National Grid Stock Price Prediction

by

Soham Adhikari

Table of Contents

Project Abstract

Problem Description:

National Grid plc is a multinational electricity and gas utility company headquartered in London, England. Its principal activities are in the United Kingdom and Northeastern United States. It has a primary listing on the London Stock Exchange and is a constituent of the FTSE 100 Index. It has a secondary listing on the New York Stock Exchange. National Grid is one of the largest "investor owned" utility companies in the world, and it provides gas to ten million customers, and electricity to ten million customers in New York, Massachusetts, and Rhode Island.

Time and again it has been observed that National Grid stock market is influenced by various parameters like changing interest rate expectations in the US and the UK, foreign exchange rate etc. One of the major influencers of stock prices has been observed to be the sentiments driven by political and regulatory environment. As in instance, the surprise general election in June 2017 and the lengthy discussions around the Brexit process have both contributed to a less favorable outlook for investors considering investing in UK stocks.

Hence, in order to confirm the degree of influence made by Sentiments, we move a step ahead and implement the underlying concepts of Text Mining. To understand what subjects of discussion, influence the stock prices. For this we pursued Tweets made by Donald Trump as the main source of data during his tenure as the President of the United States of America.

Proposed Techniques and Data Science Methodology:

The objective of the project is to predict Stock Prices of National Grid based on Tweets made by Mr. Donald Trump. Hence, the first and foremost task to approach this problem was to understand the various topics he tweets about for most of the time. In order to achieve this, we plan to implement the concepts of Topic Modeling. A popular technique called the Latent Dirichlet Allocation is used as a part of Data Science methodology for Topic Modeling purposes. Later we implement various supervised Machine Learning techniques such as the Random Forest and Gradient Boosted Trees and compare the degree of accuracy in the prediction of the stock prices.

A few business questions:

While doing the project, we were successful in analyzing the data i.e. how the tweets are responsible in affecting the stock prices of National Grid. We succeeded in doing so by carefully cleaning the data of Tweets and stock prices and then applying various models on the cleaned data. We determined the most discussed topic and the least discussed topic after extrapolating the most significant attributes from the data. Further we were able to predict the rise or fall of the stock prices of the industry by processing a tweet made by the president of the United States.

The business questions that were asked during the whole process were as follows:

- What are the different topics President tweets about?
- How can we classify various tweets under different topics?
- How are the tweets affecting the change in price of National Grid's stock?
- How well we can predict the stock prices based on the tweets?

Data Sources:

The Main Data source for the tweets made by Donald Trump was gathered from the website trumptwitterarchive.com. This website contains various tweets made by Donald Trump over different periods of time about different subjects. As our main data we considered the tweets made by Donald Trump since January 20, 2017 until March 25, 2020. The start date of the tweet represents the start of his presidential journey until the start day of our project.

The secondary data source for the stock prices of National Grid was gathered from Yahoo finance for the same dates as the tweets. In total the tweet dataset has 42K rows and 9 columns and National Grid Stock prices has 800 rows and 7 columns.

Data Description:

Tweets Dataset

The data extracted from trumptwitterarchive.com has around 42k observations with 9 columns. The relevant columns are described below.

Id – Tweet ID that is associated which each tweet posted on twitter.
Link – Link of tweet associated with the tweet ID that can be browsed to open a particular tweet
Content – Tweet content consists of the actual text posted by President Donald Trump
Date – Tweet Date specifies the date and time of a tweet posted
Retweets – This column describes a number of times that a tweet has been retweeted.

Stock Price Dataset

The National Grid Stock price dataset has been extracted from Yahoo Finance website for the dates corresponding to the tweet dataset i.e. from January 20, 2017 until March 25, 2020. This dataset comprises of 800 observations with 7 columns. The relevant columns are described below.

Date – This column corresponds to the days for which the stock market was open and operational, excluding weekends and national holidays.
Open – It corresponds to the price at which a stock opens on a particular day.
High – This column provides the maximum value a stock price achieved on a particular day.
Low - This column provides the minimum value a stock price achieved on a particular day.
Close – It states the closing price for the National Grid stock price on a given date. Further, we have used this column for analysis of stock price prediction.

Data Cleaning:

The data gathered contained tweets that were written in common language. Which contained characters and expressions that a machine is unable to understand and hence the Data was required to be cleaned and made ready for use. As a part of the cleaning process the first step was to drop the unnecessary columns. Columns that do not add any value to our objective like the tweet ID, Retweets, etc. Similar for the stock price data the unnecessary columns were high, low, adjusted close etc. that were dropped.

The next task was to remove the rows in the tweet data that were empty i.e. sometimes the data collection process experiences some glitch or perhaps there has not been any tweet made on that particular time. This is taken care by first filling the empty with "Nan" value i.e. "Not a Number" and then the "Nan" rows dropped.

Once we had a compact dataset of the tweets, it was time for us to clean the text content and filter out the characters that do not add any meaning to the machine. The tweets contained embedded hyperlinks and special characters. These hyperlinks, special characters and punctuation were removed using Pandas "apply" function and regular expressions "split" function. The data was then sorted by the dates. Hence, the final tweets data contained unique dates and tweets made on each date. This data set was further preprocessed and made ready for analysis.

Data Preprocessing:

After cleaning of data, an important step comes into play that is Data Pre-processing. Data Pre-processing is an essential step in data analysis problem, as the unprocessed data consists of noise or outliers and duplicate or wrong data. Further, we have used nltk and gensim packages that give us the power for pre-processing data. In this problem statement of tweet analysis where we have tweets grouped by date after cleaning performed in the steps above, we have performed the following steps of Data Pre-processing.

