Using Twitter Sentiment Analysis To Predict the Direction of Blue Chip Stocks' Daily Price Movement

Teddy Burns, Andrew Heimerman, Nick Loeper, Naveen Senthilkumar

December 12, 2016

Contents

1 Abstract

Twitter is a microblogging platform that allows users to share short concise messages to a wide audience. Tweets capture emotional content as well as real-time information about global events. When applied to the stock market, this information can show predictions of future stock movement when the sentiment of tweets is properly classified. We used the NLTK library and a Naive-Bayes classifier to classify the sentiment of tweets. The tweets were queried from the Twitter API, where the payload of tweets corresponds to a certain stock ticker, such as AAPL. These sentiment classifications were used as a feature, along with other tweet metrics such as favorite and retweet count, to train a neural network that is used to predict stock movement over the next day. Sentiment analysis proved to generalize well over the dataset, providing a good feature space for the neural network.

2 Introduction

Unlike traditional sources of information, Twitter is real-time thereby increasing the rate at which information is dispersed (Mao, Wang, Wei, & Liu, 2012). Being able to capture the information contained in tweets allows us to forecast the stock market by incorporating new information more quickly than the market can react (Bollen, Mao, & Zeng, 2010).

While the stock market often bases valuations on business fundamentals, it also frequently seems to react to emotional changes in the general opinion of a company. We

believe that capturing this company specific sentiment can help capture these emotional responses (Gilbert & Karahalios, 2010).

We use Twitter data queried in the format '$TICKER' to gather tweets relevant to the company of study. This data can be obtained through a script that runs continuously using the Twitter API to gather a dataset of historical tweets. One major drawback with this is that the window of available tweets for the API is limited to about 10 days.

We use a Naive-Bayes classifier to predict sentiment. This sentiment is then fed into a neural network along with metadata about the tweet such as the number of times it was favorited, retweeted etc.

Challenges include data collection, avoidance of overfitting, and identification of an optimal architecture for a neural net.

Our goal is to beat benchmark prediction results by at least 5%.

3 Methods

3.1 Data Collection

There are two conventionally adopted methods of gathering Twitter data: the Twitter API and third-party data aggregates. Using the Twitter API it is only possible to query tweets in a historical window of ten days. This is a major hurdle when trying to assemble a large dataset to model. The API is also restricted by the number of calls it accepts in a one hour window. Until recently there were large, publicly-available academic datasets for research, but due to recent changes in Twitter's data policy, most have been removed. GNIP, a data provider of historic Twitter data, was recently acquired by Twitter and is the sole provider of historic data. To train our model for Sentiment Analysis we used the Twitter API to gather 2000 tweets about four different large blue chip companies. We diversified our query to avoid overfitting our training data. We then wrote a script to allow for an easy interface when hand classifying these tweets. Our dataset for implementation of the neural net is different from the one used to train the classifier because we required a larger sample over many days. This second dataset is consistent with our initial training, however, because it was queried using the same parameters. This data was obtained from followthehashtag.com.

3.2 Preprocessing

Tweets contain many stopwords, symbols, and textual emotion. Therefore, in order to best reduce the textual input space for a natural language processor, we included several methods to preprocess the tweets.

Retweet, Favorite, and Follower Counts: These fields are present in the original tweet data. However, if any of these values were zero, the data would be represented as

Not-a-Number (NaN). For each tweet, we reset the value of these fields to 0 if a NaN was present.

Symbolic Data: Many of the tweets contained characters such as '$', '#', '@', or '%'. We removed all the special characters and punctuation that provided no contextual information. Exceptions to this were the '?' and '!', which could represent some level of implicit sentiment. We also removed "RT" from all tweets as we only wanted to use the retweet count metadata within the tweet itself. In addition to the punctuation, URLs were removed using regular expressions.

Emoticons: In the event that a tweet had emoticons, we replaced them with their corresponding emotion in a textual format. For example, ':)' would be translated to 'smile' and replaced in the tweet text.

