Predicting Short Term Stock Prices with News Data and Recurrent Networks

Brian Falkenstein
University of Pittsburgh
Dr. Milos Hauskrecht
CS2750 Machine Learning Final Project

1 Abstract

In this project, I explored the impact that relevant data from the news could have on predicting short term stock prices. Intuitively, stocks react in the short term to political and economic events. By combining recurrent features from historical stock data with latent representations of relevant news articles, we hope to see a boost in accuracy in the short term. Although the results are unconvincing when applied to a vanilla RNN, the news features helped to improve accuracy and stability in an LSTM based model. Further studies, particularly with a much larger dataset, must be conducted to better determine the impact of news features on stock prediction.

2 Introduction

Short term trading, or active trading, is a method of trading stocks where a position is held for a short period of time, no more than a few days. This is quite different than a traditional buy and hold technique, where investments are made with the assumption that over long periods of time, there will be positive returns. Short term trading seeks to profit off of short term market volatility in response to key economic data releases, company earnings, and political events [6]. Short term trades attempt to capture short term market trends, and thus is speculative. This means it comes with considerably more risk than long term holding strategies, but can lead to considerably more short term profit.

The short term market fluctuations that we want to profit from in active trading are often influenced by news events, both economic and political [2, 10]. These events may include news that a company is projecting future grown, which may cause more investors to buy the stock, causing a price increase. Or perhaps word that new sanctions will be put in place which may impact a specific industry, causing investors to sell off stocks, and a decrease in price.

Machine learning techniques have been applied to the task of predicting both short and long term investing with considerable success. However, to my knowledge, there have been no efforts to explicitly incorporate outside news information to influence the model output. Intuitively, if we wish to improve accuracy in predicting short term market trends, we should attempt to incorporate news data in some way.

3 Related Works

As mentioned, many efforts have been made to utilize machine learning in predicting the stock market. In [8], the authors applied several neural network models to 90 economic indicator features to predict market momentum (when a successful company continues to be successful) and reversal (when an unsuccessful company becomes successful) with great success. In [3], they again applied a simple neural network, as well as a regression model, onto 1 second resolution trading data. This model was successful, but only up until 2009, when it began losing money. The purpose of the study was to show that short term trading is more difficult now than it was previously, suggesting what they say is an increase in weak form market efficiency. Note that both of these methods evaluated their models by simulating trades that the model would make, and computing profit over a time period.

The idea of incorporating news data into the predictions was inspired by other members of my labs experiments into multi modal learning, such as in [14], where they employ a multi modal latent space for words and images to better understand visual advertising. Multi modal machine learning is an expanding field with many applications. Audio and video can be processed together to improve video captioning, or an images caption can be incorporated into its visual representation to improve segmentation. These applications and many more are explored in the survey [1].