Tokenization

In tokenization, the tweets grouped by date were split into sentences. After this conversion, sentences were split into words. The words were converted into lowercase and words with fewer than three characters were removed. Hence, longer format of grouped tweets was converted into array of words which was then used for further analysis.

```
def tokenization(text):
    text = re.split('\W+', text)
    return text

trumpTweet_groupedDate['Tweet_tokenized'] = trumpTweet_groupedDate['cleaned_content'].apply(lambda x: tokenization(x.lower()))
```

Stop word Removal

After tokenization of data, stop words were removed from the data in hand. Stop words includes words such as “a”, “and”, “but”, “how” and “what”. It is important to remove these stop word as they act as noise or outliers in data which can negatively impact data in future. Moreover, considering stop words is not required in this analysis as it would not make any significant contribution to Topic distribution of Trump’s tweets.

```
stopword = nltk.corpus.stopwords.words('english')
def remove_stopwords(text):
    text = [word for word in text if word not in stopword]
    return text

trumpTweet_groupedDate['removed_stopwords'] = trumpTweet_groupedDate['Tweet_tokenized'].apply(lambda x: remove_stopwords(x))
```

Lemmatization

Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma. After removal of stop words from the tokenized form of words, we performed lemmatization which involved changing words in third person to first person and future tenses were changed into present form.

```
ps = nltk.PorterStemmer()
def lemmatize_stemming(text):
  word = WordNetLemmatizer().lemmatize(text, pos='v')
  #print(word)
  stemmed_word = nltk.PorterStemmer.stem(word)
  print(stemmed_word)
  return stemmed_word
def preprocess(text):
    result = []
    for token in gensim.utils.simple_preprocess(text):
      if token not in gensim.parsing.preprocessing.STOPWORDS and len(token) > 3:
        result.append(lemmatize_stemming(token))
        return result
```

Stemming

In linguistic morphology and information retrieval, **stemming** is the process of reducing inflected (or sometimes derived) words to their word **stem**, base or root form. For example, words like “happiness” was reduced to “happy” and “executed” was reduced to “execute”. Thus, we got words that were present in their root word format and can be grouped together with other words to give meaningful topics. snowball.EnglishStemmer() contained in nltk packages was utilized to perform stemming of words.

```
ps = nltk.snowball.EnglishStemmer()

def stemming(text):
    text = [ps.stem(word) for word in text]
    return text

trumpTweet_groupedDate['stemmed_words'] = trumpTweet_groupedDate['removed_stopwords'].apply(lambda x: stemming(x))
```

Finally, we get a dataset that displays the tweet_tokenized, removed_stopwords, stemmed_words and tweet_lemmatized columns.

date	content	cleaned_content	Tweet_tokenized	removed_stopwords	stemmed_words	Tweet_lemmatized
1/1/18	HAPPY NEW YEAR! We are MAKING AMERICA GREAT AG...	HAPPY NEW YEAR We are MAKING AMERICA GREAT AGA...	[happy, new, year, we, are, making, america, g...	[happy, new, year, making, america, great, muc...	[happi, new, year, make, america, great, much,...	[happy, new, year, making, america, great, muc...
1/1/19	HAPPY NEW YEARIpic.twitter.com/bHoPDPQ7G6-MEXI...	HAPPY NEW YEARpictwittercombHoPDPQGMEXICO IS P...	[happy, new, yearpictwittercombhopdpqgmexico, ...	[happy, new, yearpictwittercombhopdpqgmexico, ...	[happi, new, yearpictwittercombhopdpqgmexico, ...	[happy, new, yearpictwittercombhopdpqgmexico, ...
1/1/20	Get this straightened out, Governor @ GavinNew...	Get this straightened out Governor GavinNewso...	[get, this, straightened, out, governor, gavin...	[get, straightened, governor, gavinnewsomwonde...	[get, straighten, governor, gavinnewsomwond, a...	[get, straightened, governor, gavinnewsomwonde...
1/10/18	Today, it was my great honor to sign a new Exe...	Today it was my great honor to sign a new Exec...	[today, it, was, my, great, honor, to, sign, a...	[today, great, honor, sign, new, executive, or...	[today, great, honor, sign, new, execut, order...	[today, great, honor, sign, new, executive, or...
1/10/19	The Mainstream Media has NEVER been more disho...	The Mainstream Media has NEVER been more disho...	[the, mainstream, media, has, never, been, mor...	[mainstream, media, never, dishonest, nbc, msn...	[mainstream, media, never, dishonest, nbc, msn...	[mainstream, medium, never, dishonest, nbc, ms...

Feature Engineering:

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data. Feature engineering turn your inputs into things the algorithm can understand. Here, we have used different approaches to create numbers out of words which would help our predictive model to train.

Three approaches that we have used to perform feature engineering are explained as follows.

Bag of Words

In this approach a dictionary is created from the pre-processed word document created in the previous step. This dictionary consists of number of times a word has appeared in the dataset. This implies, we generate a document that comprises of the words stated with their frequency of occurrence in the document.

```
bow_corpus = [dictionary.doc2bow(doc) for doc in trumpTweet_groupedDate['Tweet_lemmatized_stemmed']]
len(bow_corpus)

Out[157]: 1148
```

```
bow_doc_1147 = bow_corpus[1147]
for i in range(len(bow_doc_1147)):
    print("Word {} (\"{}\") appears {} time.".format(bow_doc_1147[i][0],
                                                     dictionary[bow_doc_1147[i][0]],
bow_doc_1147[i][1]))
```

```
Word 0 ("administr") appears 3 time.
Word 8 ("despit") appears 1 time.
Word 10 ("done") appears 4 time.
Word 11 ("everi") appears 1 time.
Word 12 ("fail") appears 2 time.
Word 17 ("give") appears 1 time.
Word 20 ("help") appears 4 time.
Word 25 ("last") appears 2 time.
Word 26 ("leader") appears 2 time.
Word 27 ("leav") appears 2 time.
Word 33 ("mani") appears 2 time.
Word 34 ("much") appears 5 time.
Word 36 ("noth") appears 6 time.
Word 37 ("obama") appears 1 time.
Word 41 ("state") appears 2 time.
Word 44 ("think") appears 1 time.
Word 45 ("thought") appears 1 time.
Word 46 ("time") appears 2 time.
Word 47 ("today") appears 1 time.
Word 48 ("u") appears 1 time.
Word 49 ("unit") appears 1 time.
```

This bag of document created can be used to perform topic modeling as this desired by our problem statement.

TF-IDF

Another approach, that can be used of topic modeling is TF-IDF which is an abbreviation of Term Frequency-Inverse Document Frequency. This statistical measure tells the importance of word in the document. TF-IDF is the dot product of term frequency and inverse document frequency. TF-IDF weight consists of two parameters which are described as follows:

- TF: Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

  TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

- IDF: Inverse Document Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus, we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

  IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

Vectorization

In the previous approaches explained, model is trained using the words and their frequencies present in the document. While Vectorization is a technique in which words are converted into a word vector dataframe. This word vector dataframe consists of various possibilities of words called as n-grams which are not originally present in the dataset.

