Stock Market Prediction Using Polynomial Regression

CS 658 – FInal Project – may 16

Sean Kramer

2019

Table of Contents

[Introduction 2](#_Toc8851011)

[Environment 2](#_Toc8851012)

[Methodology 2](#_Toc8851013)

[Data Sourcing 3](#_Toc8851014)

[Data Manipulation 3](#_Toc8851015)

[Model Determination 4](#_Toc8851016)

[Prediction Results 4](#_Toc8851017)

[Model Applications 5](#_Toc8851018)

[Active AAPL Portfolio Management 5](#_Toc8851019)

[Portfolio Management Pitfalls and Fixes 6](#_Toc8851020)

[Conclusion and Future Considerations 6](#_Toc8851021)

# Introduction

The idea for this project came from a long-standing interest of mine in the US stock market. I’ve never tried to ‘day trade’, that is buy and sell stocks throughout the day to make a profit. However, when the final project for this class was announced I felt I could apply machine learning principles to the age-old question of, “how do you beat the stock market?”

I decided to make this project two-phase, with the initial phase being prediction of stock market values. I am choosing to define the price of a stock as the closing price for that stock at any given point in time. So, phase one of this project was to build a model which given a set of input parameters would accurately produce an estimate of a stock’s value. The second phase of this project was to use that model, alongside intra-minute data from the Alpaca API, to determine whether to buy, sell, or hold positions in AAPL stock. I intended to apply this to real money, using penny stocks, but unfortunately there was not enough penny stock data available at the time of this project.

The final note for this introduction is that the project overall was a success, however there are portions of it I would like to change in future revisions. And despite running into obstacles along the way, the result was a fully functional phase one, and a functional, but not ideal phase two.

# Environment

For all previous homework in this class I used Jupyter notebook to develop, test, and run my models. Per the suggestion of Isaac from class, I decided to give Google Colaboratory a try with this assignment. I opted to use Google’s free GPU service so that my model would train quickly, and due to that am unaware of what hardware is being used to run my code.

For development purposes I worked both on my laptop and desktop computer. My laptop is a MacBook Pro (Late 2013) running macOS Mojave (v10.14.4). It has a 2.3 Ghz Intel Core i7, 16GB of 1600 MHz DDR3 RAM, and an NVIDIA GeForce GT 750M 2 GB alongside an Intel Iris Pro 1536 MB. The NVIDIA chip was unknown to me prior to this project and did not appear in the system information until this week, likely due to an update. My desktop is a custom build running Windows 10 Home (v10.0.17134), an Intel Core i7-5820K, 32GB of 2133 MHz DDR4 RAM, and a NVIDIA GeForce GTX 1070 8 GB.

For software I am running Python 3.6.7 with all the pre-installed libraries and revisions of Google Colaboratory. I was unable to determine exactly which versions of every library were being used by default, as Google Colaboratory is still relatively new and under development. I am also using a python package called apscheduler, and a stock API called Alpaca. I am using these packages to facilitate background process scheduling and pulling real time stock data alongside making buy and sell orders to a simulator based on the real market

# Methodology

In order to successfully carry out this project I began by writing out a basic roadmap of what I needed and what I wanted to get accomplished. Per Professor Tu’s recommendation I needed intra-minute data for a stock, or stocks, and a means of compiling that data into a usable dataset for my model. Following data, I needed to understand which portion of the stock I was trying to predict, and I decided upon the closing value, as that is the value traditionally reported as a stock’s value. Next, I needed a way in which I could interact with the stock market in real time, and make purchases and sales with fake money, and potentially real money. To accomplish this, I decided upon a RESTful API service known as Alpaca. They provide a host of abilities within their API, including live market trading, and intra-minute data. Finally, I needed to identify a way to combine the model, it’s predictions, and real-time to decide whether to buy, sell, or hold my stock options. To do this I used apscheduler, which allowed me to schedule ‘chron’ jobs that would trigger a function every minute.

