Table of Contents

Introduction

A friend of yours comes to you with questions about the stock market. You tell them that now would be a terrible time to buy stocks. After all, in the midst of a global pandemic, surely stock prices would drop dramatically. You quickly google to confirm and see that contrary to your assessment, stock prices have surged in recent days, despite news of rising death totals and record unemployment. You tell your friend to speak with someone with a financial background.

The rises and falls of the stock market are seemingly impossible to predict. At the same time, there is a vast wealth of historical records to explore going back decades. Armed with data science techniques and a little computing power, would it be possible to make sense of stock market prices? Are there any trends that have repeated over the years? What features are most useful in predicting tomorrow's prices?

Datasets

1. https://www.kaggle.com/jacksoncrow/stock-market-dataset
   This dataset contains multiple decades of daily historical stock prices sourced from Yahoo Finance. There are over 5,000 stocks contained in the dataset, which has been updated to the current month.

2. Data will also be sourced from https://iextrading.com/ via the API provided by https://alpaca.markets/, a trading platform with many convenient features such as paper trading and backtesting. This data can be used to trade on the current day's market information.

3. If you have a verified Alpaca API account, which is currently available only to US citizens, you can use the Polygon API. It features more symbols, more data points by default, and more accurate historical data.

What You'll Need

- Python 3 Environment
- Data Science Python Libraries (Pandas, Numpy, Matplotlib, etc.)
- Technical Indicator Library (TA-Lib, ta)
- Alpaca API Paper Account (Verified Account is optional but recommended)
- My GitHub repository which contains the code necessary to run the pipeline.

Data Acquisition

Initially, I used a Kaggle Dataset which sourced its data from Yahoo Finance. It contained over 5000 stock symbols, an overwhelming number for testing purposes. Using Volume (a value representing the number of trades performed within a period) as a metric, I reduced this number to the top 500 stock symbols.

I encountered a number of issues with this dataset. 'Zero' values were a common sight, and for time series data these missing values are difficult to impute without introducing inaccuracies. The dataset was also static, requiring manual downloads to update to a newer version.

I then switched to the Alpaca Data API, which is available to all Alpaca accounts. Using the Alpaca Python API endpoints, you can download up to 1000 previous data points for 100 symbols per call. A timeframe value can be adjusted for daily or intraday results. While day trading is allowed on an Alpaca paper account, please note that there are special rules and limits for day trading when using real money.

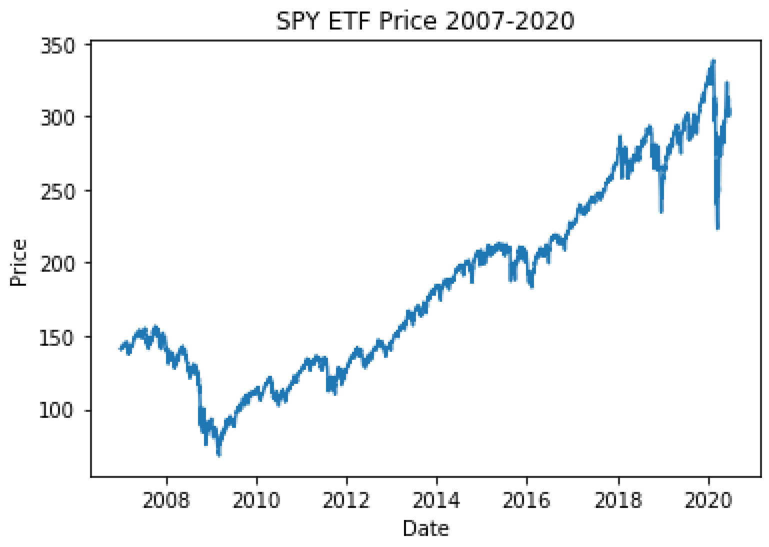
The Data API historical data was more accurate, but after encountering some strange results, I noticed that errors were still present in the data. These can be mitigated by removing the offending values after normalization. If you are a US Citizen and don't mind signing up for a verified Alpaca account, there is a third option available to you.

The Polygon API provides real time historical stock data and boasts higher accuracy. You can pull up to 3000 previous data points per call. However, you can only download one stock symbol per call, so be wary of reaching the API rate limit. In the included GitHub repository, I have separate notebooks for both the Data API and the Polygon API data acquisition. When running the pipeline, choose one or the other according to your account.

