# UGAHacks 6 Methodology

Logan Bayer, Dakota Craig, Madelyn Hendricks, Anderson Molter

February 5 - 7, 2021

## Contents



## 1 Introduction

Knowing when a public company is in the press could be helpful when determining whether or not to invest in them. We hypothesize that bad press results in a drop in stock price the next day, while good press results in an increase in stock price the next day. To test this hypothesis, we have developed a machine learning model to analyze a companies stock based on the language used in articles written about said company.

## 2 Data Collection

We obtained our stock data using the Yahoo Finance API.

We have also obtained two data sets which contain 'positive' words and 'negative' words.

# 3 Model

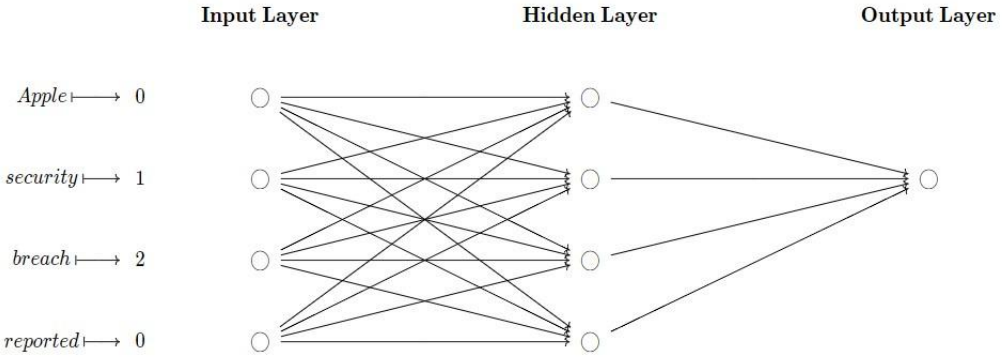
We have implemented a reinforced neural network where every neuron represents a word in the articles. If a word in the article appears in either data set (positive or negative words), a unique integer value is attributed to that word. If the word does not appear in either data set, then it is given a value of 0. We then feedforward our data and receive a real number from 0 to 1, which gets put into another function that determines whether or not to buy stock from the company.

Let our sigmoid activation function be defined as $\sigma(x) = \frac{1}{1+e^x}$

Each one of our neurons $x_i$ holds the following data:

- a weight $w_i$; where $i$ represents the integer value attributed to the word represented by the neuron.
- a bias $b$

**Example Network:** For simplicity, let our data be the following sentence: "Apple security breach reported." The words 'apple' and 'report'/'reported' are neither positive nor negative words. On the other hand, 'security' and 'breach' are negative words.




Based on our hypothesis, we theorize that Apple stock would go down in price the day after this article gets posted.

**Example Iterations:** Let $\vec{w} = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix}$ and $\vec{x} = \begin{pmatrix} 0 \\ 1 \\ 2 \\ 0 \end{pmatrix}$. Omit $b$ for now.

Then $\vec{w} \cdot \vec{x} = (0 \cdot \frac{1}{4} + 1 \cdot \frac{1}{4} + 2 \cdot \frac{1}{4} + 0 \cdot \frac{1}{4} = \frac{1}{4} + \frac{1}{2}) = \frac{3}{4}$, and $\sigma(\frac{3}{4}) = \frac{1}{1+e^{\frac{3}{2}}} = 0.321$

If $\sigma(\vec{w} \cdot \vec{x}) > 0.5$ , then our model recommends the user to buy stock in Apple.

Otherwise, the model recommends the user to do nothing. In this example, the model would recommend the user to do nothing. Now, say that the model was correct in its recommendation to the user, and Apple stock did go down the day after the article was posted. Then the weights and biases get adjusted.

