Machine Learning Engineer Nanodegree - Capstone Project

Jeremy Jordan

(Dated: September 27, 2017)

I. DEFINITION

A. Project Overview

The general goal of this project is to develop a flexible forecasting system that is capable of leveraging unstructured text data in addition to more quantitative features. This project will focus specifically on using public companies' quarterly earnings calls to aid in stock price forecasting. These quarterly earnings calls provide company executives a chance to reflect on their progress in the previous quarter and make statements looking forward into the next quarter. Given that these statements are made by company executives, they can be considered high-quality information with little speculation. However, due to legal concerns and formalities, these earnings calls can also contain highly redundant information such as a safe harbor statement, which is stated during every call. An example of one such statements is provided below.

> Some of the statements that we make today may be considered forward-looking, including statements regarding our future investments, our long-term growth and innovation, the expected performance of our businesses, and our expected level of capital expenditures. These statements involve a number of risks and uncertainties that could cause actual results to differ materially. For more information, please refer to the risk factors discussed in our Form 10-K for 2016, filed with the SEC. Any forward-looking statements that we make are based on assumption as of today, and we undertake no obligation to update them. — Alphabet's (GOOG) CEO Sundar Pichai On Q2 2017 Results

Thus, this project will develop a technique for extracting the **useful** content from bodies of text to aid in stock price forecasting.

B. Problem Statement

Given the financial reward present in keeping high-performing algorithms confidential, it is hard to conclusively make statements on the state of the art. However, based on what information is publicly available, sentiment analysis remains one of the leading techniques for processing text. Sentiment analysis is the process of analyzing a body of text and determining the overall sentiment: positive, neutral, or negative. Whereas a majority of news events are neutral, one can draw correlations between positive and negative articles and their corresponding effect on the stock price.[1]

The goal of this project is to leverage word embeddings to provide a more granular analysis of text data. Whereas sentiment can provide a one-dimensional view of a body of text, word embeddings provide a multi-dimensional perspective. Word embeddings allow for the translation of words into a vector-space representation that preserve semantic relationships between words.

By plotting a body of text in vector space, we can analyze a dynamic multitude of relationships. For example, this study posits that one could spatially observe a shift in language between a company's statements of excitement during highly-profitable periods of time and the same company making more defensive statements during highly-competitive period of time by comparing the language used during each statement in a vector-space representation.

By observing patterns in this multi-dimensional text representation space using a convolutional neural network, distinct features can be extracted - providing the ability to go beyond sentiment classification. These extracted features may then be fed into a recurrent neural network for use in making predictions.

In general, the steps in this project includes:

1. Download price and text data.
2. Convert text to vector-space representation.
3. Use a convolutional network to serve as a fixed feature extractor on the text data.
4. Combine extracted text features with price data.
5. Train a recurrent neural network to predict future values.

C. Metrics

Models will be evaluated according to the mean absolute error, as defined below. The predictive model will be forecasting stock price returns, which typically vary between -0.15 and 0.15; because the error will likely be less than one, absolute error was chosen over squared error in

an attempt to have a larger loss metric. Forecasting models which have a strong predictive capability will have a lower mean absolute error.

The mean absolute error can be calculated as

$$MAE\,(y, \hat{y}) = \frac{1}{n}\sum_{t=1}^{n} |y_t - \hat{y}_t|$$

where y_t represents the true value at time t and $\hat{y}_t$ represents the forecasted value at time t.

II. ANALYSIS

A. Data Exploration

There are two sources of data for this project: company price performance and quarterly earnings call transcripts. Each data source will be discussed individually.

Stock Price History Price performance history for all of the S&P 500 companies is downloaded from the Google Finance API. This API (accessed via the Pandas DataReader wrapper) provides the adjusted daily opening price, closing price, high, low, and volume traded. For this project, only the adjusted daily closing price will be used. This adjusted price accounts for corporate actions such as stock splits, dividends/distributions and rights offerings to provide a more accurate representation of the company's value.

Below, an example of the information available via the Google Finance API is provided. Note that the data source does not guarantee the full history for any given attribute, as evident by the lack of volume information for the excerpt below.

Date	Open	High	Low	Close	Volume
2009-01-02	154.15	160.75	152.6	160.5	NaN
2009-01-05	160.34	165.45	157.34	163.86	NaN
2009-01-06	166.32	170.23	163.03	166.86	NaN
2009-01-07	164.0	165.29	159.22	160.84	NaN
2009-01-08	158.98	162.43	158.51	162.43	NaN
2009-01-09	163.59	163.59	156.54	157.38	NaN
2009-01-12	158.0	159.32	154.96	156.19	NaN

Each company's price history is stored in a CSV file.

Quarterly Earnings Call Transcripts The quarterly earnings call transcripts are downloaded from Seeking Alpha via the Python web scraping library, Scrapy. For each earnings call, the following information is collected: title, url, date, company, executives, analysts, and the earnings call text. For the purpose of this project, only the date and earnings call text will be used. For each company, the entire history of earnings call transcripts is downloaded from Seeking Alpha and stored in a JSON file.

A list of quarterly earnings transcripts for Google is available at `https://seekingalpha.com/symbol/GOOG/earnings/transcripts` as an example to orient the reader. For companies which have acquired a public company, Seeking Alpha includes the acquired company's history in the list of transcripts. For example, pre-merger transcripts from Motorola now reside under the list of Google transcripts. For the sake of simplicity in this project, these pre-merger transcripts are not differentiated or separated.

Note: The transcripts provided by Seeking Alpha are protected by copyright and can not be used for commercial interests. However, given the educational nature of this project as part of the Udacity Machine Learning Nanodegree, use of this information is permitted under the Copyright Fair Use principle.

On average, an earnings call contains 9,450 words. This is a substantive body of text when compared to a standard news article or tweet; however, these earnings calls only occur four times a year whereas other sources of information such as news articles and Twitter feeds are published much more frequently. Ideally, a system should be able to combine text from various sources. This said, the current project will focus solely on earnings call transcripts.

B. Exploratory Visualization

When dealing with large amounts of data, it is useful to develop visualizations in order to efficiently observe and consume the data in a human-friendly manner. This section will explore the downloaded data through visualizations to get a better sense of the information we're dealing with. Google will be used as an example company for these visualizations.

Price data Figure 1 below displays the information as retrieved from the original source and prior to any preprocessing.

