library has to offer; this led to a lot of trial and failure as we dove deeper into using advanced functionality. As a result, we have found a lot of information which we believe would be beneficial to add to the documentation. We understood that these issues would be present when we decided on using the Zipline API. We accurately assessed that Zipline’s positives were worth taking extra time to work with an open-source API.

Dataset Choice: TradeStation

The final choice we made before beginning the implementation of the system was the historical data source we would use to run Zipline's simulations. Thanks to Zipline's abilities to ingest generic data sources, we were free to pick from a diverse set of dataset sources. We were mostly looking for two parameters: cost and data variety. As a common theme of this report, we were heavily biased towards finding free or open-source solutions. On top of this, we wanted to find sources that offered a variety of assets at different timescales, up to minute-bar data. With that being said, the following list below provides the pros and cons of some of the sources we considered:

- eSignal:
  - - API documentation webpage 404s
- Interactive Brokers:
  - + Available through the use of official libraries for Java, C++, Python, and C#
  - - Convoluted pricing structure
- TradeStation:
  - + Minute-by-minute market data available going back 20+ years
  - + Already available to students taking this IQP
  - - Data has to be manually exported (CSV) from the TradeStation desktop client (Windows only!) under the plan available to WPI students.
  - Web API available (JSON or XML) for a $21 / month fee after being reviewed by TradeStation.
- Quandl:
  - + CSV, JSON, and XML formats
  - + Libraries available for R, Python, and Excel
  - + Free WIKI stock price dataset available – mirrored by Quantopian, bundled with Zipline by default
    - Daily pricing data for 3000 publicly-traded American companies.
  - + A large number of additional datasets are available for free (including daily pricing data for Bitcoin and Litecoin and daily pricing data from the Tokyo Stock Exchange)
  - - A large number of more detailed paid datasets are available as well.
  - - Esoteric "alternative data" is available for an undisclosed price.
- Alpha Vantage:
  - + Freely available API with intraday, daily, weekly, and monthly historical stock and cryptocurrency data.
  - + CSV and JSON formats
  - + Provides a wide variety of standard indicators on demand
  - - Intraday data only spans the last 10 to 15 days

We were most impressed with TradeStation's ability to provide detailed minute-by-minute data for a very large variety of stock assets. We also liked Quandl's ability to generate datasets for unconventional assets like cryptocurrencies. While TradeStation typically has a large fee, our access to the TradeStation platform provided by the IQP allowed us to farm all the backtesting data we would need into CSVs for permanent use. We, therefore, decided to use TradeStation as our primary dataset source. For any specialized assets unavailable to TradeStation, we agreed that we would pay a fee for Quandl's services.

Neural Network API Choice: TensorFlow

As explained in the design overview above, one of the goals of this project is to use neural networks to classify trading algorithms. To complete this task, we needed to find an API in Python to handle the underlying design and implementation of the networks themselves. We wanted to abstract away as much of the internal neural network architecture as possible, leaving us with a high-level API that could quickly build networks of varying architectures. Tensorflow was clearly the library of choice to implement our networks. Tensorflow enables the implementation, training, and testing of many diverse network architectures. It also has the capability of reporting advanced performance metrics which are key to diagnosing the factors of network error described in Section 5. Finally, Tensorflow has rich documentation and wide community support which we took advantage of throughout the project.

The decisions detailed above were made before implementation began for the system of systems. The following three sections detail how each subsystem was designed and implemented, guided by these project-wide decisions.

8.3 Implementation of the Backtest Data Generator

The backtest data generator is the most important component of the system of systems, and by necessity the first to be completed. As a general rule of thumb, most time in any AI project will be spent writing code to generate and tune training samples. Our case was no exception, for this component of the code took half a year to reach a working state. As a result, this subsystem relies on many components which in turn use tools from a wide range of external libraries.

8.3.1 Desired Behavior of the Backtest Data Generator

As explained in Section 8.2, the backtest data generator must take raw historical financial data and manually-generated algorithms and produce a set of training examples with which a neural network could be trained. Each training example reflects the operation of the manually-generated algorithm over a period of historical training data. It consequently represents both the financial context of that historical period and the performance of the algorithm. The backtest data generator must therefore collect financial behavior in whatever context is most fitting. It then must run an algorithm backtest API (in our case, Zipline) over the same historical period to process how the algorithm would have performed over that period.

Our training examples take past performance as input features and return future performance as the target for the network to predict. Therefore, two simulations are run for each example, with the second simulation starting when the first simulation ends. The first simulation is paired with financial data to generate the input features; the second simulation is converted to some metric for the output of the network. In this way, the neural network must predict the performance of an algorithm over the next *n* time periods, given the last *m* time periods of financial data and performance. Each training example varies based on the asset it is using and the reference point in time from which the two simulations are run. For instance, a training example could be generated for AAPL on 5/07/2005. The first simulation would run from 05/01/2005 to 05/07/2005 to generate performance data for the input feature, and the second simulation would run from 05/07/2005 to 05/14/2005 to generate the performance metric that the network needs to predict. In this way, a stock can be used to generate many training examples by assigning each one a unique reference date.

To achieve the goal of generating training sets, the backtest data generator needs several components. First, the generator needs a parser to transform raw historical data from arbitrary sources into data usable by our financial API. Code must be built to leverage Zipline's backtesting abilities such that backtests can be arbitrarily defined and run with different parameters. Any algorithms written for TradeStation must be rewritten in Python so that Zipline can run backtests with them. This includes porting over indicators available in TradeStation as a custom external Python library. A system must exist that uses Zipline to measure historical data and algorithm performance. In turn, different performance indicators and financial indicators must be programmed manually into the system. Finally, a file I/O system should exist that loads training examples into .csv files and saves them for the neural network to train on.

The following subsections detail how each component of the code was implemented. Reasons are also provided for why certain financial and performance metrics were used in the training examples. Whenever possible, code will be shown to illustrate how each component was implemented into the data generator. The majority of this section will describe training set generation for the convolutional networks; a subsection will be dedicated at the end to present the peculiarities of LSTM data generation.

8.3.2 The Data Ingestion Pipeline

Zipline represents sets of financial data internally in its own data bundle format. By default, it has access to the quantopian-quandl bundle, which encapsulates quandl's WIKI dataset for use in Zipline. In order to achieve our goal of using TradeStation data, we created a Python module that can bundle financial data in CSV form for use with Zipline.

Our implementation of this is the *FileCSVBundle* class. The *FileCSVBundle* class utilizes the Pandas function *read_csv()* to represent a given csv as a Pandas *DataFrame* and the standard Zipline bundle api to package and ingest the data.

8.3.3 Integrating Zipline into the Generator

After transforming the data into a state that Zipline could accept, we began to apply Zipline's API to the generator. Most importantly, we created Python classes with member functions that had Zipline functionality built into them, so that users of the system would not need to interface with Zipline directly. This helps maintain the implementation-agnostic design goals we had in mind. We knew that future maintainers of this project might want to replace Zipline with a more powerful backtesting API, which class encapsulation makes more feasible.

The central class that implements Zipline functionality is the *Strategy* class. It maintains two types of information: first, it holds a strategy itself which Zipline can ingest, and second, it holds metadata which the backtester uses to test the strategy. In this way, each strategy is "bound" to a specific set of backtest parameters. While this limits how strategies can logically be organized, we felt it was useful in the context of how we were organizing the generator. An essential method of the *Strategy* class is the *run_backtest* function; this method instantiates and runs a Zipline simulation, returning the results of the simulation back to the data generator. Other member functions initiate, modify, and maintain the parameters that configure each Zipline backtest. This includes the data bundle to be used, the asset to trade with, and the start and end date of the simulation.

The *Strategy* class is implemented in two subclasses: the *DailyStrategy* and the *MinuteStrategy*. As the titles imply, *DailyStrategie*s are designed to operate on daily bar data, while *MinuteStrategie*s operate on minute bar data. This distinction is necessary because Zipline follows the NYSE's historic trading calendar to run its simulations. Work must explicitly be done by us to make sure our requested trading periods result in the same number of trading days or minutes. For example, our system needs to know if a trading period falls on a federal holiday and add one extra day of trading to compensate for the day lost. Trading parameters are encapsulated in a *StrategyParameters* class which handles the low-level work of transforming dates on the financial calendar to produce start and end times.

Each *Strategy* class houses its own trading algorithm which Zipline runs in its backtest. Zipline adopts arbitrary algorithms by only requiring a function that can generate Zipline-defined buy and sell orders. This function gets called at each trading bar of the backtest. Zipline updates a context object with prior financial data which the algorithm uses to make its buy and sell decisions. For example, the algorithm may ask Zipline to give it the past ten days of open and close prices, from which the algorithm could make a buy order. Zipline simulates the process of ordering an asset as factually as possible. It factors for commission, slippage, and even taxes for each order made. Orders can be made by absolute share number or by portfolio percentage. Each piece of Zipline's order functionality has a wrapper method in the *Strategy* class for the hosted trading algorithm's use. By utilizing both Zipline's context data structure and order API, the *Strategy* class allows arbitrary trading algorithms to be implemented.

Since Zipline is written in Python, its functionality is mutable at runtime. This means that while the code is running, a Zipline backtest can be reconfigured to include or omit arbitrary information. Zipline reports a day by day summary of a backtest simulation as a Pandas *Dataframe*. Since Python's runtime mutability extends to class members, this dataframe can report any information desired at any time during backtest data generation.
For example, we needed Zipline to report additional information to calculate performance metrics that we wanted to append to the input feature vector.

The trading algorithms embedded within each *Strategy* class take advantage of Zipline's mutability to record data necessary for training set generation. The *Strategy* class has a set of helper functions which must be called in each trading strategy to add this necessary data. These helpers also modify the behavior of the algorithms themselves. The central helper function, *nn_record*, is responsible for keeping track of the start and end dates of a trade, which Zipline does not do automatically. It also compensates for peculiarities in how Zipline interprets the trading calendar which make some orders execute a time unit after an order is placed.

Another set of helper functions ensure that all trades have completed by the end of a simulation. This is important because the results of a backtest would be inaccurate if all positions were not liquidated by its end date. The helpers allow for two types of behavior: forcing total liquidation and preventing any new positions. In the former case, a strategy may be forced to cancel all orders, sell all positions, and essentially stop trading. In the latter case, a strategy may only be told to stop initiating new orders after a specific date. While forcing total liquidation guarantees that no positions will remain when the simulation ends, it may force a trade to complete prematurely, skewing the performance of the algorithm. Our general technique adds extra "sell time" beyond the desired time frame of a backtest, during which no new positions can be made but positions may be sold. After that sell time, the strategy is forced to liquidate all positions. This technique makes an effort to preserve the true behavior of the algorithm, but also ensures that all positions are liquidated.

Zipline is not just used in the backtest data generator for the assembly of strategies and the operations of simulations. We also use Zipline as a generic tool for farming financial data that we want to add to our training examples. This is done by creating and running a special Zipline strategy called the Generator in parallel with each backtested strategy. The generator never actually makes trades; all it does is populate its DataFrame of results with lots of financial data and calculated indicators that can be used for the input feature vector. Zipline backtests support a Pipeline feature which calculates different indicators and maintains their current value for any strategy to use. For example, an exponential moving average indicator could be loaded into the pipeline so that a strategy is always made aware of the n-day EMA of its asset when it generates orders. We were able to farm arbitrary financial data from Zipline by loading the pipeline with the various indicators we wanted and running a dummy strategy that simply reports those indicators in the DataFrame. By running both the Generator strategy together with the actual algorithm strategy that we wanted to backtest, we had enough raw information to construct our training examples.

8.3.4 Refactoring Algorithms into Zipline Strategies

Having acquainted ourselves with Zipline's tools and integrating relevant elements of its API into our code, our next task was to build a suite of helpers to implement EasyLanguage algorithms into Zipline. The main challenge here was to learn the underlying math behind various EasyLanguage indicators and replicate them in Python code. Additionally, by adding a wrapper to each indicator we implemented, that indicator could serve as a pipeline factor and be queued in the Generator pipeline. By extension, our decisions here also affect the input feature vector of the produced training examples. If incorporating the Zipline API to our code built the framework for the data generator, then writing indicators in Python outlined the financial calculations for that Zipline framework to implement.

Each indicator extends the Zipline's CustomFactor class to make custom Zipline pipeline factors. Each class holds a wrapper function that makes the class compatible with the Zipline pipeline and a central function which holds the actual means to calculate the indicator. Within the class, an input array holds Zipline keywords that direct Zipline on the kind of data it should provide the indicator (i.e. open/close price, high/low price, etc.). The main function usually takes a set of financial data of the past *n* days and returns an array of outputs from which the wrapper can select and report an individual "indicator" value. For example, the PercentK indicator takes the high and low prices for the last *n* days and the closing price for a given day. It outputs a single value representing the difference between the closing price and the lowest low of the past *n* days divided by the difference between the highest high and lowest low of the past *n* days. By building one indicator class for each desired indicator, we populated a library of indicators for use by both the trading algorithms and the generator pipeline.

It is important to remember here that we intended the system of systems to be a proof of concept, and not a fully optimized system. We prioritized having the major components of the system implemented, and then spent our remaining time improving that implementation. It follows that we did not have enough time to consider what indicators we should use as input features, and instead tried to find a distribution of indices that each represented a unique piece of information. We believe that this is sufficient for generating a proof of concept, but we also acknowledge that more work should be done in the future to refine our indicator selection.

With that being said, our main consideration when selecting indicators was the amount of predictability they would add to the network. Recall that each training example centers on a single point in time. The input features take financial data and algorithm performance *before* that moment in time, and the output feature represents performance of the algorithm *after* that moment in time. This means that indicators help the neural network make decisions about the performance of an algorithm by better clarifying the future performance of the asset it is trading. It follows that the indicators we pick should each represent unique market trends that help inform the network of how the market will behave in the future. Indicators should never be expressly used to represent how the algorithms make orders. We selected our indicators under this main principle.

We also made an effort to gather indicators of varying complexity. We reasoned that since these indicators would be ingested as input features into a neural network, the complexity of those indicators would be a deciding factor in how useful they would be for the network to make decisions. On the one hand, simple indicators provide values based on relatively few inputs and simple calculations, making it easy for the network to infer financial behavior. On the other hand, complex indicators may represent information that the network would not have easily created on its own. We wanted to see what types of indicators would be best suited for the network, so we generated both simple and complex ones. A detailed description of each indicator we used is given in the results of this paper.

8.3.5 The Data Generation Pipeline

Having fully implemented both Zipline functionality and the algorithms, we felt prepared to build the fundamental code of the system, which runs subsequent backtests and generates training examples. This code receives a desired range of time and selection of stocks to use for building all the training samples. It also allows adjustments for the number of time periods each training examples' prior and future performance backtest should take. Users can also specify the kinds of indicators to use in the input feature vector. The training set outputs can be either categorical or regression, but both are based on the future performance of the algorithm. If categorical, each output will be assigned a numeric class based on percent gained or lost; if regression, each output will be the raw percent gained or lost. For example, a training set could be generated that runs backtests for a daily bar strategy from 2005 to 2015 for three different stocks, with four weeks reserved for each backtest (two to learn prior performance, and two to create the output feature). This code is housed in the datagen function; its wrapper, datagen_pipeline, is the calling function for running the backtest data generator.

Within the datagen function, the internal process of running a backtest is repeated many times. For each asset, the pair of backtests required to create a training example is repeated, incrementing their start times until the range of time from the desired start date and end date are covered. For example, if the user has specified that three assets should be used, the pair of backtests take two weeks, and the user decides to use one year (52 weeks) of historical data per asset, then 78 training examples will be generated. The datagen function is responsible for iterating the asset and backtest start time to achieve the user-specified parameters as described above.

Within each iteration, the datagen function, known henceforth as the data generator, must run the desired strategy twice. The data generator maintains a “current time” which is incremented as it completes backtests and dictates the start date of each backtest. It is the data generator’s responsibility to adjust the current time so that all backtests begin and end when they are supposed to. As explained in Section 8.3.4, the data generator appends extra time beyond the desired time of the strategy to allow trades to sell naturally. As a result, the data generator must decrement the current time after a backtest so that the next backtest starts immediately after the last one logically ended. The data generator thus uses the current time to run the pair of backtests necessary to make a training example.

As explained in Section 8.3.3, we created a Generator strategy whose only purpose is to farm raw financial data and indicator signals. To get relevant historical data, the Generator strategy’s backtests run in parallel with the algorithm’s backtests, i.e. over the same assets and time periods. By combining raw financial data from the Generator strategy with performance data from the algorithm’s strategy, we could generate our input features. The data generator thus runs the Generator strategy once, and then runs the algorithm strategy twice upon each iteration.

Our strategy implementation returns dataframes with backtest results, so the data generator has the responsibility of parsing those frames and pulling the raw data that it needs. Since Zipline handles the calculation of indicators and financial data internally, the data generator does not need to do any postprocessing on the data from the Generator strategy to prepare it for the training example. However, we configured Zipline to only report basic information about the trades a strategy made, with information including start/end dates and the amount made/lost. The data generator must accumulate and analyze that data to make indicator performance metrics for the training strategies. The data generator will produce performance features for the input of the training sample and a target performance for the output, which the neural network will try to predict. The methodology for how this is done is described in the next section. Upon completing this step, the generator has transformed the raw output from the backtests into data suitable for a training example.

