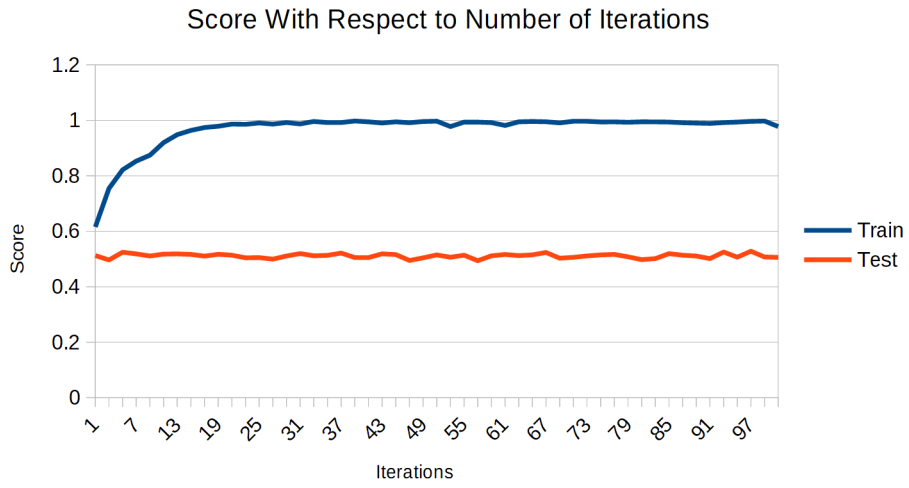
William Frank
3/9/19
Midterm Report

Introduction and Background:
With this project I will be attempting to predict stock price movement, specifically by using news articles and their headlines. Stock prices, and the stock market as a whole, react to events going on around the world, and that reaction is one that can be very hard to guage. Sometimes these events have a more obvious correlation, like a new product being announced, or stock market performance in other countries. Other times however, events are more tenuously related: natural disasters, political happenings, etc. Because my method requires interpreting news articles, specifically their headlines, it is fundamentally a natural language processing problem. Natural language processing (NLP) is a sub-field of computer science that deals with translating socially complex and nuanced text into values a computer can use. While investigating ways to use news articles to predict stock price movement, I will also be investigating ways to best extract features from news articles.

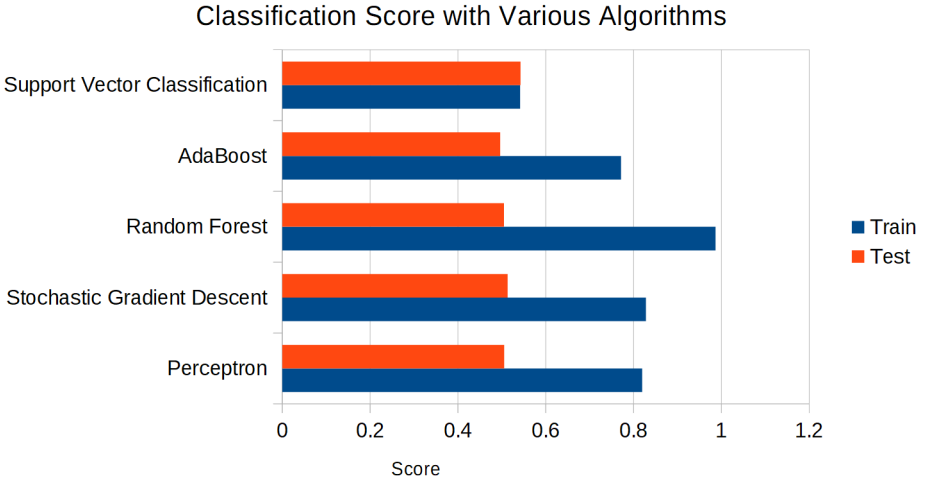
Motivation:
The stock market is affected by an incomprehensible number of factors in ways that humans could likely never properly account for with traditional models. Machine learning algorithms can construct models that take hundreds of thousands of attributes into account when assigning a final label. They can also assign weights to these attributes in a way that might be completely illogical to a human. The stock market is something that also generates a tremendous amount of data, which needs to be consumed and acted on in minimal time to maximize reward. To make the best possible prediction the model needs as much information as possible. To extract and evaluate this information to its greatest potential will require methods not practical through non-machine-learning methods.