However, it is often useful to convert this information into a relative difference between daily prices, generally referred to as a return. These returns can capture increases or decreases to the price as opposed to the absolute price, which is always positive. Figure 2 displays the log-returns for Google's stock price over the same time period as the prior figure. A log-return may be calculated as

$$r\,(t) = \log\left(\frac{p\,(t)}{p\,(t-1)}\right)$$

where p_t represents the current price and p_{t-1} represents the previous day's price. Due to the logarithm properties, returns over multiple days can be meaningfully added to represent the aggregate log-return over the entire period. Further, it is worth mentioning that converting stock prices to log-returns is a standard practice

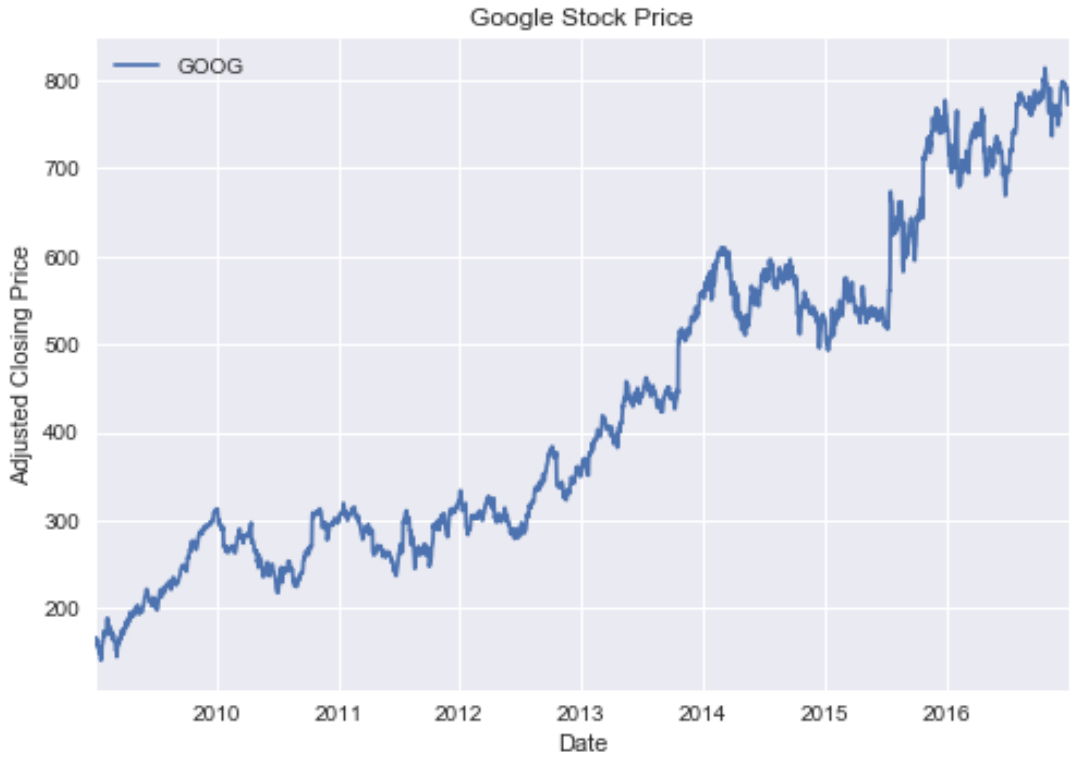

Figure 1. Google Daily Adjusted Closing Price

within the financial industry due to underlying assumptions on how assets behave in the market.[2]

Text data Foremost, it's useful to know how much text a transcript contains. Not only that, it is equally useful to look at the distribution of this value over all of the transcripts for a given company. Using the `visualize_word_count()` function in the Jupyter Notebook, one can plot the word count distribution of any company for which there is data. The word count distribution for Google is displayed in Figure 3. This histogram is especially useful for identifying erroneous data existing far outside of the rest of the distribution.

Quarterly earnings calls should occur once a quarter, totaling four calls for any given year. As a simple way to visually inspect the dates of all of the collected transcripts, `visualize_dates()` plots each transcript on a timeline. This timeline, such as the one in Figure 4, not only shows the regular period of calls expected (quarterly), but also brings to light deviations from this regular periodicity such as events that occurred more frequently than once per quarter (as shown by the overlapping squares), a case that occurs when pre-merger transcripts of an acquired company is included in the dataset. This timeline also makes it very easy to identify gaps where an earnings call wasn't available, or the web scraper failed to download it, such as the case for Google in early 2016.

Lastly, Figure 5 displays a single earnings call transcript, tokenized, and converted into a vector representation via word embeddings. Each scatter point represents a word spoken during the earnings call, with the z axis representing the importance of each word as calculated by a normalized TF-IDF score. The accompanying Jupyter Notebook contains an animation which shows the evolution of language used over the history of a company's earnings calls.

C. Algorithms and Techniques

The following section will discuss the specific machine learning techniques used during this project.

Word embeddings As mentioned in the introduction, word embeddings provide a vector-space representation of words in such a manner to preserve semantic relationships between words. Thus, the difference between vector representations of `king` and `queen` should be roughly equivalent to the difference between vector representations of `male` and `female`.

Word embeddings will be used in this project to project words used during earnings calls into a 50-dimensional space.

Principal Components Analysis The word embeddings used in this project provide 50-dimensional rep-