# Data Sourcing

I used multiple data sources to accomplish my goals for this project. It was very difficult to find historical intra-minute data for any stocks, and most websites which offered it were based upon a pay-for-data model. After much searching and scouring both US and Russian data sources I discovered First Rate Data, a company which offered free intra-minute data for select stocks as far back as 2004.

The only problem was that their data only went up until March of 2019, so there was a gap of approximately a month between my historical data and current data. I tried to find another source for this data but was unsuccessful. I was able to get daily data rates using AlphaVantage, but when passed into the model it didn’t comprehend the sudden loss of data points well and started predicting stock prices up into the $8000 range for a stock that is only around $200.

# Data Manipulation

The data source that I chose offered pre-cleaned data, so my historical stock price data was incredibly useful to me immediately after download. It came in the form of 31 CSV files, each containing six month’s worth of intra-minute daily stock values. The columns of the data set were time, open value, close value, low value, high value, trade volume, and weighted average. I chose to remove the weighted average and convert the datetime value into an integer so that the model could use it as an input. The data was loaded from Google Drive directly, and then split into feature groups and stock data groups. The feature group consisted of the date, open, high, low, and volume columns, while the stock data group consisted of close. These would become my X\_train, X\_test, y\_train, and y\_test.

For my polynomial regression model to use the data I had just received, the feature sets needed to be transformed into polynomial representations. I accomplished this using sklearn’s PolynomialFeatures function, with a degree of two. I found that any higher degree resulted in worse validation data performance.

For receiving current price data, I had to manipulate the Alpaca API’s output format. They use an object called a barset, which is essentially a dictionary. Through some minor manipulation I was able to extract all the information and feed it into a transposed numpy array for the model to give me its predicted close value. The code for this portion is very messy, but the amount of nested connections within barsets and dictionaries made it basically unavoidable.

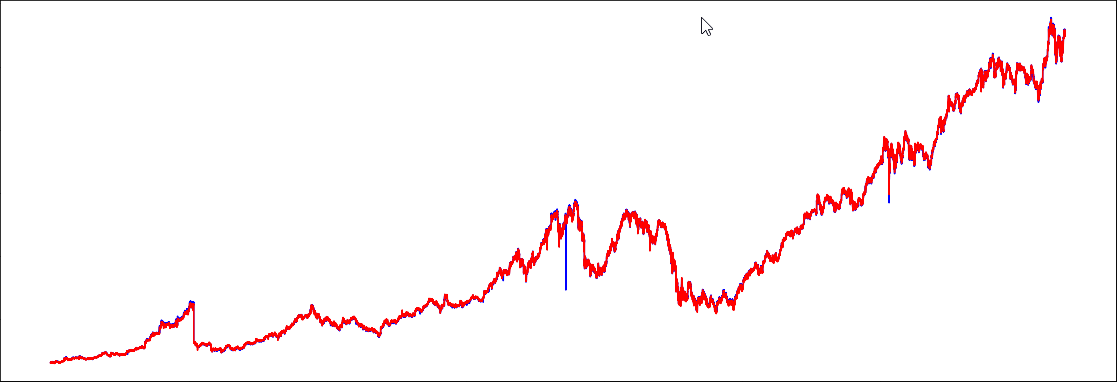
# Model Determination

I decided to use a polynomial regression model after failing to use a traditional MLP. I did a lot of research on forums such as StackOverflow and found that for my needs (predicting a number based on other numbers) it was more than likely that a regression problem of some sort would fit my needs. I tried to do a linear regression but very quickly realized that would in no way shape or form work for the bounciness of the stock market. So, with a little digging I happened upon the polynomial regression and quickly locked into it. The model runs very quickly, within 15 to 20 seconds, even for my massive dataset (1 million+ data points).

# Prediction Results

The model boasts a rather impressive 99% training data accuracy, and a 96% validation data accuracy. In real-time use it sits at approximately 92% accuracy, and as the market moves with more rapid swings it seems to fall below it. However, it still represents the swings of the market well, just in more subdued terms.

