Yale University
Department of Computer Science

Short-Term Technical and Sentiment based Stock Trading Strategies

Emmad Arif
Advisor: James Glenn

May 2, 2019
Contents

1 Introduction

2 Problem Formulation

3 Technical Analysis
   3.1 Why Technical Analysis?
   3.2 Data Selection and Collection
   3.3 Indicators
   3.4 Strategies
      3.4.1 Simple Moving Average Crossover
      3.4.2 Exponential Moving Average Crossover
      3.4.3 Simple Moving Average Convergence Divergence
      3.4.4 Relative Strength Index
      3.4.5 Linear Regression
      3.4.6 Random Forest

4 Sentiment Analysis
   4.1 Why Sentiment Analysis?
   4.2 Methodology

5 Results
   5.1 Suggested Usage
   5.2 Examples
      5.2.1 Microsoft
      5.2.2 General Electric
      5.2.3 Tesla

6 Code Documentation
   6.1 Software Requirements
   6.2 Back Test Script
   6.3 Advice Script
   6.4 Sentiment Analysis Scripts

7 Limitations and Future Work

8 Conclusion
1 Introduction

According to the 2017 Federal Reserve Bulletin, nearly half of American households do not own any stocks.¹ Stock ownership is even lower among low-income households; for families earning below the median income, this figure drops down to 30%. This project provides an advisory framework to assist persons with no prior financial knowledge to trade stocks listed on the New York Stock Exchange (NYSE).

2 Problem Formulation

To simplify the problem, we frame the question as follows: given a stock symbol and its price/volume data, will the price increase/decrease by at least a certain percentage, within a certain time period. The two variables i.e. minimum percent change and time horizon are configurable by the user.

3 Technical Analysis

3.1 Why Technical Analysis?

Technical analysis refers to the utilization of past price and volume data, to predict future movements. The goal is to identify repeating patterns and trends, and use that information to make smart trading decisions. Since individual stocks exhibit certain behaviors over and over again, it makes sense to use historical price/volume data as a starting point for our analysis. In this project, I utilized four pure technical strategies, and two basic supervised learning strategies, as outlined in section 3.4.

I discovered that while these strategies are certainly useful on their own, the accuracy of predictions increases dramatically upon combining more than one strategy, as we shall see below.

3.2 Data Selection and Collection

For this project, I primarily focused on two sources of data:

1. daily close price ($): last quoted price that day,
2. daily volume ($): total amount of shares traded that day.

The decision to use the daily close price, as opposed to the opening/average price was made primarily for convenience. The intended usage is for the user to run the scripts after the markets close every day, and make a decision about the next day’s trades accordingly; the close price lets us do that cleanly.

To access this data, I used an API provided by Alpha Vantage, which provides both real
time and historical pricing/volume data. To run advice and back test scripts, the user must
obtain an API key (see Section 6.1: Software Requirements).

For the most part, the API returns clean data. There were a few minor issues, for instance,
some companies conduct stock splits (issue new shares to existing shareholders based on
their current holding), and reverse stock splits. These events dramatically alter the stock
price, but the market capitalization remains constant, so the shareholder neither gains nor
loses any value. In order to minimize the impact of these uncommon incidents, I chose to
utilize data following 01/01/2015 only (for back testing). If, however, we missed some
unforeseen incidents, and the stock price does change dramatically overnight (i.e. > 10%
movement), both the advice and back test scripts alert the user, so that they may
investigate further.

3.3 Indicators

An indicator refers to a secondary metric derived from primary data (price/volume), to
summarize trends/patterns. Most indicators are based on certain time frames, for example n
day moving averages, which can be tweaked by the user in config.py (see Section 6.1:
Software Requirements). A short summary of the indicators used in my strategies is
provided below.

Implementations of all indicators listed below can be found in the /scripts/indicators
directory.

Simple Moving Average (n-day): Mean close price of the previous n days. It is
calculated as:

\[ SMA_{today} = \frac{1}{n} \sum_{i=0}^{n-1} price_i \]

where \( i = 0 \) is today, \( i = 1 \) is yesterday (and so on), and \( price_i \) is the close price on day \( i \).

