Predicting Nature of Stocks Based on News Articles

Team Members:

Deepan Saravanan (PennKey: `deepans`; Email: `deepans@seas.upenn.edu`)
Apurba Sengupta (PennKey: `aseng`; Email: `aseng@seas.upenn.edu`)
Chinmay Shah (PennKey: `chinmays`; Email: `chinmays@seas.upenn.edu`)
Vimell Yuvarajah (PennKey: `yvimell`; Email: `yvimell@sas.upenn.edu`)

Assigned Project Mentor:

Anant Maheshwari

Team Member Contributions:

Team Member	**Contributions**
Deepan Saravanan	Data Collection, Sentiment analysis using SVM, Logistic Regression and Naïve Bayes
Apurba Sengupta	Feature selection, Sentiment analysis using LSTM-RNN (Using PyTorch)
Chinmay Shah	Feature selection, Sentiment analysis using LSTM-RNN (using MATLAB R2018)
Vimell Yuvarajah	Data Collection, Sentiment analysis using SVM, Logistic Regression and Naïve Bayes

Code Submission:

Kindly find the code for this project at the following Github repository:
`https://github.com/DeepanS206/oracle`

Abstract

Efficient Market Hypothesis is an investment theory that states it is impossible to "beat the market" because stock market efficiency causes existing share prices to always incorporate and reflect all relevant information. Over the years a lot of research has been carried out in the field of prediction of stocks to prove its failure. In this project, assuming that the news articles have an impact on stock prices, we predict the future stock trend using financial news articles from various sources relating to certain tech companies with news sentiment classification. Here we use four different classification models which outline the polarity of news articles being positive or negative. Results show that SVM model outperforms all the other models.

1 Introduction

The problem we try to address in this project is a classification task of predicting whether the stock price of certain tech companies (like Apple Inc., Amazon, Facebook, etc.) will go up or down, based on news about the company. We obtain the data for this task by querying available online APIs for news articles from various websites like the Wall Street Journal, Reuters and Bloomberg. Based on the data gathered from these websites, we then perform sentiment analysis using different Machine Learning algorithms in order to develop a predictive model that is able to associate a positive or negative sentiment to the corresponding news articles.

2 Related Work

1. Sentiment analysis of news articles for financial signal prediction - Jinjian (James) Zhai (jameszjj@stanford.edu), Nicholas (Nick) Cohen (nick.cohen@gmail.com)

2. Application of Machine Learning Techniques to Sentiment Analysis - Anuja P Jain, Computer Science and Engineering, Gogte Institute of Technology, India - jainanu04@gmail.com

3. Stock Trend Prediction using News Articles - A Text Mining approach - Pegah Falinouss.

3 Data Set

In order to have more flexibility with the data set, we decided to scrape data from the Web using Python. The advantage of scraping our own data is that we could search for news of specific companies. This then allows us to tag the news with a positive or negative sentiment. The labeling was nodes by taking the price difference of the specific company's stock two days after the news was released. Price difference being positive implied the article contained positive sentiment and vice versa. In this attempt, we made use of several API's from different new sources to fetch the data. We initially considered news sources from tech news sites such as Techcrunch, Techradar, and Engadget. However, we realized that the news articles were rather irrelevant to the stock price's movement. Hence, we restricted our sources to financial news sources such as Wall Street Journal, Reuters, Bloomberg etc. The Python library, `newspaper`, was used to parse contents of the news article which contained important information such as the news article along with other information such as the date of publishing, the URL of the article, the site where the article is hosted, etc. We then split the data set to train and test data of 80% and 20% respectively.

4 Problem Formulation

We try to learn a binary classifier $h_S(x)$ that associates an article x_i about a certain company with label $y_i = +1$ if the stock of the company is predicted to increase in value. Similarly, a label $y_i = -1$ is associated if the stock is predicted to decrease in value. We assume that the sentiment relating to a stock increasing or decreasing in price is similar for all articles regardless of company (since we only consider the tech sector, we believe this is a safe assumption to make) Thus, all articles about all companies are pooled together, where instance x_i represents an article and label y_i represents whether the stock went up or down. The articles are transformed to feature vectors using different feature representation models such as TF-IDF, word-embeddings, etc. We then use different Machine Learning algorithms in order to learn predictive models that can capture the sentiments attached to the articles.