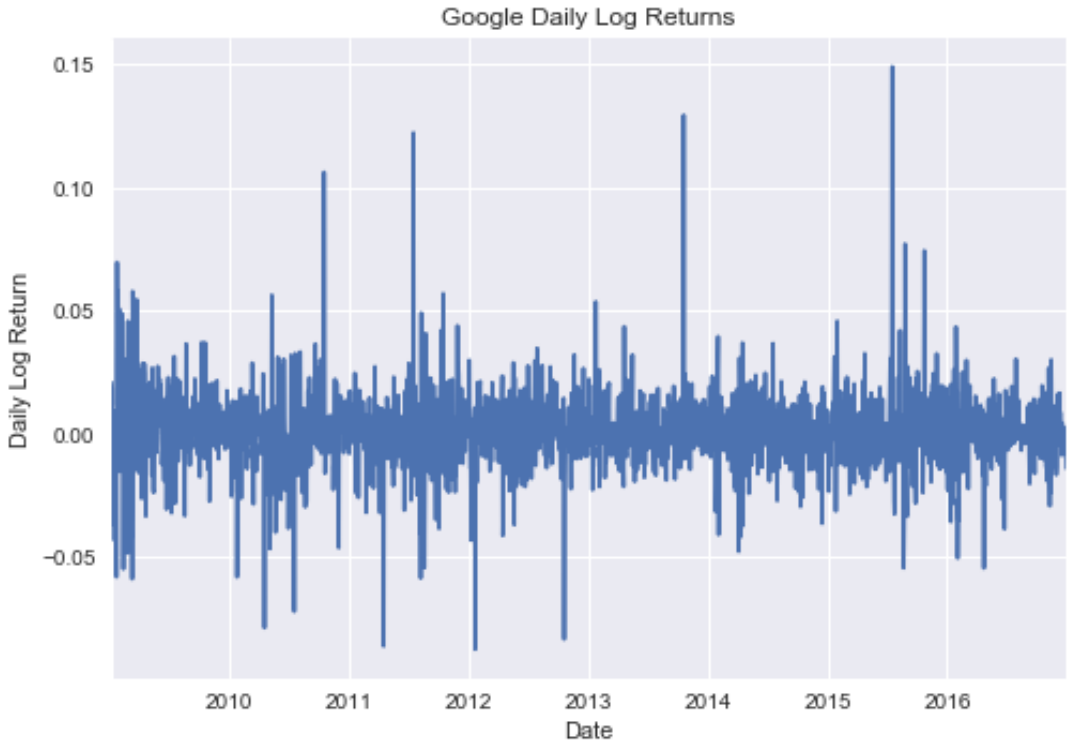

Figure 2. Google Daily Log Returns

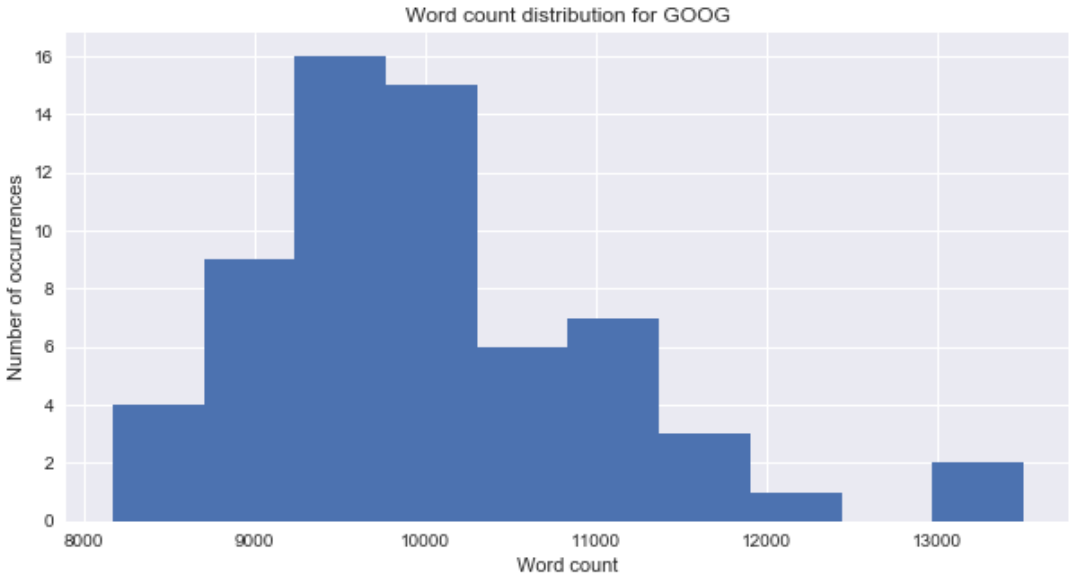

Figure 3. Google Transcripts Word Count Histogram

resentations of words. In order to plot the language of an earnings call in a manner that can be presented visually, principal components analysis (PCA) is used to reduce the dimensionality of the embeddings. PCA was chosen over t-SNE, another popular dimensionality reduction technique, given the deterministic nature of its output. PCA can reduce high-dimensional data by determining the orientation of a lower-dimensional set of axes which captures the maximum variance within the data.

PCA will be used to reduce the 50-dimension word embedding vectors down to a 2-dimensional representation, both for the sake of visualization and reduced parameter size for the convolutional neural network.