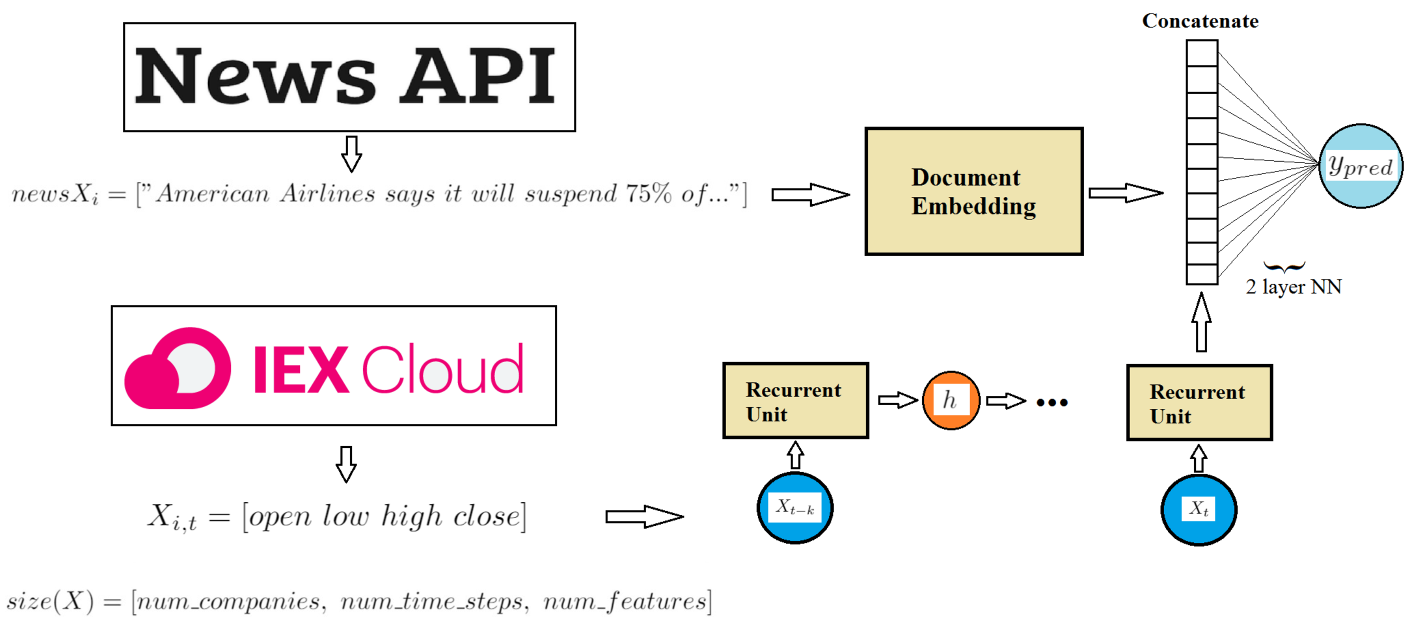
News API
$newsX_i = ["American\ Airlines\ says\ it\ will\ suspend\ 75\%\ of..."]$
Document Embedding
Concatenate
y_{pred}
2 layer NN
IEX Cloud
$X_{i,t} = [open\ low\ high\ close]$
Recurrent Unit
h
X_{t-k}
Recurrent Unit
X_t
$size(X) = [num\_companies,\ num\_time\_steps,\ num\_features]$

Figure 1: General overview of the approach taken in this study.

4 Methods

4.1 Data

A big challenge with this project was the accumulation of necessary data. I wanted the model to be able to work in real-time, and thus it must be capable of watching stock prices in real time, as well as finding relevant news articles. This required a two step approach, where historical stock data was accumulated for a list of companies using IEXCloud [5], and from that list of companies, news articles from the same time period are fetched using News-API [11], using the company name as a search term. Overall, I was only able to get historical financial data on 39 companies, a list of which can be found in *data/companies_list.txt* in the code repository. This was due to IEX clouds limits on how many queries could be made, and despite paying for premium service, I was only granted a very limited number of queries, and was able to get 1 months historical data for the 39 companies, with 1 data point for each day, consisting of the features [*open*, *low*, *high*, *close*, *volume*]. The final feature, *volume*, was ignored because of the massive disparity in range of values (although this could be fixed via normalization). Thus, we have a 4 dimensional feature vector to describe each company at each time step.

Every company queried had numerous articles found via NewsAPI during the studied time period. The articles were fetched from a large variety of news sites, over 50,000 according to the web site. In order to avoid any expensive pre-processing, only 1 article for each company was selected, that article being the one which was published closest to but before the date that we desire to do the prediction on. No attention was paid to the news source of the article. This could be a potential problem, as we do not know the bias of the source. Each article had its first 250 characters isolated, for use in a document embedding. Often, news sits will incorporate summary information in the first paragraph in an article in an attempt to reel in readers, so intuitively we are trying to capture the semantics of this summary.

4.2 Document Embedding

Document embedding is the process of converting a document, or list of sentences, into a fixed length 1 dimensional vector to be used in further machine learning applications. Ideally, the embedding encodes semantic information about the document, relevant to whatever task you are trying to accomplish. Many methods of document embedding exist, as it is a very hot topic in the field of Natural Language Processing (NLP), but we will be specifically using $Doc2Vec$ as it was proposed in [7].

$Doc2Vec$ seeks to improve upon previous embedding schemes by addressing their lack of attention to word orderings and semantics using machine learning (some previous approaches, like bag of words, required no learning). In order to understand how $Doc2Vec$ forms a representation of a document, one must understand the framework it is built off of, $Word2Vec$ [9].

