Reddit News Headlines for Stock Market Perdiction

Eric Stevens

***Abstract*— This paper details an effort to utilize a small amount of news data to make predictions about the daily movements of the stock market. A variety of techniques were attempted to model the news data but none were successfully able to extract information about the market. This paper includes a detailed description of the data itself and the effects that the preprocessing pipeline had on the data, the efforts that were made, and an analysis of why the techniques failed to capture information in the data. The techniques that were attempted include a unigram probability method, logistic regression, and an LSTM.**

I. INTRODUCTION

The stock market is notoriously difficult to predict on a day to day basis. Fundamental analysis of a security is the act of trying to predict what the value of shares will be based on the real world aspects of the company. This can include, but is not limited to, company debt, earnings reports, performance relative to competitors, etc. Another important part of fundamental analysis is the effort to reason about the current state of the economy and global affairs can affect the market value of a security. It is this aspect of fundamental analysis that this paper attempts to leverage in order to generate a trading signal.

A change in some factor can immediately alter the fundamentals of a security. For example, company A produces steel hardware in the United States but imports the raw material from a country that can provide it at a lower price. If the government of the United States suddenly passes a law that imposes taxes on steel imports, then that company is immediately less valuable. The import tax cuts into their profit margin, reducing what they are capable of pulling. The market sees this and the stock becomes over valued, shares are sold off until they reach a stable price that reflects the change to the profit margins that are possible after paying the tax.

The preceding tax law example was only intended to demonstrate the ways in which world news can directly effect the stock market. Tax laws are not passed over night. There is a long lead up to a vote on a tax law in which investors are taking bets on either side of any given outcome. However, it can be said that there are moments in time when news stories, such as the results of a vote on tax law change, can change the value of a security immediately.

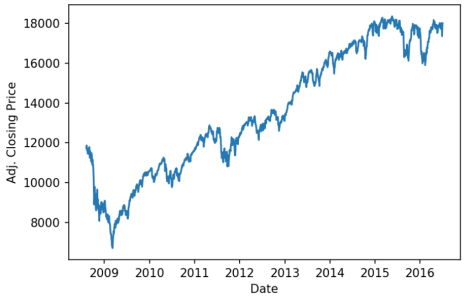

Fig. 1. Adjusted close of Dow Jones Industrial Average over the time period of analysis.

This paper lays out an experiment in an effort to use natural language processing techniques to extract information out of random news headlines that pertain to the immediate movement of the stock market.

II. DATA ANALYSIS

The data used for this task is a pre-compiled set from kaggle.com. It consists of two parallel data sets. Stock market data and news data over the same dates. The dates that are included are trading days that roughly cover the Obama presidency.

A. Market data

The first is simply the Dow Jones Industrial Average daily ticker. This includes open price, closing price, adjusted closing price, high of the day, low of the day, and trading volume. Most of this data will not be used for the actual training of the model.

```
Stock.head()
```

	Date	Open	High	Low	Close	Volume	Adj Close
0	2016-07-01	17924.240234	18002.380859	17916.910156	17949.369141	82160000	17949.369141
1	2016-06-30	17712.759766	17930.609375	17711.800781	17929.990234	133030000	17929.990234
2	2016-06-29	17456.019531	17704.509766	17456.019531	17694.679688	106380000	17694.679688
3	2016-06-28	17190.509766	17409.720703	17190.509766	17409.720703	112190000	17409.720703
4	2016-06-27	17355.210938	17355.210938	17063.080078	17140.240234	138740000	17140.240234

Fig. 2. The market data.

```
News.head()
```

	Date	News
0	2016-07-01	A 117-year-old woman in Mexico City finally re...
1	2016-07-01	IMF chief backs Athens as permanent Olympic host
2	2016-07-01	The president of France says if Brexit won, so...
3	2016-07-01	British Man Who Must Give Police 24 Hours' Not...
4	2016-07-01	100+ Nobel laureates urge Greenpeace to stop o...

Fig. 3. The news data set.

B. *News data*

The news data consists of the top 25 news headlines posted on the social media website reddit.com or each day of trading. These cover all sorts of topics that might be covered in any news paper.

C. *Combining data set for classification*

We combine these two separate data sets into a single data set for training. Since the Obama years were a time of relatively stable growth, looking at the values of the stock price itself would result in a large bias towards upward movement.

