William Frank
5/4/19
Final Report

GitHub Link: https://github.com/willfrank98/Machine_Learning_Project

Abstract

The stock market is an extremely complicated establishment which moves in ways that seem impossible to predict. However, like with any other mechanism, finding the right pattern of inputs and their associated importance will allow you to determine the output. One of the most obvious sources of information is the news, specifically such highly syndicated publications as the New York Times and Reuters. This paper hypothesizes that by using the term frequency-inverse document frequency representation of news headlines on a certain day one could predict the 10-day performance of the stock market.

Introduction

For this project I attempted to predict the stock market using news articles. The stock market is extremely complex, with an unknowable number of influences that act in ways that range from direct to unimaginably indirect. However, at its core the stock market reacts to the perceived value of a stock. This value is determined, in theory, by a company's ability to produce revenue, as well as the general trends of the market. That is to say, a company coming out with a new product is likely to see an increase in market value, though this could possibly be offset by any general recession in the market. Conversely, when the market is generally trending upwards, so will most individual companies.

Accurately predicting the stock market is something people have been trying to do since its inception, and there is no better time than the age of big data to do just that. The amounts of data currently available is unparalleled, as well as a machine's ability to process it. For this reason, machine learning is the perfect solution to attempt to predict the impossible. Machine learning algorithms can combine data in ways humans may never think to, and can try more combinations than any human could ever hope to manually. Additionally, there is the obvious monetary incentive to being able to predict the stock market.

At its core, this is a natural language processing, or NLP, problem. NLP problems typically deal with determining the emotion of a particular work of text, or assigning a part-of-speech to a certain word in a sentence. However this project attempts to assign something of an economic indication value to a selection of text by pairing a source of information with a measure of economic performance as input to a machine learning algorithm.

Process

Data Scraping

The first step of my pipeline is in nyt_scraper.py, which uses the New York Times Archive API. This API returns nearly all the information on all NYT articles for a given month, dating from today back to 1985. From these API calls I kept the articles' headline, abstract, news desk, document type, material type, and lead paragraph. All these fields were cleansed, which includes stripping irrelevant special characters, converting non-ascii characters to their most similar ascii counterpart (such as è to e), and converting to lowercase. The produced output is a single csv file for each year containing these fields for every returned news article for that year.

One old approach also included stemming, or lemmatization, as part of cleansing. This process replaces every word with its base, such as 'spending' to 'spend'. In theory this process removes any unnecessary meaning from indivudal words and decreases the overall vocabulary, but is somewhat out of place when analyzing news headlines due to the abundance of proper nouns. This process was also rather time-expensive, and resulted in no meaningful performance increase.

Data Combining
After this massive amount of data is downloaded it is combined in nyt_combiner.py. This script combines all valuable headlines (or another desired field, such as lead paragraph or abstract) for each single day in a given range of years. Valuable headlines are ones which contain business/economic-type news, and are not empty or useless (headlines such as 'Dividend Meetings'). These headlines-per-day aggregations are output into a single csv file, one day per line. Articles are determined to be business/economic-type if they come from a news desk which contains one of the following strings: ['business', 'commerce', 'career', 'invest', 'real estate', 'financ', 'job', 'foreign', 'national']. NYT appears to be inconsistent with assigning names of news desks, as well as changing their standards over the years, however any business/ economic-type news will most likely come through a news desk with a name similar to one of those strings. Unincluded news desks are primarily things such as the arts, theatre, fashion, editorials, reviews, and sports, which were deemed very unlikely to contain relevant business/economic-type information. Ideally all news articles would be considered, but this conflicts with practical data limitations of the environment these tests were run on. Another approach involved using the lead paragraph rather than the headline, which is far more information-rich, but again was too much data to feasibly work with.

After the news articles are combined into a day-by-day format, they must be combined with the relevant economic information in data_combiner.py. This script adds attributes such as the previous 10 days' performance of the Dow Jones Industrial Average, and the previous day's high, low, and trade volume. The DJIA was selected as a single good indicator of America's overall economic performance due to its collection of businesses from diverse fields. The label for a given data point is the DJIA's performance over the next 10 days, however this is created in the next script. Alternatively, the label could be modified so that the algorithm predicted the increase or decrease for any number of specific stocks. Once the data points have been augemnted they are saved again to a single csv, one day per row.

An alternative approach involved including past performance of the 30 stocks that currently make up the Dow Jones Industrial Average, however this had several flaws. While including as much additionally financial data as possible should only be a benefit, it goes against the spirit of the original problem, which is predicting market movements through the news. Secondly, including this data did not make a noticeable difference on prediction quality, and was time consuming to include. Another feature that was tested was the polarity and subjectivity of each day's headlines. These are common and typically useful measures in the field of natural language processing, specifically speech classification. However news headlines should not be subjective at all, and how polarizing they are should not have any noticeable effect on the market or investors. And as before, this attribute made no difference to performance.