$Word2Vec$ creates fixed length 1 dimensional vectors representing individual words in a corpus. It does this by first initializing a vector representation for each word, and uses this to predict a neighborhood of words in that words context. It then maximizes the probability of finding the true words in the training data. More formally, for a sequence of training words $w_1, w_2...w_T$ it seems to maximize:

$$\frac{1}{T}\sum_{t=k}^{T-k} log(p(w_t|w_{t-k}...w_{t+k})$$

Where k is a parameter of the network, defining the window size (size of the context of each word). After training, $Word2Vec$ maps words that appear in similar contexts, as well as words that have similar meaning, close together in the latent space. These word vectors have been succesful in a variety of tasks, including machine translation and language modeling.

$Doc2Vec$ functions in a very similar way. However, in addition to having a vector representation for each word, we have a vector representation for each document. This document vector is used in conjunction with the word vectors to predict words around a target word. This can be foramlized in a very similar equation to the $Word2Vec$ one, with additional paragraph vector P_i for document i:

$$\frac{1}{T}\sum_{t=k}^{T-k} log(p(w_t|P_i, w_{t-k}...w_{t+k})$$

The introduction of the P_i term allows the model to learn representations of the documents that help in the task of word context prediction. Testing on these document vectors showed very good performance for the sentiment analysis task, where the sentiment (happy, sad, angry...) of a paragraph must be inferred.

The goal of using this document embedding technique is to capture the overall sentiment of the news articles collected, with the intuition that a good article may cause an increase in a stocks price, and a sad or angry article might make a stocks price go down. Note that the dimensionality of the produced vector can be altered (which requires retraining the model).

4.3 Recurrent Modules

In addition to utilizing news articles in the prediction, we want to study a stocks prices over a series of days to attempt to determine a trend which we can use in the prediction. Pricing data is acquired for each stock daily, producing an inherently ordered sequence of data points for each stock. It is widely accepted that Recurrent Neural Networks, or RNN's, are the state of the art for processing time series data, such as what we have in this problem. For that reason, we will be testing out two types of RNN's for processing the sequenced financial data: a vanilla RNN, and a Long Short Term Memory (LSTM).

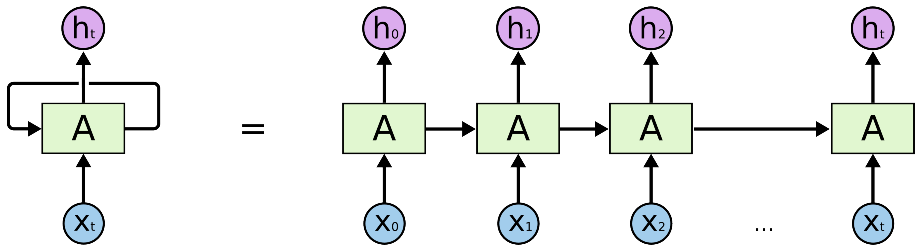

Figure 2: A basic, 'unrolled' recurrent neural network. At each time step, the module takes in as input a hidden representation from the previous step, as well as new input x. The module outputs a new hidden state, and an output h for each time step. Figure from [13]

The most basic recurrent module is a simple recurrent neural network, where at each time step, a new hidden representation is computed and passed to the next time step, and an output is produced. A visualization of this can be seen in figure 2. This is implemented with 2 linear layers, both of which take as input the concatenated input and hidden vectors: one which produces the output, and one which produces the next hidden representation (because the dimensionalities of the two need not be the same). So, for a sample X_i at time step t, the output and new hidden vector can be computed as:

$$out_{t+1} = f(concat(X_{i,t}, h_t), w_1) + b_1$$

$$h_{t+1} = f(concat(X_{i,t}, h_t), w_2) + b_2$$

With w_1 and w_2 being separate weight vectors. Thus, we operate on all the time steps in a sample X_i.

The LSTM functions in a very similar way, with a few additions. First proposed in [4], the LSTM seeks to allow for better long term dependencies, by introducing additional 'gates' whereby the input and hidden states are processed. It also introduces another vector to describe the module at a given time, called the cell state. The state from the previous time step passes through the forget gate, which determines how much information to either keep or throw away from previous time steps. The input passes through the input gate, which determines how much the new input will impact the future cell state and output. The output gate then computes the next hidden state, as a function of previous hidden state, cell state, and new input. A visualization of this can be seen in figure 3. Although the RNN was implemented from scratch using $PyTorch$, the LSTM was implemented using the already defined LSTM model in the *pytorch.nn* library.