5 Algorithms

We used different Machine Learning algorithms for the task of text classification. These include:

1. **Support Vector Classification**

   We expect the data which signals positive and negative movements in the market to be linearly separable. This is because we believe that the frequency of "positive" and "negative" words will be different for both types of data points causing the "positive" data points and "negative" data points to be clustered in different areas. As the project was coded in Python, we used the SVM package from scikit-learn to train and evaluate the SVM weights. We considered different kernels, linear, polynomial and RBF kernel.

2. **Logistic Regression**

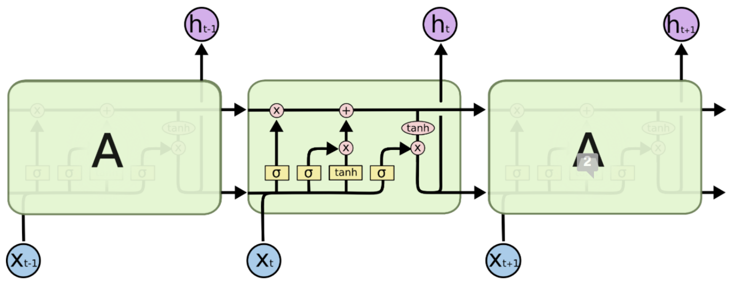
   As an alternative, we decided to relax the constraint of largest margin classifier and used the logistic regression framework to train a linear classifier. We used the LogisticRegression package from scikit-learn to train the linear classifier.

3. **Naïve Bayes**

   In order to explore different possible modeling techniques, we considered the conditional independence assumption for the sentences in the corpus. We expect the conditional independence assumption to capture the positive and negative words in the text data and assign higher probabilities that the specific sentence has a positive or negative sentiment respectively. We used the Naïve Bayes package from scikit-learn to learn the probabilities of each word given positive and negative sentiment.

4. **Long short-term memory Recurrent Neural Network**

   We utilized a deep learning model called a Recurrent Neural Network (RNN) that is used for natural language processing (NLP) applications such as text classification, speech recognition, machine translation, image captioning, etc. The beauty of the RNN architecture compared to a simple feed-forward neural network architecture is that it utilizes the activations from previous time step as an additional input to induce term-dependencies. To address the problem of vanishing gradients during back-propagation, we used a specialized RNN cell/unit called long short-term memory (LSTM). An LSTM unit consists of three gates/switches, update, forget and output gates, that help to capture long-term dependencies in the text. The dependencies on features far ahead in time is captured using bidirectional LSTM (BiLSTM) units. The training and testing procedures are carried out in mini-batches, where each mini-batch consists of zero-padded matrices (GloVe dimensions × highest number of features in the mini-batch). We used PyTorch and MATLAB R2018 for development purposes.



6 Experimental Design and Results

To utilize the algorithms discussed above, a single article had to first be converted to a vector representation, or a sequence of vector representations. This allows to reason about articles mathematically, without relying on the actual text of the article. The challenge in this process, however, is to create vector representations for a given article, that preserves the sentiment expressed in the article. We based our project on the following approaches to vectorize articles.

1. **GloVe word-embeddings**

   GloVe or Global Vectors for Word Representation is an unsupervised learning algorithm to learn vector representations of words based on word co-occurence statistics from a corpus of words. In this project, we use word-embeddings in various forms; we use the raw word-embeddings in the LSTM-RNN classifier and average & weighted average of word-embeddings for SVM, Logistic Regression and Naïve Bayes classifiers, where the weights are defined as a positive or negative integer depending on whether the corresponding word has a positive or negative context. Note that the word embeddings are of various dimensions: 50, 100, 200, 300. Each dimensionality was used on its own to construct document vectors.

2. **TF-IDF Vector Representation**

   The TF-IDF vector space model is an extension of the Bag of Words model, where a news article is represented as a collection of its words. Each word is associated with a tf-idf value, and the vector representing the document consists of the collection of tf-idf values of all the words in the article. Both unigram as well as bigram vector representations were employed.