The construction of the training example itself is fairly straightforward. First, the data generator records some metadata in the example, including the asset associated with it, the start and end date of the backtest, and its creation number. Next, raw price data over the duration of the backtest is appended. Price data can be optionally pooled by averaging together neighboring price units to limit dimensionality of the input features. The data generator appends indicator data from the Generator strategy and performance data of the algorithm over its first backtest. Together, this information builds the input feature vector of the example. Finally, the data generator appends the performance of the second backtest. This complete training example is appended to a CSV file holding the training set for the network. After all the training examples are constructed, the data generator normalizes each feature on a -1 to 1 scale so that the neural network does not overcompensate for the features at the largest scale.

8.3.6 Calculation of Performance Metrics

As mentioned in the previous section, it is the responsibility of the data generator to produce performance metrics for each training example. In doing so, the data generator must produce two things; first, it must generate a set of performance indicators from the first algorithm backtest that make up input features of the example. Second, it must use some heuristic to produce a performance value from the second backtest, which the neural network will have to predict. The process of how each of these things are done is described below.

The generation of the heuristic-driven output value is the more straightforward process of the two. As mentioned in Section 8.3.5, the algorithmic strategy will release the overall performance of its underlying algorithm over the duration of the backtest directly. This value can easily be transformed into a percent change in portfolio value, which we decided was a suitable basis for the output value. The percent change can be released directly, if the user desires a regression problem, i.e. wants the network to predict a value over a continuous scale. Alternatively, users can specify the output value to be discretized and placed into a category ranking its performance. In this case, the neural network will solve a classification problem, where it tries to "classify" the future performance of an algorithm with a discrete value. To do this, the data generator creates classes that accept certain performances and sort the percentage values into those classes. For example, a 7% increase in portfolio value could be converted to a class which represents performance in a 5% to 10% range. The user sets the number of classes desired, and the data generator will create classes of equal width.

The generation of the input features, on the other hand, take advantage of the statistical performance of individual trades over the backtest. The Strategy class is configured to return the start and end date of individual trades in the backtest. The data generator takes this information and extrapolates how much money was made or lost on each trade. From this set of data, many different performance metrics can be derived. First, the data generator tallies the percentage of trades that made money and the percentage of trades that lost money. The generator then calculates the average amount gained on a winning trade and the average amount lost on a losing trade. Finally, we decided to calculate two complex indicators that try to illustrate variability of the algorithm: average drawdown and the K-Ratio. Average drawdown measures the amount of variance experienced by an algorithm between the time when an asset was bought or sold.

Specifically, it looks for the largest drop in price during a long order that made money and the largest rise in price during a long order that lost money. The K-Ratio measures the deviance of an asset's price chart from its linear regression. These performance indicators are designed to illustrate the amount of "risk" a trading algorithm took to make its trade. All of these indicators are calculated in helpers associated with the data generator, allowing the data generator itself to append the values to the input vector for each training example.

8.3.7 Exporting Training Examples as CSV

This is the final component of the backtest data generator. The components together form the backtest data generator, transforming raw data into an organized set of training examples. As mentioned in the introduction to this section, creating this section of the code took the most effort and consequently occupies the majority of the systems' code space. In order to abstractly turn arbitrary datasets into training sets for supervised learning, the backtest data generator has to encapsulate the subcomponents outlined above to transform the data. The neural network framework accepts these output training sets; its behavior is described in detail in the next section.

8.3.8 LSTM Backtest Data Generation

Before moving on to a description of the neural network framework, it is worthwhile to briefly discuss the modifications we made to the backtest data generator to support LSTM training sets. The main modifications to the data generator lie in the datagen function, and specifically how it chooses to parse and run the algorithm backtests and Generator backtest. As explained in Section 5, the LSTM takes several input feature snapshots in sequential time order and produces a single output value associated to that sequence. In our case, the time snapshot for each day contains the price data of that day and an indicator value for the day. This information carries over directly from the CNN implementation, and only has to be called once for each time sequence, instead of once for each example. However, the performance metrics no longer make sense, since each element in the time sequence holds only one day of activity. Instead, performance can be replaced with a new, more primitive metric: on a given day, if a trade was initiated, how much did that trade win or lose? This allows each time step to represent its component of the algorithm's performance. The output value for each example remains unchanged; it still measures the future performance of an algorithm, which the AI must predict.

Since each input feature snapshot of an LSTM training example represents a single time step, each LSTM training example can represent much more data about a simulation. For example, in our implementation, each LSTM line holds the high, low, open, and close data of the asset. Furthermore, our market performance indicators can be calculated for each time step, instead of just once in the case of the CNN. While this additional information has its advantages in being able to inform the neural network, it also comes with an implementation price. Since so much more data is used in each training example, it takes far longer to an LSTM training set than it takes to make a CNN training set on our local machines. What's more, once the training sets are implemented, it takes more time to ingest and train the LSTM network with those datasets. In both cases, our lack of access to AI-specific hardware or even high-end graphics cards limits the amount that we can do and test. As a result, our testing for the LSTM networks are much less extensive than for the CNNs. If we had more time to spend on the LSTMs, we could have parallelized the data generation process over each time slice of the input to speed the process. Generally, there is a lot that could be done to improve the LSTM process as a whole.

8.4 Implementation of the Neural Network Framework

With the backtest data generator producing full training sets, our next task was to build the framework that would actually train and test the neural networks. This section, known henceforth as the neural network framework, As explained in Section 8.2, we used TensorFlow with the Keras wrappers to handle the actual implementation of the networks themselves; therefore, all we needed to do was prepare the training sets for Tensorflow and analyze the performance of the networks. This section of the code only took one month to implement, but took longer to refine and fine-tune, as there are many tunable parameters in a neural network.

The remaining subsections follow the details of the neural network implementation. In Section 8.4.1, details are given on how the training sets are transformed from a .csv file into Tensorflow-compatible training, test, and validation sets. In Section 8.4.2, a brief summary is given on how Tensorflow works and how we used Tensorflow to build our networks. In Section 8.4.3, we explain the techniques we used to analyze the predictability of the training sets. In Section 8.4.4, we describe our methods for assessing the accuracy of the trained neural networks. Finally, in Section 8.4.5, we explain how the framework was modified to train with LSTM networks.

8.4.1 Parsing the Training Sets for Tensorflow

Since Tensorflow and Keras abstract away the low-level implementation of a neural network, most of our job involved preparing our training sets for Tensorflow to use. The final task of the backtest data generator is to output its generated training samples in a .csv file. It is the responsibility of the neural network framework to reingest those training examples and parse them into numpy arrays that Tensorflow accepts to run its backtests.

At the center of this task is creating a training set, a test set, and a validation set. The training set is used to actually train the network; as explained in Section 5, the network continuously adjusts its weights and bias values to reduce its predictive error on that set. The test set is used to see how the trained neural network performs with fresh data. If the network performs poorly with the test data, it is a sign that the neural network has underfit or overfit the training data. Tensorflow uses its training and test data in "rounds" of training, where the performance results of the test set in each round updates the parameters of the network in all subsequent rounds. Furthermore, we use the k-fold cross-validation technique to train the networks, so *n* completely trained neural networks are produced for *n* folds. The validation set is applied on each model after all training and testing is complete to gain a final understanding of the model's true performance. The accuracy of the networks on the validation set can be averaged to produce the approximate predictability of the model as a whole.

To produce these sets, each training example is split into its input features and output feature, and each is appended to a separate matrix. Once all training examples have been sorted into the input matrix and output matrix, these matrices can be partitioned into the training, test, and validation sets. In order to avoid oversampling a specific subsection of time or a specific asset in any of the sets, stratified sampling is used to divide the sets. Once a validation set has been extracted, the remaining examples are partitioned into *n* subsections. These subsections, or folds, are used for k-fold cross-validation. For each round of the cross-validation, one of the folds is the test set, and the remaining folds compose the training set. Tensorflow itself ingests four arrays: two which represent the input and output array of the training set, and two which represent the input and output array of the test set.

8.4.2 Implementing Tensorflow into the Framework

To actually represent and maintain trainable neural network architectures, we decided to use Keras with a TensorFlow backend. TensorFlow abstracts away almost all of the training and testing of the networks themselves. Users need only model the neural network architecture and provide properly formed training data, and Tensorflow will returned a trained and tested network that can be used to make predictions. Therefore, we did not need to add substantial framework to implement Tensorflow into our neural network framework. However, we did spend some time standardizing the construction and organization of the networks.

We decided to encapsulate each individual network architecture with a Python class. These classes fall under a hierarchy, in which the highest class is the *Network* class. Underneath this class, subclasses are constructed for different architecture classes, like the *CNNNetwork* and the *LSTMNetwork*. The next level of subclass indicates the type of prediction, i.e. between classification and regression. The final class has a concrete implementation of a specific network architecture, specifying the number of columns in the network and the number of neurons in each column. Each of these class implementations maintains a few key functions: first, it offers a Tensorflow wrapper to train the network with a training and test set. Second, it maintains yet another wrapper to generate predictions from a trained network given an arbitrary set of inputs (for example, a validation set). Finally, each implementation has rules on how to evaluate the accuracy of the network on the validation set.

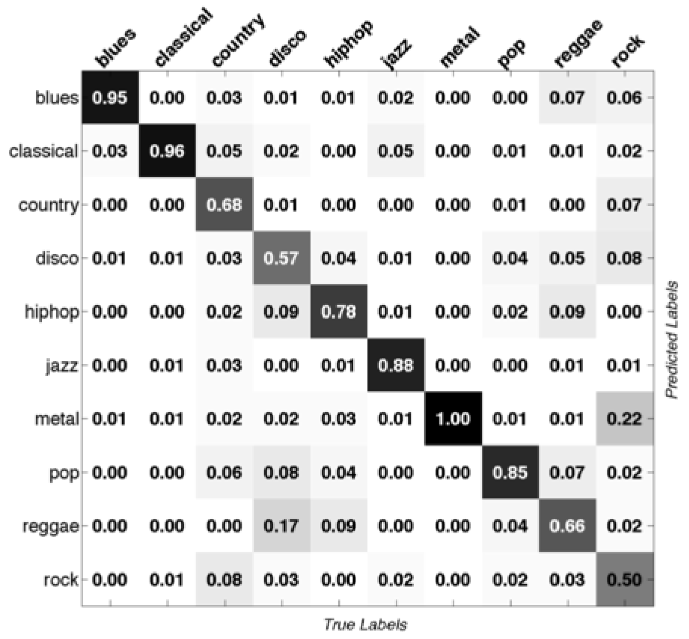
Within each neural network implementation, actual Tensorflow calls are produced to generate the network architecture. Tensorflow maintains a network architecture layer by layer, with separate configurable options for each one. Most obviously, the width of each column can be specified. On top of this, the initial values for the weights and biases of the nodes can be configured to map some distribution. The activation functions of a layer can be set to a preset function or a custom-defined one. To help combat overfitting, regularization can be applied to the weight and bias calculations during each error-reduction iteration. Parameters for the training of the network, including an optimization function and an error calculation metric, are also specified with the implementation. All of these Tensorflow function calls are encapsulated within the *build_model* method which generates and stores an untrained network.

8.4.3 Evaluation of Network Accuracy

We implemented several heuristics to evaluate the predictive capabilities of a trained neural network. To start, we measure the raw accuracy of the network on our previously-generated validation sets. This is a good one-number overview of the performance of the network, but it in no way illustrates the full performance of the network. Specifically, it is useful to understand the circumstances in which the network does well and poorly. To start, we calculate the F-score of the network's performance. The F-score measures the ratio between the precision and recall of the network. Precision here measures the number of correct classifications out of all positive classifications, and recall measures the number of correct classifications out of all classifications that should have been registered positive. As a general heuristic, the network achieves better performance when the F-score approaches 1, and a worse performance when it approaches 0.

On top of this, we also use the Area Under Receiver Operator Characteristic (AUROC) technique to measure the classes in which a neural network performs best. ROC curves measure predictive strength against truly random prediction. Represented in the false-positive/true-positive plane, truly random prediction creates a '$y = x$' line bisecting that plane. We generate a ROC for each class that the neural network predicts. By measuring the area between the ROC curve of the network and that random prediction line, we can produce a metric for the predictive capability of the network. The larger the difference in area is for a class' ROC curve, the better the network is performing for that class.

Finally, we employ confusion tables to illustrate how well the network predicts each class. A confusion table maps the predicted class of a given example against its actual class. To do this, rows of the table represent what class the neural network predicted for the example, and the columns of the table represent that example's actual class. If the network performs optimally, the diagonal elements of the table should be the only populated ones; i.e. the network correctly predicted the true class of each example. If elements populate cells in the table that deviate from the diagonal, they signal the network's error. The confusion tables are the most informative element of our accuracy analysis because they show the network's performance for each class.


True Labels
Predicted Labels

	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	0.95	0.00	0.03	0.01	0.01	0.02	0.00	0.00	0.07	0.06
classical	0.03	0.96	0.05	0.02	0.00	0.05	0.00	0.01	0.01	0.02
country	0.00	0.00	0.68	0.01	0.00	0.00	0.00	0.01	0.00	0.07
disco	0.01	0.01	0.03	0.57	0.04	0.01	0.00	0.04	0.05	0.08
hiphop	0.00	0.00	0.02	0.09	0.78	0.01	0.00	0.02	0.09	0.00
jazz	0.00	0.01	0.03	0.00	0.01	0.88	0.00	0.00	0.01	0.01
metal	0.01	0.01	0.02	0.02	0.03	0.01	1.00	0.01	0.01	0.22
pop	0.00	0.00	0.06	0.08	0.04	0.00	0.00	0.85	0.07	0.02
reggae	0.00	0.00	0.00	0.17	0.09	0.00	0.00	0.04	0.66	0.02
rock	0.00	0.01	0.08	0.03	0.00	0.02	0.00	0.02	0.03	0.50

Fig. 35 - An example of a confusion table. Notice here that most elements populate the diagonal.

8.4.4 Measuring Predictive Nature of the Network

There are methods to assess the success of a model beyond only analyzing the performance of a network after it has been trained. The training set itself can be analyzed using various techniques to test its "predictiveness". Here, predictiveness references to the separations between different classes. If each output class in the set falls into an isolated input feature range, then it should be easy for the network to build a decision boundary. To determine the separation of each class, several dimension reduction techniques are applied to the input features of each example, allowing the example to be plotted in 2-D space. We generally used these techniques as a brief check on the predictability of the set, without quantifying any values or relationships for later analysis.

We decided to use four dimension reducing techniques and examine their results in parallel: MDS, ISOMAP, TSNE, and PCA. Each of these techniques has its own method of reducing down the n-dimension input feature vector down to two features. For each technique, a plot is given of all the transformed training examples. Each training example is given a marker on the plot based on the output class it belongs to. That way, it is easy to examine the potential divisions between different classes. Below is an example of PCA applied to a dataset:

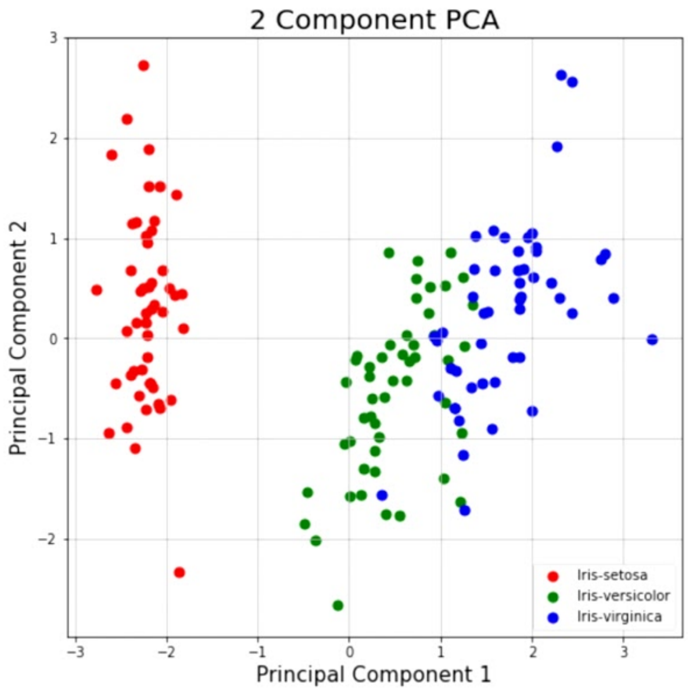


Fig. 36 - An example of PCA applied to a dataset. Here, each color represents a separate class.

8.5 Implementation of the Trading Platform

The trading platform (otherwise known as the trader) was created in order to allow trained neural networks to be used to trade assets in simulated real time. Given a given amount of starting capital, a set of strategies, and a corresponding set of trained neural networks which predict the performance of the strategies over a specific time period, the trading platform will allocate capital to each strategy weighted by its predicted performance. After one trading period has passed, the trading platform halts strategy execution, extracts the capital stored by each strategy, selling assets as needed, and repeats the process.

In order for trained neural networks to be used by the trader, they needed to be able to be saved and loaded from the file system. In order to do this, after each neural network was trained, the weights of its nodes were exported as an h5 file and metadata about the network was saved using a pickled[1] *Manifest* object. Currently, a *Manifest* object contains metadata that includes the strategy the network corresponds to, what indicators were included in the training set (and will serve as inputs to the neural network), which network class was used, and the path to the training set itself. Using the metadata and weights stored in these files, the trading platform is able to reconstruct the trained neural network so that it can be used to predict strategy performance.

[1] In Python, objects can be saved to and loaded from pickle files. When an object is saved to a pickle file, we say that the object has been pickled.

Once the trained neural networks have been reconstructed, an input vector for each needs to be generated. Each neural network expects an input vector that consists of the daily prices of the asset being traded, the corresponding indicator values, and the performance evaluation of the strategy over the course of the historical period.