Exponential Moving Average (n-day): Exponentially weighted mean close price of the
previous n days. It is calculated in three steps.

Step 1: Calculate the SMA for the \( i - n^{th} \) day. This is your initial EMA.

Step 2: Calculate \( \alpha \), the weighting multiplier, using the following formula:

\[ \alpha = \frac{2}{n + 1} \]

Step 3: Now for each successive day between \( i - n \) and today (inclusive), calculate the EMA
as:

\[ EMA_{today} = \alpha(price_{today} - EMA_{yesterday}) + EMA_{yesterday} \]
Relative Strength Index (n-day): A bounded, momentum index, calculated as follows:
\[
RSI_{\text{today}} = 100 - \left( \frac{100}{1 + \text{Relative Strength}} \right)
\]
where Relative Strength is defined as follows:
\[
\text{Relative Strength} = \frac{\text{Average Gain}}{\text{Average Loss}}
\]
and Average Gain = average percentage gain on the days the stock price increased, over the past n days, and vice versa for Average Loss.

3.4 Strategies

Below, you will find descriptions of six technical strategies, each of which uses different combinations of the indicator(s) outlined above.

In this document, long refers to a longer time period, and short refers to a shorter time period, i.e. indicator_1 (long) and indicator_1 (short) are the same indicators, applied under different time frames, for instance, long could be 100 days and short could be 30 days.

The back test tool also compares each strategy to a random buy/sell strategy, and a buy and hold strategy, in order to better gauge its success.

Implementations of all strategies listed below can be found in the /strategies directory.

Note: These strategies are primarily used for buy signals only. There are two safety mechanisms built into the exchange, which may execute a sell trade before the strategy issues a sell signal. These two cases are stop loss sells (to minimize losses), and take profit sells (to take profits once targets are met), and are elaborated in Section 5.1.

3.4.1 Simple Moving Average Crossover

Indicators used: Simple Moving Average (n day).

In order to predict whether the stock price will rise or fall in the short run, it is useful to first establish a baseline price. If the price breaks above the baseline, it is reasonable to assume there is an upward momentum, and interpret that as a buy signal. On the other hand, a break below the baseline points toward a downward trend. This strategy uses the output from Simple Moving Average (SMA) as the baseline price.

Running this strategy with a time horizon of 10 days on Tesla yields the following results.
The strategy works quite well on Tesla stock, returning 100.13% of the trader’s initial investment ($1,000,000 in this case – configurable in config.py) over a period of 4.28 years, compared to a return of -0.23% by the random strategy, and 22.55% by the buy and hold strategy.

It should, however, be noted that I had prior knowledge of Tesla being a volatile stock. As a result, I established the baseline price at a relatively short time frame i.e. 10 days, in order to minimize lag. A less volatile stock, such as Microsoft, for instance, would give better returns with a longer time frame, since the baseline changes slowly.

Since the user may not always have complete knowledge of the volatility of their chosen stock, a solution to this problem is provided in the following strategy.

### 3.4.2 Exponential Moving Average Crossover

Indicators used: Exponential Moving Average (n day).

The EMA Crossover is a strategy very similar to the SMA Crossover, except in this case, when calculating the baseline price, we assign greater weights to recent prices. So, for instance, while the SMA indicator would assign the same weight to the close price on 04/03/2019 that it would to 04/10/2019’s close price, the EMA indicator would assign a greater weight to the latter, since that accounts for potential recent volatility.

This strategy slightly outperforms SMA Crossover for a 10-day time horizon, returning 108.03% over a period of 4.27 years, compared to the 100.13% returned by SMA Crossover.

### 3.4.3 Simple Moving Average Convergence Divergence (MACD or SMALSC)

Indicators used: Simple Moving Average (m day), Simple Moving Average (n day), where m > n.

Note: This terms MACD, and SMALSC (Simple Moving Average Long Short Convergence) are used interchangeably in this document and the code.
This strategy is similar to SMA Crossover, except in this case, we issue a buy signal when a shorter term SMA indicator breaks above the longer term SMA indicator. The intuition behind is that the longer Simple Moving Average represents the baseline, and when the shorter Simple Moving Average breaks above it, there is a positive momentum indication. This is a slightly less risky strategy than SMA Crossover, since we are relying on the average price of the past n-days instead of a single price point.