Text Tokenization: The text was tokenized by the NLTK library's tweetTokenizer. This class was used to turn the text to lowercase and tokenize user handles. Additionally, the tokenizer removed excess repeated letters from words when the repetition of the letter was greater than 3. For example, 'baaaad' would be replaced with 'baaad.'

Stopword Removal: From the tokenized text stopwords were removed using the NLTK stopwords corpus. Additionally we removed the query term (stock ticker) to ensure that the tweets are analyzed without a bias.

Upon completion of the preprocessing, we replaced the values of the input pandas dataframe with the preprocessed data. This modified dataframe was sent to be analyzed by the NLP/Sentiment Analyzer (Pagolu, Challa, Panda, & Mahji, 2016) (Bonzanini, 2015).

3.3 Sentiment Analysis

Since sentiment is subjective, we thought the best way to create sentiment classifications was to do so by hand. Before training our Naive-Bayes classifier, the tweet text was preprocessed as described. There was some concern for overfitting, since the 2000 tweets may have been overly specific to their respective company, although preprocessing created better textual data standardization.

Our Naive-Bayes classifier was implemented using NLTK, Python's natural language toolkit. The NLTK package is built on Python's scikit-learn (machine learning library) with incorporation of TF-IDF (Term Frequency times Inverse Document Frequency) (NLTK). Because of NLTK's powerful natural language processing capabilities, the environment was ideal for creating a sentiment classifier.

In order to evaluate performance, we used cross-validation, specifically a 4-fold cross-validation. 4-fold was optimal because it was a small enough number that helped to avoid underfitting, yet it was large enough to ensure that there was variance among each fold. The data set consisting of 2000 tweets was split into training and test sets using sklearn's

KFold module. The KFold module created 4 different variations of training and test sets, since there were 4 folds. Each training set contained 1500 samples, while each test set contained 500 test samples.

As a comparison for our cross-validation performance, we established three baseline tests: most common classification, random classification using the classification distribution, and a random classification with a fair probability density function. The most common classifier baselines assigned a classification of zero to every tweet since zero, or neutral, was our most common sentiment classification. The random baselines assigned a classification based off a probability density function corresponding to the classification distribution, as well as a fair (equal chance) probability distribution.

3.4 Neural Network

The neural network was implemented in Python (3.5) using TensorFlow. The data was transformed from a Pandas dataframe into a set of Numpy NDArrays in order to process using the TensorFlow network. The output is encoded in two numbers using a softmax regression on the outputs, resulting in a pair of probabilities that add up to 1. The softmax function is used to generate the probabilities that the correct classification for the output is in each class. The equation is as follows:

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^T \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^T \mathbf{w}_k}} \tag{1}$$

Due to the variability of the results given by different architectures of neural networks and the scarcity of data, we chose to experiment with a variety of architectures.

Constants Two things were held constants throughout the varying architectures: 7 inputs and two outputs, as well as having 1000 iterations for training. We saw that this was enough in every case to see some sort of minimum in the out of sample error.

Variables

Hidden Nodes/Layers: We experimented with neural networks that have both one and two hidden layers. For the networks with only one hidden layer, the number of nodes in the layer were for all n such that $n \in [3, 8] \wedge n \in \mathbb{R}$. For the networks with two hidden layers, with the number of nodes in the first layer denoted by f and the number of nodes in the second hidden layer denoted by s, we used all architectures for every $f \in [4, 7] \wedge f \in \mathbb{R}$ and every $s \in [2, 4] \wedge s \in \mathbb{R}$

Activation Functions: After conversing with Professor Alvarez, we decided to try multiple activation functions. The first activation function is the standard sigmoid function, which acts as a squashing function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