To generate these values, a child class of the *GeneratorStrategy* class originally created for the backtest data generation process, *AdaptableGeneratorStrategy*, is used. The key difference between the *GeneratorStrategy* and the *AdaptableGeneratorStrategy* is that the latter stores the price data in *FixedSideSelfPruningQueue*, which is a queue of fixed size that prunes the oldest value when a value is being enqueued and the rest of the queue is full. This allows the *AdaptableGeneratorStrategy* to be more efficiently used in cases where the backtest data generation period is different from the trading period, the interval of time in which strategies are run with a given set of weights.

Using the input vectors populated using the data from *AdaptableGeneratorStrategy*, the performance rating for each strategy being balanced by the trader is estimated by the neural networks. Since each neural network can potentially have a different number of categories (and therefore performance estimations) than the rest, each performance estimation is normalized to a between 0.0 and 1.0, which represent the worst and best performance estimations, respectively.

The normalized performance estimations are used to allocate capital to each strategy using the following equation:

$$p_i = 1 + w_i \frac{P - n}{n}, \qquad (8)$$

where p_i is the capital assigned to strategy i, w_i is the normalized performance estimation of strategy i, P is the total capital possessed by the trader, and n is the total number of strategies. There are a few notable things about this equation. For one, you may notice that the trader always allocates at least \$1 to every strategy. This is because Zipline refuses to allow a strategy with a principal of \$0 to be run. Another notable thing about this equation is that if every strategy has a performance estimation of 1 (a very rare event), the trader goes all in and allocates P / n dollars to each strategy. In the future, options for different position sizing approaches will be added to the trader.

Once capital is allocated to each strategy, they are allowed to run for one trading period. Once the trading period is finished, each strategy is forced to sell any positions it may still have and relinquish its capital to the trader. After this is done, the process begins again, with the input vectors for the next set of estimations being generated.

The trader itself, whose development started at the end of D-term last year, is still in a very early state, with a lot of room for improvement. One thing that we plan to add in the future is a better way of handling the reclaiming of capital from the strategies at the end of the trading period. Right now, the strategies are forcibly stopped at the end of the trading period and are restarted without any context from the previous trading period. Ideally, we could find a way of “pausing” each strategy between trading periods so that longer-running algorithms are not impacted in the way they are currently.

8.6 System of Systems Performance

To test our system of systems, we wanted to gather a fundamental understanding of the system’s ability to predict the behavior of its trading algorithms. We needed to understand how each component of the system contributes to its accuracy (or lack thereof). Given the scope of this project as a proof of concept, we decided to restrict our testing to several daily-bar algorithms. As explained in Section 8.3, our selection of market indicators, performance indicators, and assets were based off of a simple heuristic, and we used a fixed four-week period of trade data for each training example. Doing so gave us enough information to assess the success of the system of systems and provide insight as a proof of concept. We acknowledge that much more work can be done to test our system which extends past the scope of this IQP.

The remainder of this section will describe in detail the structure of the training sets we used and the performance of the system of systems. Section 8.6.1 summarizes the structure of the training sets and the parameters of each training set configuration with which we tested the neural network. Section 8.6.2 describes the configuration of the neural network that gets trained in the system. Section 8.6.3 describes the performance of the system of systems under each of those configurations, and Section 8.6.4 provides an analysis of that performance and a set of conclusions.

8.6.1 Training Set Structure and Parameters

As explained in the introduction to this section, we only had the time to reasonably provide a proof of concept for the system of systems. Consequently, many of the parameters we used to build our training sets were set to arbitrary values, or values based on simple heuristics. Below, we describe how we set each parameter for the training set, and where possible, we provide an explanation for our design choices.

Each training example uses two adjacent four week periods, the first of which gathers the context of the trading algorithm performance, and the second of which reflects the subsequent performance of the algorithm. As explained in Section 8.4, the first backtest generates the input vector of each training example, and the second backtest generates the output value of the training example. Each asset was given around ten years of data to generate training examples, ranging from 2007 to 2017. 70 training examples were produced from each asset.

The asset selection itself was driven by the desire to test assets with high variance in performance. While we did not employ modern portfolio techniques to build a portfolio of assets with high covariance, we did ensure that stocks which have both gained value and lost value were represented.

For example, we use both Apple and General Electric in our portfolio of assets; while Apple has seen a meteoric rise in its share value, General Electric's share price has been consistently devalued over the past few years. We also made sure that our assets reflected the market as a whole, and not just some specific sector. We believe that training the system on a portfolio of assets with varying performance is necessary for the system to properly understand how an algorithm does under all contexts. Below, we list the assets we used to build our training sets:

- Apple
- General Mills
- United States Steel Corporation
- Microsoft
- Walmart
- Intel
- Advanced Micro Devices
- Amazon
- EBay
- General Electric
- NVidia
- Mattel
- Range Resources Corp.
- Under Armor
- Apache Corp.
- Goldman Sachs
- J.P. Morgan
- Hess Corp.
- Applied Materials Inc.

Since we decided to have a four-week backtest period to build the input vector of each training example, the input vector has 40 price features, one per day. Each feature holds the closing price for the given day of trading. The next set of features in the input vector are indicators calculated at the trading day between the two backtests. These are the indicators calculated by the Generator strategy as explained in Section 8.4. Our main heuristic for selecting indicators was to gather both relatively simple and complex indicators. We were curious about which kinds of indicators would provide better predictability to the network, as explained in Section 8.4. Some indicators, like the EMA and Percent D, find averages of the data, while other indicators, like the Commodity Channel Index, require multiple sub-indicators and are extremely complex to implement. We also wanted each indicator to provide unique information to the input vector. Our list of indicators, with a brief description of each, is provided below:

- Exponential Moving Average (EMA)
    - A moving average of the last 14 days which biases recent price changes more that price changes further in the past
- Long EMA
    - An EMA with a 42-day range, instead of 14 days
- RSI
    - A momentum oscillator which signals the speed and change of price movements. The RSI indicates when an asset is “overbought” or “oversold”.
- Percent K
    - An indicator which shows the ratio between the current price deviation from average and the highest price deviation from average over the last 14 days.
- Percent D
    - An average of PercentK indices calculated over the last 3 days.
- Slow Percent D
    - An average of PercentD indices calculated over the last 3 days.
- Momentum
    - An indicator which reports the difference in closing price over a 4-day period.
- Rate of Change (ROC)
    - An indicator which reports the percent change in closing price over a 4-day period.
- Percent R
    - An indicator which finds the percent difference between the current closing price and the difference between the highest high and lowest low over the last 14 days. The Percent R indicator shows volatility.
- Accumulation Distribution
    - Relates the price and volume of trades for an asset. Indicates momentum of an asset by gauging whether the asset is accumulating or distributing among buyers.
- Chalkin Oscillator
    - Takes the difference between the 3-day EMA of accumulation-distribution and the 10-day EMA of accumulation distribution.
- Commodities Channel Index

- Finds the difference between the “typical price” of a commodity and its current price. Indicates the stage of a cycle an asset is in.

The final set of indicators in our training example are the performance metrics as described in Section 8.3.6.

We decided to experiment with two types of outputs for the training examples: an 8-class categorization and a 2-class categorization. The 8-class category set classifies output performance at a 5% gradient. Any performance producing greater than 20% returns or losses are grouped into the highest and lowest classes, respectively. The 2-class categorization asserts if there has been positive or neutral performance, and de-asserts if there has been negative performance.

We use two strategies to test the system of systems. Each strategy has its own generated dataset and neural network performance. The first strategy, an EMA strategy, buys if the current price is higher that an EMA of the last 10 trading day price highs, and sells when the share price has moved more than three dollars in either direction. The second strategy, a TurtleStratrgy, is a much more useful and applicable strategy which buys and sells based on the basic turtle strategy rules. The simpler and less applicable EMA strategy is useful to test and debug our system to make sure trades are properly executing, while the TurtleStrategy helps show how the system of systems balances a robust algorithm which could realistically be used on its own. We believe that a combination of the results from these two strategies helps indicate the predictive nature of the system of systems, and is enough to validate our proof of concept.

8.6.2 Neural Network Structure and Parameters

For the purposes of our proof of concept, we chose to test with relatively fixed network parameters. Since we were working mainly with classification problems, our input layer size is always the length of the input feature vector, and the output layer size is the number of classes. The final layer uses a softmax activation function to ensure that the final layer returns a single class. We always use SGD as the optimizing function, and we use categorical cross-entropy for our loss function. Both of these are fixed because they are applicable to classification problems.

We did experiment with modifying several parameters to minimize overfitting and underfitting. The first and most prominent feature is the number of layers in the network. We also looked at regularization of the weight and bias correction in the network to minimize overfitting the data. Finally, we looked for the activation functions for each layer that produced the best output. However, we almost immediately settled on using layers with a tanh activation function following a single layer with a ReLu activation function, and experimentation with those parameters were limited.

We also chose to leave some parameters in their default state, namely the initialization functions for the weights and biases of each layer. The default random initialization and our k-fold cross validation helps ensure that we can holistically understand the performance of the system without a bias for a particular test run.

8.6.3 System of Systems Performance

We decided to test our system of systems under several of the configurations described in the prior two sections. Each test optimized a neural network for one strategy with one type of classification (2-class and 8-class). These networks are then tested in "real-time" to determine the amount of money they made/lost.

8.6.3.1: EMA Strategy with 2-Class Classification

We first balanced the neural network for our EMA strategy, the simplest one out of the free. We also tested with two-class classification. The first class, class 0, represents trading periods that lost money. The second class, Class 1, represents trading periods that made money. In general, we test whether the system of systems can make the fundamental determination on what trading periods are profitable and unprofitable before testing anything else.

As explained in Section 8.6.3, we knew from the start that our network would be composed of alternating layers with ReLu and tanh activation functions. We also decided as a general principle that no single neural network should have more than ten layers, to ensure that training could be done efficiently. The first parameter we tuned was weight and bias regularization. It was almost immediately apparent that substantial regularization was forcing the network to underfit the dataset, so we decided to leave only minimal weight regularization in the network. The next parameter tuned was the length of each neuron layer. We tested several lengths, ranging from the length of the input vector to four times that length. It was found that the network did not experience significant differences in performance from its layer lengths, so we settled on setting the length of twice the size of the input feature vector. Finally, we tuned the number of layers in the network. After trying a four-layer, seven-layer, and nine-layer configuration, we determined that on average the nine-layer network performed the best while avoiding major overfitting of the dataset.

The final, optimized network configuration has eight alternating ReLu and tanh layers, preceding a final softmax layer. Weight regularization of 0.001 is applied to all layers, and each layer has a length of twice the input feature vector. The weights are initialized in a uniform distribution. Since we use k-fold cross validation with 10 folds to test our network, we have ten samples which represent the accuracy of the network. We have chosen one of these samples to illustrate specific accuracy information on the network. Below is a sample AUROC curve of one of the folds:

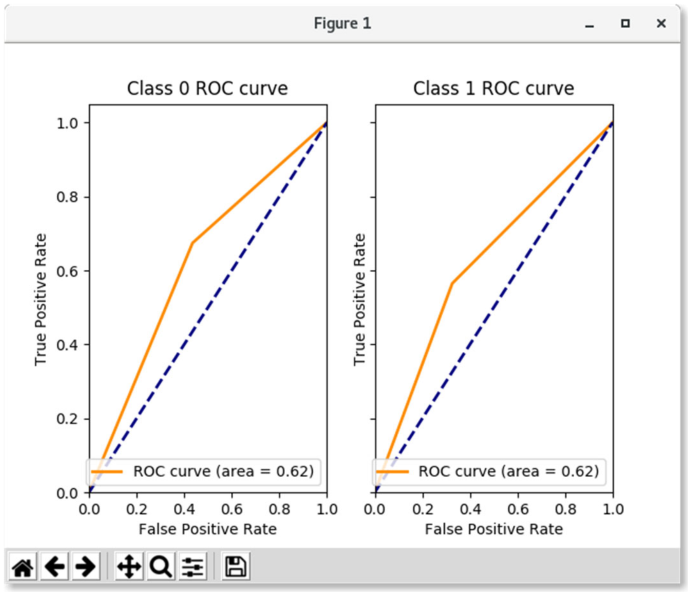


Fig. 37 - AUROC curves for both classes in the optimal configuration.

In this case, Class 0 indicates a loss over the trading period, and Class 1 indicates a gain. Both Class 0 and Class 1 have an AUROC value greater than 0.5, we can confidently say that the neural network has at least some measure of predictability with the dataset. The following confusion illustrates the predictability of the network on each class:

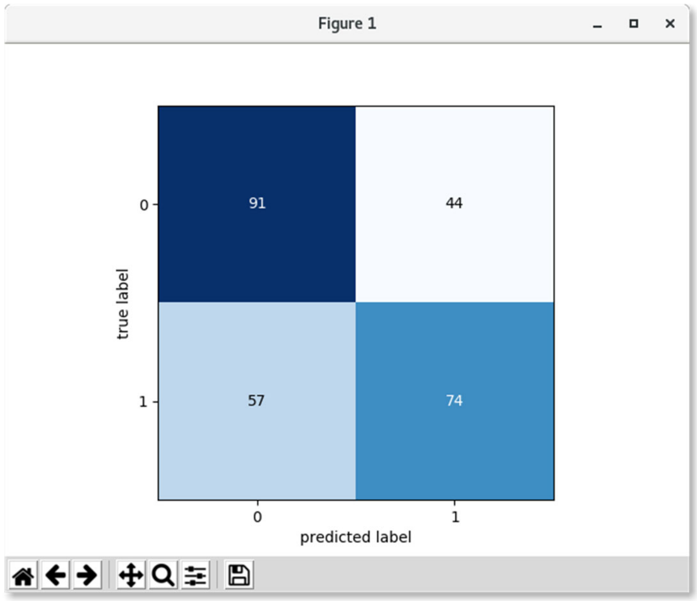


Fig. 38 - Each row illustrates the difference between correctly and mispredicted training examples for both classes.

The network is most successful in determining what trading periods will be unsuccessful, with a precision of 67%. It is less successful in determining what trading periods will be successful, with a precision of merely 56%. In general, we are biased towards precision for lower classes against precision for higher classes. This is because we are more interested in stopping the trader from losing money than we are with the trader seizing every opportunity to gain money, since the former yields a loss while the latter yields only a lack of a gain. For this specific run of the k-fold cross validation, overall accuracy is 62%, and the unweighted F-score is 0.59. Of course, these two values only apply to one element of the k-fold, and cannot necessarily be used to judge the accuracy of the entire system.

To better gauge the accuracy of the network as a whole, we averaged out the performance of each fold. As a whole, the network has 56% accuracy. This indicates a moderate degree of predictability. All folds had accuracies above 50%, which validates the predictability of the system. The AUROC curves and confusion tables for each testing fold are listed below in the appendices.

After being trained, the network was saved using the *Manifest* class and tested using the trader. Using the network and the *SimpleEMAStrategy*, the trader was run with the following parameters:

- Trading began on or after January 1, 2017
- Trading ended by January 1, 2018
- Historical and trading periods of 40 days
- AAPL was the asset to be traded
- The principal balance was $1000

At the end of the trading simulation, the final balance of the system was $983.59. This performance leaves a lot to be desired, especially considering that AAPL increased in price by about 46 percent between January 1, 2017 and January 1, 2018. We suspected that increasing the classes of our classifier would yield better results.

8.6.3.2: EMA Strategy with 8-Class Classification

In addition to balancing a two-class neural network for our EMA strategy, we also balanced an eight-class neural network. Classes 0 through 3 represent trading periods which lost money, where the class number is inversely proportional to the amount of money lost. Classes 4 through 7 represent trading periods which earned money, which the class number is directly proportional to the amount of money gained. Like the work we did for the two-class neural network, our primary goal was to test whether the network could predict trading period profitability, and beyond that, whether it could predict how profitable (or unprofitable) a trading period would be.

Our starting point for the configuration of the eight-class neural network was that of the two-class neural network we tuned in section 8.6.3.1. Through experimentation, we found that a neuron layer length of 8 times the input length worked better than the original input layer length of 2 times the input length, and also performed better than input layer lengths of 4 and 16 times the input length. After we made this change, we noticed that the network was overfitting in some cases (the accuracy on the training set approached and hit 100%, whereas the accuracy on the validation set was stuck around 20%). This was mitigated by adding a Dropout layer after then input layer, which reduced the accuracy on the training set but increased the accuracy on the validation set. Finally, we tried varying the number of pre-output layers between 4, 8, and 10. Through our testing, we found that the best compromise between performance and training time was 8 layers.