For both methods, we are only interested in the output at the last time step. In the baseline case (with no news data), this output will be a single value which is the predicted value of the stock for the next time step. In the case where we wish to incorporate news data, the output of the recurrent unit is a fixed length vector (of size $(|hidden| + |input|)/2$) which we then concatenate with the document vector computed with $Doc2Vec$ and feed that into a 2 layer neural network (with ReLU activation) to compute the predicted price for the next time step.

Models were trained with adaptive learning rates which halved every 10 epochs ($\alpha_{init} = 0.001$ for RNN, $\alpha_{init} = 0.01$ for LSTM). The loss function used for all models was the Mean Absolute Error (MAE) computed as

$$\frac{1}{n}\sum_{i=1}^{n} |y_{pred}(x_i) - y_i|$$

The reason for MAE instead of squared error is that error values can sometimes be very large, given the range of values in the dataset. Thus, when a large error is squared, the loss landscape may become impossible to navigate, as the derivative approaches positive or negative infinity. This is a phenomenon known as exploding gradient, and is common in RNN's.

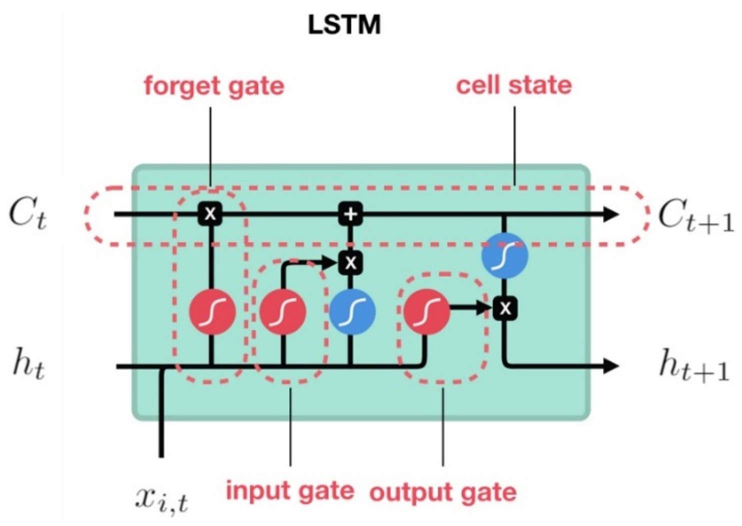

Figure 3: An LSTM unit at time t. Image originally from [12].

5 Results

There are multiple parameters that can be modified in this experiment to see how they impact performance. For the experiments, I attempted to hone the performance on the baseline RNN / LSTM models, and then compare these performances with models that have the same parameters, but include the news features. Parameters that were modified for the recurrent units are: dimensionality of hidden state, and the sequence length of the input data (number of time steps). Altering the dimensionality of the document vector was also studied.

Results were computed by randomly splitting the dataset 80/20 into training and testing sets, training the model and evaluating on the test set. This was repeated 10 times for each model, to obtain a more robust performance metric.

Table 1: RNN Test Errors

Hidden Dim	Sequence Length	MAE
10	10	1.761 +/- 0.640
20	**10**	**1.333 +/-0.536**
30	10	1.538 +/- 0.822
50	10	1.791 +/- 1.215
20	1	1.710 +/- 0.8188
20	**5**	**1.325 +/- 0.606**
20	20	2.061 +/- 1.022

Table 2: RNN with News Features Test Errors, Hidden dim=20 Sequence length=5

Document Feature Dim	MAE
5	1.722 +/- 0.371
10	**1.574 +/- 0.597**
20	1.726 +/-0.580

Table 3: LSTM Test Errors

Hidden Dim	Sequence Length	MAE
10	10	107.14 +/- 68.427
50	10	62.037 +/- 44.274
100	**10**	**55.155 +/- 25.839**
100	1	72.409 +/- 65.199
100	5	66.954 +/- 60.184
100	20	88.994 +/- 67.606

Table 4: LSTM with News Features Test Errors, Hidden dim=100 Sequence length=10

Document Feature Dim	MAE
5	56.633 +/- 19.001
10	49.607 +/- 11.433
20	**48.870 +/- 10.467**

6 Discussion

Clearly, the LSTM performed significantly worse than the simple RNN, which achieved very decent scores. The LSTM was likely harder to fit to the small data set, given that it has significantly more parameters.