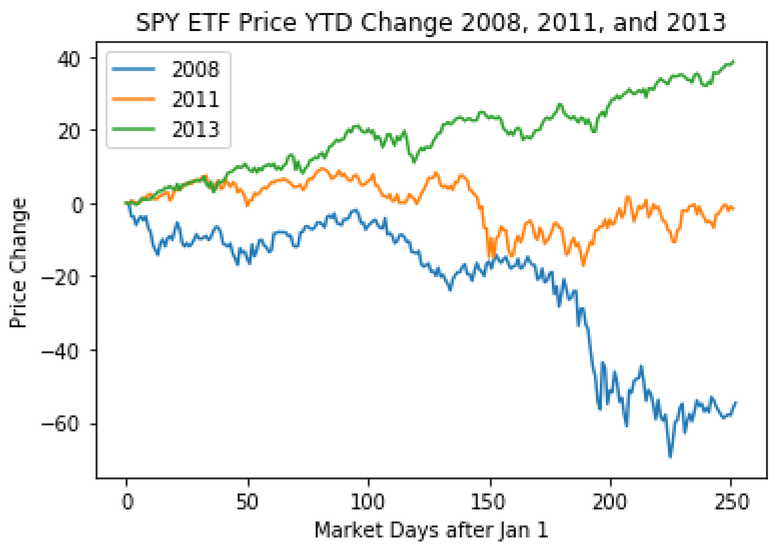
In addition to individual stock data, I also acquired information on the market performance as a whole. This can be done either by averaging values from all stocks, or more simply using data from an ETF (Exchange-Traded Fund). ETFs use multiple stocks to derive their value. The more popular ETFs include SPY, which tracks the S&P 500, and DIA, which tracks the Dow Jones Industrial Average. Utilizing ETFs for market data saves time and disk space.

Exploratory Data Analysis:

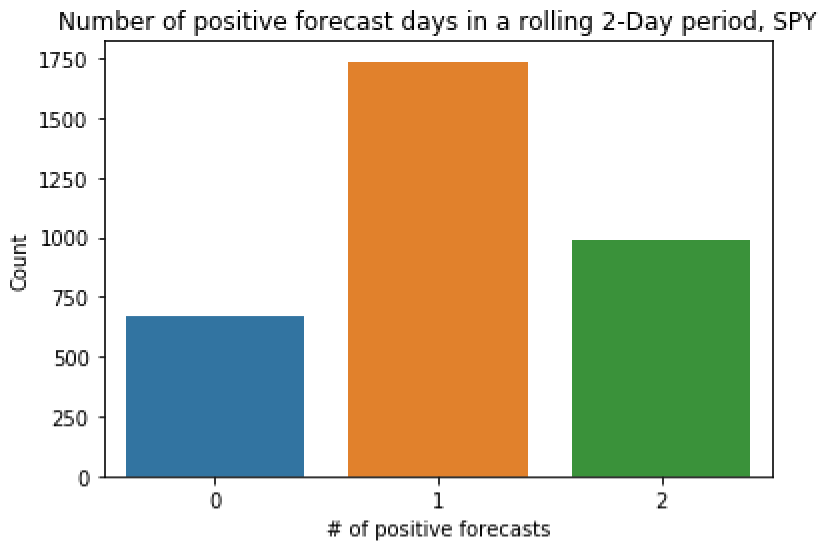
What trends can be found with over a million data points? To begin, it's best to look at the market as a whole. The SPY ETF approximates the market using a collection of stocks. From 2007 to 2020, the market has doubled its value. However, there have been periods of growth and recession.



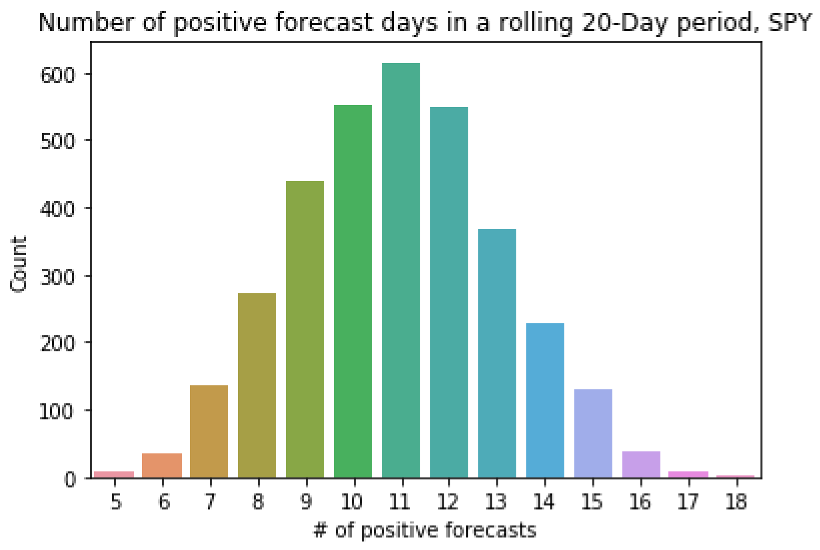
Let's take a closer look. The years 2008, 2011, and 2013 represent the different types of markets. 2008 shows a market in decline. In 2011 the recovery had yet to take full effect, and the price at the beginning and the end of year were nearly the same. 2013 was a year of consistent growth.