After tuning, the configuration of our eight-class neural network consisted of an input layer with a tanh activation function, a Dropout layer, seven layers of with alternating ReLu and tanh activation functions, and a softmax output layer. The neuron layer length of the layers (other than the Dropout and output layers) was 8 times the input layer length. Like the two-class neural network, each neuron layer has its weights initialized using a uniform distribution and a weight regularization of 0.001. Since the network was tested over ten folds, there are ten samples available to determine the accuracy of the network. One of these samples was selected to exemplify the performance of the network, and the rest can be found in the appendix. Below is a sample AUROC curve of one of the folds:

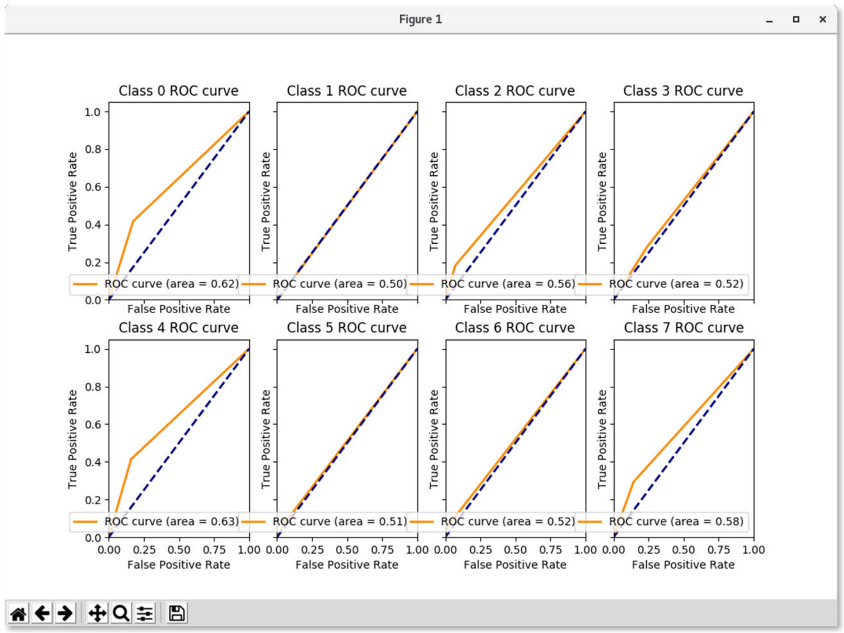
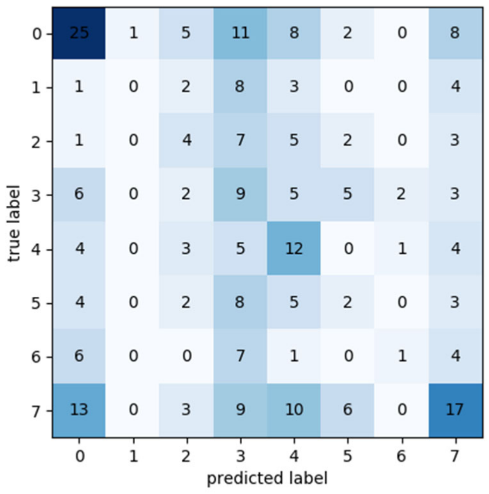


Fig. 39 - AUROC curves for both classes in the optimal configuration.

In this case, Classes 0, 1, 2, and 3 indicate a loss over the trading period, and Classes 4, 5, 6, and 7 indicate a gain. Every class other than class 1 has an AUROC value greater than 0.5, so we can confidently say that the neural network has at least some measure of predictability with the dataset. The following confusion illustrates the predictability of the network on each class:



true label \ predicted label	0	1	2	3	4	5	6	7
0	25	1	5	11	8	2	0	8
1	1	0	2	8	3	0	0	4
2	1	0	4	7	5	2	0	3
3	6	0	2	9	5	5	2	3
4	4	0	3	5	12	0	1	4
5	4	0	2	8	5	2	0	3
6	6	0	0	7	1	0	1	4
7	13	0	3	9	10	6	0	17

Fig. 40 - Confusion Table

Each row illustrates the difference between correctly and mispredicted training examples for both classes. The network is most successful in determining what trading periods will be wildly unsuccessful (class 0), with a precision of 42%. It is a bit less successful in determining what trading periods will be wildly successful (class 7), with a precision of 37%. For this specific run of the k-fold cross validation, overall accuracy is 27%, and the unweighted F-score is 0.20. Of course, these two values only apply to one element of the k-fold, and cannot necessarily be used to judge the accuracy of the entire system.

Looking at whether the network can determine whether a trading period will be profitable or not, these results look a bit better. The network is around 56.2% accurate at predicting that a trading period will be unprofitable and is 56.4% accurate at predicting that a trading period will be period will be profitable. This suggests that the system has some predictive ability, but is not great at determining exactly how profitable a trading period will be.

To better gauge the accuracy of the network as a whole, we averaged out the performance of each fold. As a whole, the network has 26% accuracy. This indicates that it cannot accurately predict how profitable or unprofitable a given trading period will be. Most folds had accuracies above 50%, which validates the predictability of the system. The AUROC curves and confusion tables for each testing fold are listed below in the appendices.

After being trained, the network was saved using the *Manifest* class and tested using the trader. Using the network and the *SimpleEMAStrategy*, the trader was run with the same parameters as the two-class neural network:

- Trading began on or after January 1, 2017
- Trading ended by January 1, 2018
- Historical and trading periods of 40 days
- AAPL was the asset to be traded
- The starting principle was $1000

At the end of the trading simulation, the final principle of the system was $1026.20. While better than the performance of the two-class network (this network actually made money!), its performance is still worse than just running the *SimpleEMAStrategy* by itself, which ended with $1127.04 when run with the same parameters. Part of the reason why the strategy itself may have done better without the trader is that each time a strategy is run by the trader, it loses its context from previous runs. This is something we plan to fix in the future.

8.6.4 Analysis of Performance and Conclusions

The testing we have performed on the system of systems yields mixed results; while we can generate neural networks which are predictive with our datasets, we have yet to translate those networks into a successful trader. We therefore have a valid proof of concept for the first two components of the system of systems, but not the third. We believe that while we correctly calibrated the system's artificial intelligence to adapt to our generated backtest data, our generated backtest data did not accurately capture the particularities of the algorithms they represented. Specifically, we neglected to tune features like the length of the backtest to build the input features and output value of each training example. This leads to a mismatch between the backtest data and the underlying algorithm it represents. In our opinion, this is a fundamental reason why the trader does not make money, despite the fact that the underlying neural networks are predictive.

This leads to the most fundamental lesson from our proof of concept: that the system needs specific tuning for each algorithm that it wishes to balance. Any broad-brush techniques, especially when generating the backtest data, will invariably lead to the backtest data misrepresenting when the algorithm buys and sells. Most importantly, the time periods for both backtests must be tuned correctly to capture the amount of data the algorithm uses to make a decision. The use of daily closing price bars as input features may also not represent what the algorithm looks at when it makes a decision. Our backtest data generator should have subclass implementations per algorithm, similar to our neural network framework.

There are many things we would change about how we implemented our backtest generator. Discrete Zipline backtests with small trading periods for each training example are clunky, especially if the backtest period is shorter than the average life of a trade in the algorithm. This is despite our best efforts to let trades start and end naturally with selloff periods past the end of the backtest. Future work could involve running only one backtest over a range of many years and dividing that backtest into segments. This would allow each backtest to capture how the algorithm naturally trades. On top of this, the trading signals should be reworked to represent the entire period of trading, rather than a snapshot for one given point in time. This would lend credence to the LTSM approach, where an entire feature vector can be plotted for each time unit. Furthermore, it is clear that we will not get performance beyond mild predictiveness while we still use shallow networks which can be computed on CPUs. Future efforts should be made to train deep networks, and preferably data-rich LSTMs, on GPUs.

Taking a step back, the system of systems fundamentally answers two questions. First, it predicts the performance of an algorithm given a discrete pattern. Second, it predicts what a stocks future patterns will hold. We suspect that a single neural network is not well suited to answer both questions at the same time. We perhaps should have prepared and trained two networks for each algorithm, one per question, and design a trader to make informed decisions with both. We are certain that our training sets were not large enough and our networks not deep enough to have the predictive capacity to drive the trader.

We were very satisfied with our choice of Python as a language and Tensorflow as our neural network library. We found excellent documentation and support from the Python community as we implemented the system. What's more, we were able to borrow from existing finance APIs in Python to do the heavy lifting of our indicator calculations. Zipline in general was serviceable, but in some cases was not satisfactory for what we desired. The top-level API exposed via documentation is incomplete, requiring us to dig into the library itself and expose data and attributes we needed from the trader. Furthermore, Zipline's metadata generation relies on deprecated Google Finance APIs which intermittently fail, thus crashing the entire library. We have some interest in merging the improvements we required back into the upstream branch.

9. Conclusions and Recommendations

We were able to successfully implement trading strategies and a system of systems with predictive capability. Although our individual trading strategies were not successful, they provided us with great practice in the proper design of trading strategies with a scientific approach. Most of the strategies that we wrote show potential for real world trading if they are further developed to overcome their individual problems. We strongly believe that these strategies and the principles they outline form a good starting point for anyone interested in the development of trading strategies. Our system of systems showed mild predictability with a basic EMA trading strategy. We believe that this forms a proof of concept which validates our belief that neural networks can be used to balance a set of trading algorithms. We learned a lot about how artificial intelligence techniques inform financial technology; as our discussion in Section 8 described, there are several architectural changes that could be made to the system to boost its ability to predict algorithm performance. The framework which we have designed could be expanded in future work to implement those suggested changes.

Our results indicate that the world of finance will irrevocably change as artificial intelligence is further applied to predict the markets. As we discussed in our introduction, investment, and especially trading, require the prediction of patterns which reliably produce profit. It is abundantly clear that neural networks excel at pattern prediction, in some cases more-so than humans. Already, data science and machine learning have had a transformative effect on the financial system; as that influence grows, we may see markets becoming less volatile, as more traders in the system take advantage of and therefore nullify opportunities in the market. On the other hand, when these prediction schemes all make the same mistake, they can amplify the resulting effect on the market. Trading at its core revolves around the question of whether or not one could systematically beat the market. While we have shown that it is indeed possible, the mass adoption of artificial intelligence in the financial industry might very well change that answer.

We find the success of our system of systems particularly exciting because it shows that individual actors can play the markets in a more involved way than simply buying and selling shares. One could foreseeably construct their own trading strategies, load them into a system of systems, and eventually run a trading system more akin to that of a professional freelance trader than an amateur investor. Like we emphasized at the beginning of this paper, it is a social good for individuals to independently build and maintain their wealth. Often, a hegemon monopolizes wealth to become a de facto rule-setter in society. Our work offers a means for people to counteract those influences by maintaining their own wealth independent from any large institution.

More concretely, however, we recognize our work is merely a proof of concept, and realistically could not be fully deployed as a reliable trading system. That speaks to the time and work commitment which must be made to trade successfully on the markets. We hope at the very least that our work can be continued by IQPs in the future to turn our prototype into a fully-fledged system. Our contributions to the Trading and Investment IQP can hopefully influence projects for years to come. This project was an invaluable learning experience for us; we leave it with seasoned trading experience and a better understanding of the mechanics one must employ to successfully invest.

References

[1] *(March 3,). The State of American Retirement*
How 401(k)s have failed most American workers | Economic Policy Institute. Available: https://www.epi.org/publication/retirement-in-america/.
[2] (). *Type of Financial Markets and Their Roles*. Available: https://www.investopedia.com/walkthrough/corporate-finance/1/financial-markets.aspx.
[3] (Jan 2,). *Comparing Institutional and Retail Traders*. Available: https://www.investopedia.com/articles/active-trading/030515/what-difference-between-institutional-traders-and-retail-traders.asp.
[4] R. Wray, *Modern Money Theory: A Primer on Macroeconomics for Sovereign Monetary Systems.* (2nd ed.) Palgrave Macmillan, 2015.
[5] (). *Introduction to Bonds*. Available: https://www.investopedia.com/walkthrough/corporate-finance/3/bonds/introduction.aspx.
[6] (). *Bond Ratings*. Available: https://www.investopedia.com/walkthrough/corporate-finance/3/bonds/ratings.aspx.
[7] (). *Equity*. Available: https://www.investopedia.com/terms/e/equity.asp.
[8] (Feb 27,). *Commodities Trading: An Overview*. Available: https://www.investopedia.com/investing/commodities-trading-overview/.
[9] (). *Derivative*. Available: https://www.investopedia.com/terms/d/derivative.asp.
[10] (). *Credit Derivative*. Available: https://www.investopedia.com/terms/c/creditderivative.asp.
[11] (). *Credit Default Swap - CDS*. Available: https://www.investopedia.com/terms/c/creditdefaultswap.asp.
[12] Murphy, J. J. (2009). The visual investor: How to spot market trends. Hoboken, NJ: John Wiley & Sons.
[13] (). *Intermarket Analysis*. Available: https://stockcharts.com/school/doku.php?id=chart_school:overview:intermarket_analysis.
[14] (). *Yield Curve*. Available: https://www.investopedia.com/terms/y/yieldcurve.asp.
[15] (). *Market Breadth*. Available: https://www.investopedia.com/terms/m/market_breadth.asp.
[16] (May 3,). *What All Investors Should Know About ETFs*. Available: https://www.investopedia.com/articles/investing/050316/what-all-investors-should-know-about-etfs.asp.
[17] D. Kansas, *Complete Money &amp; Investing Guidebook.* New York City: Three Rivers Press, 2005.
[18] (Aug 4,). *Support and Resistance Basics*. Available: https://www.investopedia.com/trading/support-and-resistance-basics/.
[19] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," *IBM Journal of Research and Development,* vol. 3, *(3),* pp. 210-229, 1959. . DOI: 10.1147/rd.33.0210.
[20] (). *The MNIST Database of Handwritten Digits*. Available: http://yann.lecun.com/exdb/mnist/.
[21] John Markoff, "Computer Wins On 'Jeopardy!': Trivial, It's Not," *New York Times,* 2011. Available: https://search.proquest.com/docview/851916264.
[22] (Dec 21,). *Deep Learning Achievements Over the Past Year*. Available: https://blog.statsbot.co/deep-learning-achievements-4c563e034257.
[23] G. James *et al*, *An Introduction to Statistical Learning.* New York: Springer, 2013*103*.
[24] G. James *et al*, "Regression vs. classification problems," in *An Introduction to Statistical Learning*Anonymous New York: Springer, 2013.
[25] G. James *et al*, "The classification setting," in *An Introduction to Statistical Learning*Anonymous New York: Springer, 2013.
[26] (Mar 25,). *Simple Guide to Confusion Matrix Terminology*. Available: https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/.
[27] G. James *et al*, "Resampling methods," in *An Introduction to Statistical Learning*Anonymous New York: Springer, 2013, pp. 175-201.
[28] G. James *et al*, "Tree-based methods," in *An Introduction to Statistical Learning*Anonymous New York: Springer, 2013.
[29] G. James *et al*, "Principal component analysis," in *An Introduction to Statistical Learning*Anonymous New York: Springer, 2013.
[30] B. K. Wong and Y. Selvi, "Neural network applications in finance: A review and analysis of literature (1990–1996)," Information *& Management,* vol. 34, *(3),* pp. 129-139, 1998. Available: https://www.sciencedirect.com/science/article/pii/S0378720698000500. DOI: 10.1016/S0378-7206(98)00050-0.
[31] S. Shanmuganathan and S. Samarasinghe, *Artificial Neural Network Modelling.* Cham: Springer, 2016*628*.
[32] Aalst, van der, WMP Wil *et al*, "Process mining: A two-step approach to balance between underfitting and

overfitting," BPMcenter.org, 2008.
[33] (). *Improve Shallow Neural Network Generalization and Avoid Overfitting*. Available: https://www.mathworks.com/help/deeplearning/ug/improve-neural-network-generalization-and-avoid-overfitting.html;jsessionid=d7be1ad036363848cb8116524b70.
[34] S. Lawrence and C. L. Giles, "Overfitting and neural networks: Conjugate gradient and backpropagation," in 2000, pp. 119 vol.1.
[35] Steven J. Nowlan and Geoffrey E. Hinton, "Simplifying Neural Networks by Soft Weight-Sharing," *Neural Computation,* vol. 4, *(4),* pp. 473-493, Available: http://www.mitpressjournals.org/doi/abs/10.1162/neco.1992.4.4.473. DOI: 10.1162/neco.1992.4.4.473.
[36] N. Srivastava *et al*, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* 2014. Available: http://www.jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf?utm_content=buffer79b43&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer.
[37] A. K. Jain and B. Chandrasekaran, "39 dimensionality and sample size considerations in pattern recognition practice," in *Handbook of Statistics*Anonymous Elsevier Science & Technology, 1982, pp. 835-855.
[38] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science,* vol. 313, *(5786),* pp. 504-507, 2006. Available: http://www.sciencemag.org/cgi/content/abstract/313/5786/504. DOI: 10.1126/science.1127647.
[39] (). *SKLearn API Reference*. Available: http://scikit-learn.org/stable/modules/classes.html#module-sklearn.manifold.
[40] A. K. Jain and B. Chandrasekaran, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Handbook of Statistics 2*Anonymous Elsevier B.V, 1982, pp. 835-855.
[41] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM,* vol. 60, pp. 84-90, May 24, 2017.
[42] (Jun 15,). *"Convolutional Neural Networks (LeNet)*. Available: http://deeplearning.net/tutorial/lenet.html.
[43] Z. Tang and P. A. Fishwick, "Feed-forward neural nets as models for time series forecasting"
[44] (). *TIME SERIES PREDICTION WITH FEED-FORWARD NEURAL NETWORKS*. Available: http://neuroph.sourceforge.net/TimeSeriesPredictionTutorial.html.
[45] T. Kimoto *et al*, "Stock market prediction system with modular neural networks," in 1990, pp. 6 vol.1.
[46] J. Han *et al*, "Road boundary detection and tracking for structured and unstructured roads using a 2D lidar sensor," *Int. J Automot. Technol,* vol. 15, *(4),* pp. 611-623, 2014. Available: https://search.proquest.com/docview/1530310920. DOI: 10.1007/s12239-014-0064-0.
[47] T. Loughran and B. McDonald, "When is a Liability not a Liability? Textual Analysis, Dictionaries, and 10-Ks." *Journal of Finance, Forthcoming,* 2009.
[48] P. C. Tetlock, "Giving Content to Investor Sentiment: The Role of Media in the Stock Market," *Journal of Finance,* vol. 62, *(3),* pp. 1139-1168, 2007.
[49] H. Chen *et al*, "Wisdom of Crowds: The Value of Stock Opinions Transmitted Through Social Media," *The Review of Financial Studies,* vol. 27, *(5),* 2014.
[50] X. Huang, S. H. Teoh and Y. Zhang, "Tone Management," *The Accounting Review,* vol. 89, *(3),* 2013.
[51] A. K. Davis *et al*, "The Effect of Manager-Specific Optimism on the Tone of Earnings Conference Calls," *Review of Accounting Studies,* vol. 20, *(2),* 2014.
[52] S. L. Heston and N. R. Sinha, "News Versus Sentiment: Predicting Stock Returns From News Stories," 2015.
[53] K. D. Allee and M. D. DeAngelis, "The Structure of Voluntary Disclosure Narratives: Evidence from Tone Dispersion," 2014.
[54] M. López de Prado, "Advances in financial machine learning," 2018.
[55] M. J. Cooper, M. T. Cliff and H. Gulen, "Return Differences between Trading and Non-Trading Hours: Like Night and Day," *SSRN Electronic Journal*. DOI: 10.2139/ssrn.1004081.