Instead, the problem is turned into a binary binary classification problem by, at any given day, looking at the closing price of that day and comparing it to the adjusted closing price of the following day. If the price of the following days adjusted close is higher than the current day then the current day is marked with a 1 label. If the closing price of the following day is lower than the current day than the current day is labeled with a 0.

With each of these labels there are the top 25 news headlines for that day from Reddit. They are usually between 15 and 30 words long each.

D. *Market data exploration*

The problem of an upward bias in the market data was spoken about previously. It was stated that this issue would be solved by turning the problem into a binary classification problem. Turning the problem into a binary classification problem does not, in itself, make this issue go away. If the stock market was growing so steadily because it had many more days where the market rose then days where the market fell then this same issue of bias would be present in a binary classification problem. However, if the market grew because, even though there were a close to an even split between number of days the market went up and down, the days that the market went up it went up much further than it went down on the down days, then the binary classification will remove the bias. We examine the result of turning the problem into a binary classification problem by inspecting the label counts. As can be seen in figure 5, there is a slight bias towards a 1 label, meaning that the data is biased towards saying that there was a higher close on the following day. That being said, the label split is almost even, reducing the bias that would be present had this problem not been reduced to a binary classification problem.

```
data_combined.head()
```

	Date	Label	Headline
0	2008-08-11	0.0	b'Why wont America and Nato help us? If they w...
1	2008-08-12	1.0	b'Remember that adorable 9-year-old who sang a...
2	2008-08-13	0.0	b' U.S. refuses Israel weapons to attack Iran:...
3	2008-08-14	0.0	b'All the experts admit that we should legalis...
4	2008-08-15	1.0	b"Mom of missing gay man: Too bad he's not a 2...

Fig. 4. The data sets are combined with a binary label for training.

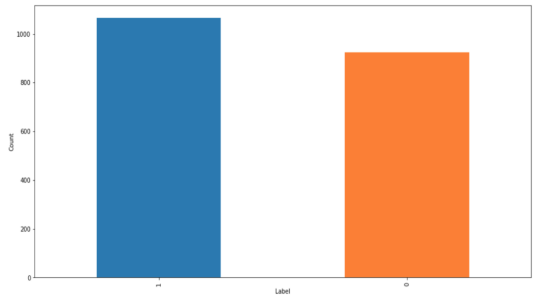

Fig. 5. There are close to an even number of each label.

E. *News data exploration*

The news data consists of the top 25 news headlines on Reddit of every trading day over the time period in question. In other words, there is no news data for weekends or holidays since the markets are closed on those days. Just as we

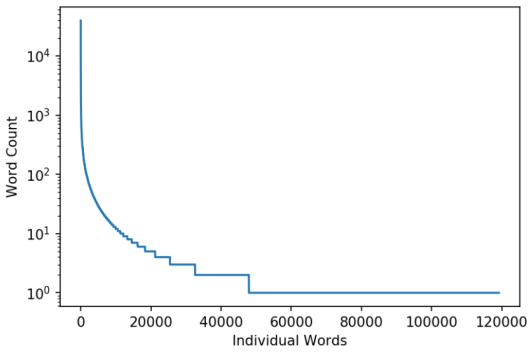

Fig. 6. Individual word counts.

```
Run 0 accuracy: 0.5202020202020202
Run 1 accuracy: 0.494949494949495
Run 2 accuracy: 0.5606060606060606
Run 3 accuracy: 0.51010101010101
Run 4 accuracy: 0.5151515151515151
Run 5 accuracy: 0.5404040404040404
Run 6 accuracy: 0.5909090909090909
Run 7 accuracy: 0.5202020202020202
Run 8 accuracy: 0.5858585858585859
Run 9 accuracy: 0.5707070707070707

Average accuracy: 0.5409090909090908
```

Fig. 7. Performance of the baseline model.

did with the market data labels, it is important to ensure that the data is regular enough to ensure that we are able to train models without bias in the data. Figure 6 shows the word counts. What is being plotted is the simple splitting the news headlines everywhere there is a space, using the resulting string subsections as tokens, and counting occurrences. On the x axis of figure 6 are the individual words (not listed), and on the y axis are the number of times each word appears throughout the text data. Note that figure six is plotted with a log y axis. We can see that there are only a few words with counts over 1000 appearances, these are likely stop words. There are also many words, around 60% of all the words, with only a single occurrence. This is a major sparsity issue that will have to be managed.