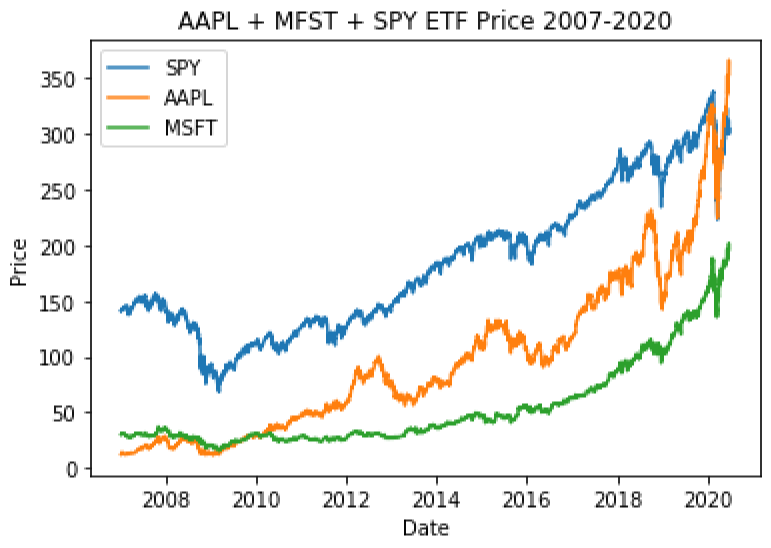
A naive trader might buy when the previous day's price change was positive, and sell when the previous day's price change was negative. However, when a rolling period of two days is sampled over the entire dataset, days where the price change trend reverses are more common than days that continue a trend. In the short term, the market is volatile.



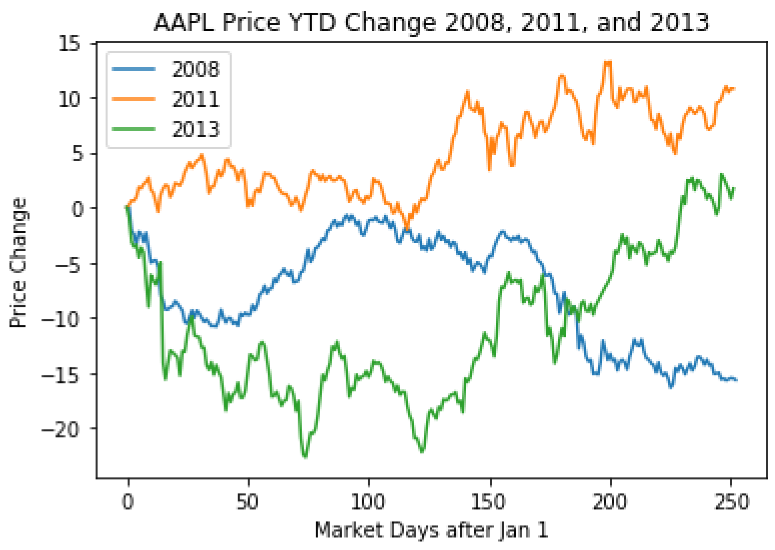
Over a rolling period of 20 days, the market shows an overall positive trend. The expected number of positive days is 11. The expected monetary return during a 20 day period is also positive. Going by these results, long term strategies have been safer for the average trader.



How closely do individual stocks match the market? To visualize this, I chose two well known stocks to plot alongside SPY, AAPL (Apple) and MSFT(Microsoft). Just by looking at the prices from 2007 to 2020, both stocks follow a similar pattern of growth.



Microsoft's prices in 2008, 2011, and 2013 matched the SPY prices. However, Apple actually performed worse in 2013 than it did in 2011. Slowing sales and profit margins triggered a decline in the stock price, despite the rest of the market performing well.



Individual stock price forecasts match the market forecasts only 63% of the time. The market is a driving force for stock prices, but individual events can override this effect. Certain industries do match the market more closely, however. For example, financial institutions make up five of the top ten stocks by market influence and match the market forecast over 75% of days.

Technical Indicators

Just the market data alone is not enough to predict future prices. To create a robust model, feature generation is required. To accomplish this, I will use technical indicators. A technical indicator is a fancy term for a function that modifies historical stock prices. This modification usually reveals a trend or a signal that can be used to make decisions when buying or selling stocks. For example, a simple Moving Average may be used to determine if a stock is trading above or below its average price for a specific time window.

Technical Indicators are not perfect. While they do provide additional insight into a stocks current strength, they have a few drawbacks. Firstly, they are reactionary. A moving average can only go up once the stock has already climbed a little. Most indicators have various levels of “lag” before they reveal any sort of trend. At best, you can catch onto a trend quickly, or be more sure of the strength of a trend.

Additionally, technical indicators cannot predict world events. Stocks are (usually) a reflection of the strength of the company they represent. If something happens that boosts the value of the company, it will (usually) be reflected in the stock’s value as well. Technical indicators cannot account for sudden changes in a stock’s future value.