Now, let $\vec{w} = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \\ 1/3 \end{pmatrix}$

Then $\vec{w} \cdot \vec{x} = (0 \cdot \frac{1}{3} + 1 \cdot \frac{1}{3} + 2 \cdot \frac{1}{3} + 0 \cdot \frac{1}{3} = \frac{1}{3} + \frac{2}{3}) = 1$, and $\sigma(1) = \frac{1}{1+e} = 0.269$

The model still makes the correct decision, and it is closer to 0 than the previous one, so the weights and biases get adjusted slightly.

The idea behind our model is that it reiterates this process over and over until it finds the most optimal setting for weights and biases.

# 4 Algorithm Design

Our algorithm takes in two sets of data:

- stock data
- word data

Our word data gets parsed into two different sets: positive words and negative words. Words which are neither positive nor negative are considered neutral, and neutral words have no impact on the decision that the model makes. We then connect all three sets of data to our new neural network. Our new neural network then learns to determine the overall connotation of the article (overall negative or overall positive). Based on the overall connotation of the article, the model makes a prediction that the companies stock will go up or down the next day. No matter what the algorithm decides, the weights and biases are adjusted, and the process is repeated. This process can be summarized by Figure 1 in the appendix.

# 5 Statistical Testing

To test whether or not our model is able to accurately predict the connotation of the article, we used a two-sample t-test. First, we took a random sample of 30 positive articles and 30 negative articles. The assumptions required for a valid two sample t-test are the following (sourced from JMP):

- Data values must be independent
- Data in each group must be obtained via a random sample
- Data in each group are normally distributed

- Data values are continuous
- The variances for the two groups are equal

Our data meet these assumptions, as the reader can verify by looking at the data given in the appendix. Our null hypothesis is that the populations of positive and negative articles are the same, meaning our model could not distinguish between a positive and negative article. The alternative hypothesis is that the populations of positive and negative articles are not the same. That is, our model can correctly predict the connotation of an article 95% of the time.

Let $\alpha = 0.05$

$d_f = n_1 + n_2 - 2 = 58$

So our t-value $t_{\alpha,d_f}$ is 1.6716.

The difference between averages is given by $\bar{x_1} - \bar{x_2} =$

Pooled variance $s_p^2$ is given by

$s_p^2 = \frac{((n_1-1)s_1^2)+((n_2-1)s_2^2)}{d_f} = \frac{((29)s_1^2)+((29)s_2^2)}{58} = \frac{s_1^2+s_2^2}{2}$

So our pooled standard deviation is

$s_p = \sqrt{}$

Finally, our test statistic is

$t = \frac{\bar{x_1}-\bar{x_2}}{s_p\sqrt{\frac{2}{30}}} =$

$t_{\alpha,d_f} > t$

# 6 Conclusion

# 7 Improvements, Thoughts and Looking Further

Our model is far from perfect. We chose to build a relatively simple model for learning purposes, and because of time constraints. Introducing new variables into our model complicates nearly everything, and we felt that executing a simple idea is the best course of action in this situation. This section highlights what we would do differently if we ran this experiment again.

Ideally, we would have liked to not use an existing model to train ours. Because of this, we had to scrap our original idea of using positive and negative reinforcement to teach our neural network. If given more time, we would have implemented a reward and punishment system similar to the one described below:

Our reward system is designed by a total of four cases:

- If the model predicts to buy a companies stock, and the stock goes up the next day, then the model is given a reward.
- If the model predicts to buy a companies stock, but the stock goes down the next day, then the model is punished.
- If the model predicts to not buy a companies stock, and the stock goes up the next day, then the model is punished.
- If the model predicts to not buy a companies stock, and the stock goes down the next day, then the model is rewarded.

Let $\epsilon > 0$. Let $p$ denote the model's prediction, and let $o$ denote the correct outcome. If our model is punished, then the function $P(o, p) = -\frac{|o-p|}{2}$ is applied. If the model gets rewarded, then the function $R(o, p) = \frac{1}{2}(\frac{p}{o+\epsilon})$ is applied. The range of $P$ is $[-0.5, 0]$ and the range of $R$ is $[0, 0.5)$ to ensure the model incrementally learns how to process the given data.