III. Baseline Model

Since it is our goal to try to capture some information in the text and leverage it for the sake of predicting market movement on the following day, a baseline should be determining if any information is actually be extracted from the data. In the previous section it was explained that there are slightly more days in the data set labeled 1 (market moves up) than 0 (market moves down). This means that if a model is implemented that just guesses that the market will always go up then that model should have a performance slightly better than 50%. This model is tested on a 10-90 test train split, shuffled each time the model is ran. The results can be seen in figure 7.

We can see that the performance ranges from 50% accuracy up to about 60% accuracy with an average accuracy of 54% over these 10 tests. This was basically what was expected based on figure 5.

This 54% will serve as a benchmark for what any model that is able to extract information from the text pertaining to the stock market should be able to beat.

IV. Unigram Probability Model

Now an effort will be made to extract some information out of the text data that pertains to the market. The first thing that will be attempted is to see if any of the words in the text hold any meaning that will be associated with movement of the market price on the following day. In order to do this several steps will need to be taken. The fist step is to trim the fat out of the text data. The data will be fed through a preprocessing pipeline before the being fed into the model. This will take care of the sparsity problems that were identified in figure 6.

After the prepossessing pipeline the remaining words will be fed into a unigram count based MLE model that will look for relationships between the collections of words found in the news headlines and the labels associated with those words. This method is described in the following p subsections.

A. News data prepossessing

There are several steps that will need to be taken in order to both reduce the dimensionality of the training data as well as remove the sparsity from it. The first step that is taken to reduce dimentionality is to remove token case dependency. This is done by forcing all the words in the text to be lower

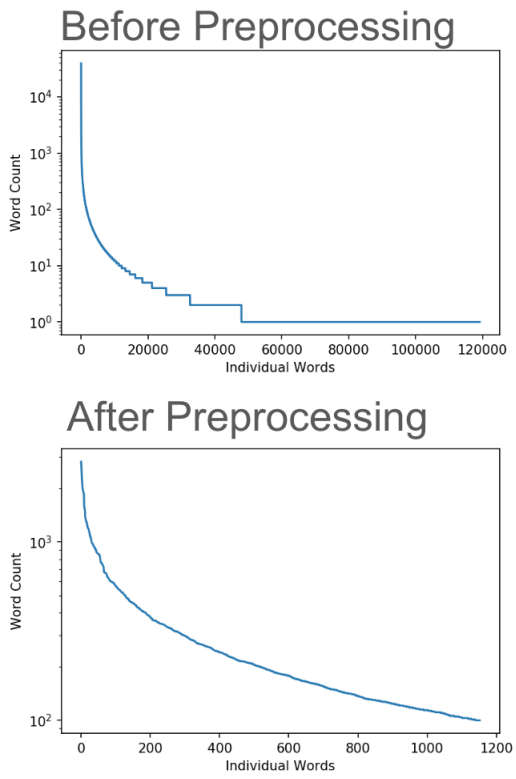

Fig. 8. Word counts before and after preprocessing.

case. The next step is to tokenize the text. Unlike the case that we started out with, here we will use the NLTK tokenizer, which will do the work of separating out punctuations form other tokens when they are not related to the word they are attached to. Now we can remove stop words and punctuation tokens from the text, which will bring down the numbers on the left side of figure 6 dramatically. Finally we perform sparsity reduction by only considering words that appear 100 times or more in the entire document. This way we can be pretty sure that none of the words that are being used to base label probabilities have not had ample opportunity to show up in both label categories. Figure 8 demonstrates the before and after of the data preprocessing pipeline for the text data.

From figure 8 we can see that both the number of words that are being considered and the number of times that the most frequently occurring words appear have been dramatically reduced. Before the preprocessing there were roughly 120,000 unique tokens; whereas now there are only about 1,200. The original data had tokens that appeared up to

Least Common	MostCommon
('qatar', 100)	('country', 1207)
('countrys', 100)	('iran', 1229)
('push', 100)	('uk', 1257)
('meat', 100)	('first', 1285)
('drop', 100)	('killed', 1302)
('forest', 100)	('one', 1306)
('disease', 100)	('u.s.', 1352)
('early', 100)	('president', 1370)
('hands', 100)	('war', 1521)
('offers', 100)	('years', 1534)
('tens', 100)	('russia', 1572)
('daily', 100)	('people', 1871)
('towards', 100)	('israel', 1879)
('thailand', 101)	('police', 1905)
('hiv', 101)	('china', 1974)
('suggests', 101)	('government', 1999)
('conditions', 101)	('world', 2179)
('showing', 101)	('new', 2350)
('500', 101)	('says', 2561)
('reactor', 101)	('us', 2811)

Fig. 9. 20 least common and 20 most common words after preprocessing.