Appendix

Types of Gaps

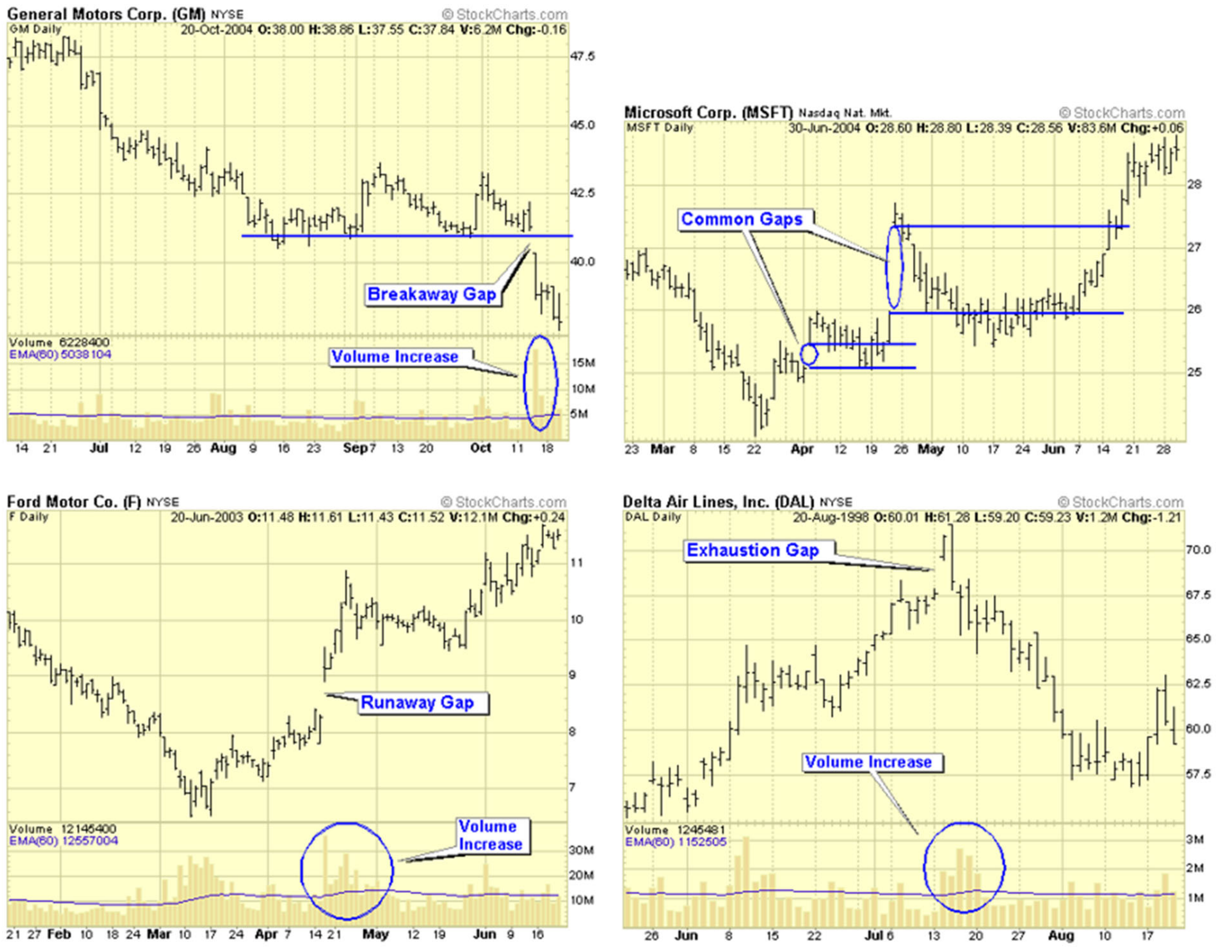


Fig. 41 - Adapted from StockCharts: Gaps and Gap Analysis

Monte Carlo Analysis and Equity Curves for Alan Fernandez’ Systems

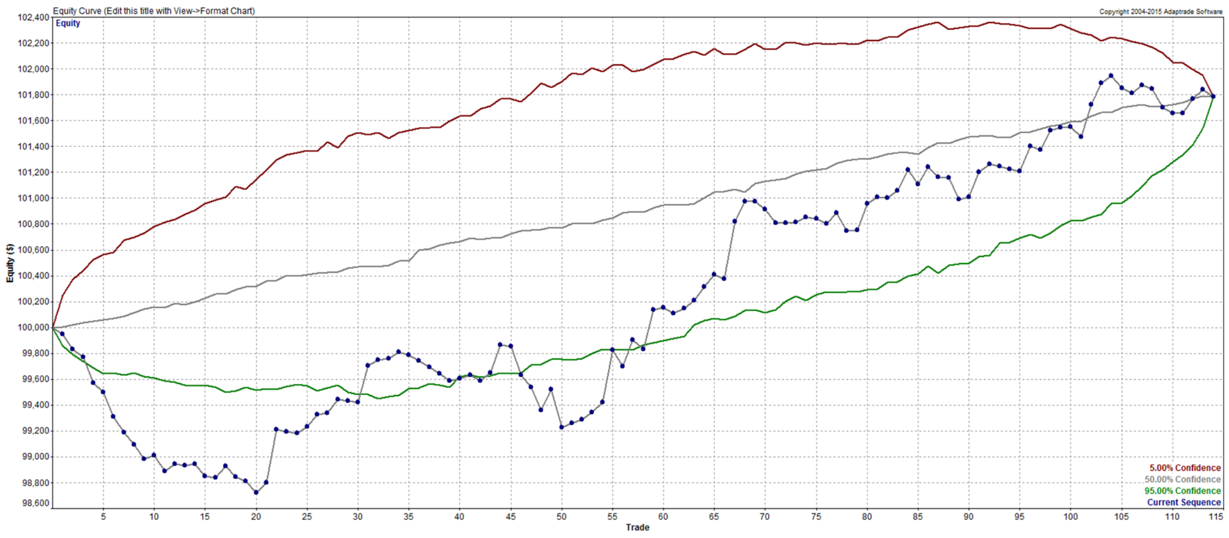


Fig. 42 - Monte Carlo Analysis - Sentiment Trader Strategy: USDJPY

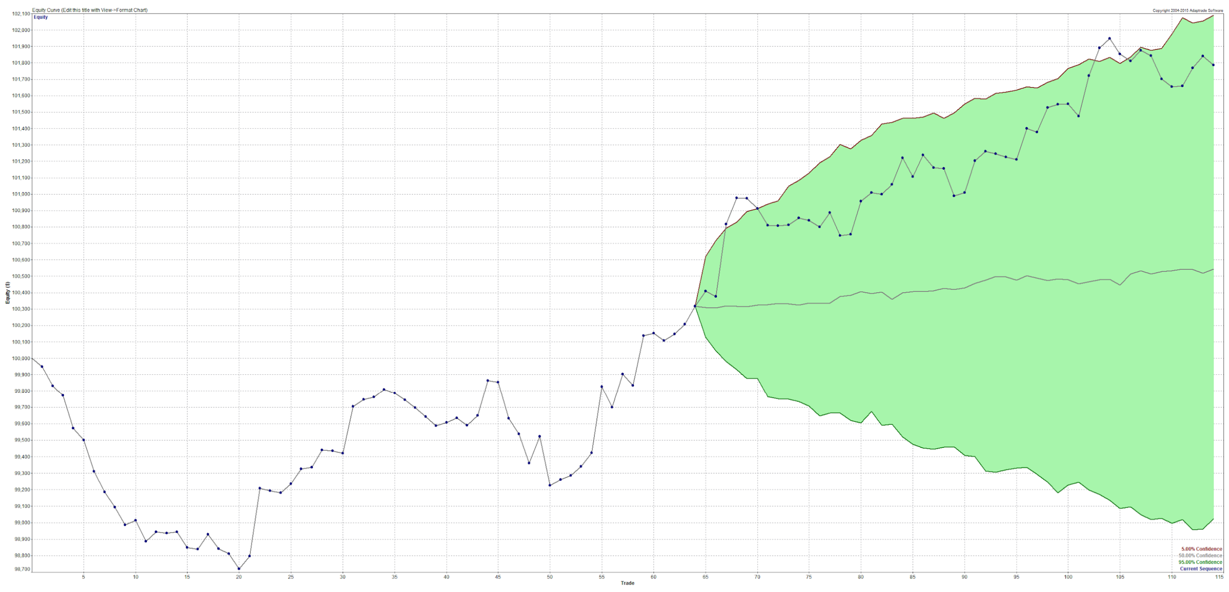


Fig. 43 - Monte Carlo Prediction - Sentiment Trader Strategy: USDJPY

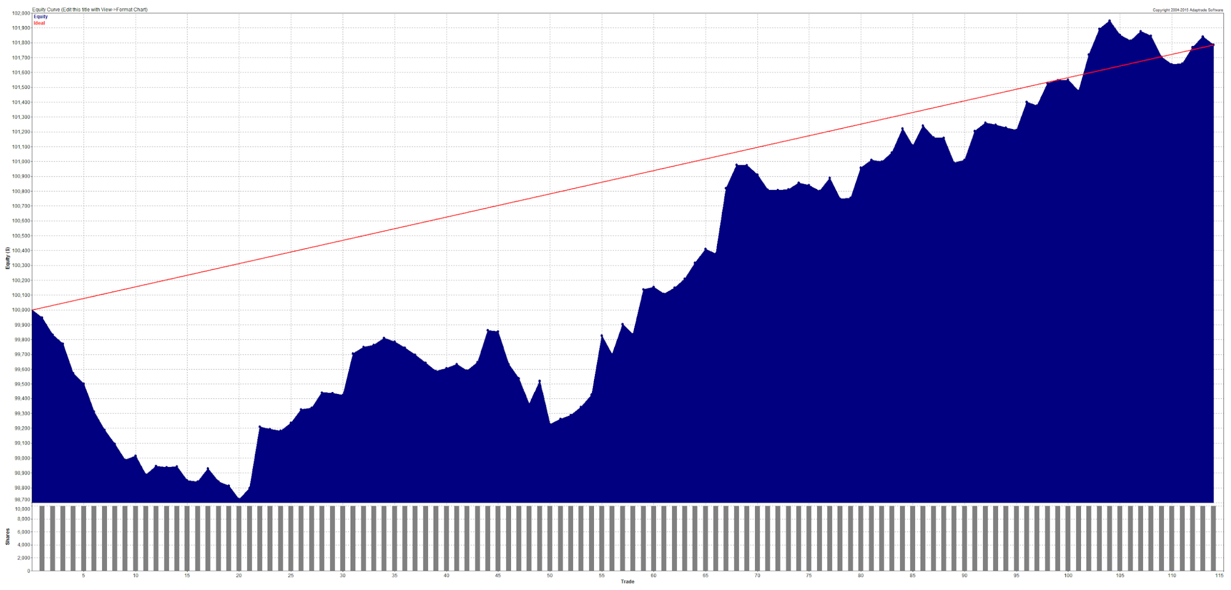


Fig. 44 - Trade Profit - Sentiment Trader Strategy: USDJPY

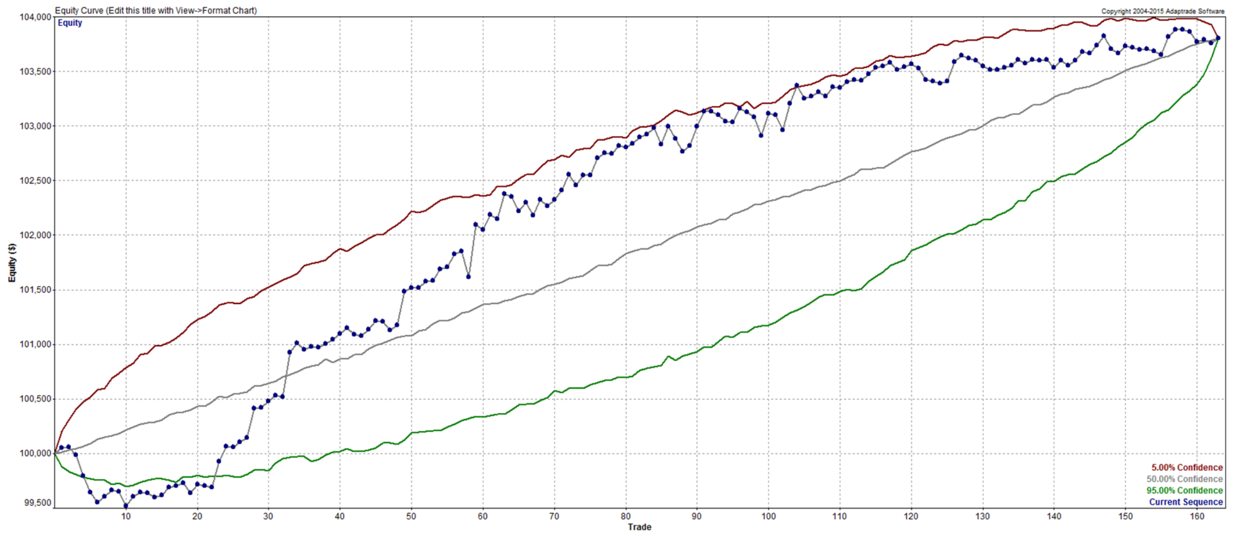


Fig. 45 - Monte Carlo Analysis - Sentiment Trader Strategy: USDJPY (ngram)

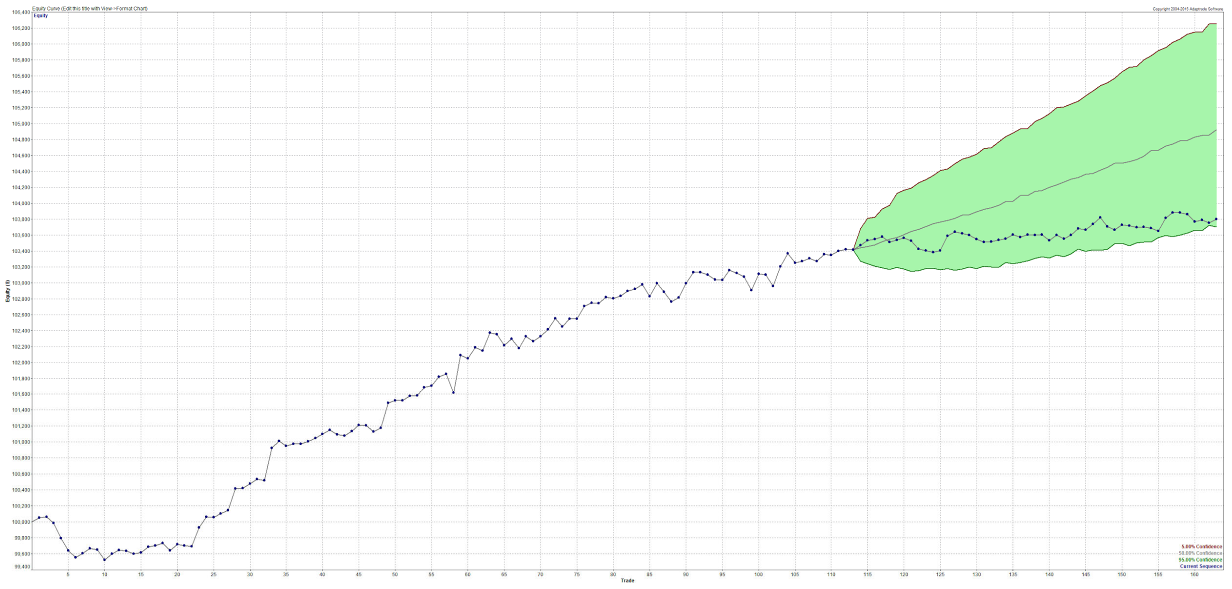


Fig. 46 - Monte Carlo Prediction - Sentiment Trader Strategy: USDJPY (ngram)

