Predicting Nasdaq Movements using Daily News

Sushant Bansal | Github |sushant@uchicago.edu

Abstract

In the past few years we have seen how recurrent neural networks can be heavily effective in processing and realizing sequential data. Here we do a comparative study of various Machine Learning Models such as Logistic Regression, MLP and RNNs (Word and Character Level) to process daily news and predict the NASDAQ index trends based on daily news articles.

Introduction

Predicting the stock market behaviour is an area of heavy research and is influenced by too many factors, which makes it really convoluted to predict the movements. Even a little bit better predictions from the competitors can yield huge profits in the financial markets, which makes this area even more exciting because there is a clear monetary benefit.
The daily commentaries about the stock market in general or specific companies influence the investors a lot, which can move the market in both directions according to the sentiment of the news. Also, in this age where information travels at such a high pace we can see the markets move minutes after a breaking story. We can draw some parallels, from the 'security theatre' concept which states making people believing that they are safe is more effective than actually making them more safe. That is saying that the sentiment about the news can have a huge impact on the market. And we're trying to quantify that correlation.

Due to the advancements in the Machine Learning community with the introduction of models such as Recurrent Neural Networks and related work, we are now able to process text sequences much more powerfully than before. Also, the introduction of word embeddings such as Glove [1]. These embeddings can help the machine understand the natural language better and give significantly better results.
We use these advancements and do a comparative study to find which models work the best and can be taken forward as next steps to build an automatic trading suggestion system.

[1] "GloVe - Stanford NLP Group - Stanford University." https://nlp.stanford.edu/pubs/glove.pdf. Accessed 28 Aug. 2019.

Related Work

The amount of publicly available research done on this topic seems inversely proportional to the amount of interest in the topic due to monetary reasons. Corporations heavily investing in this research do not want to lose the competitive advantage they gain from this research. But we still use two papers both trying to predict trends of the market using news.

- "Using NLP on news headlines to predict index trends", Marc Velay and Fabrice Daniel (2018)[2]
- "Stock Market Prediction with Deep Learning: A Character-based Neural Language Model for Event-based Trading", Leonardo d. S. Pinheiro, Mark Dras (2017)[3]

The first paper mentioned above works with a Bag Of Words model, Neural Networks and various other non deep learning approaches such as SVMs, Decision Trees etc to predict the DJIA Index using Global news headlines And they find that the bag of words Logistic Regression models, performs the best of all the other approaches, also no embeddings were used for any of these models. We investigate this claim and add embeddings to our models except the Bag Of Words Model.

The second paper listed here, works mostly with the deep learning approaches such as LSTMs on the character level with character embeddings, which they generate using a character language model on the dataset they use. And then they use the embeddings in the stock market predictions using a character level LSTM. They report the highest accuracy using character level LSTM with character embeddings, we try to replicate this finding.

Description

We're going to work our way through the topics as they come in the accompanying notebook. So that it's easy to follow and makes more sense.

We start by data collection, which was gathered from two sources:

- All The News,[4]. Collection of news articles
- Market Data, [5]. Collection of Nasdaq Index Data

[2] "Using NLP on news headlines to predict index trends." 22 Jun. 2018, https://arxiv.org/abs/1806.09533. Accessed 28 Aug. 2019.
[3] "Stock Market Prediction with Deep Learning - Association for ...." https://www.aclweb.org/anthology/U17-1001. Accessed 28 Aug. 2019.
[4] "All the News - Components." 18 Jan. 2019, https://components.one/datasets/all-the-news-articles-dataset/. Accessed 28 Aug. 2019.
[5] "Historical Intraday Nasdaq Composite (COMP) Price ... - FirstRate Data." http://firstratedata.com/i/index/COMP. Accessed 28 Aug. 2019.

Data Pre-Processing

Now because the time ranges, these datasets span are different we keep the data only which shares dates in both the databases. This involves making the date formats same and dropping all the data with malformed dates. We store this data separately still as pre_news and pre_market data. {Done by pre_pre_processing.py}
With the next step being combining the data. But now because we have the market data at intervals of a minute, we filter the data for the morning and ending price of the data and then take the difference from the previous row to get the opening and closing difference. Which we predict positive movement and negative movement. {Done by combine_data.py}
We'll be going through the notebook now {Named: MainColab.ipynb}.