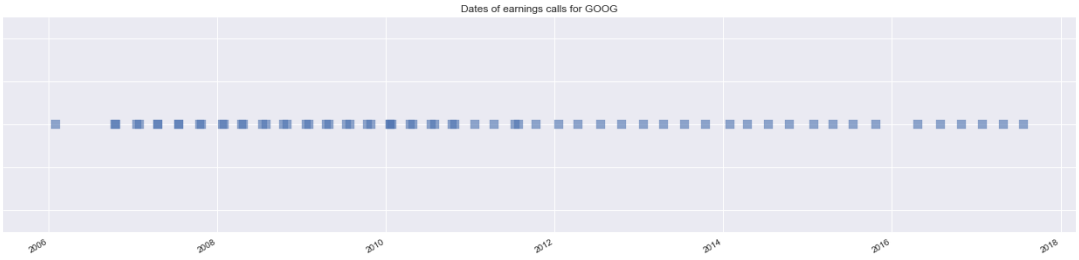

Figure 4. Google Earnings Call Dates

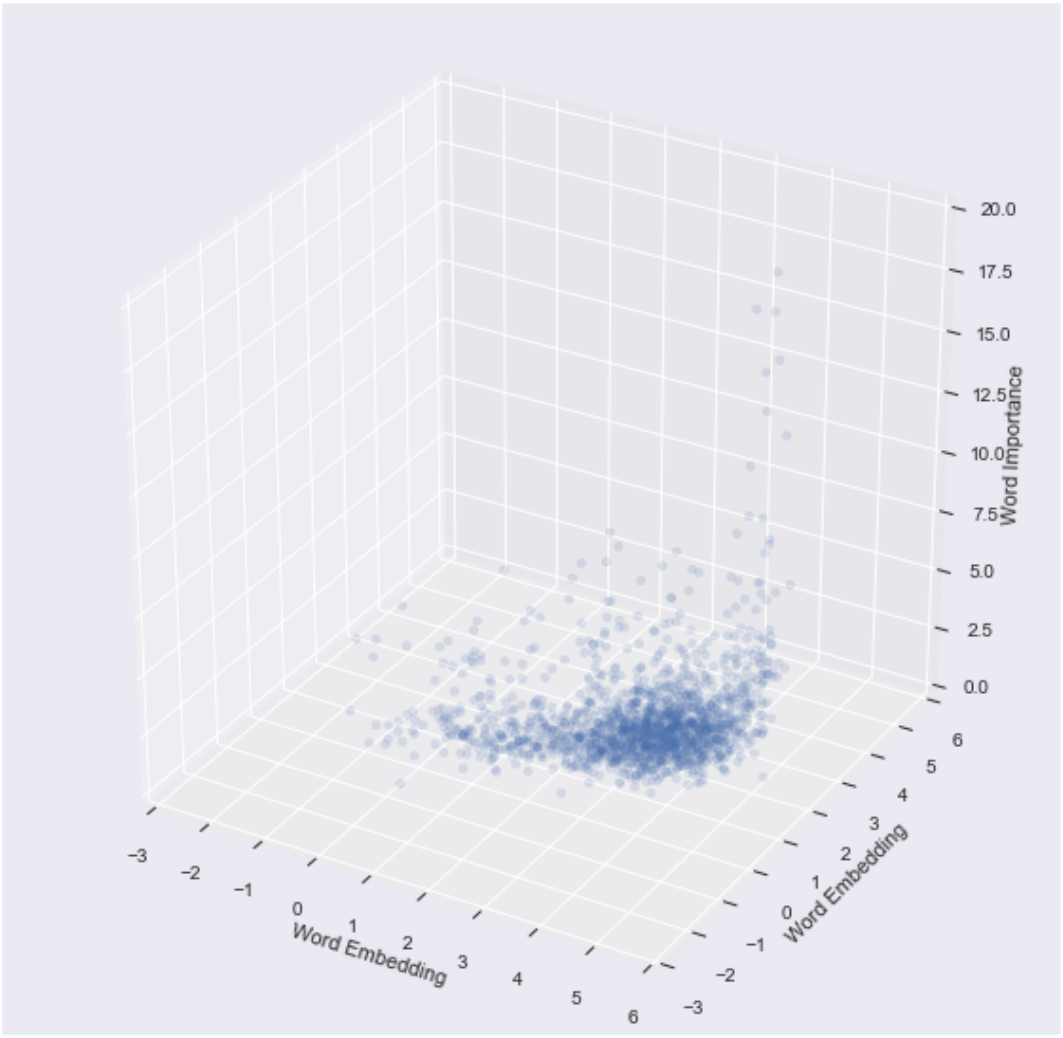

Figure 5. Word Embeddings Visualized

TF-IDF Scores In an attempt to ignore redundant language and give emphasis to words that uniquely describe a document, the TF-IDF score will be added as a third dimension to the text data. A TF-IDF score combines the term frequency within a document with the inverse document frequency of the given term across a collection of documents. By weighting the word accounts against the inverse of their frequency across all documents, it is possible to focus on the words which uniquely describe a document while ignoring common words ("the", "at", etc.) that may appear frequently within a document, but also appear frequently in all of the other documents as well.

Convolutional Neural Networks Convolutional neural networks (ConvNets) are a type of neural network

capable of learning local spatial relationships within the input data by convolving the input to generate feature mappings. The features mapped depend on the filter parameters used during convolutions. Because these filter parameters are learned during training, the network has the capability to determine which features to look for. These networks are used most notably in computer vision applications such as image classification, but they can also be used as a fixed feature extractor from a multi-dimensional input space.

ConvNets will be used to identify patterns in the vector-space representation of earnings call language and their corresponding effect on the stock price.

Long Short-Term Memory Networks Recurrent neural networks are useful for instances where either the input, output, or both are a sequence of individual values. In other words, recurrent neural networks are capable of learning one to many, many to many, and many to one relationships from data. These networks are capable of maintaining an internal state of the network of its previous values, combining with new input to predict an output.

Long short-term memory networks were proposed as a type of recurrent neural network that overcomes the vanishing/exploding gradient problem of standard recurrent networks, allowing for the network to learn long-term dependencies.

D. Benchmark

The ARIMA model is an industry-adopted standard of econometrics forecasting. This model is a statistical tool which consists of three components:

- **Autoregression (AR)** refers to a model which combines (1) a linear dependence of an observation on previous values with (2) a stochastic term for predicting future values.
- **Integrated (I)** makes use of differencing observations in order to enforce stationarity of the data. This differencing may occur multiple times before the data becomes stationary.
- The **moving-average** model **(MA)** measures the dependency between an observation and residual error, ε, of previous values.

The model may be written as:

$$y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + ... + \theta_q \varepsilon_{t-q}$$

where ϕ are the parameters for the autoregressive model and θ are the parameters of the moving average model. p and q respectively determine how many terms to "look back" when predicting the value at the current time step.

This project will compare the predictive performance of three models: a univariate ARIMA model, a univariate LSTM model, and a multivariate LSTM model which combines previous price performance with company's quarterly earnings call transcripts.

A comparison between the univariate LSTM model which makes predictions solely on price performance and the univariate ARIMA model will evaluate the predictive power of recurrent neural networks against the industry standard in forecasting. Further, a comparison between the univariate LSTM model and the multivariate LSTM model, which incorporates the features extracted from text, will evaluate whether the additional information resulted in an improvement in predictive power.

III. METHODOLOGY

A. Data Preprocessing

Price data The stock price data contains daily information for all of the days in which the markets were open for trading. However, the markets close for certain holidays and thus the time series does not have a common frequency. Thus, the pricing data was reindexed to include all business days. This causes the days which were originally not included (due to the markets being closed) to now contain null values; these values were filled by copying the price of the previous day (known as a "forward fill").

Next, the log-returns were calculated as described previously in the Exploratory Visualization section. By using a forward fill, no future information about the price is included, and the returns will simply calculate to be zero for the days in which the markets were actually closed.

In summary:

1. Reindex to a common frequency (business day).
2. Forward fill all null values.
3. Calculate the log-returns.

Text data The earnings call transcripts required a fair amount of preprocessing. The data downloaded during web scraping is stored as a JSON file, which is then loaded into memory as a Pandas DataFrame. This DataFrame contains the entire collection of earnings calls for a given company.

Foremost, it is possible for a transcript to be scraped from the web without extracting the date of the event. Given the fact that this project is attempting to learn the effect of earnings calls on the stock price, the two sources of information need to be tied to a common time series. Thus, if the date for an earnings call is missing, the transcript will simply be discarded.

Further, it is possible for the scraping engine to have extracted the same earnings call, whether the call was published multiple times or the web crawler visited the same web page twice, these duplicate events will also be discarded.

The text from an earnings call transcript is extracted from the web page according to the web page's DOM (Document Object Model) structure. As a result, the transcript is extracted as a list of individual collections of text. These collections of text are now joined together into a single body of text.

At this point, an additional column is added to the DataFrame to calculate and store the word count of each document in the collection; empty documents (ie. word count is zero) are discarded at this point.

Next, a TF-IDF matrix is calculated where each row represents a body of text and the columns represent a collection of all of the words present across documents. A dictionary is created for each document mapping words to their corresponding TF-IDF scores. The collection of dictionaries representing individual transcripts for a company are stored in a list.