In this approach, words are converted into vectors that have a direction and magnitude. This conversion of words to vectors helps to find similarities between the words. A cluster of words having similar words or pattern is generated which can be used for better training the algorithm for topic modeling. The vectorized dataset that was created in our analysis is displayed below.

```
#Vectorizing the words
countVectorizer = CountVectorizer(analyzer=clean_text, token_pattern='[a-zA-Z0-9]{3,}', lowercase=True)
countVector = countVectorizer.fit_transform(trumpTweet_groupedDate['content'])
print('{} Number of tweets has {} words'.format(countVector.shape[0], countVector.shape[1]))
```

```
1148 Number of tweets has 16133 words
```

```
count_vect_df = pd.DataFrame(countVector.toarray(), columns=countVectorizer.get_feature_names())
count_vect_df
```

Out[172]:

	aaa	aap	aapour	ab	abandon	abba	abbaspictwittercomuzpivvrpgreat	abbott	abc	abcmr	abcnew	abcpoliticspictwittercomjixcsnzcpictwittercomyowcqlmad	abcwashington	abcworldnew	abduct	abdul	abdullah	abe
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Descriptive Statistics:

Descriptive statistics are used to describe the basic features of the data. They provide summaries about the sample and the measures. Here our only feature was the tweet content. The tweet contents contained words which are previously converted into list of lemmatized lists. There are few ways how we decided to visualize the various words in our dataset.

WordCloud:

Figure 1

In the above figure the wordcloud of the tweets represent the variety of words that were used by Donald Trump in majority of his tweets during his tenure as the President

Frequency of words before the stop words were removed

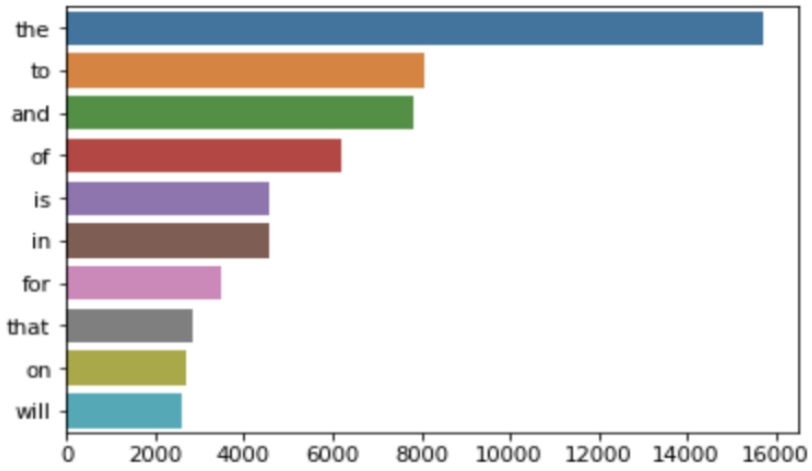


Figure 2

The above figure represents the frequencies of the tokenized words before the removal of the stop words. The visualization helps us understand the fact that stop words take up most of the space in a text space. The underlying principal of text mining is to set importance of a specific word based on its frequency of usage. Considering the fact that stop words do not add any value to our text, we shall remove them from the list of tokenized words.

Frequency of words after removing the stop words

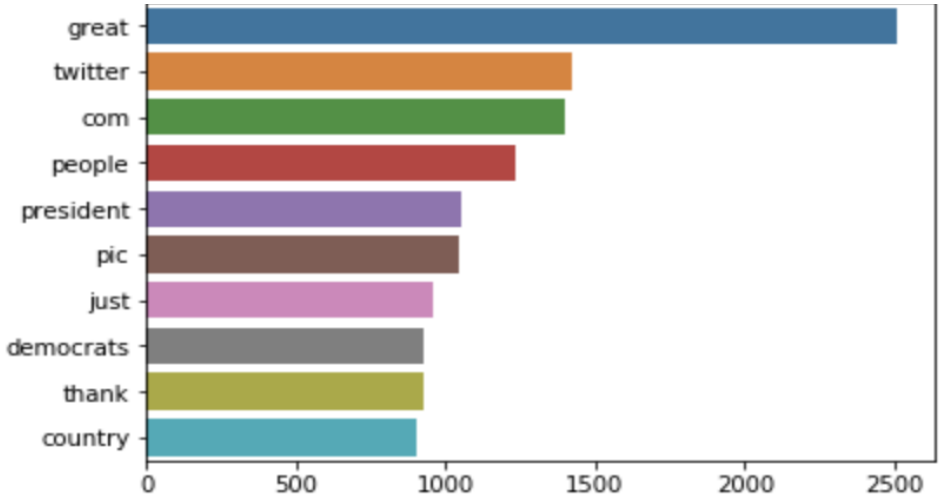


Figure 3

The above figure represents the frequencies of the top 10 words in the whole corpus i.e. the collection of all the tweets.

Latent Dirichlet Allocation (LDA for topic Modeling)

Topic modeling is a type of statistical modeling for discovering the abstract "topics" that occur in a collection of documents**. Latent Dirichlet Allocation** (LDA) is an example of topic model and is used to classify text in a document to a particular topic. It builds a topic per document model and words per topic model, modeled as Dirichlet distributions.

Finally, after pursuing the technique of converting words to vector i.e. the dataset is ready for LDA implementation and topic modeling. For designing the LDA model we use Scikitlearn's LDA package LatentDirichletAllocation and set the Hyperparameters manually first.

Initially the hyperparameters are set manually in order to observe the working of the model and get an idea of the performance based on the hyperparameters we set. Following is a snapshot of the algorithm and its parameters

```
lda_model = LatentDirichletAllocation(n_components = 20,               # Number of topics
                                      max_iter=10,               # Max learning iterations
                                      learning_method='online',
                                      random_state=100,          # Random state
                                      batch_size=128,            # n docs in each learning iter
                                      evaluate_every = -1,       # compute perplexity every n iters, default: Don't
                                      n_jobs = -1,               # Use all available CPUs
                                     )
lda_output = lda_model.fit_transform(countVector)
```

The hyperparameters set are as follows:

- Number of Topics we want our model to distribute the whole corpus into (n_components) which is set to 20. So, the output will give us 20 topics that will contain words closely associated.
- We set max_iter to 10 indicating the machine the times it needs to run over the data to learn its pattern. The more it runs the better it learns. This parameter may lead to over fitting (i.e. it should not high enough that leads to the machine memorizing the inputs) or under fitting (i.e. it should not be too low for the machine to not comprehend the inputs efficiently)
- Batch_size is set to 128 i.e. the number of documents (tweets per date) in one batch for the machine to process it for learning.