There are many types of technical indicators. I used the well organized documentation of TA-Lib, a technical indicator library, to learn about many of the features used in my model.

Types of Technical Indicators

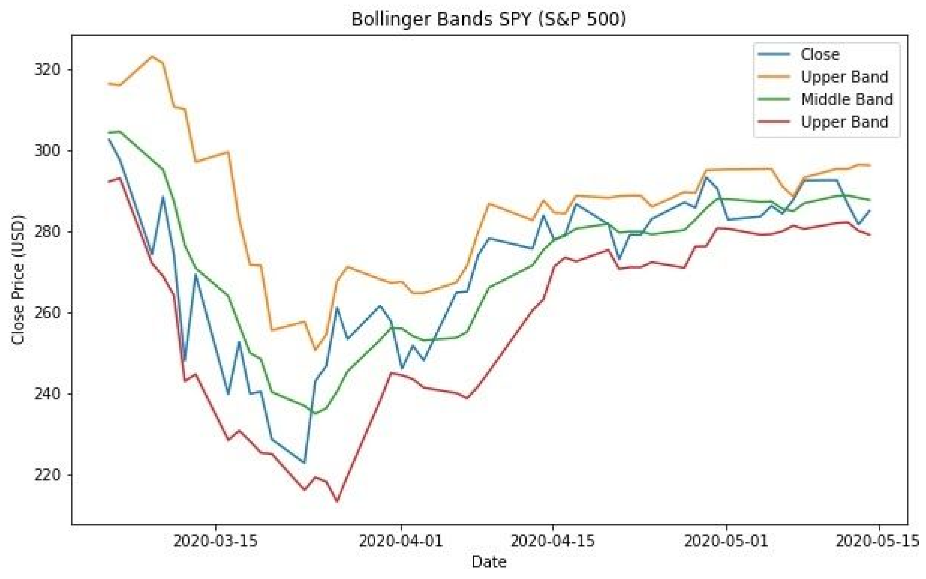
- Overlap Studies
- Momentum Indicators
- Volume Indicators
- Statistic Functions

Overlap Studies

Overlap studies use moving averages to measure stock performance. The term overlap refers to periods in time when the stock price "overlaps" with a particular moving average. This may indicate a time to buy or sell a stock depending on the direction of movement.

Bollinger Bands

The Bollinger Bands are likely the most well known set of technical indicators. They consist of three features. The upper and lower bands represent (by default) two standard deviations above and below the middle band. The middle band is a simple moving average of the stock price, but can be modified to use other types of moving averages. The width between the upper and lower bands may also be used as an indicator, as the distance between them expands when the price changes, and contracts when the price stays the same.



As you can see from the plot above, The closing price crosses over the middle Bollinger Band at multiple points. The width of the bands contracts as the price stabilizes. From just three lines, a lot of information is revealed about the stock's directional trend, as well as the strength of the trend.

Momentum Indicators

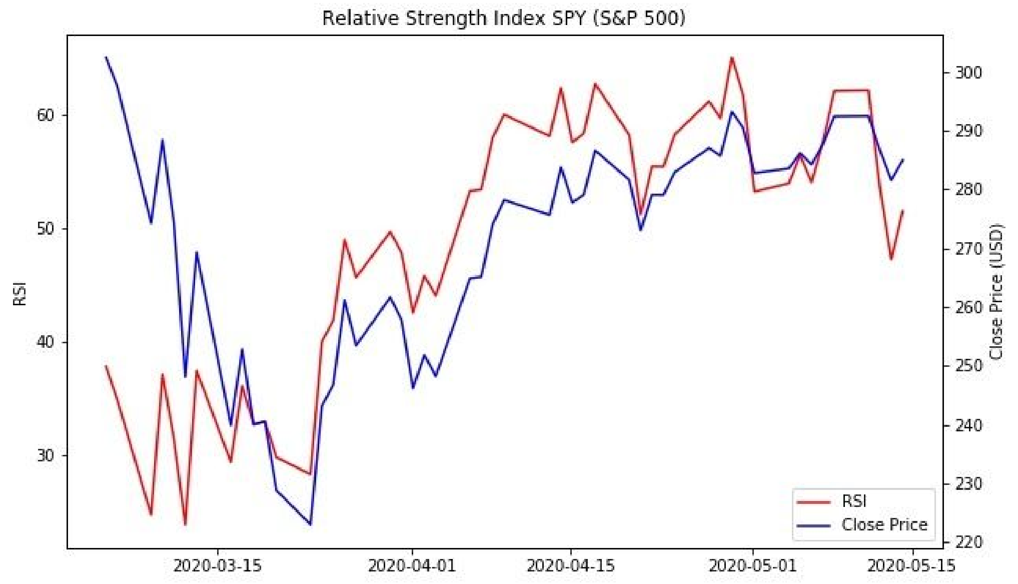
Momentum Indicators look at changes in a stock's price to measure how fast a stock is rising or falling. Unlike overlap studies, many momentum indicators return a score within a specific range.