The second activation function is the rectifier, which is a function that just ensures that the input is floored at zero. It is represented as below:

$$f(x) = max(0, x) \tag{3}$$

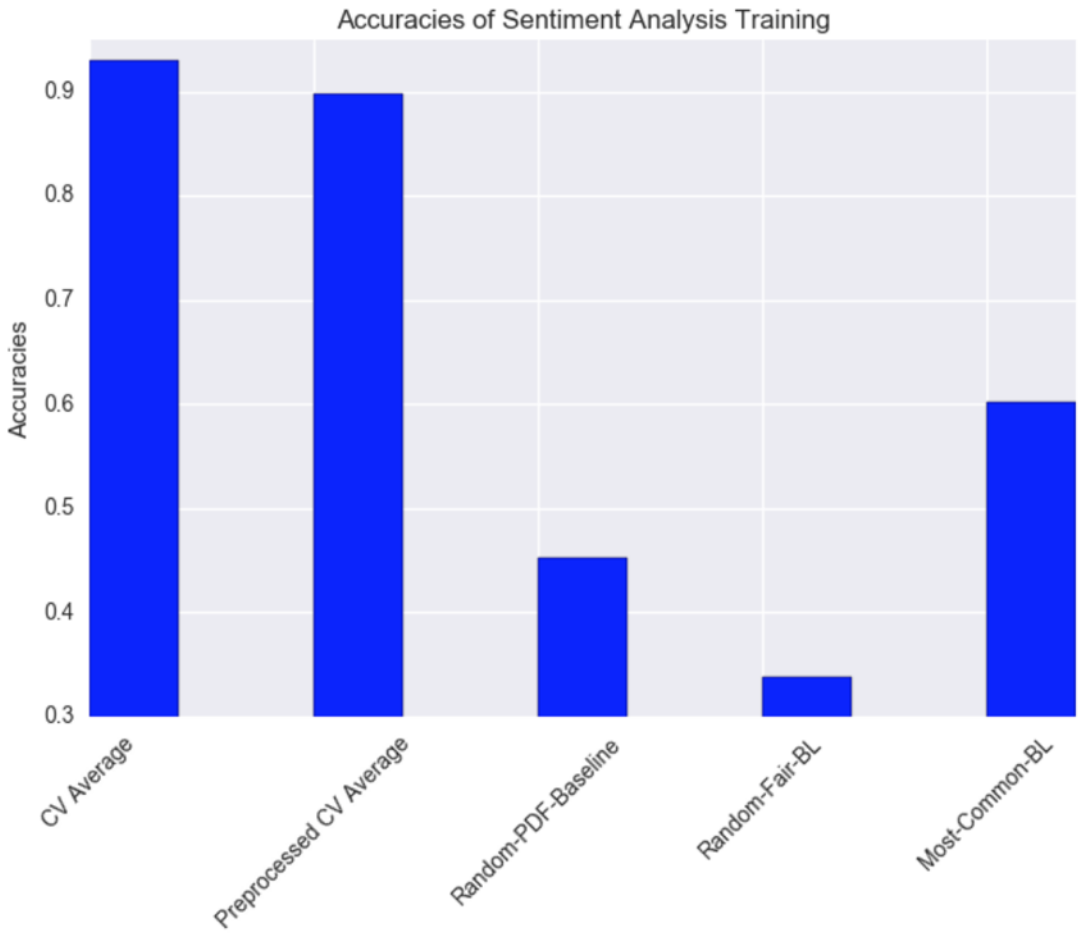
Lag period We decided to experiment with lag periods for the predictions of 1, 2, and 3 days. This means that we were predicting stock prices 1,2, and 3 days in advance. Our main method for stopping overfitting is early stopping, as it has been proven to be an effective method for regularization for neural networks.