One limitation of my current data combination approach here is that no weekend news is considered. Because there is no new market data on weekends and holidays, these days were removed from the final dataset. Potential ideas for incorporating this data are to roll it into the preceding Friday or following Monday, or to use market data from one of those days to make the weekends a seperate data point. It is hard to say if either of these approaches would be more valid, but they certainly merit investigation.

Final Preparation
The final data preparation is done in final_prep.py. This script first creates the new label for our data, the 10 day future performance of the DJIA, scaled relative to the closing value of the 10th day. Before any further feature transformation is done the data is split into train and test sets, so that test data is not taken into account when calclating feature scalings. Then the headlines are transformed into term frequency-inverse document frequency format, which means each headline is converted into a count of its tokens, and then those counts are scaled by the frequency of that term divided by the inverse frequency of that word in all documents. In essence, very common words will be downweighted while uncommon words will be given greater significance. The date field

is also removed]and blown up into several features designed to capture any seasonal stock market trends, such as day, month, day of week, day of month, quarter, and more. Finally all the attributes are normalized with scikit-learn's RobustScaler, which removes the median and scales the data to the interquartile range, doing so individually for each feature. This transformation was calculated using the training data then applied equivalently for train and test data. Finally the train and test sets are made into pandas DataFrames and output into separate HDF5 files, a format chosen for its speed with large datasets.

Previous label approaches were also considered here. The initial approach was to make this a binary classification problem, with the two labels being the market decreased, or the market increased or stayed the same, before the market closed the next day. However my models were unable to improve past almost always predicting the market to increase, something that happens on roughly 60% of days in my test data. I then changed the labels back into a linear regression problem which captured how much the market increased or decreased by, rather than simply whether or not it did. When this approach yielded no significantly better results it was deemed that 1 day is too short a time period to see any meaningful impact of news articles, and 10 days, or two business weeks, was decided on.

Training

For training and regression two main approaches were used. The first is a rather basic densely connected feed-forward neural network, implemented with keras from tensorflow. The first layer has 256 neurons, followed by 3 layers of 128 neurons, and finally an output layer with a single neuron. This architecture was chosen as the smallest which could properly prepare the input features. Making it significantly bigger slowed down training with no appreciable performance gains in the model, while making it significantly smaller runs the risk of an overly simple model. All neurons use a rectified linear unit for activation, with the exception of the output neuron which has no activation, and all neurons are initialized with the truncated normal initializer. The only difference between this initializer and a normal gaussian initialization is that values more than two standard deviations from the mean are discarded and redrawn. This is the keras' recommended initializer for neural network weights. The loss function is mean squared error, the standard for regression problems, and the selected optimizer is Adadelta. Adadelta is an optimizer which "adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients", from the keras documentation. The benefits of this optimizer include no manual setting of a learning rate (though you can), insensitivity to hyperparameters, individual learning rate per dimension, and minimal additional computation over vanilla gradient descent.

A neural network was chosen for its ability to create connections between data which a human could never imagine. This is a perfect quality for natural language processing tasks, as well as something as complicated as the stock market. In theory, the right neural network could uncover any bizarre links between recurring current events and market fluctuations which a human may never imagine.

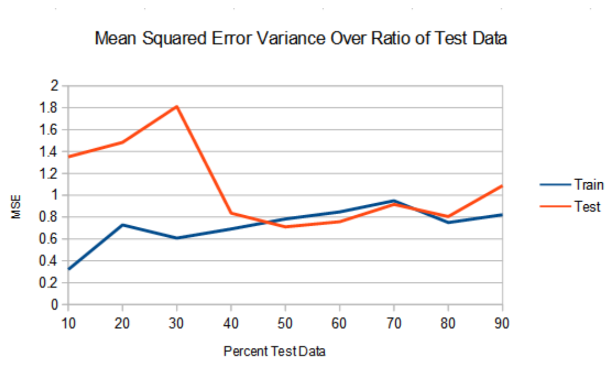
The next approach for prediction is with LightGBM, a light gradient boosting framework that uses "tree based learning algorithms". For this training method the training data is split into a main training set and a validation set, so that all tuning can automatically be checked against some validation data. When the loss score does not improve on the validation data for 50 consecutive iterations the model is considered to be converged and training stops. This split was done by randomly selecting 20% of the training data to become validation data. This is a potentially flawed method for selecting validation data, as at will theoretically contain information about the future, though this is irrelevant considering what data is stored and how it is accessed.