RSI

The Relative Strength Index, or RSI, is a value between 0 and 100 that determines the strength of a stock's positive trend. A value of 0 indicates that the stock has had no positive days in the chosen time period. A value of 100 indicates that the stock has only had positive days in the chosen time period.

On its own, the RSI has two weak points. First, the indicator has inherent lag. An RSI can only go up after a positive day, so you might miss the entirety of a quick upward trend. Second, the direction of the RSI matters. A value of 50 will not tell you if the previous day was

40 or 60. For a deep learning model which processes days one at a time, this information will be lost.



The RSI (in red) follows the close price closely, but there are some subtle differences. A value below 50 indicates a negative price trend, which holds steady at the beginning despite the price dropping. The value only goes above 50 at the tail end of the stock's increase. While the RSI does miss trends on its own, it has value in understanding the current strength and momentum of a stock, which works well in combination with other technical indicators.

Volume Indicators

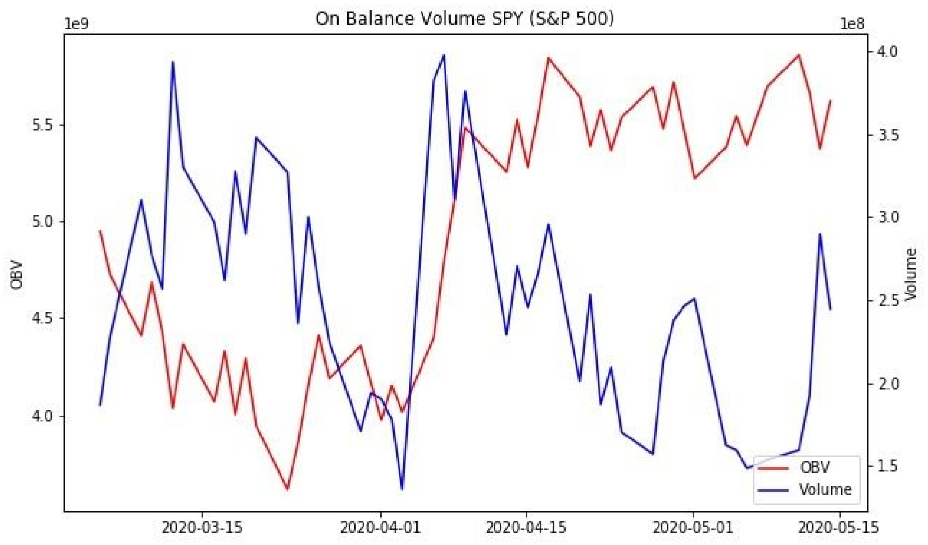
Volume is a measurement that tracks how often a stock is traded. The volume of a stock is the absolute number of trades that occurs on a given trading day for a particular stock. Both buys and sells count as trades, so a stock's volume may be considered "buyer-controlled" or "seller-controlled" depending on the price movement. Some indicators incorporate volume to predict price changes.

On Balance Volume

On Balance Volume (OBV) is a unique indicator. Unlike other indicators, it does not use a time period to calculate its value. Because the calculation is cumulative, the starting date determines the end result. Each value depends on the previous value. If the price goes up, the day's volume is added to the OBV. If the price goes down, the volume is instead subtracted from the OBV.

In theory, the On Balance Volume can show disconnects between a stock's price and volume. A stock which is increasing in price but has a sluggish OBV may indicate a drop in price in the near future. This makes the OBV a leading indicator, or an indicator that may show a change in price before it happens.

However, OBV on its own is not enough, and requires additional information to show a definitive trend. Because the indicator's value is cumulative, it is difficult to normalize to a value that makes sense across multiple stocks, limiting its use in machine learning. Finally, OBV is very sensitive to large spikes in volume, which affects the value long after the spike is over.



Notice the spike in the value of OBV which matches the spike in the stock's volume. This raises the OBV level for over a month. For this reason I do not recommend OBV when creating a machine learning model, despite the many uses it has in manual trading.

Statistic Functions

In addition to technical indicators, more traditional approaches may be used to predict stock prices and trends. Statistic Functions rely on mathematical principles

Linear Regression

The Linear Regression is the best fit line for a stock within a specific time period. While the stock may seem volatile on the day-to-day scale, the Linear Regression line may reveal a long-term trend that will outperform short term-trends.

There are hundreds of technical indicators out there. Many of the indicators I used in my final model were sourced from TA-Lib. I used only a handful of the available indicators, so if you wish to test my model with additional features, the TA-Lib documentation is a great place to start.

Machine Learning

I decided to test three different approaches for creating a machine learning model. By doing so, I could test how the different models compared to each other.