Progress:
As the basis of this project is interpreting news articles, my first step was to gather as many as possible. I was able to use a Python script to scrape information about the most recent 20,000 Reuters articles on various subjects, initially their Economics section specifically. The information I was able to scrape was the headline, a short summary, and the date of posting. This gave me news articles dating from February 2nd, 2015 to March 1st, 2019. I paired this with data on the Dow Jones Industrial Average over the same time period. For my first tests I combined all headlines on any given date, and assigned them a label of 1 if the Dow Jones increased or stayed the same that day, and -1 if it decreased that day. To create attributes from the headlines I first used simple token counts, starting with tokens of length 1. This gave me 1025 samples to train and test on, of which I randomly sampled 80% for training and left the rest for testing. The reason for random samples was so that my model might eventually generalize to future news articles better, rather than always training and testing on the same set of events.

To start with I used Perceptron, as it is a classification algorithm I am familiar with. To get an idea of how well my current attributes map to their labels I measured testing and training error over varying numbers of iterations of Perceptron, reported below. Values are averaged over 20 iterations per Perceptron iteration count.



Even though training error drops to nearly zero percent, test scores are no better than guessing. Either another model needs to be chosen, or something needs to be done to better prepare the attribute values. To test the first option I trained and tested with several other learning algorithms, as shown in the below chart.



All algorithms are implemented by scikit-learn and ran with default parameters. Obviously something needs to be done to better prepare the attributes.

While browsing the documentation for scikit-learn I found tf-idf, a method to transform text into tokens just like the token count vectorizer, however the tokens are weighted in a way that "scales down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus" [1]. I combined tf-idf preprocessing with Perceptron, and used a cross-validated grid search to tune the parameters. I changed the following parameters: ngram_range=(1, 4) and max_df=0.6 for the initial count vectorizer, sublinear_tf=True for the tf-idf transformer, and max_iter=6 for Perceptron. Still, the maximum prediction accuracy achieved was 0.536. At this point I searched for a larger data set, thinking a larger number of news articles ranging a wider array of subjects might allow for a better model to be developed. The current objective is to predict large market trends, something that could be influenced by any number of domestic or foreign events. I

1 https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

found a list of one million Australian news headlines ranging from February 19th, 2003 to August 23rd, 2007. I combined this data from the S&P ASX200, which is considered the benchmark for Australian equity performance. I then repeated the grid search and achieved the best prediction accuracy with the following parameters: max_df=0.7 and ngram_range=(2, 3) for count vectorizer; norm=None, sublinear_tf=False, and use_idf=True for tf-idf transformer; and penalty='l1', alpha=0.00001, max_iter=5, and tol=0.1 for Perceptron. Again my best results were hardly better than random prediction at 52.5% correct predictions.

After this result I assembled a data set that targeted the most relevant news articles, in this case the most recent 20,000 Reuters Business headlines (at the time of assembling). The headlines were again combined with data from the Dow Jones to give a set spanning July 13th, 2017 to March 1st, 2019. Here I was able to get up to a prediction accuracy of 58.6%, which is still quite poor, but promising. Here are the 10 strongest positive and negative coefficients for the entire data set:

Word	Coefficient
tariffs	0.23772
hopes	0.209591
hit	0.207209
jobs	0.197433
production	0.19649
earnings	0.193861
take	0.190247
dollar	0.189975
start	0.177817
uk	0.175928

Word	Coefficient
fall	-0.292383
probe	-0.290845
group	-0.289265
air	-0.205643
win	-0.185183
reuters	-0.184809
battery	-0.166699
korea	-0.166648
air berlin	-0.16215
companies	-0.157488

These tokens were made from n-grams of lengths 1 to 4. Some of them make sense, like ‘jobs’, ‘production’, and ‘earnings’ being positive coefficients, but most of these do not, like ‘reuters’ and ‘air berlin’ being negative coefficients. When the model is instead trained on data from before 2019 and tested on data from 2019 the prediction accuracy jumps to 67.5%, suggesting that chronology might be important to the development of the model. This is also a result from only 40 non-random test samples and could merely be a fluke. Either way, more clearly needs to be done to generate a more accurate model.