**TRAINING DATA: MODEL PREDICTIONS – ACTUAL STOCK MARKET**



**VALIDATION DATA: MODEL PREDICTIONS – ACTUAL STOCK MARKET**

**A close up of a logo

Description automatically generated**

As you can see the model is very good at mimicking the stock market with testing data, which makes it OK that it is not a perfect estimate. That is because the major portion of the model that I want to use to deploy phase two of the project is estimating if the stock price will raise, lower, or hold steady.

# Model Applications

I think this model has a lot of applications. The first being the most obvious, to predict stock market prices in real-time. However, beyond that I think it could be used to predict almost any series which has the end goal of obtaining a numerical estimate from numerical features. I don’t think it would be good for categorical predictions, but this could be used to estimate gas prices, housing market swings, pretty much any market-esque data source.

# Active AAPL Portfolio Management

Phase two of this project was to implement the previously discussed model in such a way that it would ‘day trade’ a stock for me. I chose to use Apple stock (AAPL) since it had intra-minute historical data back to 2004, where as a lot of other stocks which are more volatile (read: better suited to day trade) such as Tesla and NVIDIA did not have that historical data.

I’m going to begin by saying that day trading AAPL is not a good idea. It is not a stock that is made for this type of application, and it is not a stock which has high volatility. In the one day which I tried running this program it failed to work until about noon, and then through the course of the afternoon it made a mere 0.04% profit, which if I were to have to pay commissions would be a net loss.

**AAPL STOCK PRICE – 5/15/19**

**A close up of a map

Description automatically generated**

**ACTIVE PORTFOLIO MANAGEMENT – 5/15/19**

A close up of a logo

Description automatically generated

As you can see the AAPL stock price rose over the course of the day, in fact it rose by 1.2%, however the portfolio management ended nearly where it began, rising a mere 0.04%.

The portfolio I used was through Alpaca, and it began the day with an equity of $100,000. The way my application works is it stores a history of its predictions, and after getting the minute data from Alpaca at the 59 seconds mark (so just before the minute ends) it spits out its prediction for the next minute. That prediction is taken and depending on whether it is higher or lower than the previous stock price, the algorithm determines if it should by, sell, or hold the account’s current position.

There are measures in place to ensure that the account does not result in a negative position, and it always retains a 10% buffer of initial equity so that the account holder will only ever lose 90% of their position, WORST CASE scenario.

# Portfolio Management Pitfalls and Fixes

There are a few pitfalls I encountered during this program. The first being that due to the model’s inaccuracy when it came to the exact prices and giving more of a general it will go up or down result, it was difficult to determine whether to buy or sell. I would like to make the model more accurate, getting as close to 99% as possible.

Another issue I’d like to address is the format of the data, currently it is set to predict close price as if the data it has been passed is for the current minute. I would like to implement one minute ahead future forecasting so that it can better predict the actual future value and determine if holding is a good idea, because as it stands now it can only really see one minute or into the market. This leads into my next point, that the model does not have consistent data recently, which I believe is part of why it has a difficult time determining exact prices, but still mimics market movement well.

To fill this data void, I would like to continuously capture stock price data throughout the day and build my own dataset to fill the voids in mine.

# Conclusion and Future Considerations

This project was an incredible learning experience for me. It introduced me to time-series, forecasting, polynomial regression, and a whole host of python libraries and methodologies that I had not been able to experience before.

I really enjoyed being able to work with newer technology, and it was extremely satisfying seeing the first couple graphs come to life. Although I am disappointed in my portfolio managements performance, I believe the heart of it is still solid. The model is going to be continued to be worked on by me, and I plan to post this project to GitHub so that others can fork it and see what they can get out of it.

One day I would like this program to reach a point where I can throw $50 into it and come back a few months later to a couple hundred dollars, that would be amazing. Stock market predicting is no joke and trying to organize all the data for it, especially in a Google Colaboratory notebook is incredibly difficult. Overall the project was a success in my eyes, and I am very glad that I took this class. Thank you for being our teacher, and I look forward to using what I’ve learned in this class in my future places of work and side projects!