The same model is then fed by the vectorized words that we created in the previous segment of process. This is done by scikitlearn's function "fit_transform". This function first trains the model with the data and then runs the model on the data to create the topics.

In order to evaluate the performance of the above model's performance, we retrieve the performance parameters. The following snapshot describes the main performance parameters

```
# Log Likelyhood: Higher the better (Likelyhood score tells us how good the model is performing. The more the magnitude the better the model)
print("Log Likelihood: ", lda_model.score(countVector))

# Perplexity: Lower the better. Perplexity = exp(-1. * log-likelihood per word)
print("Perplexity: ", lda_model.perplexity(countVector))

# See model parameters
pprint(lda_model.get_params())
```

```
Log Likelihood:  -1276599.725280423
Perplexity:  2737.463145699325
{'batch_size': 128,
 'doc_topic_prior': None,
 'evaluate_every': -1,
 'learning_decay': 0.7,
 'learning_method': 'online',
 'learning_offset': 10.0,
 'max_doc_update_iter': 100,
 'max_iter': 10,
 'mean_change_tol': 0.001,
 'n_components': 20,
 'n_jobs': -1,
 'n_topics': None,
 'perp_tol': 0.1,
 'random_state': 100,
 'topic_word_prior': None,
 'total_samples': 1000000.0,
 'verbose': 0}
Command took 0.85 seconds -- by soadhika@syr.edu at 4/29/2020, 8:44:57 PM on Soham
```

The “score” function of LDA model gives us a measure called “Log Likelihood” that gives us a performance indicator. From the above image we see that log likelihood of the model -1276599.73. The more the magnitude of log likelihood, the better the performance of the model would be. Having said that, it is highly important to increase the efficiency of model. In this case we have specified number of topics as 20. Less number of topics with a high magnitude of Log Likelihood give us an efficient model.

Hence, in order to find the best combination of hyperparameters that the model could use to become best efficient we use a feature called “grid search”. This process is called Hyperparameter Tuning.

LDA with Grid-Search (Hyperparameter Tuning)

A model hyperparameter is a characteristic of a model that is external to the model and whose value cannot be estimated from data. The value of the hyperparameter has to be set before the learning process begins. For example, in the above model we set the parameters *n_components* and *max_iter*.

Grid-search is used to find the optimal hyperparameters of a model which results in the most ‘accurate’ predictions. In Grid-search function, we have the scoring parameter where we can specify the metric to evaluate the model on. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set. It adds grids and evaluates the prediction against various combinations across a set of values of the hyperparameters.

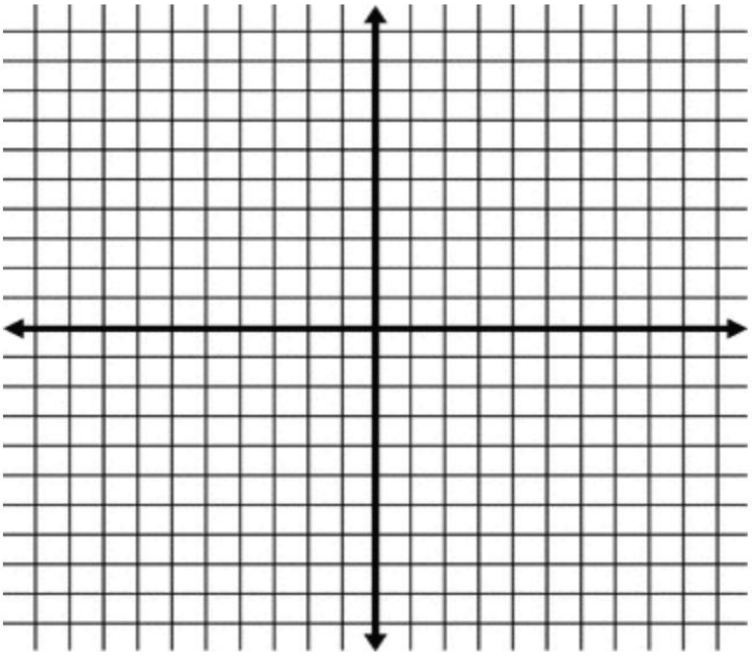


Figure 4

The above figure shows the way the algorithm works to tune the hyperparameter by evaluating various combinations of the hyperparameters to get the best combination that gives the most accurate results. Following snapshot of the code demonstrates the functionality of grid search on an LDA model.

```
# Define Search Param
search_params = {'n_components': [10, 15, 20, 25, 30], 'learning_decay': [.5, .7, .9]}

# Init the Model
lda = LatentDirichletAllocation()

# Init Grid Search Class
model = GridSearchCV(lda, param_grid=search_params)

# Do the Grid Search
model.fit(countVector)
```

```
/databricks/python/lib/python3.7/site-packages/sklearn/model_selection/_split.py:2053: FutureWarning: You should specify a value for 'cv' instead of relying on the
default value. The default value will change from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)
Out[181]: GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=LatentDirichletAllocation(batch_size=128, doc_topic_prior=None,
             evaluate_every=-1, learning_decay=0.7,
             learning_method='batch', learning_offset=10.0,
             max_doc_update_iter=100, max_iter=10, mean_change_tol=0.001,
             n_components=10, n_jobs=None, n_topics=None, perp_tol=0.1,
             random_state=None, topic_word_prior=None,
             total_samples=1000000.0, verbose=0),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_components': [10, 15, 20, 25, 30], 'learning_decay': [0.5, 0.7, 0.9]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)

Command took 6.13 minutes -- by soadhika@syr.edu at 4/29/2020, 8:44:57 PM on Soham
```

As shown by the image, the hyperparameters "n_components" is set over a range (10,15,20,25,30) with a learning decay (0.5, 0.7, 0.9). This suggests that the model will train on different combinations of number of topics with different portions of the entire corpus ranging from 10 to 30 topics and 50% to 90% of tweet content respectively.

The best model will be verified against the number of topics, the learning rate (the batch size) and the log Likelihood score of the model. Next step is to retrieve the hyperparameters of the best model.

```
# Best Model Parameters
best_lda_model = model.best_estimator_

# Model Parameters
print("Best Model's Params: ", model.best_params_)

# Log Likelihood Score
print("Best Log Likelihood Score: ", model.best_score_)

# Perplexity
print("Model Perplexity: ", best_lda_model.perplexity(countVector))
```

```
Best Model's Params:  {'learning_decay': 0.7, 'n_components': 10}
Best Log Likelihood Score:  -506056.5870967257
Model Perplexity:  2787.436974171556
Command took 0.78 seconds -- by soadhika@syr.edu at 4/29/2020, 8:44:57 PM on Soham
```

From the above output we reckon that the best model is with 10 topics learning on 70% of the tweet contents from the whole corpus. With the best model the likelihood score is -506056.59.