The Future

Going forward I wish to explore two main avenues to improve prediction accuracy: improved attribute generation and improved input data. In the area of improved attribute generation I will go further into the natural language processing side of this problem.

There are better processes for extracting keywords alone from text. One such set of processes that I will implement is described by Iglesias JA, Tiemblo A, Ledezma A, and Sanchis A in Web news mining in an evolving framework. The specific step from this article I am missing is called stemming or lemmatization, and involves replacing words with their stems (also known as their root or base) to account for words with different forms but the same meaning. Another direction I want to go with preprocessing is word embeddings, a method for representing document vocabulary that is designed to capture semantic and syntactic similarity between words, as well as other relations. The article Visual exploration and comparison of word embeddings by Juntian Chen, Yubo Tao, and Hai Lin analyzes two methods for generating word embeddings: CBOW and skip-grams, as used by word2vec. Word embeddings are able to capture much more information than simple n-grams, and will hopefully be able to provide more insight into whether a news article is a positive or negative economic indicator. Several more methods for keyword extraction are described by Janghyeok Yoon: information filtering, co-occurrence information, domain knowledge, and graph analysis. These currently require further research and may be implemented if promising.

On the front of improved input data, I will create data sets with more examples and more articles per example. Web news mining in an evolving framework mentions one of the New York Times' API called Article Search. It can be used to query up to 10 articles at once, 100 articles per minute, and 40,000 articles a day. Because Article Search allows for searching by keyword, news desk, and subject I will be able to select news articles which are of the highest relevancy. The New York Times has another API, their Archive API, which returns metadata on all articles for a given month dating back to 1851. This will be invaluable for creating the most training examples possible. Another API that might be useful is their Semantic API, which gives a list of all the descriptors used to create article tags. These tags can be used to more efficiently search articles, as well as a method for generating the most information-dense tokens from text. Additionally, I will eventually use my model to evaluate real time news, which will likely be facilitated by their Times Wire API, a real-time feed of article publishes.

While those approaches are both on the data collection and preparation side of machine learning, I will also explore more sophisticated learning techniques. This problem is an ideal one to be solved using neural networks. The complex relations between words are unable to be properly captured by insufficiently complex models. A Unified Architecture for Natural Language Processing describes a neural network ideal for natural language processing which automatically generates various features from a given sentence.

The specific plan to complete these areas of investigation is as follows. First I will generate the rich NYT data set. From there I will use the various preprocessing methods described, starting with the easiest to implement, in this case stemming. After that I will begin experimenting with neural networks and more sophisticated methods for feature extraction. If a sufficiently good classification model can be created I will move to regression to predict how much the market will change by. Following these steps I hope to create a model that accurately predicts how current events will impact the stock market.

References:

Chen J, Tao Y, Lin H. Visual exploration and comparison of word embeddings. *Journal of Visual Languages & Computing*. 2018;48:178-186. doi:10.1016/j.jvlc.2018.08.008

Collobert R, Weston J. A unified architecture for natural language processing. *Proceedings of the 25th international conference on Machine learning - ICML 08*. 2008. doi:10.1145/1390156.1390177

Iglesias JA, Tiemblo A, Ledezma A, Sanchis A. Web news mining in an evolving framework. *Information Fusion*. 2016;28:90-98. doi:10.1016/j.inffus.2015.07.004

Yoon J. Detecting weak signals for long-term business opportunities using text mining of Web news. *Expert Systems with Applications*. 2012;39(16):12543-12550. doi:10.1016/j.eswa.2012.04.059