1. **XGBoost**

The first approach I tested was a gradient boosting algorithm. Gradient boosted models use decision trees to make predictions. In gradient boosting, a decision tree is “boosted” in each iteration by improving upon a defined loss function. This is different from a Random Forest algorithm, where many trees are aggregated all at once. The specific implementation of gradient I used was a library called XGBoost, which stands for eXtreme Gradient Boost. XGBoost is a popular tool used in many data science competitions.

2. **Deep Learning**

One of the more exciting applications of machine learning lies in deep learning. Using a neural network, a deep learning model can find complex connections in a feature set. I used the Keras library to compile and train a neural network model.

3. **Market Features Included**

The first two approaches use only the features computed using individual stocks. By including features computed for the SPY ETF, I doubled the feature set. This approach will test the market’s influence on individual stock prices.

Preprocessing

Two preprocessing steps were critical for improving the models. First, I needed to resample the data for class balance. When the two target classes are unbalanced, a model will over-predict the majority class. In the case of stock data, the forecast is more likely to be positive than negative. My models would just predict one class only, as that would provide the best results.

To counteract this, there are two options. You can over-sample the data and copy random instances of the minority class until the dataset is balanced. You can also under-sample, picking out balanced data from the existing data points. Under-sampling is significantly faster for large datasets, and is the solution I ultimately used.

Second, the data must be scaled to improve accuracy. Because many of the features have significant outliers, the StandardScaler and MinMaxScaler are not appropriate for this dataset. Scikit-learn provides a QuantileTransformer that accounts for these outliers, ignoring them when scaling the data. Any outliers are then placed at the minimum or maximum values. Changing the scaler accounted for one of the most significant improvements in my final model.

Model Capacity

When deciding on how many hidden layers to include, I came to understand the concept of model capacity. There are two ways to increase capacity in a neural network. You can either increase the number of hidden layers, or you can increase the node count in any of the layers. Increasing the capacity too much will make your model slow to train, but a low capacity model may not pick up on all of the possible feature connections. Because of my large feature set, I was able to increase the model's capacity significantly before experiencing diminishing returns.

Results Comparison (Test Set)

	w/ Market Features	w/o Market Features	XGBoost
Accuracy	**~71%**	**~56%**	**~53%**
F1 Score	**0.737**	**0.621**	**0.567**

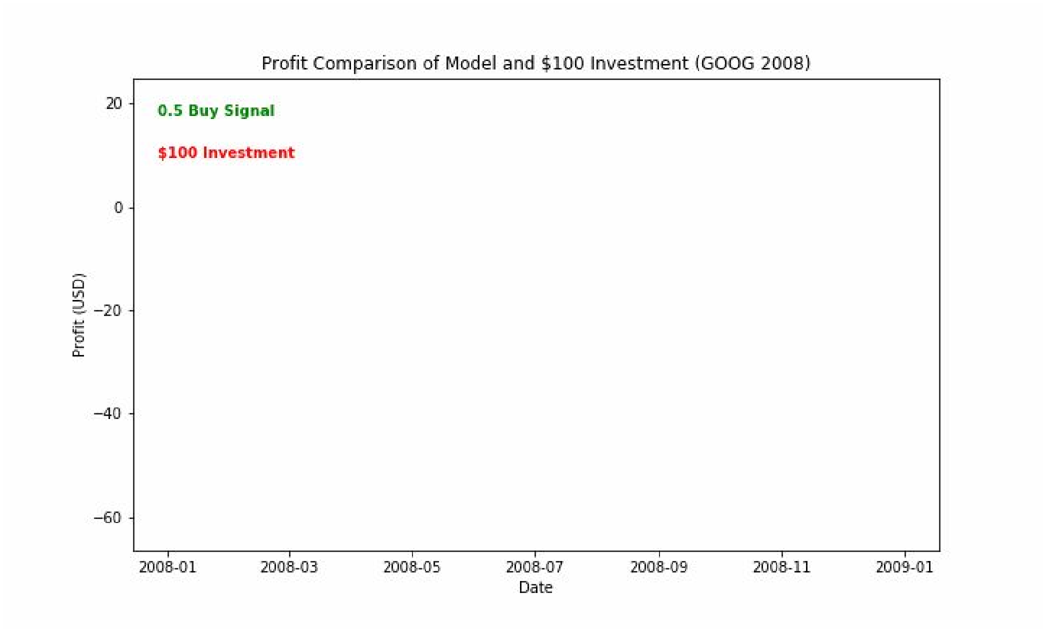
Going by the metrics of accuracy and F1 Score, the market model far outperforms the other two. However, the best metric for an investment model is profit. Do these models hold up when tested on sequential market data?