20,000 times in the data set; now the maximum number of appearances is less than two thousand. In figure 9 we examine the 20 most common and least common words.

We can see that the words that are the least common are not words that are totally obscure. They are not names of random people or small towns in which some event took place. They are commonly occurring terms that have some relevance to the world. One can even imagine situations in which some of these words may lead to a change in some stock market attribute. For example, one of the tokens that appears in the least commonly appearing tokens after preprocessing is the token 'hiv'. If 'hiv' appeared in the headline 'u.s. pharmaceutical manufacturer phizer discovers cure for hiv', it becomes pretty apparent how this news could effect the stock market. The 'hiv' token the main subject of that sentence, so its importance can not be overstated.

On the side of the figure that contains the 20 most common tokens we see many words that could have a dramatic effect on the fundamentals of the stock market. 'u.s.' is in there, 'president' is in there, and government in there. These are the types of words that were spoken about in the introduction to this paper as likely having an impact on the fundamentals of the stock market. So it appears that the preprocessing pipeline is functioning as intended.

B. Model implementation

The simple unigram probability model that will be used to see if and of the words in the document hold any meaning in terms of the stock market. What is meant by unigram probability is the probability that when a word appears in the document it is appearing an a headline set with the 1 label. The unigram probability of a 1 label can be expressed as follows,

$$P(label = 1|w_i) = \frac{\#(w_i|labeled =)1}{\#w_i} \quad (1)$$

For a given headline set the total probability that that headline will be correlated with a 1 label will be the average of the unigram probabilities of the individual words that make up that headline. The equation that represents this is,

$$P(S = 1) = \frac{\sum_i^N P(label = 1|w_i)}{N} \quad (2)$$

Where $P(S = 1)$ denotes the probability that a given headline has a label of 1 and N is the number of words in the headline.

This model is built up by using an embedded default dictionary to count the number of 1 and 0 labels associated with each word. After all the labels have been counted for each word they the conditional probability can be calculated by cycling through each word in the default dict and dividing the 1 label counts by the sum of both label counts.

After the training is done on the training data, the test can just query the default dict for only words that are in the default dict. The labeling of the test data is as follows,

$$Label = \begin{cases} 0 & \text{if } P(S = 1) < 0.5 \\ 1 & \text{if } P(S = 1) \geq 0.5 \end{cases} \quad (3)$$

C. Results

Figure 10 shows the performance result from running the model discussed int the previous subsection for 10 random iterations.

We can see form this that there was actually a slight drop in performance in comparison to the baseline model but is statistically is the same. Figure 11 plots the values of equation 2 used to determine the label through equation 3. Looking at the values of the y axis in figure 11, it is easy to see that the system will always guess one because all of the values that result from equation 2 are above 0.5. This means that the unigram probability model and the baseline model are effectively doing the same thing; both of the models guess 1 every time.

```
Run 0 accuracy: 0.5151515151515151
Run 1 accuracy: 0.5454545454545454
Run 2 accuracy: 0.5757575757575758
Run 3 accuracy: 0.6060606060606061
Run 4 accuracy: 0.5303030303030303
Run 5 accuracy: 0.51010101010101
Run 6 accuracy: 0.5353535353535354
Run 7 accuracy: 0.494949494949495
Run 8 accuracy: 0.5353535353535354
Run 9 accuracy: 0.494949494949495

Average accuracy: 0.5343434343434343
```

Fig. 10. Results from running unigram probability model.

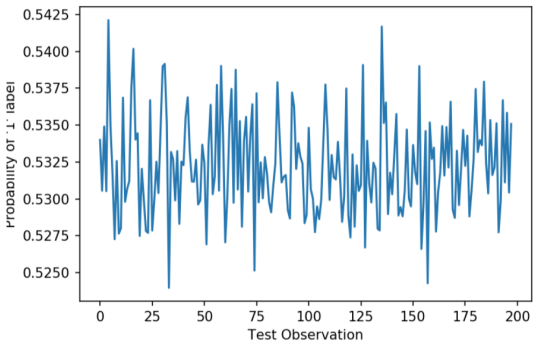

Fig. 11. Probabilities of 1 label for each observation.