Each dictionary is then converted into a Pandas DataFrame, indexed by word, and two columns are added to include the word embedding dimensions. This results in a data structure that can be queried by word for the word embedding and TF-IDF values.

At this point, the word embeddings exist in a continuous vector space. In order to prepare this information for a ConvNet, the data is sorted into a 2D histogram where bin values represent the sum of TF-IDF scores for all words that exist in the bin.

In summary:

1. Drop events that don't have a date.
2. Check for duplicate events.
3. Combine scraped transcript into a single body of text.
4. Count words in transcript.
5. Check for empty transcripts.
6. Calculate square root TF-IDF values for a company's history of transcripts.
7. Calculate the 2D reduced word embedding vectors for each word.
8. Digitize the input space with a 2D histogram.

B. Implementation

ARIMA An ARIMA model is specified by three parameters, the order of the autoregressive component (p), the order of differencing (d), and the order of the moving-average component of the model (q). These parameters are chosen via a hybrid of statistical analysis of the characteristics of the underlying data, and a grid search approach. After determining the parameters to specify a model, one can build and train an ARIMA model using the Python Statsmodels package.

Autocorrelation and partial autocorrelation plots are highly useful when examining time-series data, as they show the degree to which previous time steps are correlated with the current time step. An autocorrelation function directly measures the correlation between time series events and a lagged copy of the time series event, essentially answering the question "how correlated is this price today with the price x days ago"? The partial autocorrelation function similarly measures the correlation between times series events and a lagged copy of the time series event, but also controls for the linear dependence on time steps between the current and lagged copy, essentially answering the question "how correlated is this price today with ONLY the price x days ago"?

For example, the autocorrelation and partial autocorrelation plots for Google's price in 2013 are displayed in Figure 6. One can see that the price is highly correlated with previous time steps for many steps prior to the current. However, when examining the partial autocorrelation plot, it becomes clear that this correlation is embedded only in the previous day's price, but because each day is chained to the day prior, the autocorrelation extends quite far into the past.

However, it is important to have a stationary time series for accurate modeling. Thus, we'll need to difference the time series until stationarity is achieved. Differencing converts a time series from its original state to the difference at each time step between the current value and the value immediately prior. We'll also take the natural log of the data in an attempt to reduce the variance and prevent our model from overfitting. The autocorrelation and partial autocorrelation plots for Google's log-returns in 2013 are shown in Figure 7. Here, it is evident that the returns for any given day have zero correlation with the returns from prior time steps. This phenomenon, known as a random walk, makes a difficult case for prediction.

For seasonal data, there exists a specialized ARIMA model capable of handling seasonal time series data. Due to the well-known seasonal trends found in stock prices, a seasonal ARIMA model was used. A grid search over (p, d, q) values from 0 to 4 suggests optimal parameter values of ARIMA(0, 1, 2)x(0, 1, 2, 12) where the first set of parameter values define the standard ARIMA model and the second set of parameter values define the seasonal component. The last parameter value denotes one season as a 12 month period.

Because it is not standard to train an ARIMA model over multiple time series unless the time series data is generated by the same process. For the case of stock prices, there exist independent influences on individual companies that does not uniformly affect all companies. Thus, 5 different ARIMA models were trained for different companies, evaluating each model individually and

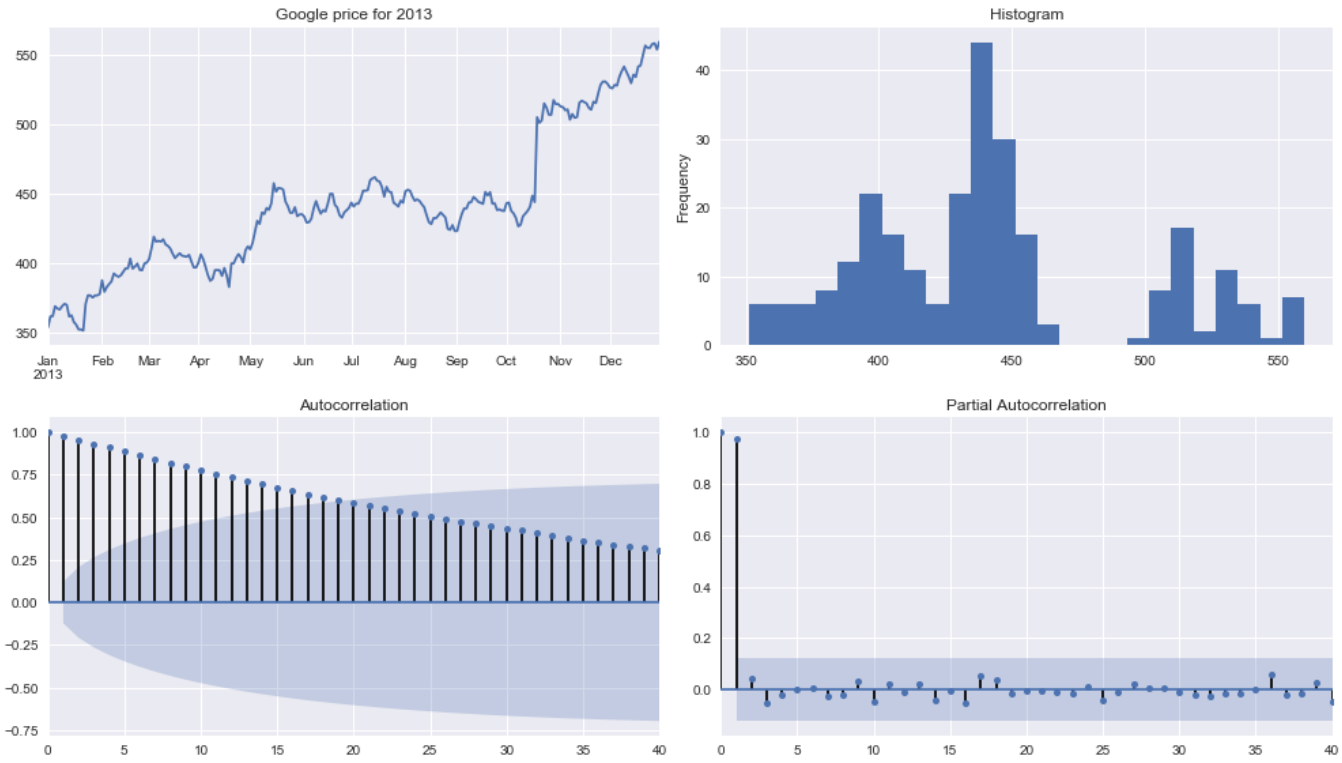

Figure 6. Time series statistical analysis of Google stock price in 2013.