Backtesting

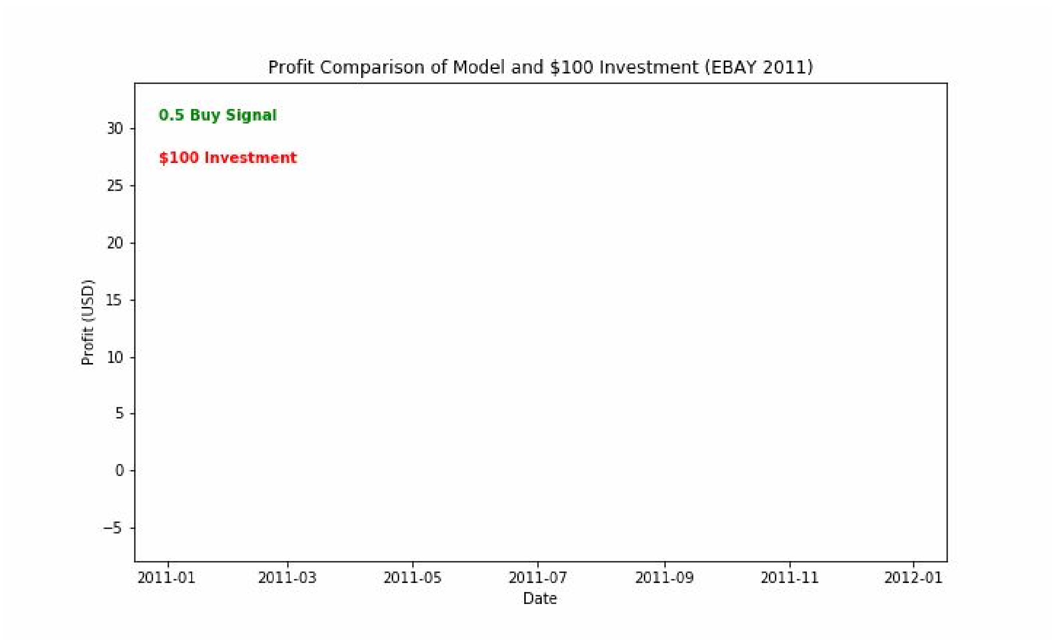
Backtesting is the act of testing a model's performance on past market data. This step can help catch bugs and inefficiencies in your model, and should never be skipped. I used three years of market data to backtest my models. I created a backtesting function that invested two sets of $10,000 into 100 stocks. The first set was invested equally into each stock, and left until the end of the test year. The second set relied on a model's predictions. A stock was bought when the model predicted that the stock would go up, and the stock was sold when the model predicted that the stock's price would go down. The backtesting function kept track of the profits made for both sets, and recorded the results.

Results during 2008, 2011, 2013 (w/o Market Model)

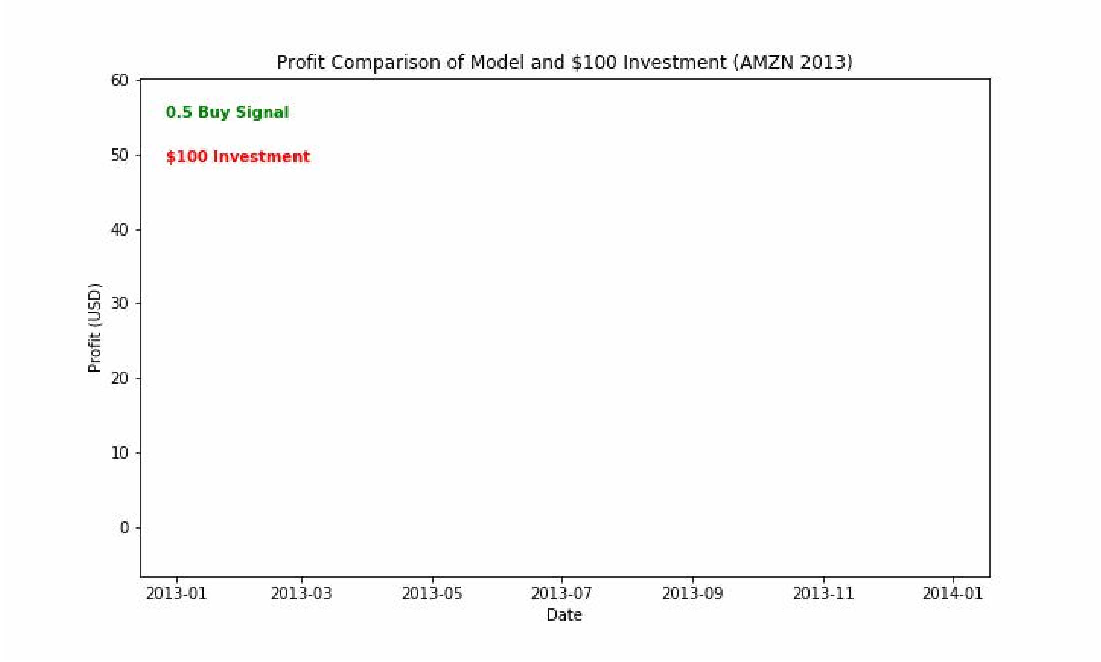
To visualize the results, I made animated plots of profit over time for each year. The model used for these plots is the Deep Learning model with no market features. In 2008, Google stock plummeted. If you invested $100 into GOOG, you would have roughly $40 remaining at the end of the year. However, the model avoided most of the decline, and actually made a slight profit when trading Google stock in the year 2008.