Below, you will find the output of the back test tool executing this strategy with the -plotIndicators command line argument, which plots all indicators used in a particular strategy. Whenever the 10 day SMA (yellow line) breaks above the 100 day SMA (black line), a buy signal is issued. As the graph shows, the strategy nicely captures the ride up Microsoft over the years, and avoids some losses, such as the drip at the end of 2018.
3.4.4 Relative Strength Index

Indicators used: RSI (n day – usually 14 days).

The RSI is probably one of the most accurate strategies implemented. Based on the heuristic chosen, a buy signal is issued whenever the RSI value breaks below 30 (indicating the stock is oversold, and a sell signal is issued when the RSI value breaks above 70 (overbought). For a more conservative portfolio, you may want to use values of 20/80 instead, although you will not get as frequent buy/sell signals.

![RSI Backtest Output]

Figure: RSI 14 strategy back test on MSFT stock with -display command line argument (which displays all trades made). Image cropped for space.

As the image shows, this strategy underperformed both the random and Buy and Hold strategies on MSFT stock. This is not surprising. The heuristic I’ve implemented is very rigorous, and only issues signals when it is extremely confident. As you shall see in Section 5.2.2, this strategy performs very well with underperforming stocks such as GE, and is a good tool to use when markets are down. Feel free to experiment by raising sell thresholds from 0.3 to 0.4 in /strategies/RSI.py, which will issue more frequent (but less accurate) buy signals.

3.4.5 Linear Regression

One of the most basic supervised learning techniques, linear regression performs reasonably well for stable stocks. The default implementation uses the close price of the past 5 days as predictors, but you can change this in /config.py. Below, you will find output from
advice.py, which runs linear regression, and the other six strategies on the last available price, and gives predictions.

Figure: advice.py output for AAPL

For Apple stock, linear regression has a decent 70% historical accuracy for upward predictions, but the accuracy for downward predictions is an unimpressive 50%. Random Forest performs better for downward predictions. As the other indicators are either uncertain, or give conflicting predictions, perhaps it would be unwise to open a position in Apple stock now.

3.4.6 Random Forest

Since we are treating this as a classification problem with a binary output, a random forest estimator is a reasonable tool to use. Since random forests aggregate several decision trees, overfitting is not a huge issue, which is useful for our situation, since individual extreme price movements (such as a flash sale), might impact the accuracy of other strategies, especially linear regression.
The default implementation uses 1000 estimators, and uses the close price of the past 30 days as features for the model.

4 Sentiment Analysis

4.1 Why Sentiment Analysis?

Imagine you are a stock market novice, and you would like to purchase GM stock, since you like their vehicles. Where would you start your research?

It seems logical to go through the business section of the New York Times, or search through Twitter and see what people have to say about the company and investing in it. This is exactly why we decided to incorporate sentiment analysis of this data into our model. Its aim was to account for the human aspects of trading stock, since technical/statistical analysis often lags behind the latest news related to the company.

4.2 Methodology

Twitter seemed like a reasonable source of data, to be able to gauge public sentiment about the stock and company. For instance, a single tweet by Elon Musk in August 2018, tumbled Tesla’s shares by almost 10% within a day. Therefore, Twitter, and social media at large, evidently plays a role in influencing the stock market. To analyze tweets, I chose certain keywords to investigate. For instance, to relate Tesla’s stock price, I looked for tweets containing the words “TSLA” and “stock”, using the Tweepy API.

A second, more reliable source I used was official news outlets. I looked at New York Times articles’ headlines and abstracts (using the NYT API), as well as Google Search results (using web scraping).

Sentiment analysis was then performed on this data using the Textblob package, which returned a subjectivity measure, and polarity measure. Subjectivity refers to the whether the tweet is based on facts or opinion, while polarity is a measure of positivity/negativity of the tweet.