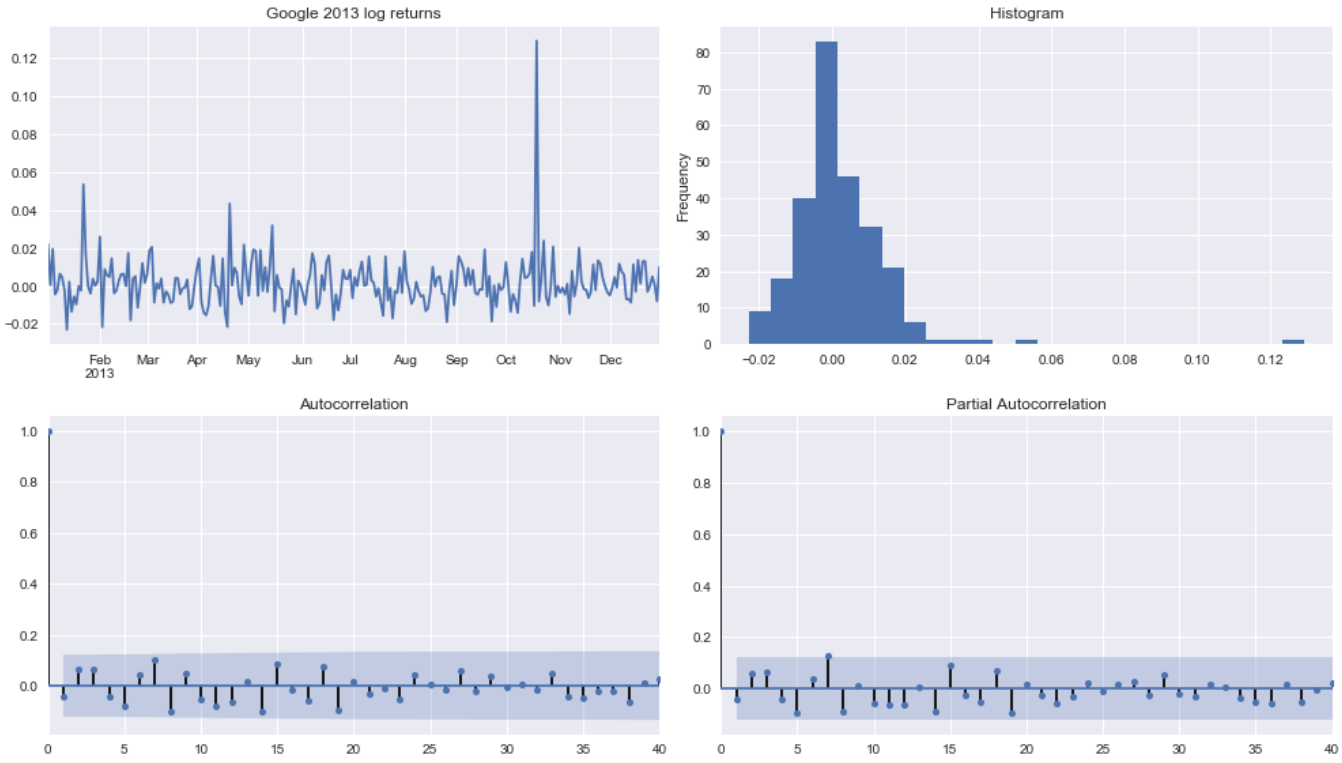

Figure 7. Time series statistical analysis of Google stock price daily returns in 2013.

averaging the result.

LSTM (price only) The benchmark LSTM model attempts to forecast future stock prices using only past stock prices as input. The network will accept a sequences of stock prices as input, and attempt to predict the next day's output for each step within the sequence, as visualized generally in Figure 8. The length of sequences provided to the network during training is a hyperparameter that can be tuned. However, when analyzing the statistical properties of the time-series data in the previous section, it was discovered that the price, in isolation, behaves as a random walk. Due to the improbability of predicting random events, the best a model can do is to consistently predict the mean return. Thus, once the model exhibits this behavior there is little incentive to perform additional tuning.

The network consists of two LSTM layers, with the first layer containing 20 units and the second layer containing 60 units. Lastly, a time-distributed fully connected layer is used to regress the next day's stock price return.

ConvNet feature extraction Although the daily price returns exhibit behavior of a random walk, it appears as if there are external influences which exhibit some degree of control over the price movement. Additionally, it was earlier observed that the largest price movements occur close to a company's quarterly earnings call. Thus, the goal of this project is to discover and extract some meaningful signal from the text data on earnings calls in an attempt to predict the direction of the price swings. The ConvNet is trained on a number of samples consisting of the 2D histogram representation of an earnings call transcript as the network input, and the 5 day log-return immediately following the text event as the network target. A standard convolution network architecture is developed consisting of the following layers:

```
Conv2D(32, (3, 3), activation='relu',
     padding='same', input_shape=(250, 250, 1))
Conv2D(32, (3, 3), activation='relu')
MaxPooling2D((4, 4))
Conv2D(16, (3, 3), activation='relu')
GlobalAveragePooling2D()
Dense(100, activation='relu')
Dropout(0.4)
Dense(text_features, activation='relu')
Dense(1, activation='linear')
```

The network is trained to identify features useful for regressing the 5 day log-return, where the values in the last fully-connected hidden layer will be used as a feature-representation of the text.

LSTM (price and text) Lastly, the forecasting model which incorporates text features in addition to price is developed. This model copies an identical architecture as the price only network, with the exception that the input now contains text features concatenated to the daily price input. For days with no text information available, a zero-filled vector is concatenated to the daily price input, as visualized generally in Figure 9.

C. Refinement

Initially, the LSTM model which incorporates text information had a time-distributed ConvNet to "read" the text and extract features, concatenating these extracted features to the price input, and then feeding the combination through the recurrent layers. For days with no text information available, a zero-filled array of the same dimensions as the 2D array representation of text data was created and fed as input. However, it quickly became clear that this was an inefficient approach. Foremost, only a few days each year contained text information, so it did not make sense to train a time-distributed ConvNet over sparse input data. Additionally, this approach required a large amount of memory to hold the input data. By training the ConvNet separately, features can be extracted from text data directly before training on the recurrent network. This refinement resulted in a 10x decrease in parameter size and made training much more manageable.

Additionally, I spent a fair amount of time structuring the text input in a way to expose quarter-to-quarter changes in the language used during earnings calls. Initially, I had simply used term frequency as the third dimension to represent the word significance. However, I changed to using a TF-IDF score in an attempt to draw out the differences between documents, essentially quieting any stop words that might occur across all documents. Even with this improvement, there is very little change between earnings calls, suggesting that the language needs to be viewed from a different perspective than just words in isolation.