In a last ditch effort to see if the model was extracting any actual information out of the text, the results of equation 2 were normalized across the range from 0 to 1 and a sweep of where the class division occurred form 0.1 to 0.5 was taken to check for performance of the model using the range of classification barriers. A graph of the normalization and an example sweep point can be seen in figure 12. At no point in the range did the this help the model performance. Most of the time the performance was worse than the baseline model, leading one to believe that the unigram probability model was not actually capable of pulling any actionable information out of the data set.

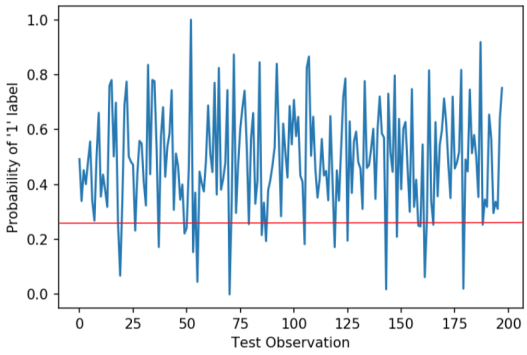

Fig. 12. Sweep across normalized unigram probabilities to see if information is in the signal.

```
tfidf = TfidfVectorizer(sublinear_tf=True,
                        min_df=100, norm='l2',
                        encoding='latin-1',
                        ngram_range=(1, 4),
                        stop_words='english')
```

Fig. 13. Function call for tf-idf vectorization and preprocessing.

```
Run 0 : 0.5105633802816901
Run 1 : 0.5080482897384306
Run 2 : 0.5187122736418511
Run 3 : 0.5099597585513078
Run 4 : 0.5142857142857142
Run 5 : 0.5172032193158954
Run 6 : 0.5127766599597585
Run 7 : 0.5259557344064386
Run 8 : 0.5214285714285715
Run 9 : 0.5045271629778671

Average Accuracy: 0.5143460764587525
```

Fig. 14. Logistic regression performance.

```
              precision    recall  f1-score   support

         0.0       0.48      0.31      0.38      4582
         1.0       0.55      0.71      0.62      5358

avg / total       0.51      0.52      0.51      9940

0.5240442655935613
```

Fig. 15. F score for single logistic regression run.

V. Logistic Regression

When the unigram MLE model failed so completely, a decision was made to not continue down the road of hand crafting models. An effort was made to leverage Python's data science packages in order to try to see if any information could be extracted from the data set.

A. Preprocessing and vectorization with Scikit-Learn

We will replace the preprocessing pipeline that was performed by hand in the unigram model with a call to Scikit-learn's tf-idf vectorization function. The function call can be seen in figure 13.

There are a few things about this method that should be noted. Because we are setting the minimum document frequency to 100, we are doing basically the same type of sparsity removal as we were doing before. It is almost the same and not exactly the same because before we were allowing an word that appeared anywhere in the document over 100 times to be included. The minimum document frequency asserts that the word must be in at least that many separate observation documents. The reason that it is almost the same thing is because words that are that sparse are unlikely to appear in the same document.

Next, notice the 'ngram_range' attribute in figure 13. Setting this attribute to (1,4) means that the vectorization will include the consideration of ngrams. This means that this model will consider word ordering to add some importance to the model. This will be helpful in distinguishing the difference when coming across the token 'market' in the context of either 'bull market' or 'bear market'.

B. Results

The results of running 10 randomized 10-90 test train splits can be seen in figure 14.

We can look at the performance statistics of one of these runs in figure 15.

We can see that the performance of the logistic regression model does not improve on that of the baseline model.

VI. LSTM

Although it is very unlikely that the amount of training we have is sufficient to train a neural network, one more experiment will be performed solely for the sake of experiment. We will attempt to train a recurrent neural network on the news data with the Keras Python library using TensorFlow for the back end. The construction of this model can be seen in figure 16.

```
print('Build model...')
model = Sequential()
model.add(Embedding(2000, 16, dropout=0.2))
model.add(LSTM(16, dropout_W=0.2, dropout_U=0.2))
model.add(Dense(2))
model.add(Activation('softmax'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

print('Train...')
model.fit(X_train, Y_train, batch_size=8, epochs=3,
          validation_data=(X_test, Y_test))
score, acc = model.evaluate(X_test, Y_test,
                            batch_size=8)
print('Test score:', score)
print('Test accuracy:', acc)
```