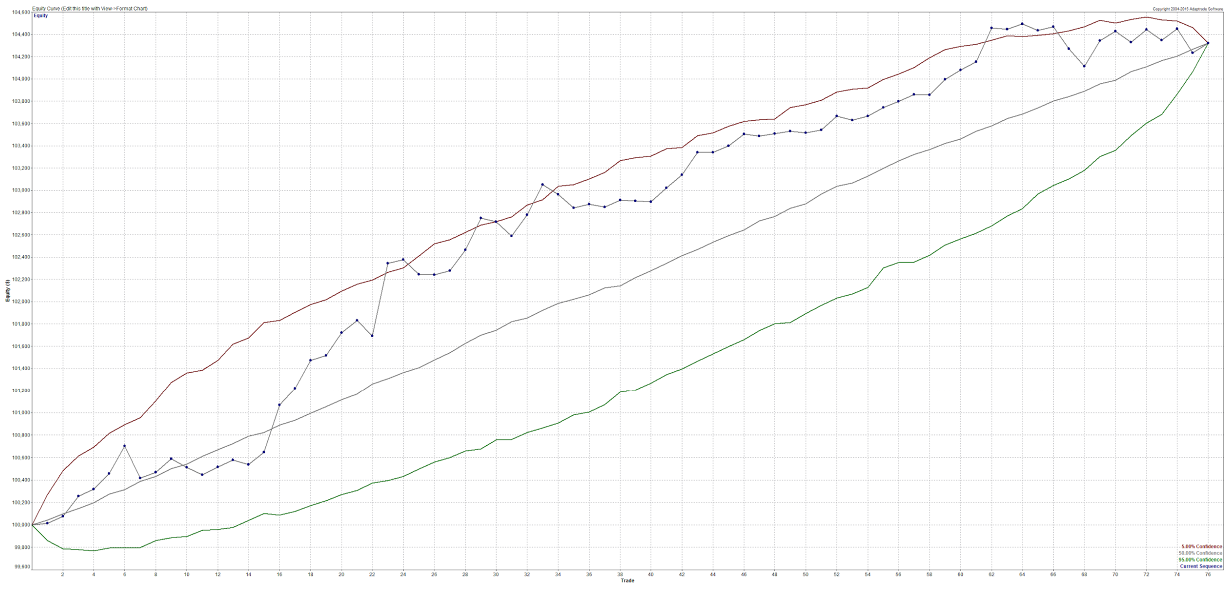


Fig. 47 - Monte Carlo Analysis - Sentiment Trader Strategy: EURUSD

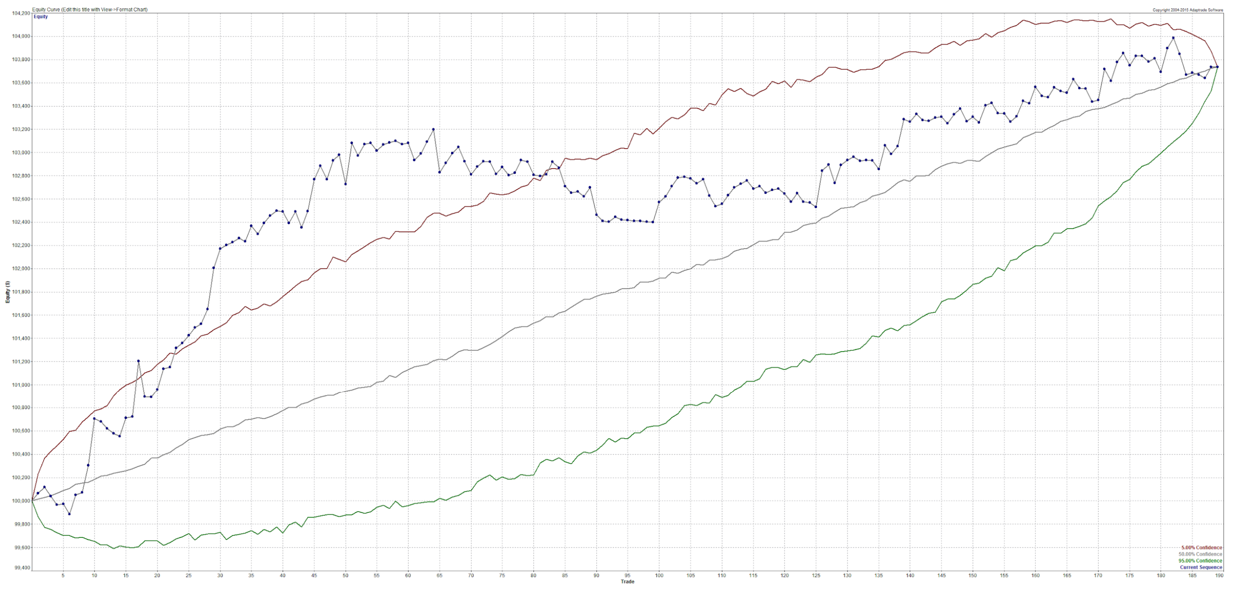


Fig. 48 - Monte Carlo Analysis - Sentiment Trader Strategy: EURUSD (ngram)

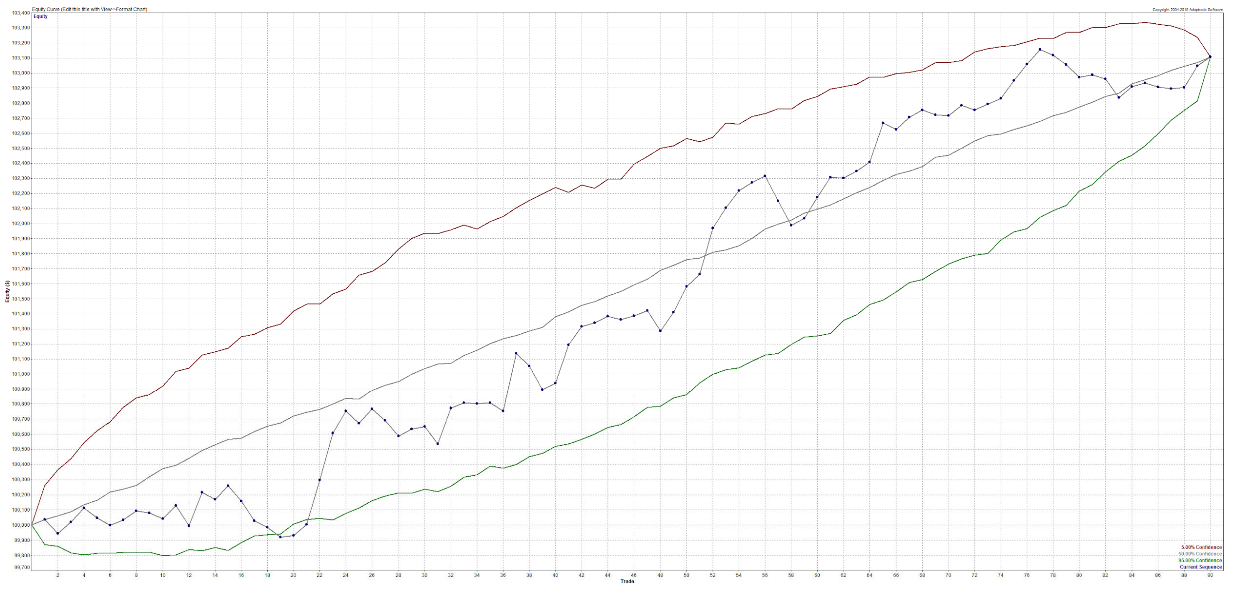


Fig. 49 - Monte Carlo Analysis - Sentiment Trader Strategy: AUDUSD

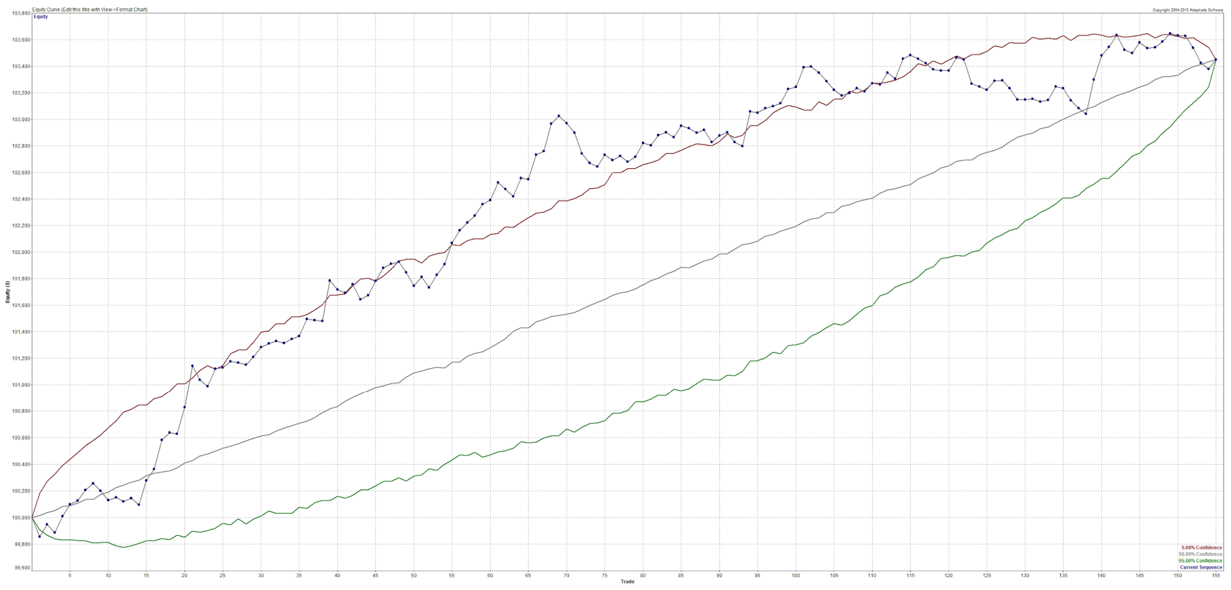


Fig. 50 - Monte Carlo Analysis - Sentiment Trader Strategy: AUDUSD (ngram)

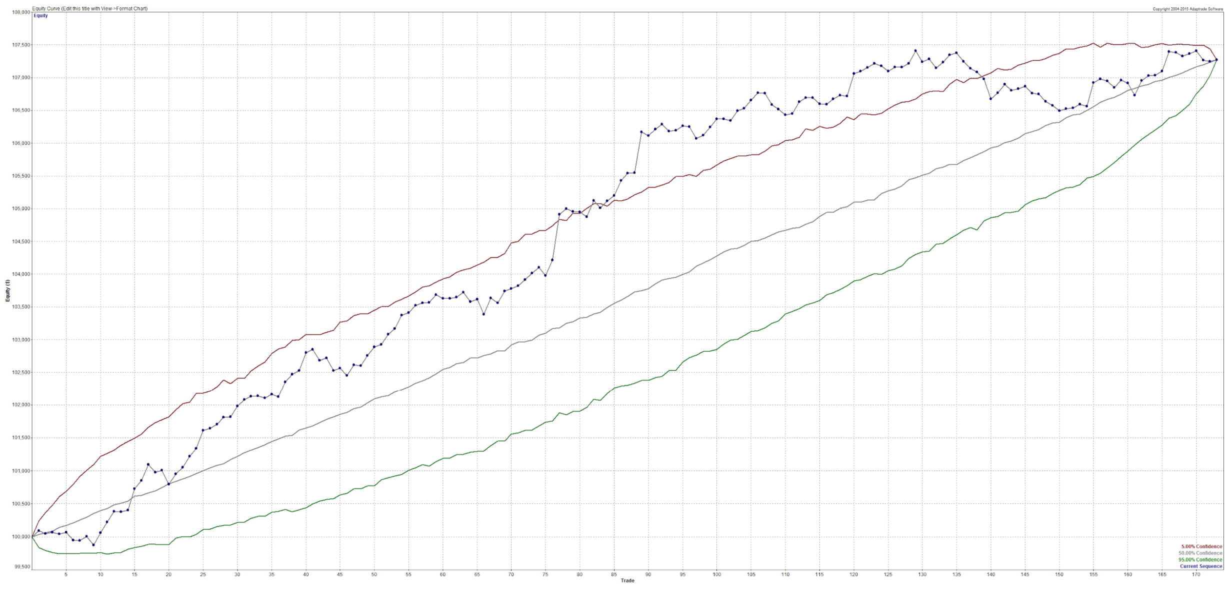


Fig. 51 - Monte Carlo Analysis - Sentiment Trader Strategy: GBPUSD

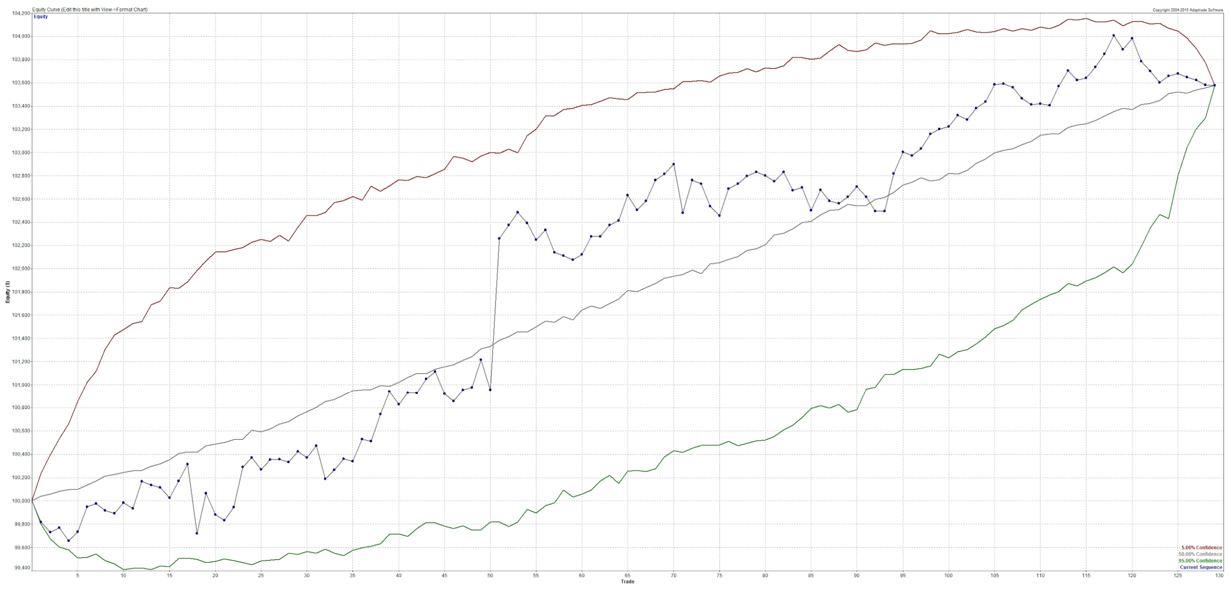
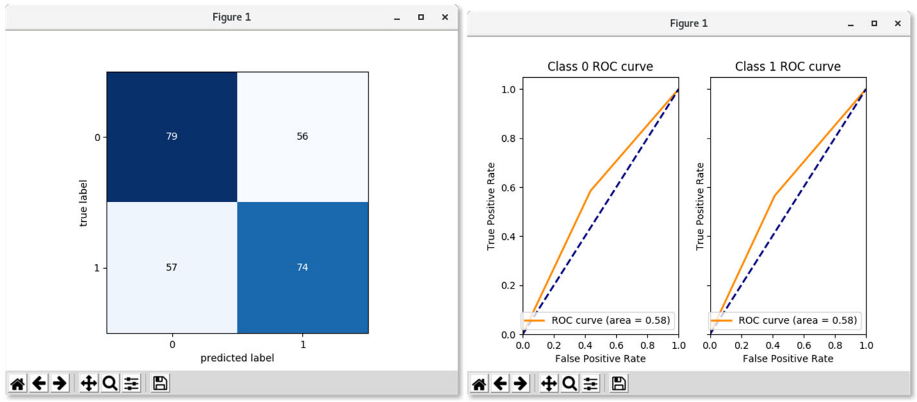


Fig. 52 - Monte Carlo Analysis - Sentiment Trader Strategy: GBPUSD (ngram)

System of Systems Performance Results

EMA 2-Category Performance:



	0	1
0	79	56
1	57	74

Fig. 53 - Fold-1 AUROC Curve and Confusion Table

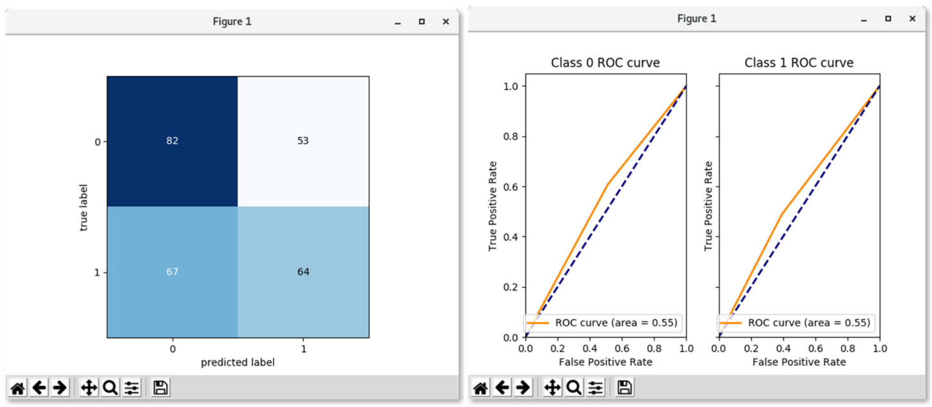


Fig. 54 - Fold-2 AUROC Curve and Confusion Table

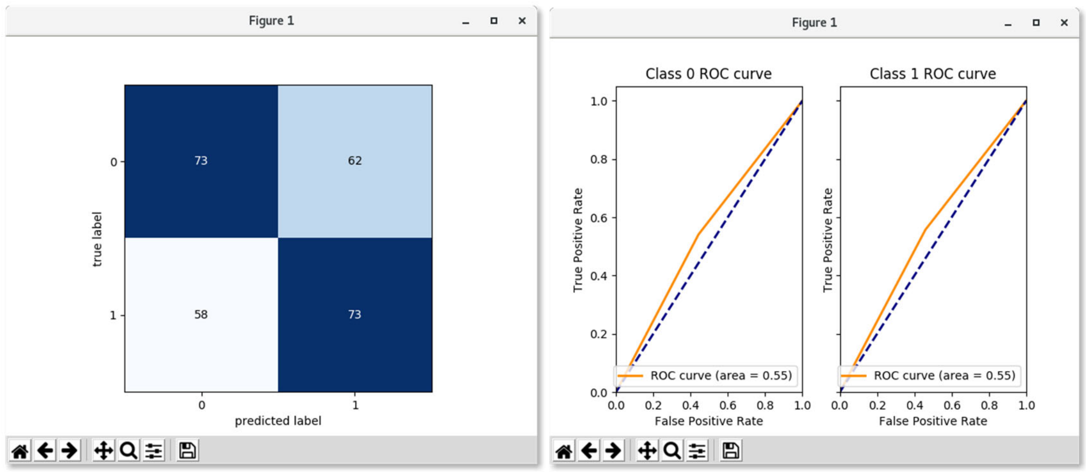


Fig. 55 - Fold-3 AUROC Curve and Confusion Table

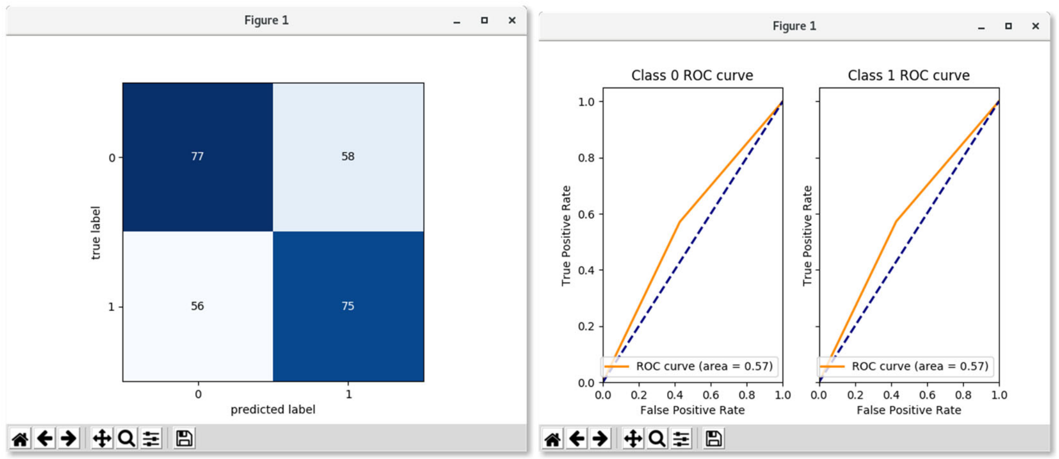


Fig. 56 - Fold-4 AUROC Curve and Confusion Table

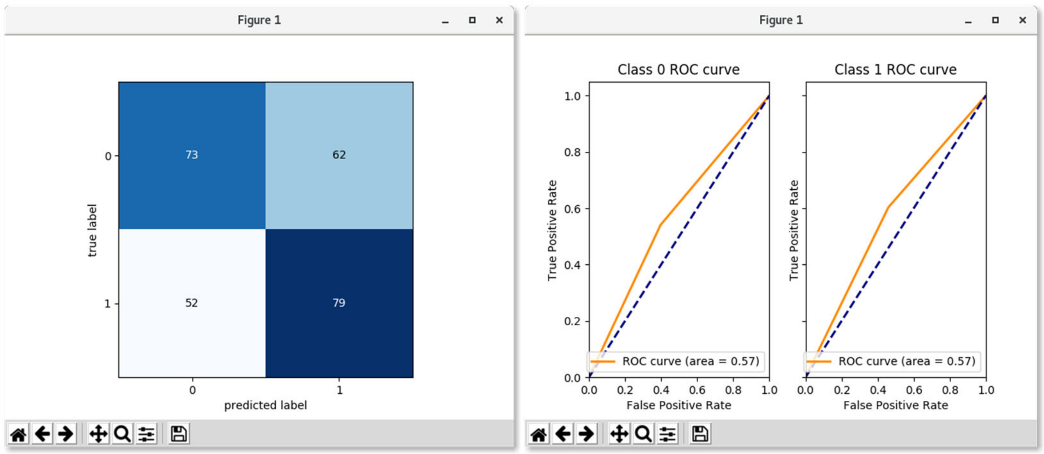


Fig. 57 - Fold-5 AUROC Curve and Confusion Table

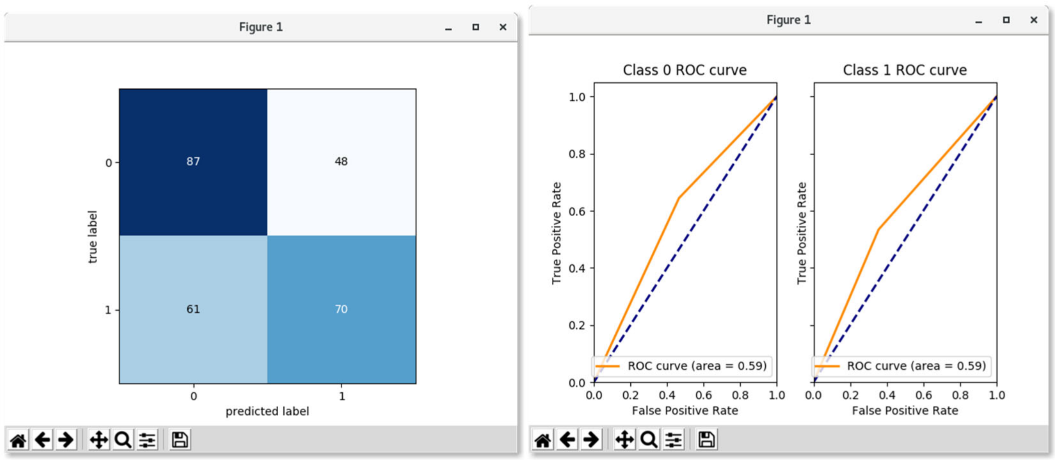


Fig. 58 - Fold-6 AUROC Curve and Confusion Table

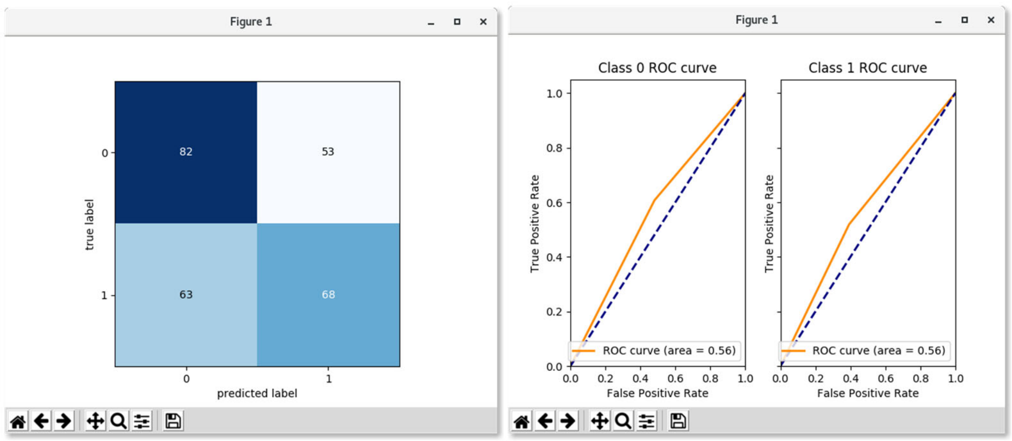


Fig. 59 - Fold-7 AUROC Curve and Confusion Table

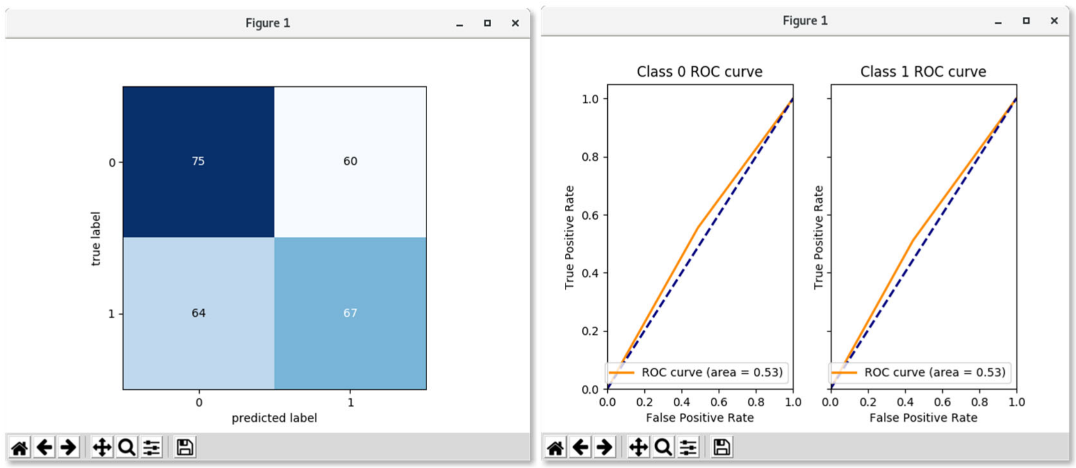


Fig. 60 - Fold-8 AUROC Curve and Confusion Table

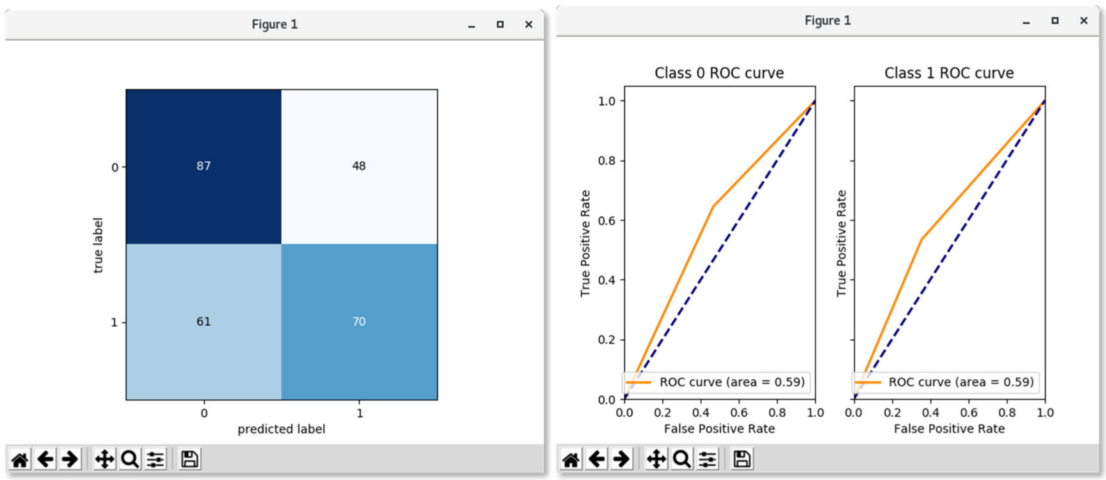


Fig. 61 - Fold-9 AUROC Curve and Confusion Table

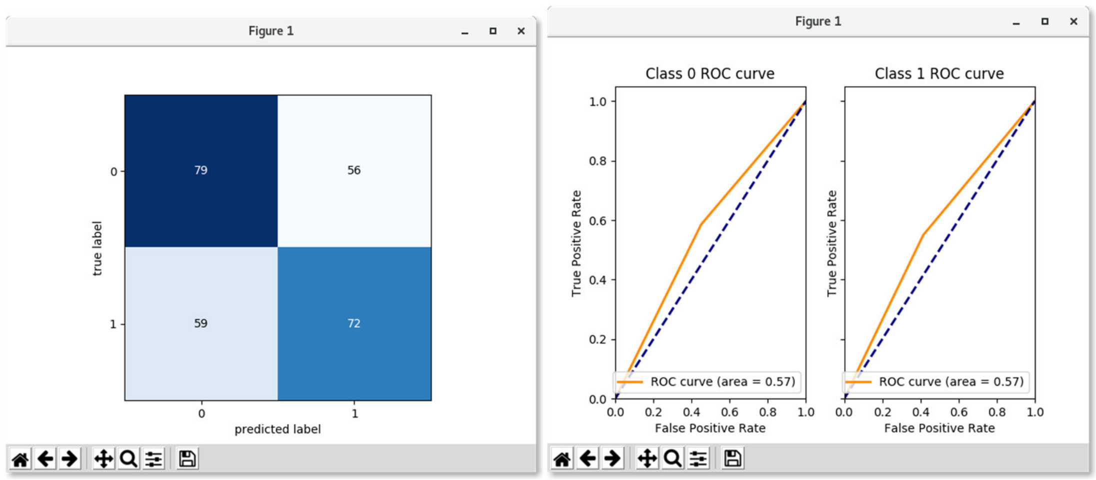
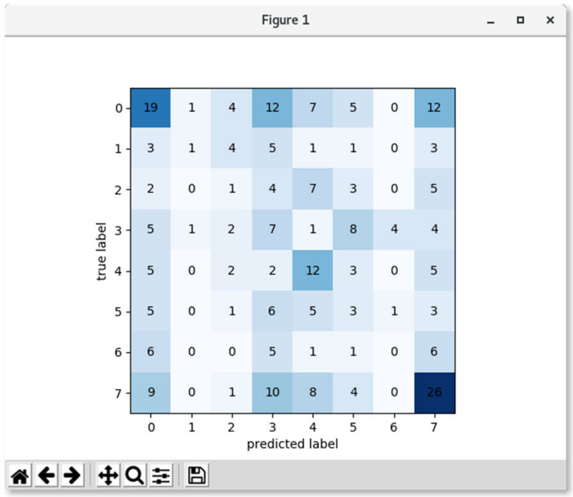


Fig. 62 - Fold-10 AUROC Curve and Confusion Table

EMA Strategy 8-Category Performance:



true \ predicted	0	1	2	3	4	5	6	7
0	19	1	4	12	7	5	0	12
1	3	1	4	5	1	1	0	3
2	2	0	1	4	7	3	0	5
3	5	1	2	7	1	8	4	4
4	5	0	2	2	12	3	0	5
5	5	0	1	6	5	3	1	3
6	6	0	0	5	1	1	0	6
7	9	0	1	10	8	4	0	26

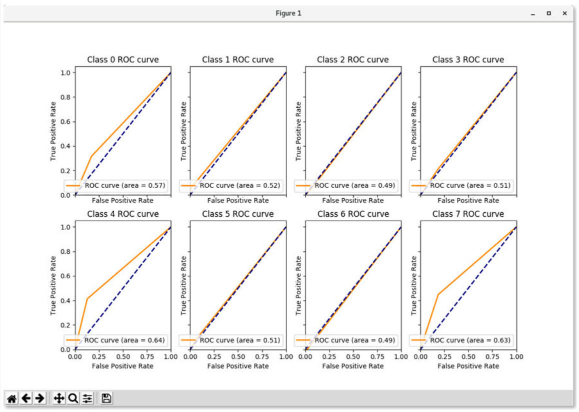
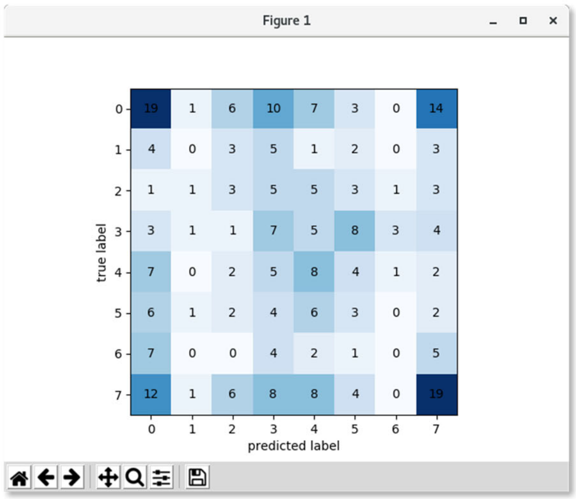


Fig. 63 - Fold-1 AUROC Curve and Confusion Table



true \ predicted	0	1	2	3	4	5	6	7
0	19	1	6	10	7	3	0	14
1	4	0	3	5	1	2	0	3
2	1	1	3	5	5	3	1	3
3	3	1	1	7	5	8	3	4
4	7	0	2	5	8	4	1	2
5	6	1	2	4	6	3	0	2
6	7	0	0	4	2	1	0	5
7	12	1	6	8	8	4	0	19

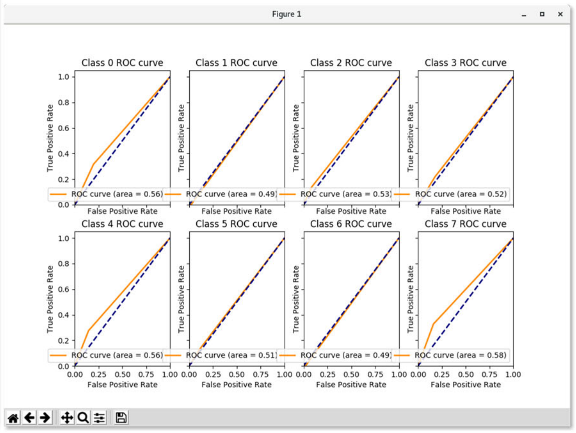


Fig. 64 - Fold-2 AUROC Curve and Confusion Table

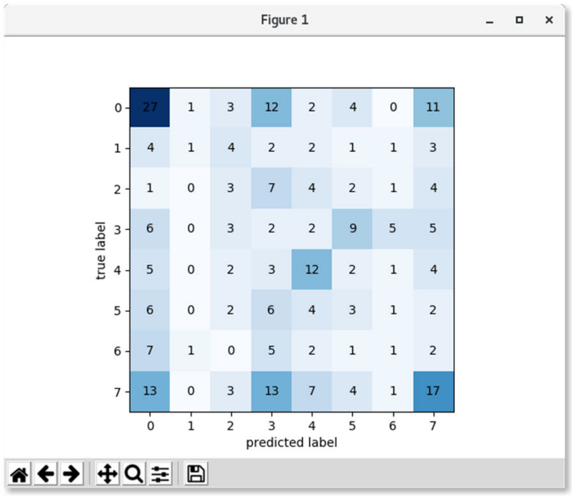


Figure 1

true label \ predicted label	0	1	2	3	4	5	6	7
0	27	1	3	12	2	4	0	11
1	4	1	4	2	2	1	1	3
2	1	0	3	7	4	2	1	4
3	6	0	3	2	2	9	5	5
4	5	0	2	3	12	2	1	4
5	6	0	2	6	4	3	1	2
6	7	1	0	5	2	1	1	2
7	13	0	3	13	7	4	1	17