3. **Doc2Vec Vector Representation**

   The Doc2Vec model focuses on capturing document information as a whole, in the same way Word2Vec encodes semantic relationship between words. The way Doc2Vec vectors are trained are very similar to the process in which Word2Vec vectors are learned. The process involves maintaining and extra vector representing the entire document, when training the word2vec model. Thus, after training is finished, the model contains vector representations for both the words as well as the documents used in training. The network can also be used afterwards to infer vector representations for newly seen documents (this is used to generate vector representations for the testing data).

Using the models of vector representation detailed above, several classification tasks were carried out. The training and test accuracy for each model learned is provided below (see table at end). The performance measure we used to evaluate the quality of the model is 0-1 classification error. We note that the representations of training and testing data used by the neural network algorithms differ from that used by the general classification (i.e. SVM, Logistic Regression, Naive Bayes). This is for reasons relating to batch training of the RNN explained below. However, the data used to train the models are the same.

1. **Support Vector Classification/Logistic Regression**

   Support vector classification with tf-idf vectors provided the best testing accuracy, compared to the other methods of vectorization. As we had expected, a linear kernel provided the highest testing accuracy values, since expect the data to be linearly separable. Cross Validation was used to determine the regularization parameter. Results showed that a parameter value of 0.8 provided the highest cross validation accuracy. Similarly, with Logistic Regression, cross validation was used to produce a lambda value of 1 when regularizing with the L2 norm.

2. **Naïve Bayes**

   The Naïve Bayes model was used to test the effect of modeling the entire joint distribution of the instance space as well as label space. We did not expect the Naive Bayes model to perform better (due to the lack of enough data), and as expected, the model gave accuracy scores lower than that of SVM or Logistic Regression. Cross validation was used to finalize the smoothing parameter for Naive Bayes, which was found to be 0.6.

3. **Long short-term memory Recurrent Neural Network**

   We used word-embedding representation of input features using 50d and 100d GloVe and iterated over different architectures of the LSTM-RNN using PyTorch and MATLAB R2018. The variation of the sorted document lengths against the sequence of documents is shown in Appendix. We restrict our document count in the range where the gradient on sequence length vs sequence number graph was not very high as the padding in such cases would be very high. In the graph in Appendix, we restrict ourselves to documents from number 100 to 800 having lengths between 80 and 400; thus having a total of 700 documents, from which we take 550 documents as train set and 150 as test set.

   PyTorch architecture – The architecture consists of a sequential input layer connected to a unidirectional LSTM network (with `tanh` activations). The last unit of the LSTM layer is fully connected to a 32-unit layer, which is then connected to a `softmax` layer that outputs the probability estimates of the sentiment attached with the input text sequence. The number of hidden features used are 44 for 100d GloVe and 42 for 50d GloVe.

   The loss function used is cross-entropy/log loss and the weight updates are done using the Adam optimization algorithm with a learning rate of 0.001. The training and test sets consist of mini-batches of 50 documents for both 100d GloVe and 50d GloVe. We use Google Colaboratory (Colab) in order to utilize GPUs for parallel computing capabilities.

MATLAB architecture – The architecture consists of a sequential input layer connected to a BiLSTM network (with `tanh` activations). The last unit is connected to a `softmax` layer which outputs the probability estimates of the sentiment attached with the input text sequence. The number of hidden features used are 120 for 100d GloVe and 100 for 50d GloVe.

The loss function used is cross-entropy loss and the weight updates are done using the Adam optimization algorithm with a learning rate of 0.001. For 100d GloVe, the training set consists of mini-batches of 50 documents and for 50d GloVe, the mini-batch size for the same is 25 documents. The test set mini-batch size was validated and the optimal batch-size were obtained as 55 and 10 for 100d and 50d GloVe respectively.

The above mentioned algorithms and feature representations gave the following results,

Algorithm	**TF-IDF (Unigram)**	**TF-IDF (Bigram)**	**Doc2Vec**
SVM	**0.72**	0.70	0.57
Naïve Bayes	0.65	0.66	0.55
Logistic Regression	0.69	0.68	0.63

Algorithm	**50d GloVe**	**100d GloVe**	**200d GloVe**	**300d GloVe**
SVM	0.57	0.56	0.53	0.50
Naïve Bayes	0.59	0.54	0.62	0.59
Logistic Regression	0.58	0.56	0.59	0.53
LSTM-RNN (MATLAB R2018)	0.610	**0.672**	–	–
LSTM-RNN (PyTorch)	0.607	0.58	–	–

7 Conclusion and Discussion

In conclusion, we observe that SVM algorithm under TF-IDF (unigram) and TF-IDF (bigram) feature representation of the given data gives the best results. As expected the polarity of positive and negative sentiment is well captured by SVM since the frequency of positive and negative words are different causing the positive and negative data points to be clustered in different areas.