The LGBM framework was chosen for several reasons: very fast training speed, low memory usage, and very capable with large-scale data. Additionally, this algorithm has been used in many competition winning solutions. While far less overtly complicated than a neural network, boosted tree algorithms have been shown to perform on par with neural networks in many applications.

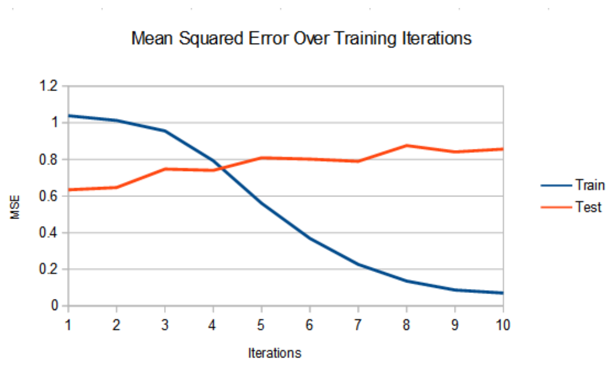
Evaluatation

Performance

Bad news. Despite the best efforts in data preparation and model selection, prediction was unable to perform significantly better than the market average. However this figure of performance over test to train data ratios in LightGBM suggests that there is potential for a good model.



Initially when training data is small a very accurate model is built, though it generalizes incredibly poorly. However, as more training data is introduced the training score decreases and the testing score increases, suggesting more training data can create a more accurate and general model. Unfortunately as even more training data is introduced both MSE's begin to creep upward. This model of MSE over iterations of training the neural network re-affirm any grim suspicions.



While the training data performance has a nice logarithmic descent, the test performance seems totally unrelated. Instead of decreasing alongside training MSE and eventually rising when overfitting becomes a problem, testing MSE increases with almost every iteration, as if overfitting from the very start.

This model was trained on data from 2005 to 2016, and tested on data from 2017 to 2018. This was done to simulate real world evaluation, where a financial model can only be trained on currently available data and must be tested online. Ideally a much larger range would have been used, however dealing with much more data posed significant problems. Another common approach when training financial algorithms is to throw out data from outlier periods, such as during the subprime mortgage crisis of 2008. I chose not to do this in the interest of the most complete data, and because my model does not use strictly financial data it should not be influenced by bubbles and recessions as directly. Additionally, in theory, an algorithm should be trained on recession data in order to create the most generalizeable model.

Results

Though these results are disheartening, they are not hopeless; if this was an easy problem someone else would have done it long ago. The main constraint I faced was with the size of data. Even using only the selected headlines still generated a vocabulary of over 40,000 words, which would likely get exponentially bigger if a larger date range was used, drastically so if headlines were replaced with something like the lead paragraph. The main issue with this approach is that news is not generally cyclical. In theory, knowing nearly everything that is happening in the world should give one a tremendous advantage when trying to predict the stock market. However, a machine can only view this newly input news data in terms of what it has already seen, not allowing it to make new inferences and connections. Because economic events rarely repeat themselves in entirely the same manner, it makes it hard for a machine to truly detect patterns. In an ideal world with perfect information this would surely be an easy task, so the search for a middle ground must continue.

Instructions for Replication

Here are the necessary instructions to clone my github repository and run my code. My preferred environment was Anaconda 64-bit running on python 3.7.1, which ships with almost all the dependencies.

1. Make sure all necessary dependencies are installed:
    1. numpy
    2. pandas
    3. scikit-learn
    4. lightgbm
    5. tensorflow
2. Clone my repo from https://github.com/willfrank98/Machine_Learning_Project.git
3. First run 'python final_prep.py' to generate the final datasets which were too large for github.
4. Here you can replicate my results by running 'python main_lgb.py' or 'python main_nn.py'.

If you wish to replicate the full pipeline then do the following:

1. Either obtain a NYTimes API key from https://developer.nytimes.com/ or use mine, 'fPBRdiPMKv8E4lUH4knZ0EhzxzpuB3J8'.
2. In nyt_scraper.py, replace 'API_KEY_HERE' on line 28 with the API key, and set the appropriate year range on line 33 (add 1 to 2nd year). Run nyt_scraper.py.
3. Set the appropriate year range (add 1 to 2nd year) on line 5 of nyt_combiner.py and run.
4. Set the appropriate year range on line 5 of data_combiner.py and run.
5. Set the appropriate year range on line 8 of final_prep.py and run.
6. Finally point the test and train file paths to the correct files in main_lgp.py and main_nn.py and run those.
7. To modify the train/test ratio do so on line 16 of final_prep.py.