Text Pre-Processing

Once we have the data ready we can start preparing our text pre-processing pipeline. The pipeline consists of three stages, normalizing the data i.e. removing the punctuation and making all the letters to lowercase. The major reason for lowercase is the embedding matrix. Which contains embeddings corresponding to words in lowercase. More on this later. Then we remove all the stop words which don't add too much meaning to the sentence. Keeping them might be useful in some cases but we drop them in our case. Once this is done, we turn to the most important part i.e. Lemmatization i.e. reducing the words into their root forms from their inflected forms. We avoid stemming here, even though it's way faster but it doesn't work so well with embeddings because it reduces words too much (features turns into featur, which isn't a word and no embedding exists for it).
We use NLTK[6] to get the correct part of speech tag and then lemmatize the word. It improves the lemmatization accuracy, by default lemmatizers consider every word to be a noun. All this preprocessing is done in parallel using python's in built multiprocessing module.(We divide the data frame and use a pool to process text in parallel) Needless to say, that we use Pandas[7] ,Numpy[8] and PyTorch[9] throughout the notebook.
Now, we drop the rows which have articles of length more than Torchtext[10] can read in, we lose less than 0.01% of the data by doing this, so we just drop those outliers.

[6] "Natural Language Toolkit — NLTK 3.4.5 ...." https://www.nltk.org/. Accessed 28 Aug. 2019.
[7] "Pandas." https://pandas.pydata.org/. Accessed 28 Aug. 2019.
[8] "NumPy — NumPy." https://www.numpy.org/. Accessed 28 Aug. 2019.
[9] "PyTorch." https://pytorch.org/. Accessed 29 Aug. 2019.
[10] "pytorch/text: Data loaders and abstractions for text and NLP - GitHub." https://github.com/pytorch/text. Accessed 28 Aug. 2019.

To make ingesting data easier into Torchtext we split the pre-processed data into training, validation and test sets using the script {split_data.py}. Which also stores the data onto disk, so that we don't have to pre-process the data again and again. This saves a lot of time. There are pre-processed samples of processed data in the {ProcessedData} folder in the repository to test the working of the code.

Once the data is pre-processed and loaded into Torchtext we can start talking about the Models.

Models (and helpers)

We work on four different models, we'll go through each of them, one at a time.

Bag of Words Model

The first model we discuss is the bag of words model.

The data loader for this model turns each article and turns it into a vector of length of the vocabulary, with each word being incremented each time we see the word in the text.
We use this long vector and pass it through a single layer network (Logistic Regression) that outputs two nodes. One for positive and other for negative movement.
The training for this model is done via the training module written for this model which trains in batches and also runs on GPUs if available. Otherwise PyTorch will just run the training on as many cores available. Also, we use CrossEntropy Loss and Adam as the optimizer. There are no dropouts in this model but we do use early-stopping to prevent overfitting, we keep the model which performs best on the validation set.

This model takes the longest to train even being the simplest of all the models because of the huge size of the input vectors.

Training Module

We take a segue here to talk about the Training Module because it is shared by the rest of the models. This module contains functions for training and evaluating models. It takes in the model object to be trained, when initializing the module. And then we call the train_model function, which takes in the training and validation set iterators. Early stopping is also implemented here, we deepcopy the best performing model on the validation set. The function in the end returns the model which performed the best on the validation set, which we then use to evaluate the test data.

Glove Loader

Continuing the segue, we have a few functions that help us load Glove Embeddings and create the word to index and index to word mappings as well as creating the Embedding Matrix, which can be fed to PyTorch. PyTorch has a function which can ingest embedding matrices.

Neural Network Model

This model uses a different data loader which clips the sequences to a max length of 1200. And each number in the vector is the word corresponding to that index in the dictionary. But the main difference here is the introduction of word embeddings from Glove. We averaged the embeddings, for all the words in the sequence and pass that into the neural network with two hidden layers and one output layer which are all fully connected. We just use one output here because of binary classification. We again use CrossEntropy Loss but PyTorch has a separate cross entropy loss for the binary cases called Binary Cross Entropy Loss (BCE Loss). We use the one with Logits (BCE Loss with Logits) which just an extra sigmoid activation included i.e. it applies it directly to the outputs and we don’t have to do it. The optimizer used is Adam again.
This model is fastest to train and when running on GPU it really shines. We get very good accuracy with this model.

Word Level RNN Model

This model uses the same data loader as the previous model. But this model is conceptually very different from the previous model. Here we preserve the sequence nature of the news data. That is saying that we use a GRU to summarise the entire sequence of text which then we pass onto the fully connected linear layer to get the result. Each word is represented by it’s embedding (glove). The hidden layer is initialized to all zeros.
Also, we use the BCE Loss with Logits again and Adam as the optimizer.

This model is second in the training speeds. Slower than the Neural Network but still faster than others.

Character Level RNN Model

For this model we have to re-write the data loader because we work with characters instead of words. So the sequence is changed to length of 5600. Also we don’t remove the punctuation or case normalize the data, because we want as much diversity as much as possible because our vocabulary is very limited consisting of numbers, letters or both cases and printable punctuation.

We do something similar as we did in our previous model, such that we summarise all the sequence into the hidden vector and then pass that vector to a fully connected layer which then produces the output for us.
We use BCELoss with Logits again here as well as Adam optimizer.