Fig. 16. LSTM model.

```
Train on 1611 samples, validate on 378 samples
Epoch 1/10
1611/1611 [==============================] - 14s 9ms/step - loss: 0.6918 - acc: 0.5289 - val_loss: 0.6973 - val_acc: 0.5079
Epoch 2/10
1611/1611 [==============================] - 13s 8ms/step - loss: 0.6196 - acc: 0.6511 - val_loss: 0.7099 - val_acc: 0.5794
Epoch 3/10
1611/1611 [==============================] - 15s 9ms/step - loss: 0.2629 - acc: 0.8951 - val_loss: 0.9444 - val_acc: 0.5185
Epoch 4/10
1611/1611 [==============================] - 15s 9ms/step - loss: 0.0649 - acc: 0.9851 - val_loss: 1.3301 - val_acc: 0.5370
Epoch 5/10
1611/1611 [==============================] - 14s 9ms/step - loss: 0.0143 - acc: 0.9963 - val_loss: 1.4155 - val_acc: 0.5344
Epoch 6/10
1611/1611 [==============================] - 14s 9ms/step - loss: 0.0045 - acc: 1.0000 - val_loss: 1.6761 - val_acc: 0.5635
Epoch 7/10
1611/1611 [==============================] - 14s 9ms/step - loss: 0.0033 - acc: 0.9994 - val_loss: 1.8303 - val_acc: 0.5582
Epoch 8/10
1611/1611 [==============================] - 15s 9ms/step - loss: 0.0011 - acc: 1.0000 - val_loss: 1.8914 - val_acc: 0.5556
Epoch 9/10
1611/1611 [==============================] - 14s 9ms/step - loss: 0.0083 - acc: 0.9975 - val_loss: 1.5082 - val_acc: 0.5450
Epoch 10/10
1611/1611 [==============================] - 14s 9ms/step - loss: 0.0093 - acc: 0.9988 - val_loss: 1.9239 - val_acc: 0.5291
378/378 [==============================] - 1s 2ms/step
Test score: 1.9239365114736808
Test accuracy: 0.5291005287851606
```

Fig. 17. LSTM training progress.

This model predictably failed to extract any meaningful information about the data as well. The results of running the model for 10 training epochs can be seen in figure 17.

VII. Failure Analysis

This experiment was unsuccessful for a variety of reasons. In this section the main reasons for the failure are covered.

A. Too little training data

There was far to little training data to have a chance of separating what is important in the stock market form what wasn't. The small token count of several hundred tokens by a thousand or so days is not nearly enough information to extract meaning form the English language let alone relate any information it does find to the stock market.

B. Bad choice of data

Using random news data from Reddit was also a bad choice. In random news data the signal to noise ratio is far lower than it might be in other types of data sets. For example, one could use a news wire feed coming from CNBC that pertains directly to the stock market.

C. Bad experiment design

Performing this task as a binary classification for looking at the only the current days data and trying to model the next days close may not have been the ideal way to capture how the stock market works. By forcing the problem into a binary classification problem we don't allow for news days that do not have any effect on the market. In reality, most of the news will not have an effect on the market. We could rephrase this problem as a multi-class classification problem with a neutral class. This would allow a lot of meaningless information to be collected in the category that has no effect on the market.

It may also be important to look at the time domain of the problem as more important than this experiment did. What the stock market is going to do the next day may depend on events that have been leading up to that day in weeks. Big news stories may be in the news for more than one day, but there are many stories that may only have high profile headlines in a time before when the stock market moves on the information contained within them. A longer time window may have to be examined to fully capture the information.

Both of the concerns above would require much more data than we are currently working with, which ties back to the topic of the previous sub-section.

VIII. Future Works

There are several strategies that will be taken on the next effort to try to extract stock market information from the news. First much more data will be collected prior to the model fitting stage. Second, alternative strategies will be used in terms of modeling. The best option forward may be to utilize transfer learning to point a neural network that has already gone through the work of learning the intricacies of the English language and can then be pointed at a stock market analysis task.

REFERENCES

[1] Aurlien Gron, Hands-On Machine Learning with Scikit-Learn & TensorFlow, ISBN-10: 1491962291

[2] Susan Li, Multi-Class Text Classification with Scikit-Learn, https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f

[3] Kaggle:Iseiyjg, Use News to predict Stock Markets, https://www.kaggle.com/lseiyjg/use-news-to-predict-stock-markets

[4] Kaggle:aaron7sun ,Daily News for Stock Market Prediction, https://www.kaggle.com/aaron7sun/stocknews