To get a better idea about the results of the different combinations of the hyperparameters we visualized the log likelihood scores with the different learning rates and number of topics. The following graph shows the comparison.

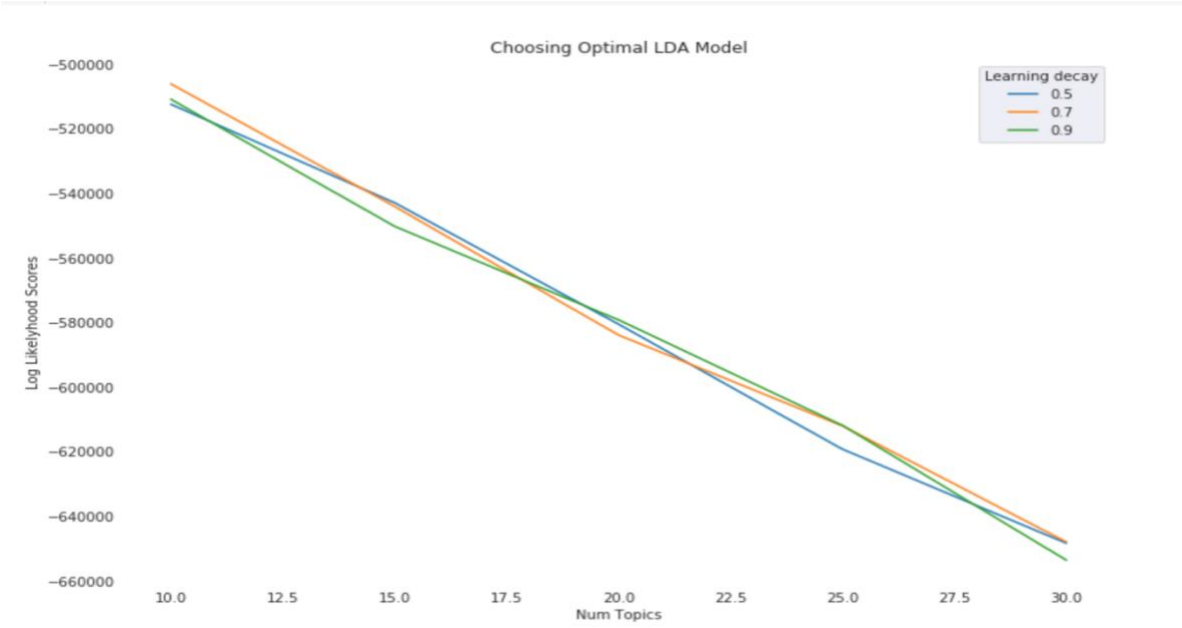


Figure 5

As mentioned above, topic modeling creates a list of topics containing closely associated words. According to the above figure we have the best model would have 10 topics. Every document (Tweet content) is associated with one or more topics, and each has a dominant topic. We regard the dominant topic of a document (tweet) as the most influencing topic for that particular tweet content. The following table gives us an idea of the how the documents are associated with the topics.

	Topic0	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	dominant_topic
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	9
1	0.29	0.00	0.00	0.00	0.00	0.00	0.00	0.16	0.00	0.55	9
2	0.61	0.00	0.00	0.00	0.00	0.38	0.00	0.00	0.00	0.00	0
3	0.00	0.00	0.73	0.00	0.00	0.00	0.00	0.00	0.00	0.26	2
4	0.00	0.00	0.00	0.00	0.13	0.00	0.00	0.00	0.00	0.86	9
5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	8

The above figure shows the association of the tweets (in sequential order as the sorted dates) with the various topics and the dominant topic.

The result of the LDA model also tells us how the vectorized words are associated with each topic by assigning the vectors different weights. This is the most significant advantage of pursuing the word2vector approach for topic modeling. Every n-gram is assigned a weight by the model and that increases the efficiency in predicting the topic for words that were never fed to the machine while training.

But in order to get a better idea let us take a look at how the regular words are associated with the topics. The following image represents the different words and their level of association with each topic.

	Word 0	Word 1	Word 2	Word 3	Word 4	Word 5	Word 6	Word 7	Word 8	Word 9	Word 10	Word 11	Word 12	Word 13	Word 14
Topic 0	great	democrat	presid	impeach	peopl	republican	noth	get	vote	countri	us	state	go	border	thank
Topic 1	great	us	presid	china	state	year	trade	get	mani	work	countri	go	make	american	honor
Topic 2	great	peopl	state	countri	job	big	america	thank	border	work	new	make	american	get	love
Topic 3	elijah	cum	baltimor	corker	convinc	million	fort	myer	hurricaneharvey	contribut	inspir	racist	tennesse	space	marsha
Topic 4	switzerland	davo	rememb	harbor	pearl	daytona	enjoy	meddl	america	wisconsin	dnc	interview	econom	collect	foxandfriend
Topic 5	first	epa	honor	w	bless	nick	enforc	l	enter	law	polic	california	god	antitrump	jurisdict
Topic 6	american	thank	great	honor	commit	night	see		vote	florida	berni	state	everyon	forward	obamacar
Topic 7	great	tax	go	must	korea	countri	peopl	cut	today	bad	north	nation	news	test	everi
Topic 8	great	news	fake	get	us	state	job	time	never	american	border	presid	peopl	work	republican
Topic 9	great	peopl	democrat	fake	news	presid	countri	trump	year	time	want	get	us	border	big

Visualizing word Vectors using PCA (Principle Component Analysis)

At this stage of the work, we introduce an approach to visualize and get a better idea of how the words are distributed across various topics. We introduce an approach called Principle Component Analysis. Our goal is to visualize the words in the form of vectors to understand their degree of association by measuring the distance between them creating clusters

Principle Component Analysis is a technique for feature extraction — so it combines our input variables in a specific way, then we can drop the "least important" variables while still retaining the most valuable parts of all of the variables. This is exactly what is done on the Word Vector and Topic data frame to visualize how far apart the words are and clusters are formed depending on their directional distance between them.

The following figure shows the clusters of words under each topic on the left side and on the right side it shows the weights of the words that are within the selected cluster. The distance between each cluster is measured from the center of each cluster. This is an interactive visualization created using pyLDAvis package, but the below figure is only a screenshot representation of the actual visualization.