Unfortunately, I was unable to establish a correlation between stock prices and the data collected. This result should not come as a surprise; the sheer amount of data available online makes it very difficult to sift through it, and discern noise from actual analysis.

For instance, the Twitter accounts most likely to tweet about an arbitrary stock, say Apple, are also likely to post about other stocks, say Microsoft, often within the same tweet. Then, it is very difficult for a machine to decide which stock the tweet is positive about, and which tweet it is negative about. As the figure below shows, our program classified the following tweet as 0.25% positive (polarity = 0.25) for Apple, however the tweeter was talking about Microsoft outperforming Apple.
Figures: A sample ambiguous tweet related to Apple stock.

Similar problems arose with news articles i.e. articles mentioning Tesla were very likely to mention Ford as well.

No correlation was observed between NYT headlines and stock prices either.

Figure: A plot of NYT headlines related to Tesla and Tesla’s corresponding stock price.

However, tools have been provided to the user to use all three of these data sources, and use the output from sentiment analysis as they see fit. While this analysis may not be useful for individual stocks, it might be useful to gain an understanding of the broader market, and then apply an offset to trading positions accordingly. For details on how to use these tools, please see Section 6.4: Sentiment Analysis Scripts.

5 Results

5.1 Suggested Usage

A safety mechanism is built into the trading exchange. It will automatically execute a “stop loss sell”, i.e. your position will be closed if the price falls below 4 times PERCENT_CHANGE (/config.py), in order to minimize losses. This sale is executed in the function stopLossSell() in /tradingUtilities.py. Similarly, a takeProfitsSell() is executed if your desired PERCENT_CHANGE is achieved, in order to take profits and close your position. Feel free to tweak this value to suit your risk profile. It is recommended, for example, to make your stop loss sells less sensitive when trading less volatile stocks such as Microsoft, as opposed to more volatile stocks such as Tesla. The rationale behind this is
that Microsoft is more likely to rebound, and prematurely closing a position to minimize losses is not a smart move.

5.2 Examples

5.2.1 Microsoft

The advice.py tool uses the six strategies outlined above, and gives you a future prediction based on your preferred time horizon and desired percent gain (configurable in /config.py). See below, the output of this tool on Microsoft stock.

![Python code output]

Figure: Results of /advice.py on Microsoft stock on April 29, 2019.

5.2.2 General Electric

GE stock is an excellent demonstration of the strength of these tools. This is a stock that has been underperforming severely, falling 59% over the past four years. However, the RSI 14 strategy not only protected us from these losses, but actually managed to make a 0.26% profit, as the following results show.
Figure: Results of RSI-14 back test on GE. Each trade made is displayed.

If you use the -plotBacktests command line argument, you can visualize all the buys and sells executed by backtest.py, as shown in the following plot.

Figure: RSI-14 back test on GE. Vertical green lines indicate a buy and red lines indicate a sell. Note how we managed to make small profitable trades, and avoid the massive fall.
5.2.3 General Motors

Not all strategies are as successful on every individual stocks, however. The 50/10 day MACD strategy (aka SMALSC) has a lower return rate on GM stock, compared to a buy and hold strategy.

Figure: MACD/SMALSC strategy on GM. It does not outperform the Buy and Hold strategy.

An argument could be made that this strategy has a lower risk than the Buy and Hold strategy, and thus the returns are lower. You can experiment with your preferred time horizons, desired percentage gain etc. in /config.py.

Using the -plotBacktests command line argument, you can visualize the indicators used in each strategy. For the above mentioned strategy, we used the two Simple Moving Averages (50 day and 20 day), and the following plot shows the points where they intersect, and a trade is executed.

Figure: Indicators used for the 50/10 day SMALSC strategy on GM stock.
6 Code Documentation

6.1 Software Requirements

1. Obtain an API key from https://www.alphavantage.co/support/#api-key, and store it in /config.py under the variable “ALPHA_API_KEY”.

2. Obtain API keys from https://developer.twitter.com/en/docs.html, and store them in /config.py under the variables TWITTER_CONSUMER_KEY, TWITTER_CONSUMER_SECRET, TWITTER_ACCESS_TOKEN, TWITTER_ACCESS_TOKEN_SECRET.