This model comes second last in training speeds, again because of it being a RNN and a big vector size.

Results

We get some expected and some unexpected results from the models. But first look at the results below.

MODEL	**MAX TRAIN ACCURACY**	**MAX VAL ACCURACY**	**TEST ACCURACY**
Bag of Words	90.93	57.02	56.78
Neural Network	79.07	59.14	60.83
RNN Word Level	60.77	59.07	60.79
RNN Char Level	59.75	59.16	60.30

The results we got for the Bag of Words Model and the RNN Character Level were in line with the related material we were looking at but the RNN Word Level and Neural Network Model outperform the models. But that is because of the fact we added pre trained embeddings, which help the model learn a lot more.

A lot of people have said, you can’t predict stock markets reliably because there are too many factors involved. But we still see an accuracy of 50%+ which is not by random chance. With that in mind these results are respectable but we have to take them with a grain of salt, because the quality of the data wasn’t so good and we were training on the news that came even after the market was closed, which isn’t good. Because we added future information, which we were trying to predict.

The Bag of Words model, performs the worst because it just looks at the word and doesn’t impart any meaning to it. Just generalize the concept so well also loses the sequence nature of the data. The Neural Network model, performs the best with embeddings because of the meaning it has and we sort of summarize the sentence well which we pass to the network which adds multiple nonlinearities which helps approximate the true function more than just a single layer of Logistic Regression. The Word-RNN model works well, the summarizing of the text is done really well by the RNN which when fed to a single linear layer gives us good results as in comparable to the multi layer perceptron. The final model with the Character Embeddings and a RNN works well because it can capture the numbers like $100 or 2.69% better than any other which is very frequent in the market news. This kind of capturing helps summarise the data well which when fed to a single layer produces good results.

Future Work

Now, that we have seen the model does infact gets the predictions right more than 50% of the time we can fine tune it for specific stocks and train the model on the news articles that contain information about those stocks. With having predictions about individual stock is much more useful than predicting the index as a whole. Also, one big improvement that can be made is utilizing the time dependent nature of the market data. Right now the model looks at each day as independent of all the previous day which is definitely not true in the case of market data. We want the model to make a prediction using the current news and market data from the past. And making it multi-class by adding classes according to the percentage change in the market. This would make the model even more reliable and probably give us at least marginally better results.

Self Evaluation

The effort increased as time grew through the project but the stress was the maximum in the start. Collecting the data was the hardest part in the project. And then cleaning the data and then making the model work. I was comfortable with Numpy and Pandas but not so much with PyTorch, which I am now. Most of the time was spent trying out different parameters, it wasn’t so much of fine tuning but more of curiosity to see what change will this parameter cause.

We all learn better by applying what we learned in class and this project was no different. I personally can confidently say that this project helped me become way more confident about writing deep learning models and wrangling the data around.
At the start of the Spring quarter, I had no clue how to even make a logistic regression model and at the end of it, I was able to implement an unsupervised algorithm K-Means in 30 minutes. And now at the end of this course, I can make various deep learning models, play with the dimensions and not be afraid of any data challenges corporations throw at me when I’m seeking employment.
Also, I would like to mention that this course helped me gain confidence in reading papers, papers don’t scare me anymore and I enjoy reading them, which I feel is a great achievement. And all of this in just 10 weeks.

Bibliography

1. "GloVe - Stanford NLP Group - Stanford University." https://nlp.stanford.edu/pubs/glove.pdf. Accessed 28 Aug. 2019.
2. "Using NLP on news headlines to predict index trends." 22 Jun. 2018, https://arxiv.org/abs/1806.09533. Accessed 28 Aug. 2019.
3. "Stock Market Prediction with Deep Learning - Association for ...." https://www.aclweb.org/anthology/U17-1001. Accessed 28 Aug. 2019.
4. "All the News - Components." 18 Jan. 2019, https://components.one/datasets/all-the-news-articles-dataset/. Accessed 28 Aug. 2019
5. "Historical Intraday Nasdaq Composite (COMP) Price ... - FirstRate Data." http://firstratedata.com/i/index/COMP. Accessed 28 Aug. 2019.
6. "Natural Language Toolkit — NLTK 3.4.5 ...." https://www.nltk.org/. Accessed 28 Aug. 2019.
7. "Pandas." https://pandas.pydata.org/. Accessed 28 Aug. 2019.
8. "NumPy — NumPy." https://www.numpy.org/. Accessed 28 Aug. 2019.
9. "PyTorch." https://pytorch.org/. Accessed 29 Aug. 2019.
10. "pytorch/text: Data loaders and abstractions for text and NLP - GitHub." https://github.com/pytorch/text. Accessed 28 Aug. 2019.
11. Course H.Ws