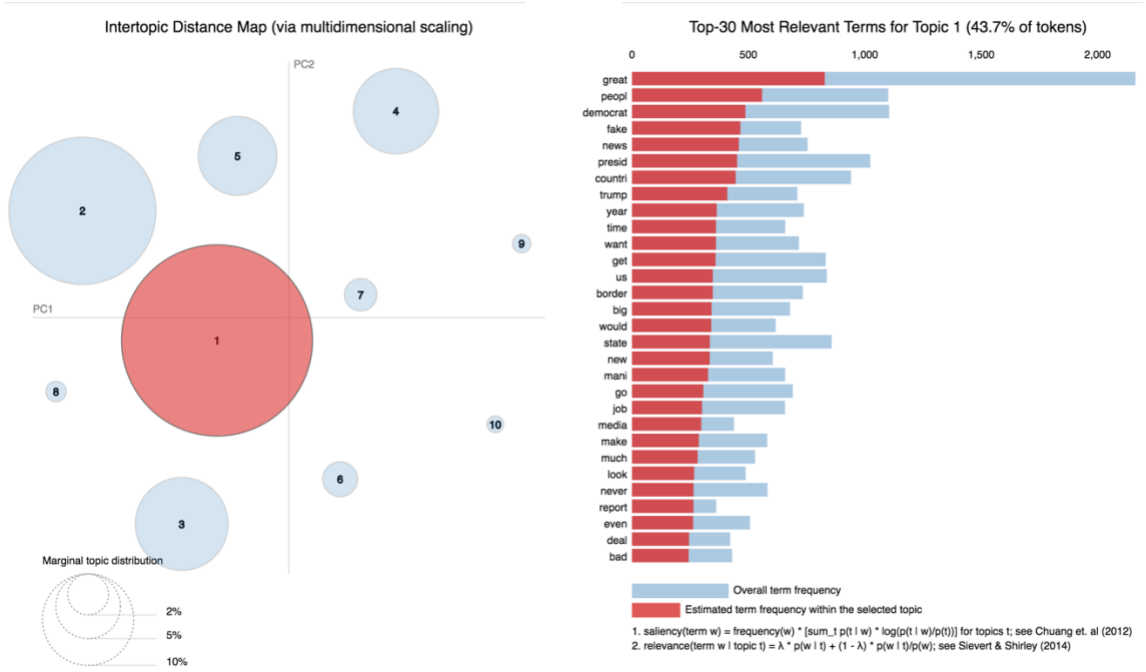


Figure 7

Hence, our model is now ready to consume any tweet made by the president of the United States of America and tell us its degree of association with each topic. As we saw earlier every tweet will have a maximum association with one specific Topic. We could ideally categorize the tweet to be having subjects from the topic it is most associated with. The following snapshot shows us the implementation of the topic model, where we take a random tweet made by Donald trump and watch its degree of association with each topic.

```python
# Define function to predict topic for a given text document.
nlp = spacy.load('en', disable=['parser', 'ner'])

def predict_topic(text, nlp=nlp):
    global sent_to_words
    global lemmatization

    # Step 1: Clean with simple_preprocess
    mytext_2 = list(sent_to_words(text))

    # Step 2: Lemmatize
    mytext_3 = lemmatization(mytext_2, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

    # Step 3: Vectorize transform
    mytext_4 = countVectorizer.transform(mytext_3)

    # Step 4: LDA Transform
    topic_probability_scores = best_lda_model.transform(mytext_4)
    topic = df_topic_keywords.iloc[np.argmax(topic_probability_scores), :].values.tolist()
    return topic, topic_probability_scores

# Predict the topic
mytext = ["I have instructed the United States Navy to shoot down and destroy any and all Iranian gunboats if they harass our ships at sea."]
topic, prob_scores = predict_topic(text = mytext)
print(prob_scores)
```

```
[[0.01250419 0.01250404 0.01250588 0.0125     0.0125     0.01250083
  0.0125024  0.0125011  0.01250184 0.88747971]]
```

The above figure takes a tweet that reads “I have instructed the United States Navy to Shoot down and destroy any and all Iranian gunboats if they harass our ships at sea”. Our model tells us that it has the highest association with topic 9 with a magnitude of 0.89 approximately. If we map Topic 9 to the word’s topic dataframe we can easily analyze the subjects that are associated with this tweet made by Donald Trump. The words are [great, people, democrat, fake, news, presid, country, trump, year, time, want, get, us, border, big]. This gives an idea that President Trump is talking about protecting his country at the border and reassuring the people of the US that nothing can harm them.

Data Preparation for Supervised Machine Learning

After implementation of Topic modeling using LDA in previous step, we prepared data to perform supervised machine learning in order to predict stock prices based on topics discussed in President Trump’s tweet. This preparation step involves extraction of National Grid stock prices from Yahoo Finance and preparing it accordingly to fit the final dataset achieved after topic modeling.

Stock price dataset is downloaded for the period of January 20, 2017 until March 25, 2020 which corresponds to tweet data in the tweet dataset. A snapshot of data is displayed below.

```
national_grid_stock_df.show()
```

```
+----------+---------+---------+---------+---------+---------+-------+
|      Date|     Open|     High|      Low|    Close|Adj Close| Volume|
+----------+---------+---------+---------+---------+---------+-------+
|2017-01-23|63.810043|64.159386|63.722706|64.104805|54.674591| 459000|
|2017-01-24|63.799126|64.181221|63.777294|63.897381|54.497673| 652000|
|2017-01-25|64.104805|64.126640|63.635372|63.755459|54.376633| 425300|
|2017-01-26|63.133186|63.242359|62.783844|63.176857|53.883141|1828600|
|2017-01-27|63.329693|63.427948|63.122272|63.187775|53.892460| 427400|
|2017-01-30|62.838428|63.133186|62.783844|62.893013|53.641052| 666600|
|2017-01-31|63.351528|63.962883|63.286026|63.831879|54.441807| 845500|
|2017-02-01|63.373363|63.722706|62.936680|63.351528|54.032120| 795200|
|2017-02-02|63.198689|63.449783|63.111355|63.351528|54.032120| 479700|
|2017-02-03|63.406113|63.613537|63.176857|63.329693|54.013500| 401800|
|2017-02-06|63.569870|63.668121|63.384281|63.471615|54.134544| 551700|
|2017-02-07|63.679039|64.497818|63.526199|64.323143|54.860798| 677200|
|2017-02-08|65.447601|65.862442|65.131004|65.338425|55.726730| 894700|
|2017-02-09|65.338425|66.331879|65.196510|65.360260|55.745354| 894900|
|2017-02-10|65.032753|65.458519|65.032753|65.371178|55.754665| 371200|
|2017-02-13|65.393013|65.447601|65.120087|65.382095|55.763981| 538600|
|2017-02-14|65.633186|65.655022|64.901749|65.229256|55.633621| 474400|
|2017-02-15|64.421394|65.065506|64.355896|65.065506|55.493965| 473000|
|2017-02-16|64.956329|65.469429|64.956329|65.436684|55.810535| 311900|
|2017-02-17|65.502182|65.556770|65.032753|65.382095|55.763981| 398600|
+----------+---------+---------+---------+---------+---------+-------+
only showing top 20 rows
```