IV. RESULTS

A. Model Evaluation and Validation

The benchmark ARIMA model and benchmark LSTM (price only) model are compared according to the mean absolute error for 5, 20, and 100 day forecasts. The results are summarized in the table below.

	ARIMA	**LSTM (price only)**
5 day forecast	0.009500	0.012039
20 day forecast	0.009690	0.012486
100 day forecast	0.009965	0.011844

The ARIMA model appears to have performed slightly better than the LSTM model, but not to a significant degree. When plotting the forecasts for each model in Figures 10 and 11, one can see that the ARIMA model

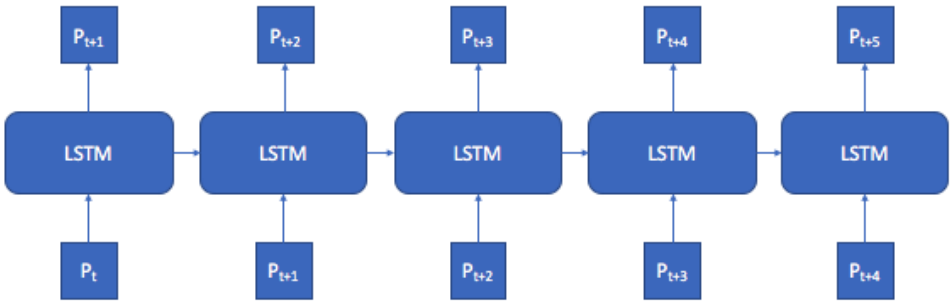

Figure 8. Price Only LSTM

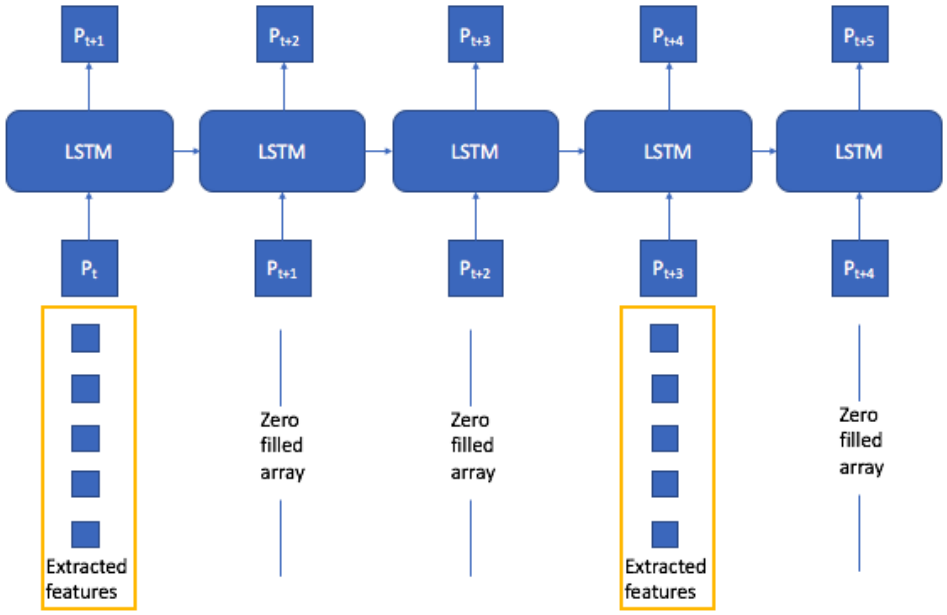

Figure 9. Price and Text LSTM

has small deviations around the mean return while the LSTM model more consistently predicts the mean return.

The LSTM (price only) model and LSTM (price with text features) model are evaluated more exhaustively using the Keras `evaluate_generator` function to generate numerous samples to evaluate the model performance over.

	LSTM (price only)	**LSTM (price with text)**
MAE	0.009176	0.008930

According to these results, the proposed model which incorporates text features from earnings call transcripts does not significantly outperform the benchmark model.

When inspecting the ConvNet performance, it becomes evident that the text data presented in the 3-dimensional representation described previously is not useful input for predicting price movements. In fact, the ConvNet resorts to predicting practically the same return no matter the input. This is not a surprise, given the degree of similarity found across text documents when visualized in a 3-dimensional space. The original hypothesis that one could visually observe changes in a company's language used as they experienced different aspects of the business cycle is disproven by these results.

In summary, given the random walk behavior of price returns, it was not expected to develop a robust forecast-