Of course, adding more parameters likely increases the accuracy of the model. Some potential candidates for extra parameters include: change in volume of stock, 52 week range of stock price, PE Ratio, PB Ratio, and many others. We also could have taken more information from the articles to see how much that article is being viewed. Some parameters which could have helped with that would be number of comments, the connotation of comments, number of times the article has been viewed, number of times it has been shared, etc. The reason why we chose not to include other parameters comes down to time, complexity, and to test whether news articles alone can influence stocks.

Overall, the stock market is tough to crack. If it were easy to tell when a stock is going to drop, then a lot of people would be richer than they currently are. In time, however, we do think that the stock market can be predicted using machine learning.

# 8 Appendix

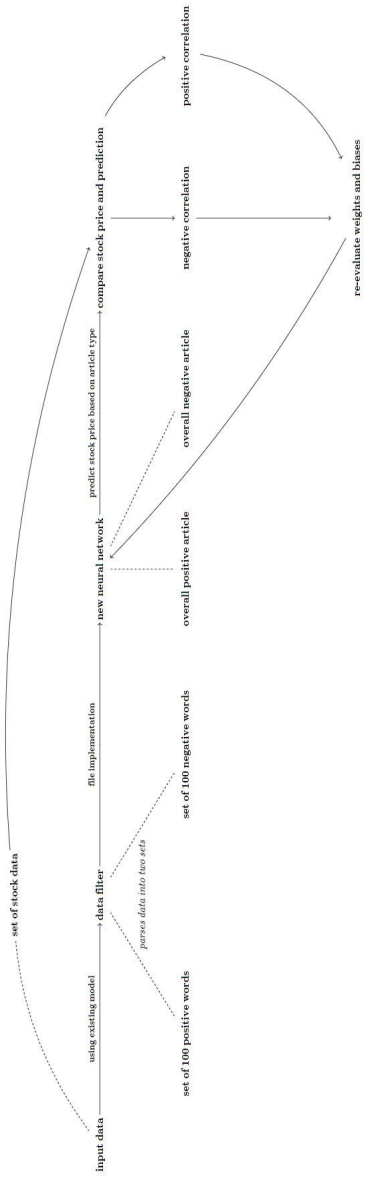



Figure 1

# References


[1] *https://www.jmp.com/en_us/statistics-knowledge-portal/t-test/two-sample-t-test.html* **The Two-Sample t-test**

[2] *https://towardsdatascience.com/a-beginners-guide-to-sentiment-analysis-in-python-95e354ea84f6* **A Beginner's Guide to Sentiment Analysis with Python**

[3] *https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce* **Recurrent Neural Networks**

[4] *https://www.baeldung.com/cs/reinforcement-learning-neural-network* **Reinforcement Learning with Neural Networks**

[5] *https://github.com/cjhutto/vaderSentiment/blob/master/README.rst* **VADER Sentiment Analysis**

[6] *https://keras.io/guides/sequential_model/* **The Sequential Model**

[7] *https://keras.io/api/models/sequential/* **The Sequential Class**

[8] *https://github.com/kofmangregory/Drag-and-Drop-Deep-Learning* **Drag and Drop Deep Learning**

[9] *https://www.kdnuggets.com/2018/06/basic-keras-neural-network-sequential-model.html* **Building a Basic Keras Neural Network Sequential Model**

[10] *https://machinelearningmastery.com/develop-character-based-neural-language-model-keras/* **How to Develop a Character-Based Neural Language Model in Keras**

[11] *https://towardsdatascience.com/sentiment-analysis-with-text-mining-13dd2b33de27* **Sentiment Analysis with Text Mining**

[12] *https://towardsdatascience.com/a-beginners-guide-to-sentiment-analysis-in-python-95e354ea84f6* **A Beginner's Guide to Sentiment Analysis with Python**