In 2011, the stock market was volatile. This can be seen in the EBAY stock, with many ups and downs throughout the year. The model performs similarly for most of the year, but turns a profit near the end.



In 2013, the market soared. Many stocks made a profit, including Amazon stock. In 2013, the model cannot beat a $100 investment into AMZN stock, but it does make a profit as well.



2008 Backtesting: Model Comparison ($10,000 invested)

For the year 2008, $10,000 turns into $7,800 when simply invested. Despite the recession, all three models make a profit. XGBoost performs the best. Note that all three models make more unprofitable trades than profitable trades.

	w/ Market Features	w/o Market Features	XGBoost
Investment Performance	$7,799.75		
Model Performance	$10,395.47	$10,242.37	$10,640.04
Profit	$12,548.55	$10,613.46	$11,756.63
Loss	$12,153.08	$10,371.09	$11,116.58
Profitable Trades	3,979	3,521	3,644
Unprofitable Trades	4,130	3,682	3,981

2011 Backtesting: Model Comparison ($10,000 invested)

Investing $10,000 in 2011 also leads to a net loss. Again, trading with the three models leads to profits. This time, the neural network model without market features performs the best. The model using market features makes the most base profit, but makes more unprofitable trades.

	w/ Market Features	w/o Market Features	XGBoost
Investment Performance	$9,443.43		
Model Performance	$10,367.26	$10,590.19	$10,277.79
Profit	$8,113.20	$7,246.60	$7,191.98
Loss	$7745.94	$6656.41	$6,914.19
Profitable Trades	4,676	4,230	4,192
Unprofitable Trades	4,476	4,102	4,174

2013 Backtesting: Model Comparison ($10,000 invested)

For 2013, an investment of $10,000 becomes $13,000 at the end of the year. No model outperforms this investment. The best performing model is again the neural network model without market features.

	w/ Market Features	w/o Market Features	XGBoost
Investment Performance	$13,198.35		
Model Performance	$11,190.33	$11,972.52	$11,361.44
Profit	$6,016.23	$6,938.99	$6,421.43
Loss	$4,825.90	$4,966.47	$5,059.99
Profitable Trades	4,429	5,149	4,874
Unprofitable Trades	4,132	4,551	4,365

Chasing the wrong goal: Loss vs Profit

What happened? Why did the best model according to machine learning metrics perform the worst when backtesting? It took me a long time to reconcile this seeming contradiction. After reflecting on it, I came to an understanding of the problem. The model with market features is not predicting whether an individual stock will go up or down. It is predicting primarily whether the market will go up or down. Thus, when the market goes up, it buys nearly every stock. THis can lead to higher losses when stocks go against the market. Additionally, it misses stocks that go up when the market is trending downward. Despite having a higher prediction accuracy, it makes a higher number of unprofitable trades.

Automated Trading

Using the Alpaca API, you can buy and sell stocks using code. I created a simple script with a timed loop. It first pulls the most recent data for the stock symbols you wish to trade on. It then computes the features needed for the model. The model then predicts whether the stock will go up or down based on the features. Finally, it buys or sells the stock based on the model's prediction.

Trying Your Luck on the Paper Markets

Even with good backtesting results, you should never trade on the real markets without first trading with paper money. Alpaca provides a paper account that lets you buy and sell stocks with any amount of money. Backtesting will give you idealized results. All stocks are bought at market price and sold at market price. In reality, market prices shift constantly. What may be a good trade in one moment may turn into an unprofitable trade in the next. By testing in a safe but real environment, you can iron out any bugs in your trading algorithm.

Options for Automation

My code must be run manually. A better idea would be to run it automatically, without any required input. To get this to work, you may need a cloud service such as Google Cloud Compute. I've gotten it to work myself, but the memory requirements for the technical indicators pushed me out of the free tier zone. If you are willing to accept that cost, then you can look **here** for more details.

Possible Improvements

I have only scratched the surface of the stock market features when creating my models. There are many areas where improvements could be made in all stages of the pipeline. More years of data could be acquired. More technical indicators could be included in the feature generation stage. Backtesting could include different thresholds for buying and selling stocks. Stocks could be bought and sold at higher or lower amounts based on the model's confidence level. All of these improvements are just a few of the ideas I plan on implementing on my own time.

Conclusions

I learned a lot while working on this Capstone project. My advice for anyone thinking of trading would be: don't. While my models outperformed the market in specific circumstances, almost every model was beat by the market in the long run. Investing is always a sound strategy, but any attempt at trading stocks should be seen as gambling. Stocks are influenced by so much more than the sum of their past prices. However, the analysis of prices can be used to gain a better understanding of past events and their impact on individual companies and the market as a whole.