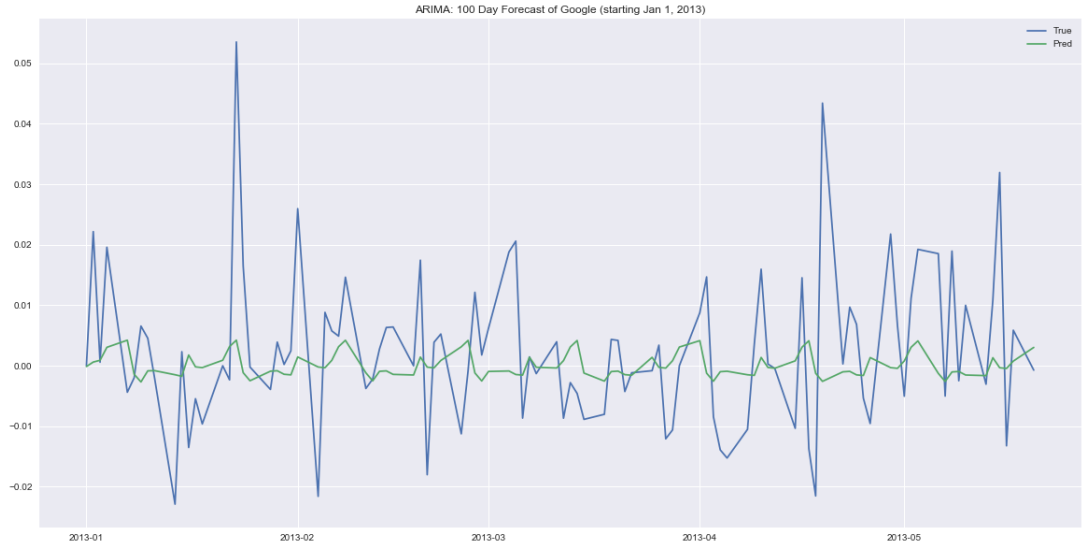

Figure 10. 100 day forecast using the ARIMA model

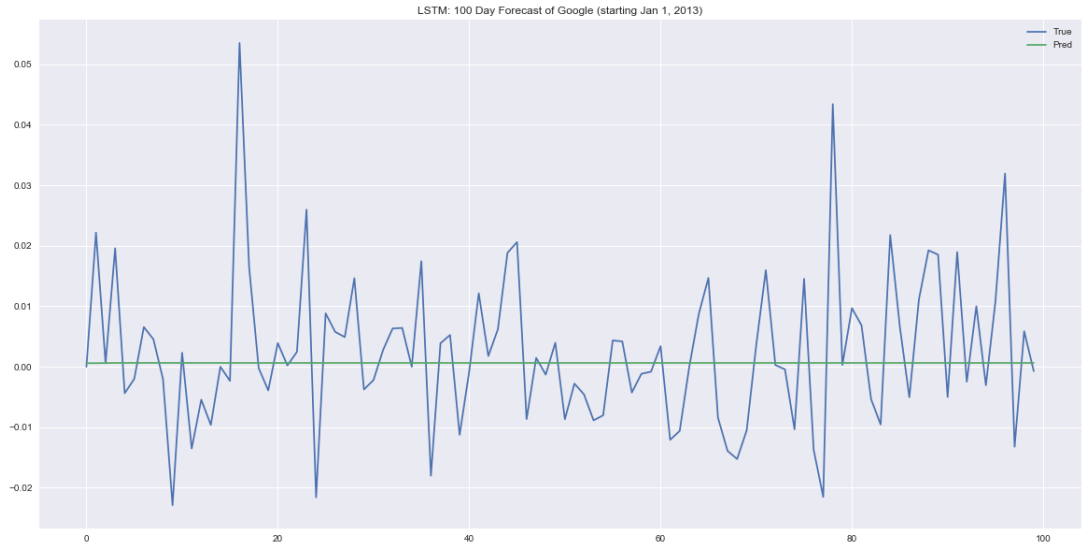

Figure 11. 100 day forecast using the price-only LSTM model

ing model using only price. Further, the text data did not prove to contain useful information to aid in forecasting future stock returns. Given these circumstances, each model performs adequately by consistently predicting a return close to the historical mean. For this reason, little time was spent tuning hyperparameters.

B. Justification

The proposed model, in its current form, does not prove to be a useful model to aid in forecasting future stock prices. However, it was discovered that a company's largest price movements occur surround the time of quarterly earnings calls. This suggests there must be some information present at this time that is causing large movements in price. Perhaps, the stock price exhibits not a random walk, but a drunken one, where day-to-day movements appear random but are in fact guided by periodic influences that steer the general direction of the random walk.

Due to the redundant and formal nature of these earnings calls, it is clear that looking at TF-IDF scores of words in isolation do not prove useful in extracting some meaningful signal. However, improvements to this model will be discussed in the following sections that might provide a better insight on text events.

V. CONCLUSION

A. Free-Form Visualization

Figure 12 displays the daily returns for Google with the dates of quarterly earnings calls (and the following five days) highlighted in green. Although the proposed model in this project was not able to learn meaningful signal from text data, this alignment of large price movements surround the time of quarterly earnings calls motivates further investigation.

B. Reflection

This project sought to leverage text data for use in forecasting future stock price performance. Analyzing the statistical properties of daily closing stock prices, it was discovered that stock returns exhibit random walk behavior, which proved to be difficult to make an accurate forecast using only the daily stock returns. However, it was also found that a majority of a company's largest price movements were concentrated surrounding the days immediately following quarterly earnings calls, suggesting that some event in this timeframe was influencing market behavior.

Extracting useful information from these quarterly earnings calls, however, proved challenging and ultimately unsuccessful given the highly redundant and formal nature of the calls. Earnings call transcripts were split into individual words, calculating a TF-IDF score for each word in the document and converting the word to a spatial representation using word embeddings and the calculated TF-IDF score.

Largely, the success of this project revolved around extracting meaningful information from text data. Over the duration of this project, it became clear that looking at words in isolation would not yield such information; simply, when looking at the 3D vector representation of quarterly transcripts, the entire history of a company's quarterly earnings calls appear largely identical. This suggests the need for more advanced natural language processing techniques to better present the information in a manner which a machine can understand the content of an earnings call.

C. Improvement

One promising possibility to improve the interpretation of this text data would be to extract events from earnings calls, rather than individual words. For example, given the text "Microsoft sues Barnes & Noble" an event may be extracted as (Actor = Microsoft, Action = sues, Object = Barnes & Noble). The same techniques for developing vector-space representations of words can be used to develop a vector-space representation of events. With this approach, less emphasis is placed on which words were said during a call and more emphasis is placed on what information is communicated.[3]

Moreover, acquiring additional, more frequent text data may prove to be useful in developing more robust forecasts. While it appears that the largest price movements are surrounding quarterly earnings calls, other large news events can also have an impact on stock price performance and thus, incorporating this information will aid in developing a more accurate forecast.

REFERENCES

[1] U. Berkeley, *Stock price forecasting using information from yahoo finance and google trend* ().

[2] 2017.

[3] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Deep learning for event-driven stock prediction.", in Ijcai (2015), pp. 2327–2333.

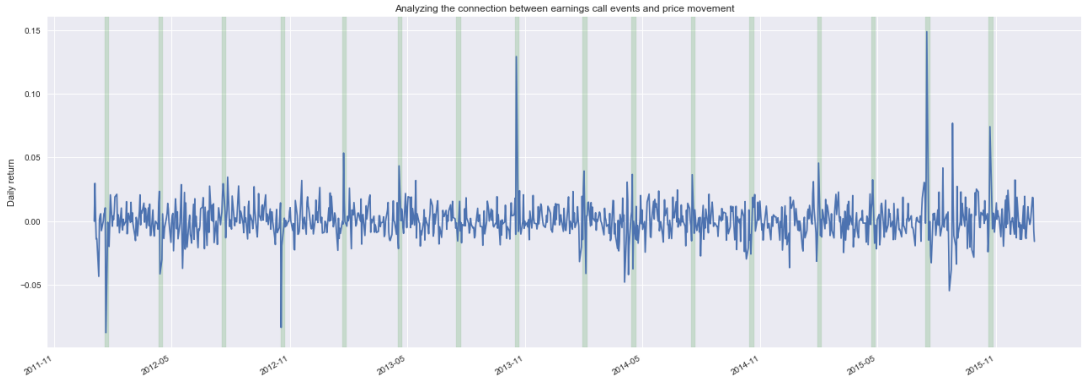

Figure 12. Daily stock price returns with days surrounding earnings calls highlighted.