Daniel Bergstein
Sam Lesser
Bradley Turcios

# Tesla's Tweets

We analyzed the connection between Elon Musk's tweets and the price of Tesla's stock. In order to accomplish this, we needed Elon Musk's Tweets as well as historical Tesla stock data. Obtaining both of these data sets was far more involved than initially expected.

With regards to obtaining the Twitter data, we initially considered two options: build a web scraper or utilize Twitter's developer API. We initially decided on using the developer API, which definitely required a learning curve to master, as there was no substantial documentation outside of the official Twitter documentation (note that we did all of this before the text-mining lecture in which Twitter was mentioned). This required forming a developer account, submitting an application, and securing access to the proper API's. After some initial testing, everything stopped working. We had expired our monthly credit on just a few queries. Because of the limitations and quotas of the free developer account, we attempted to switch over to the web-scraping option. We constructed a basic web scraper. However, this plan soon failed to. In order to access a profile on Twitter's website, one must be logged in. We adjusted our code to first sign into Twitter and then start scraping, but this also did not work. We were able to login, but Twitter kept freezing our account for violating their rules. We then returned back to our initial plan of using Twitter's developer API. We made four separate accounts and sent four different applications in order to secure access to Twitter's Full Archive API. By using these four accounts to secure tweets from different time periods, we were able to obtain all ~9 thousand of Elon Musk's tweets along with some meta-data on each tweet.

With regards to obtaining Tesla stock data, we also explored a few options. In order to perform meaningful analysis, we needed price data in small increments (per minute or hour). However, all of the free resources only provided historical stock data on a per day basis. It would not have been helpful to just have the opening and closing price of the Tesla stock. After consulting with the business school librarian, we discovered Bloomberg Terminal. Although it took a little while to learn how to use, we were able to obtain the stock price during every minute of Tesla's stock market existence (~9 years).

Once we obtained both of the above data sets, we thought we were ready to start analyzing the data. However, we soon realized that the data sets were on different time scales. This was a huge problem as our data analysis directly relied on aligning the timing of the two data sets. The twitter data was all in UTC, but the stock data was in Eastern time and was adjusted for daylight savings. This introduced a difficult problem because we would have to do more than just offset the stock data. We needed to account for the fact that the stock data was really in two time zones depending on the date: Eastern and Eastern DST. We found a python library that helped us address this issue, but it also took time to master and the results required a lot of testing as this was crucial to the success of our data analysis.

After finally obtaining and adjusting all of the data, we were ready to begin the analysis. We spent a long time figuring out how to best represent a shift in the stock. After some online research, we decided to use a moving average to accomplish this. We observed the average movement (per minute) of the stock in the 30 minutes before a tweet and compared that to

the average movement of the stock during the 30 minutes after a tweet. We then subtracted the two and took the absolute value. A large difference indicates a large change in the movement of the stock.

We also thought about neat ways to filter the data and extract trends. We looked at the data on a per week per year basis. We divided the tweets based on whether they were retweets, replies, or original posts. We set thresholds for the favorite/retweet count. After a bunch of different threshold levels, we agreed on the ones included in the final file.

In order to justify our data, we also looked at some of the most stock-influencing tweets. The fact that these tweets shifted the stock made a ton of sense (e.g one was regarding a restructuring of the company).

For the sentiment analysis, we used the Vader method learned in lecture. The goal of this section was to see if sentiment strength was correlated with larger changes in the stock price. We first segmented the tweets by sentiment strength and sign. Then we had to integrate the sentiment data with the stock data. This took much longer than it should have. Most of the debugging centered around comparing dates in the two dataframes. Eventually we were able to ensure that the dates were the same datetime format, that neither was switched to a string, and we eliminated a couple of pesky tweets whose dates were especially problematic. Despite our efforts, this entangling of the two dataframes is still the least efficient part of the sentiment analysis portion. We believe the inefficiency arises from both reindexing the dataframe and performing a time-eating function on one of the columns. A far easier confluence of stock and sentiment data occured when we examined 544 target tweets.

In our sentiment analysis we also segmented the tweets into months. Again there was copious amounts of datetime debugging, especially in functions like ‘month_data’. In that particular case we had to develop a set of conditionals such that the function could be applied to various months and years without compromising the previously established date format. The results here were especially rewarding to graph because they demonstrate some correlation beyond the outlier tweets that are obviously quite important. All in all good fun.

Using the api rewarded us with a file of data with columns and rows, but there was no way to go from an individual row back to the original post. We decided to create a small function that would allow us to select a date, and it would use the google api to find the tweet from the text, and then scrape the html to embed a twitter post onto the notebook. We created a google custom search that only searches on the twitter.com/ElonMusk url. We then navigate to the first result, and uses twitter api to generate the html. We tried web scraping directly from twitter for the embedded link, but the html is generated dynamically with an ajax request that could not be easily replicated, nor is the necessary html embedded in the websites html. We first tried to use kivy, a gui creator, but it was not meshing well with jupyter notebook. It would create a fullscreen window that blocked the rest of the notebook, and kivy itself does not support html rendering. So we decided to use the ipython widgets as previously. The problem with the widgets is that there is limited functionality in terms of dynamically changing results. It was trivial to create 3 dropdown menus, one for year, month and day. However, Elon Musk did not tweet at every possible date. So the months dropdown menu changes to only display the months where Elon Musk Tweeted, which, once set, changed the available options available for the day. For example, selecting 2010 immediately restricts the user to one day**.** The documentation allowed for such an implementation, and we would then

run code to obtain the html and display it. The problem comes when we want to update the html. Ideally, the user selects one date, gets results, then selects a different date and the results update. However, the display functionality does not allow for the updating of what has been updated. We tried using an output widget, unfortunately, the updating of information does not happen on a loop, but with events, so a widget has to both exist outside a function to be kept in scope, but within the function to be rendered properly. Ultimately, we decided to keep it so that the output does not clear when a new date is selected, but the html is appended to the output.

We trained an SGD Classifier in an attempt to see if the rise or fall of stock could be predicted by features of a tweet. At first, we used retweet count, like count, and whether or not it is a reply or retweet. This gave inconsistent random random results. Even with a grid search, there was not much improvement. A random forest performed better. The most important features were the popularity of the tweet. The retweet count was a much better predictor than if it were a retweet or not. However, it was when combined with the sentiment analysis that we got the most accurate results, that is almost 60%, so still not very accurate, but more accurate than just the initial four features. So why we can conclude that twitter is not a good predictor of the stock market, we can say that there are some features which correlate more with the stock market than others. It was expected to not be accurate, if the stock market could be predicted from twitter, everyone would be rich.

We attacked this data from a lot of different angles, but our work only raised more questions. If we had more time and resources, we would want to dive deeper into two additional questions. First, how does Elon Musk's impact on Twitter compare with other CEOs or perhaps even world leaders. Since the Twitter data was so hard to access we never seriously considered pursuing this question, but nonetheless it would provide a control group to compare Mr. Musk's tweets to. The second point for further research is the effect of saying something on Twitter instead of elsewhere. For example: would Tesla's stock have changed more or less if Mr. Musk had announced the company's reorganization via a more traditional medium instead. Since major events are relatively infrequent, answering this question would have again likely involved broadening our scope into major announcements from other companies.