The quality of data highly affects the accuracy of sentiment prediction. Since the articles collected from various sources have large number of words we need to pre-process the data to achieve better results. In this case, performing lemmatization and stemming, and removing stop words and proper nouns in the sequential input data (in the case of LSTM networks) boosted the accuracy by about 7%. Additionally, deep learning models usually need to be iterated over and over with different configurations and hyperparameters till a desired result is obtained. So, exploring a vast variety of models could lead to better accuracy results.

Further, based on the misclassification error obtained, we observe that implementing learning algorithm on news articles just based on company specific news has less impact on the stock prices. There are several other factors which should be taken into account. As an example we can say that a positive news of Samsung (like releasing a new product) has positive impacts on Samsung stocks but might have negative effects on Apple stocks. Similarly, positive news for one sector maybe negative for some other sector. Hence, if such co-relation are taken into consideration, then accuracy of stock prediction can be increased further.

8 References

1. Le, Quoc; Mikolov, Tomas; "Distributed Representations of Sentences and Documents", 2014

2. Mikolov, Tomas; et al.; "Efficient Estimation of Word Representations in Vector Space", 2013

3. Jeffrey Pennington, Richard Socher, and Christopher D. Manning, "GloVe: Global Vectors for Word Representation", 2014

4. Hochreiter & Schmidhuber, "Long Short Term Memory networks", 1997

5. Understanding LSTM Networks, `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

6. Sequence Models and Long-Short Term Memory Networks, `http://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html`

7. Coursera course on Sequence Models, `https://www.coursera.org/learn/nlp-sequence-models`

8. News API: `https://newsapi.org/docs`

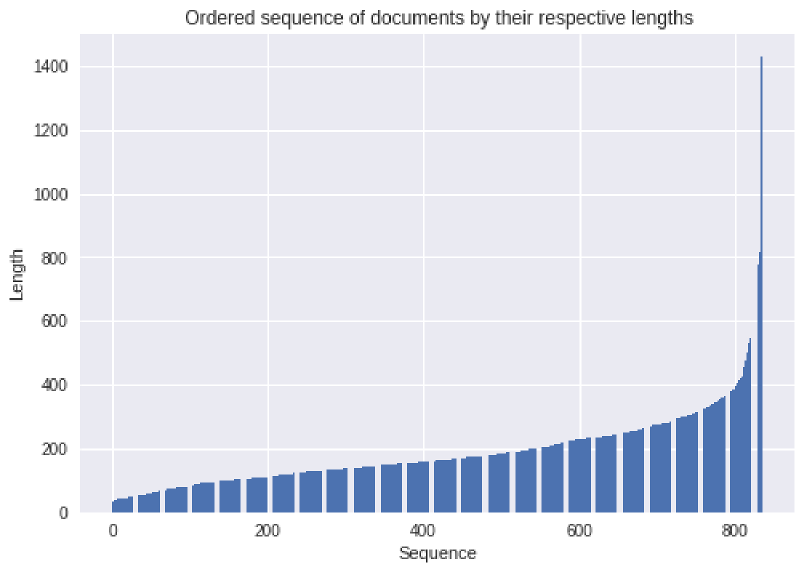
9. Newspaper3k: Article scraping & curation, `http://newspaper.readthedocs.io/en/latest/`

10. Genism: Doc2Vec implementation (API for Python), `https://radimrehurek.com/gensim/`

11. Scikit-Learn: ML Library user for SVM, Logistic Regression, and Naive Bayes, `http://scikit-learn.org/stable/`

12. Deep Learning Development with Google Colab, TensorFlow, Keras & PyTorch, `https://www.kdnuggets.com/2018/02/google-colab-free-gpu-tutorial-tensorflow-keras-pytorch.html`

Appendix


Ordered sequence of documents by their respective lengths
Length
Sequence
0
200
400
600
800
1000
1200
1400
0
200
400
600
800