3. Obtain an API key from https://developer.nytimes.com/apis, and store it in /config.py under the variable “NYT_API_KEY”.

4. Make sure you run the both scripts (backtest.py and advice.py) with python 3.x. Make sure you have the following dependencies installed: textblob, json, unicodedata, urllib, datetime, tweepy, requests, googlesearch, pycurl, certify, io, os, sys, bs4, pandas, math, random, matplotlib.pyplot, matplotlib.dates, pandas.plotting, mpl_finance. Use “pip install [dependency]” to install them.

5. Please clone all directories and sub-directories e.g. /data, as is, even if they’re empty, to ensure smooth execution.

6.2 Back Test Script (/backtest.py)

This script will pull a stock’s price/volume data, and back test strategies for you. Please see /backtestHelp.py for instructions on how to run this script. Make sure to use the -forcePull command line argument to pull fresh stock data from the API.

6.3 Advice Scripts (/advice.py)

This script will pull a stock, and give you future price predictions based on six strategies. It will also compare returns to an average of 10 runs of the random strategy (buy and sell each day, randomly), and buy and hold strategy. Please see /adviceHelp.py for instructions on how to run this script. Make sure to use the -forcePull command line argument to pull fresh stock data from the API.

6.4 Sentiment Analysis Script

Three scripts (twitterSentiment.py, nytSentiment.py, googleSentiment.py) are provided. These scripts will pull recent data from Twitter, New York Times and Google Search, and run sentiment analysis on them. Run them as “python twitterSentiment.py –[ticker]”, and similarly for the other two. Make sure to set date parameters for NYT data in /config.py.
7 Limitations and Future Work

The methods outlined above work reasonably well when used in conjunction with each other, however it would be unwise to blindly rely on these tools, without putting any safety mechanisms in check. This is primarily because of three limitations.

Firstly, the responsiveness of these tools is not very comprehensive. They are designed to be run every night, and provide advice/predictions based on the day’s close price. They do not capture intra-day movement; for instance, if stock A opens at $100, rises to $180 at noon, and then drops down to $90 at the end of day, our tools are unaware of the midday price movement, which could potentially be valuable. A future version of the project could account for limitation, and provide a more comprehensive model.

Secondly, there is no guarantee that the strategies will give mutually agreeable results. It is entirely possible, and for some volatile stocks even likely, that the SMA Crossover is giving a buy signal, while MACD is giving a sell signal. In cases such as this, it is best to not open/close a position, which may not be ideal. This is a limitation of the strategies themselves, and not so much our implementation, so it is hard to find a workaround. A deeper analysis of the company’s financials, debt status, broader market health etc. would be necessary to make predictions, which are quite subjective, and a hard problem to formulate and automate.

Thirdly, as we discovered in Section 4, sentiment analysis of the sources we used (Twitter, The New York Times and Google Search), wasn’t particularly helpful; the sentiment values showed little correlation with stock price, and while they might be useful in conjunction with technical analysis, on their own, they are not a good predictor of future stock price. This is partly because of data limitations. Not only was it difficult to clean data, for instance distinguishing the company Apple from the fruit apple in search results, but a lot of data sources limit access to past results. The Twitter API for instance, limits you to accessing only a certain set of recent tweets. Perhaps a more robust tool, which utilizes a paid search service would be more helpful, but for the beginning trader, a holistic sentiment value quantifying recent company news should be good enough.

8 Conclusion

Overall, the project has been very useful in helping me polish my software development skills. The one thing it made me realize is how difficult it is to get clean, non-quantitative data, such as the data required for sentiment analysis. As outlined above, this part of the project did not yield results useful enough to trade based on them alone.

However, the tools created should be very useful for beginning traders. They provide users with a starting point to make their trades in a relatively low-risk environment, considering
the safety mechanisms put in place. Meanwhile, the templates created for the strategies are very portable (see /strategies), and the user is encouraged to tweak variables such as time horizons, desired percent gains etc., design their own strategies, and use the back test tool to gauge their accuracy.