4 Results and Discussion

4.1 Sentiment Analysis

The reported results from our experiment and simulations of the sentiment analysis classifier are as follows:

Accuracies	
Cross-Validation Average (No Preprocessing)	92.95%
Cross-Validation Average (Preprocessing)	89.74%
Random (PDF-Distribution) Baseline	45.25%
Random (PDF-Fair) Baseline	33.80%
Most Common Baseline	60.20%



Although the average accuracy for cross-validation trainings are reported, the accuracies

for each fold in the preprocessed and non-preprocessed trainings were relatively similar, which implies a very low variance. Since the Naive-Bayes classifier is ideal for sentiment analysis, the accuracy across all folds was very high. We originally expected the accuracy to be lower since classifying sentiment via text is not necessarily an easy task.

When comparing our cross-validation results with the baseline tests, we heavily outperform our baselines. Outperforming our baselines creates more confidence in our model since we are performing better than the 'bare minimum' performance accuracy. One of the main factors that contributes to this high performance is the choice of a Naive-Bayes classifier. A Naive-Bayes classifier is a probabilistic classifier (multi-class) that assigns a classification based off of a probability. This paradigm is ideal for sentiment analysis since sentiment is based off subjectivity. Using probabilities to classify sentiment makes the most sense since a tweet's sentiment is probable to be a certain classification, rather than a definite classification.

These results demonstrate minimal overfitting due to the Naive-Bayes classification. Since the Naive-Bayes classifier has a limited hypothesis set and usually does not perfectly fit a dataset, the overfitting is held to a minimum (Domingos & Pazzani, 1997). This does negatively affect the in-sample error because the training data set is usually never classified with 100% accuracy. However, this model generalizes well and minimizing the out-of-sample error via limiting overfitting is more important than a low in-sample error rate.

4.2 Neural Network

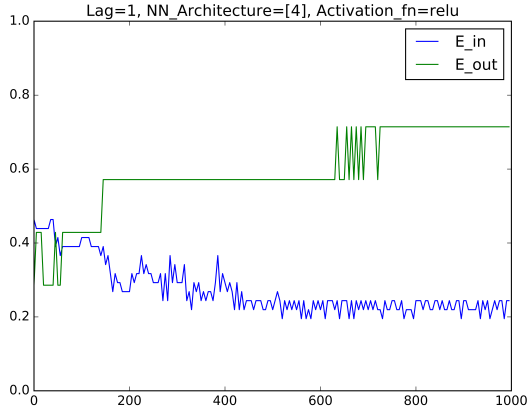
We must discuss the three variables that we had: architecture, lag period, and activation function.

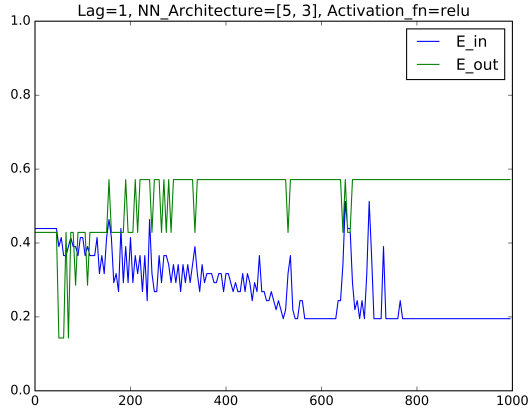
Architecture In class it was discussed that the more complex a neural network (more nodes per layer, more layers), the more it tends to overfit. As shown below in our charts, surprisingly enough, two layer models with a moderate amount of nodes had the best accuracy, and that best out of sample accuracy was short-lived. As training continued, the more complex the model, the faster the out of sample error rate moved away from the local minimum