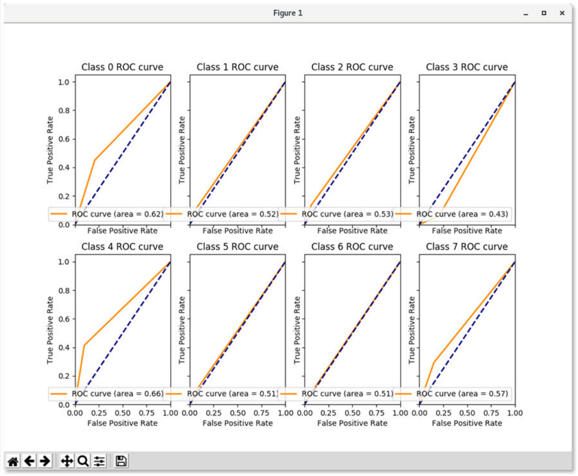


Fig. 65 - Fold-3 AUROC Curve and Confusion Table

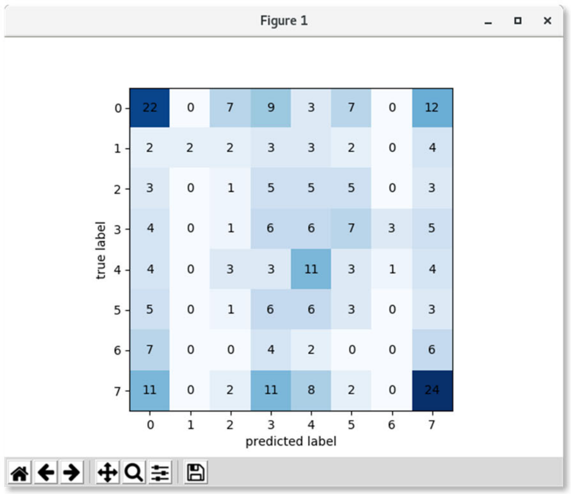


Figure 1

true label \ predicted label	0	1	2	3	4	5	6	7
0	22	0	7	9	3	7	0	12
1	2	2	2	3	3	2	0	4
2	3	0	1	5	5	5	0	3
3	4	0	1	6	6	7	3	5
4	4	0	3	3	11	3	1	4
5	5	0	1	6	6	3	0	3
6	7	0	0	4	2	0	0	6
7	11	0	2	11	8	2	0	24

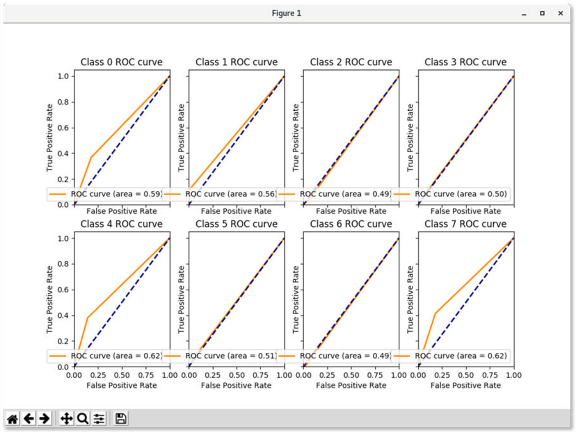
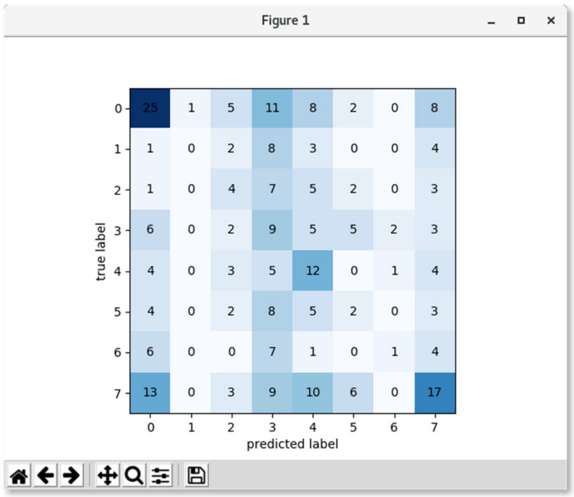


Fig. 66 - Fold-4 AUROC Curve and Confusion Table



	0	1	2	3	4	5	6	7
0	25	1	5	11	8	2	0	8
1	1	0	2	8	3	0	0	4
2	1	0	4	7	5	2	0	3
3	6	0	2	9	5	5	2	3
4	4	0	3	5	12	0	1	4
5	4	0	2	8	5	2	0	3
6	6	0	0	7	1	0	1	4
7	13	0	3	9	10	6	0	17

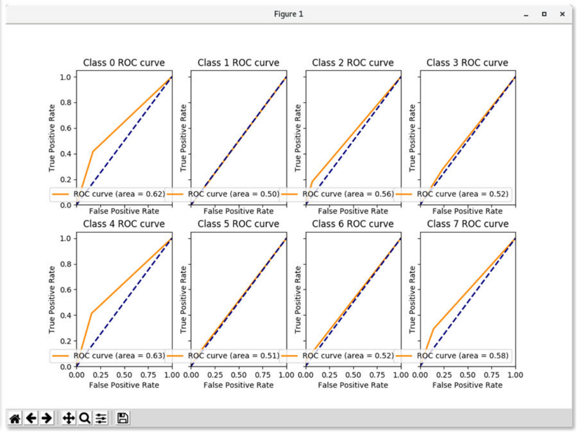
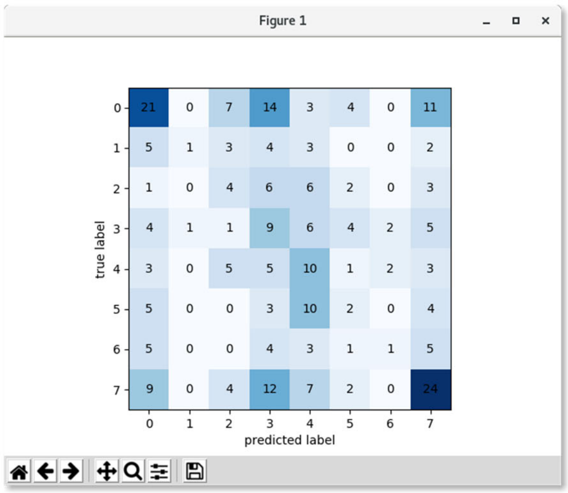


Fig. 67 - Fold-5 AUROC Curve and Confusion Table



	0	1	2	3	4	5	6	7
0	21	0	7	14	3	4	0	11
1	5	1	3	4	3	0	0	2
2	1	0	4	6	6	2	0	3
3	4	1	1	9	6	4	2	5
4	3	0	5	5	10	1	2	3
5	5	0	0	3	10	2	0	4
6	5	0	0	4	3	1	1	5
7	9	0	4	12	7	2	0	24

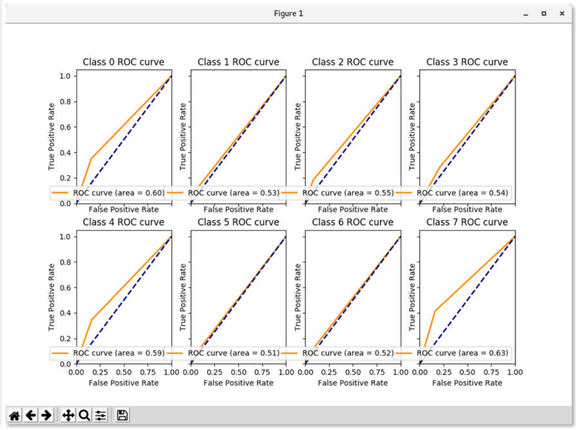
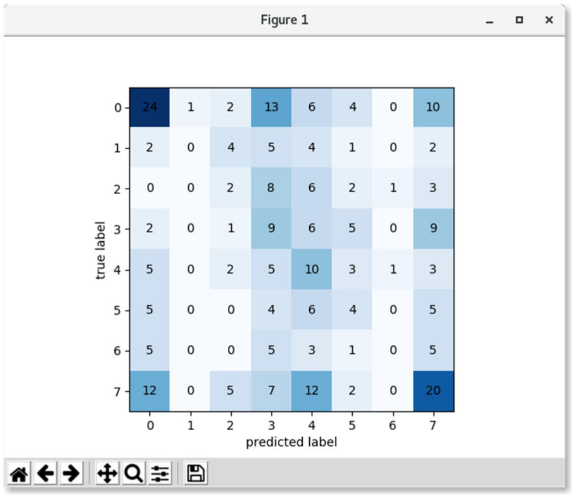


Fig. 68 - Fold-6 AUROC Curve and Confusion Table



	0	1	2	3	4	5	6	7
0	24	1	2	13	6	4	0	10
1	2	0	4	5	4	1	0	2
2	0	0	2	8	6	2	1	3
3	2	0	1	9	6	5	0	9
4	5	0	2	5	10	3	1	3
5	5	0	0	4	6	4	0	5
6	5	0	0	5	3	1	0	5
7	12	0	5	7	12	2	0	20

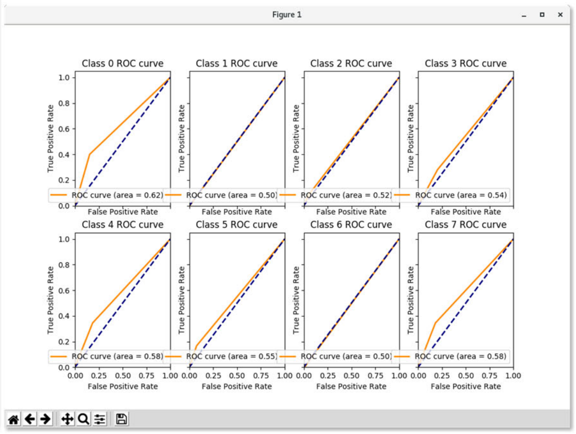
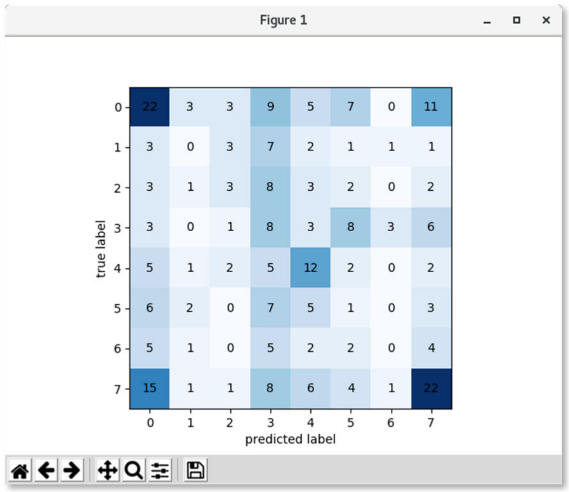


Fig. 69 - Fold-7 AUROC Curve and Confusion Table



	0	1	2	3	4	5	6	7
0	22	3	3	9	5	7	0	11
1	3	0	3	7	2	1	1	1
2	3	1	3	8	3	2	0	2
3	3	0	1	8	3	8	3	6
4	5	1	2	5	12	2	0	2
5	6	2	0	7	5	1	0	3
6	5	1	0	5	2	2	0	4
7	15	1	1	8	6	4	1	22

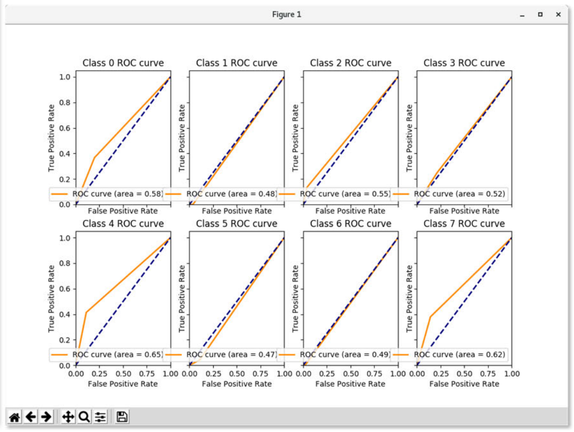
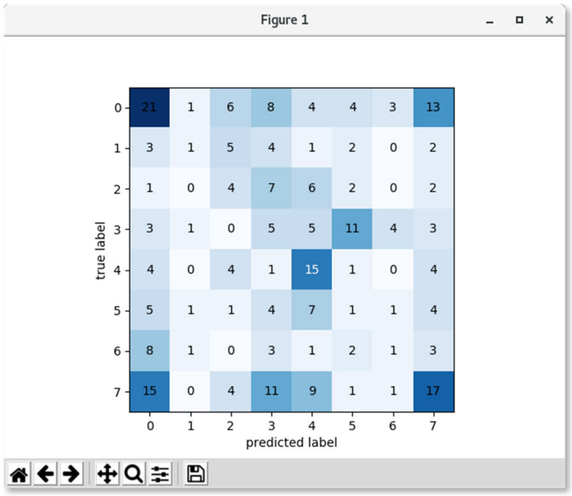


Fig. 70 - Fold-8 AUROC Curve and Confusion Table



	0	1	2	3	4	5	6	7
0	21	1	6	8	4	4	3	13
1	3	1	5	4	1	2	0	2
2	1	0	4	7	6	2	0	2
3	3	1	0	5	5	11	4	3
4	4	0	4	1	15	1	0	4
5	5	1	1	4	7	1	1	4
6	8	1	0	3	1	2	1	3
7	15	0	4	11	9	1	1	17

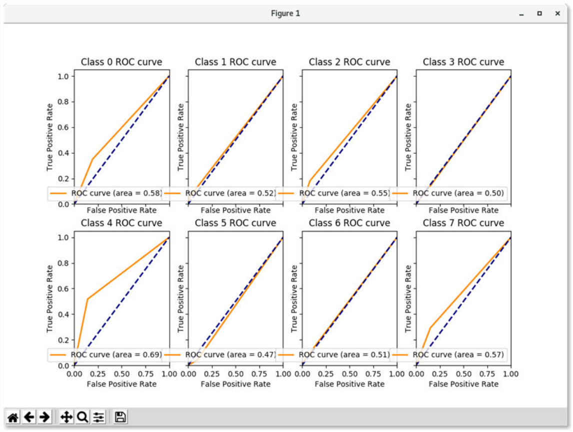
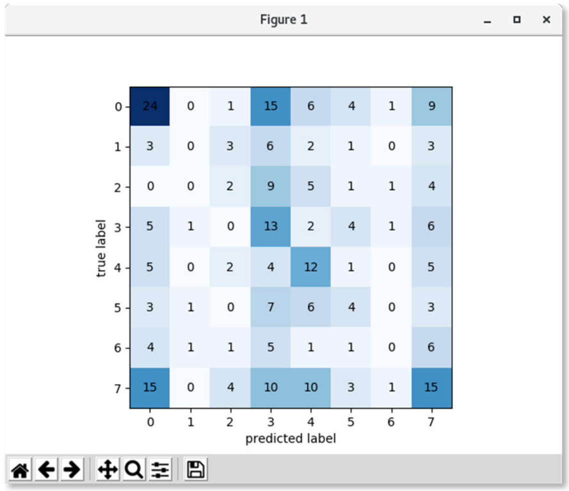


Fig. 71 - Fold-9 AUROC Curve and Confusion Table



	0	1	2	3	4	5	6	7
0	24	0	1	15	6	4	1	9
1	3	0	3	6	2	1	0	3
2	0	0	2	9	5	1	1	4
3	5	1	0	13	2	4	1	6
4	5	0	2	4	12	1	0	5
5	3	1	0	7	6	4	0	3
6	4	1	1	5	1	1	0	6
7	15	0	4	10	10	3	1	15

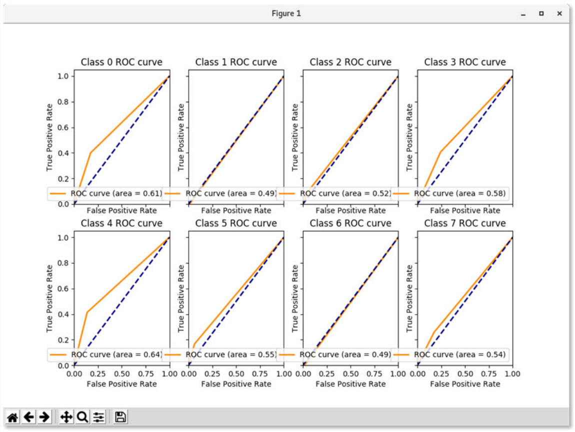


Fig. 72 - Fold-10 AUROC Curve and Confusion Table