We merged this dataset retrieved from yahoo finance with the resulting dataset from topic modeling which comprised of President's tweet and the topics discussed in each tweet with their weights. Two datasets were merged based on the tweet date. Further, columns such as open, High, Low, Volume etc. which were not important for analysis were removed and Close column that indicates the closing price of the stock on the particular day was kept. The merged dataset is displayed below.

```
Topic_stock_df_clean.rename(columns={'Close': 'Prices'}, inplace=True)
Topic_stock_df_clean.head()
```

Out[218]:

	date	content	Topic0	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	dominant_topic	Prices
0	2017-01-20	Thank you for joining us at the Lincoln Memori...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	9	NaN
1	2017-01-21	THANK YOU for another wonderful evening in Was...	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.00	0.00	0.87	9	NaN
2	2017-01-22	Had a great meeting at CIA Headquarters yester...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	9	NaN
3	2017-01-23	Busy week planned with a heavy focus on jobs a...	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.94	9	64.104805
4	2017-01-24	Will be meeting at 9:00 with top automobile ex...	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1	63.897381

This merged dataset consisted of Null values in the prices column for days such as weekends and national holidays. We needed to fill in these values in order to perform further analysis and apply machine learning algorithm. Two approaches were used to fill these empty cells in the price column. Firstly, we used a techniques called Forward Fill in order to fill last day's (yesterday's)

stock price in the empty cells. ffill() function that is designed for Pandas is used that stands for forward fill which fills the empty cell last valid observation.

Secondly, when there was no data available for the previous day as in the case displayed above i.e. for the first three days in the dataset, we used average stock price to fill this empty cell. The following figure shows the final dataset which was prepared for supervised machine learning.

```
Topic_stock_df_clean.head()
```

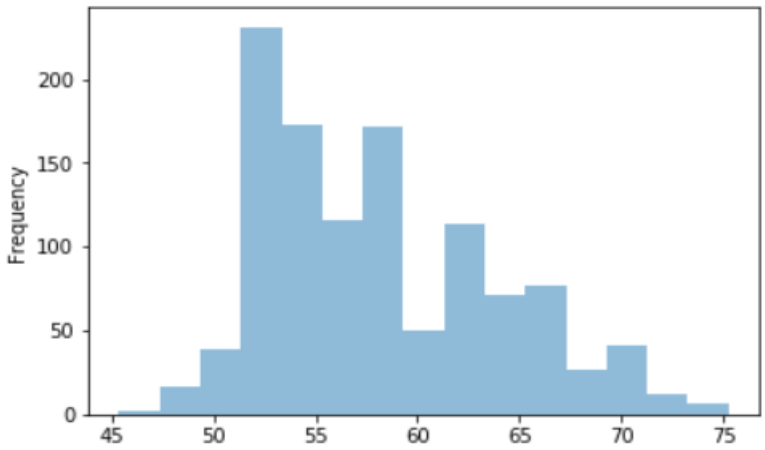
Out[223]:

	date	content	Topic0	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	dominant_topic	Prices	label
0	2017-01-20	Thank you for joining us at the Lincoln Memori...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	9	58.154025	58.154025
1	2017-01-21	THANK YOU for another wonderful evening in Was...	0.00	0.00	0.00	0.00	0.00	0.09	0.00	0.00	0.00	0.87	9	58.154025	58.154025
2	2017-01-22	Had a great meeting at CIA Headquarters yester...	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.98	9	58.154025	58.154025
3	2017-01-23	Busy week planned with a heavy focus on jobs a...	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.94	9	64.104805	64.104805
4	2017-01-24	Will be meeting at 9:00 with top automobile ex...	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1	63.897381	63.897381

Further, while applying any machine learning algorithm it is essential to know the skewness present in the dependent variable. In our case, Stock Price is the dependent variable. It is very important to review the skewness in the data as it could have a direct impact on the accuracy of our machine. As the regression model requires multivariate normality in its data that is it requires its dependent variable to be normal.

By presenting any data with high skewness level to our regression model can result into confidence intervals be too wide or too narrow as they are based on the assumption of the normally distributed errors. If there was any skewness present in the data, we needed to use log transformations and use log transformed dependent variable for our analysis. Therefore, we verified the skewness of the data (dependent variable) and got that the distribution is normally distributed, the result is displayed below.

```
#Verifying skewness of the data
Topic_stock_df_clean.Prices.plot.hist(bins=15, alpha=0.5)
```



Supervised Learning Model

As we enter the second half of our project, we move our focus toward selecting the best supervised learning model for the scenario. After several studies of different models and the nature of our data we came to decide to implement Random Forest Regressor for making the stock price predictions.

Random Forest Regressor

To explain about Random Forest Regressor, it is important to ensure if understand the underlying working principles of a Decision Tree.

A decision tree is a super simple structure we use in our heads every day. It's just a representation of how we make decisions, like an if-this-then-that game. First, you start with a question. Then you write out possible answers to that question and some follow-up questions, until every question has an answer. Let's look at a decision tree for deciding whether or not someone should play baseball on a specific day.