Among the RNN results, it seems a hidden dimensionality of 20 was optimal. This is likely because it allows it to capture enough data in the latent space, without overfitting. We also see that shorter sequences of input data perform just as well / better than longer sequences. Sequence length 1 (just looking at previous day) achieved a decent score compared to sequence length 20. Further, sequence lengths 10 and 5 have similar performance, but shorter sequence lengths lead to faster convergence, so we choose parameters $hidden\_dim = 20$ and $sequence\_length = 5$ for comparing RNN with and without news features.

Addition of the news features, along with the 2 layer neural network for the regression, showed a slight decrease in performance when compared to the baseline RNN. Although the best performance with document dimensionality 10 is comparable, and better than, many of the baseline RNN models, we cannot say that the news features showed an improvement in performance for the RNN.

The LSTM scores show that the model is nearly useless in most cases, achieving incredibly high average absolute errors on the test sets, as well as very high variance, indicating instability in addition to inaccuracy. Generally, increasing the hidden dimensionality improved performance in the baseline LSTM, with best performance achieved with a hidden dimensionality of 100, significantly higher than that of the RNN. Unlike the RNN, the LSTM sees a drop in performance when looking at shorter sequences of input data, indicating that it requires those longer sequences to learn anything useful. However, too long of a sequence again shows a drop in performance, with the optimal sequence length being 10. Thus, we compare LSTM with and without news using fixed parameters $hidden\_dim = 100$ and $seq\_length = 10$.

Interestingly, addition of the news features to the LSTM model actually showed a significant boost in performance. With document feature dimensionality 20, the LSTM with news model achieved significantly MAE than any of the baseline LSTMs. Additionally, all of the LSTMs with news features were much more stable than the baseline LSTMs, showing significantly lower variance in test errors.

Overall, I believe that perhaps the LSTM was better able to learn with the news feature data than the RNN, which is why it showed better performance with the news feature than without. Perhaps, with a more reasonably sized training set, the LSTM would be better able to converge and achieve better performance than the RNN.

7 Conclusion

In this study, we were able to show a boost in performance in short term stock price prediction by incorporating relevant news data, but only with the LSTM models. This makes the results inconclusive, as the LSTM results were still fairly unnacceptable, even with the news features. Although the significant boost in performance from the news features is still interesting.

There are many further studies that could be done to make these results more conclusive. For starters, a much bigger dataset could be accumulated. Given more access to the IEXCloud API, I could get more historical information for more companies, which can then be linked to news articles from the relevant time period. A large dataset is very important for training more complex deep networks like the LSTM. Additionally, further exploration of the hyperparameters could be done, potentially via meta-learning (wherein model parameters are learned along with the weights of the model). Further, additional vector embedding methods can be explored (latent semantic analysis, bag of words, auto-encoders), as well as different recurrent network frameworks (GRU).

One other thing that could make the results a lot more clear would be to test this method in a similar way as [3] and [8], where they tested their model in a simulated environment where it could act on its decisions by trading stocks. These results could be compared to more traditional strategies of buy and hold. Model performance can then easily be interpreted as how profitable the model was over a time period when compared to other methods.

References

[1] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. Multimodal machine learning: A survey and taxonomy, 2017.

[2] Brian Beers. How the news affects stock prices, Feb 2020.

[3] David Byrd and Tucker Hybinette Balch. Intra-day Equity Price Prediction using Deep Learning as a Measure of Market Efficiency. Papers 1908.08168, arXiv.org, August 2019.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[5] IEX. Iex cloud api: Iex cloud.

[6] IG. Short-term trading strategies for beginners.

[7] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents, 2014.

[8] Zhixi Li and Vincent Tam. A machine learning view on momentum and reversal trading, Oct 2018.

[9] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[10] Mark L. Mitchell and J. Harold Mulherin. The impact of public information on the stock market. *The Journal of Finance*, 49(3):923–950, 2020/04/22/ 1994.

[11] NewsAPI. News api - a json api for live news and blog articles.

[12] Michael Nguyen. Illustrated guide to lstm's and gru's: A step by step explanation, Jul 2019.

[13] Christopher Olah. Understanding lstm networks, Aug 2015.

[14] K. Ye, N. Honarvar Nazari, J. Hahn, Z. Hussain, M. Zhang, and A. Kovashka. Interpreting the rhetoric of visual advertisements. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2019.