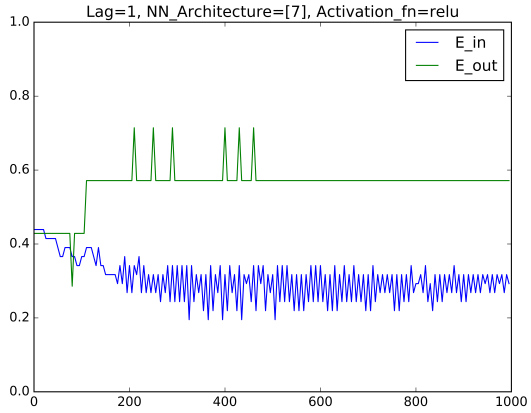
Activation Function After the recommendation to look at using ReLU nodes (the rectifier function), we noticed significant improvement in both accuracy (both in and out of sample) and speed at which it hits the minimum out of sample error. In the graphs below, the ReLU nodes resulted in somewhere between 0.12 and 0.28, which was significantly better than with the sigmoid function. When using the sigmoid function, the network had the tendency to only classify the stock as going up, even at very large amounts of training iterations. The graph below with the sigmoid function was the most accurate (although one of the slowest convergences).

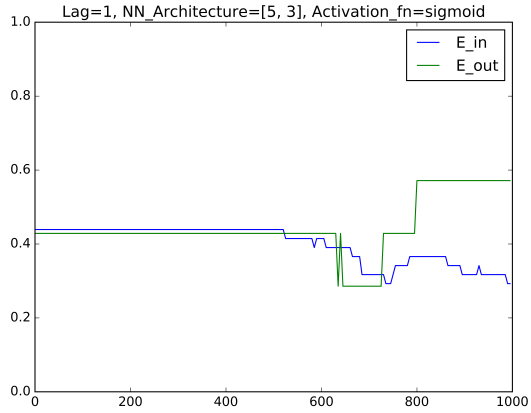
Lag As expected, when the lag was greater than one day, the network was unable to classify the data more reliably than only classifying each day as a positive movement in stock price.

As a whole, the model performed significantly better than a baseline. For a baseline, we used a gaussian Naive Bayes classifier (from Python's SKLearn). For in sample error, the error rate was 0.38. For out of sample error, the error rate was 0.57. Both of those are not particularly accurate, with the out of sample error generalizing to worse than just classifying every day as positive (indicating a stock moving up, which would result in an error of 0.42). The model was able to, at its best, have an out of sample error of nearly $\frac{1}{4}$ the error that would result from classifying every day as positive (again, 0.42).









5 Conclusions

Overall, the project suggests that, at least for Apple stock (hopefully more bluechip stocks as well), Twitter sentiment relating to the stock does signal a movement of price the following day in the same direction as the majority sentiment. The selection of architecture and models was by far the most difficult part of this process, but through significant experimentation and some intuition, we were able to come up with somewhat conclusive results.

Our next steps are to check these correlations for other stocks. While we were not able to test our findings with other stocks, we believe there is strong reason that this will work with others, and that will be tested.

6 References

1. Bollen, Johan, Huina Mao, and Xiaojun Zeng. "Twitter Mood Predicts the Stock Market."Journal of Computational Science2, no. 1 (March 2011): 1?8. doi:10.1016/j.jocs.2010.12.007.

2. Bonzanini, Marco.Mastering Social Media Mining with Python. Place of Publication Not Identified: Packt Publishing Limited, 2016.

3. Domingos, Pedro, and Michael Pazzani. "On the optimality of the simple Bayesian classifier under zero-one loss."Machine learning29, no. 2-3 (1997): 103-130.

4. Gilbert, Eric, and Karrie Karahalios. "Widespread Worry and the Stock Market." InICWSM, pp. 59-65. 2010.

5. Mao, Yuexin, Wei Wei, Bing Wang, and Benyuan Liu. "Correlating S&P 500 stocks with Twitter data." InProceedings of the first ACM international workshop on hot topics on interdisciplinary social networks research, pp. 69-72. ACM, 2012.

6. Pagolu, Venkata Sasank, Kamal Nayan Reddy Challa, Ganapati Panda, and Babita Majhi. "Sentiment Analysis of Twitter Data for Predicting Stock Market Movements."arXiv preprint arXiv:1610.09225(2016).