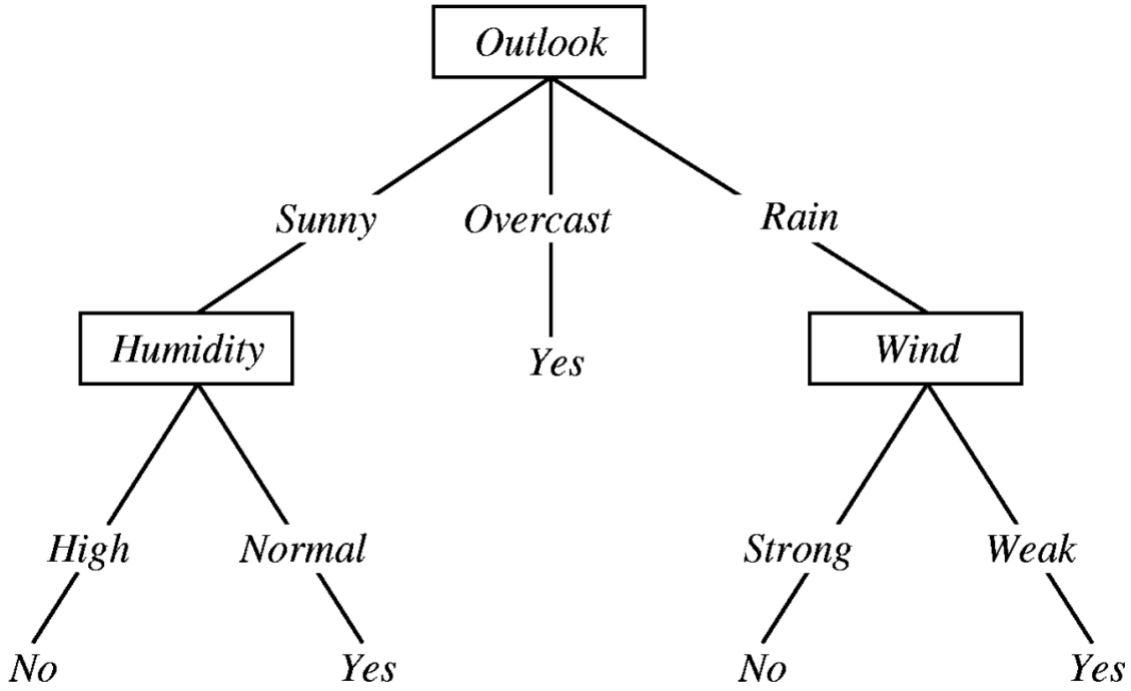


Figure 8

Random Forest is just a bunch of decision trees, Ensemble models like Random Forest are designed to decrease overfitting and variance by using bagging algorithms. As we know that Decision Trees are prone to overfitting. In other words, Decision Tree works best for finding solution for a specific problem but fails to solve for a problem it has not seen before. ensemble models use many decision trees that are good at their particular task to make a larger model that's great at many different tasks.

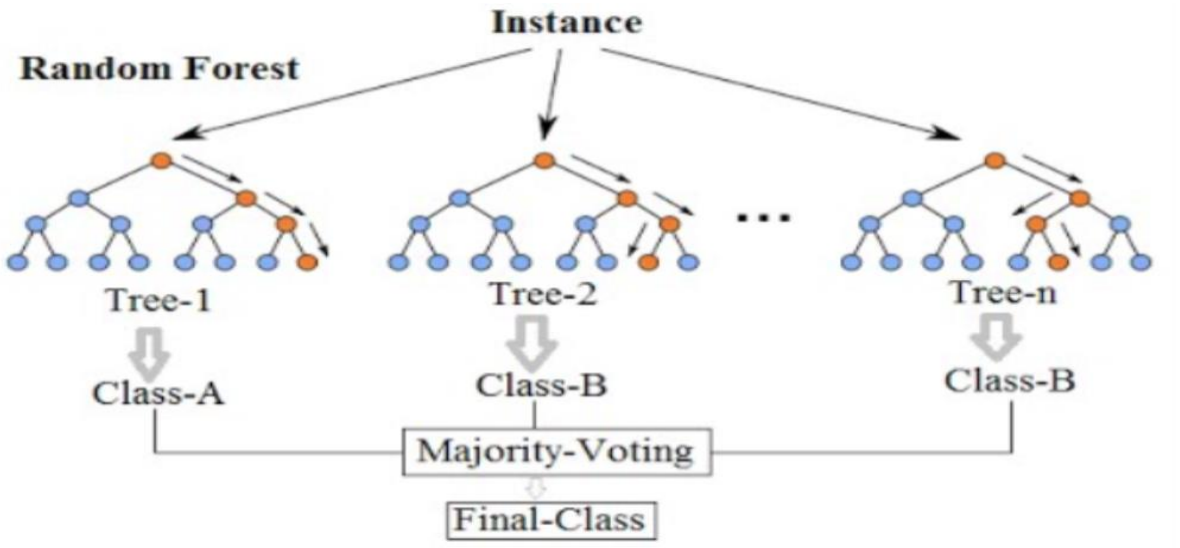


Figure 9

Hence, Random Forest regressor, randomly picks a set of features from the total number of features given and creates a decision tree for each feature. In the end it aggregates the results given by all the decision trees hence reducing the chances of over fitting.

Our features from the current dataset are our Topics, i.e. from Topic 0 to Topic 9. As the objective is to predict the stock prices based on the tweets, we now have all the topics from those tweets (that has weights from the tweets).

For Random forest regressor we used the function from pyspark and hence we decided to bring our dataset to the spark world. The following snapshot shows the process of assembling all the features under one column for RandomForest Regressor.

```
from pyspark.ml.feature import VectorAssembler

feature_list = []
for col in Topic_stock_df_clean.columns:
    if col == 'label' or col == 'Prices' or col == 'date' or col == 'content' or col == 'dominant_topic' or col == 'log_value' :
        continue
    else:
        feature_list.append(col)

assembler = VectorAssembler(inputCols=feature_list, outputCol="features")
print(feature_list)
```

Spark's "VectorAssembler" function is used to gather the features under one column for RandomForest Regressor.

As we want the machine to identify the best set of hyperparameters for our model, we make use of Grid Search to tune our Hyperparameters. The following snapshot shows the grid search configuration for grid search.

```
#Configuring Grid Search Parameters
from pyspark.ml.tuning import ParamGridBuilder
import numpy as np

paramGrid = ParamGridBuilder() \
    .addGrid(rf.numTrees, [int(x) for x in np.linspace(start = 25, stop = 35, num = 5)]) \
    .addGrid(rf.maxDepth, [int(x) for x in np.linspace(start = 6, stop = 12, num = 5)]) \
    .addGrid(rf.featureSubsetStrategy, ['auto', 'sqrt']) \
    .build()
```

The above image shows that for every hyperparameter, the grid is configures with a range of values. The number of trees for each decision tree has been set to a range of 25-35 (with 5 as the maximum number of values to be taken). The depth of the trees for the decision trees is set to the range 6-12 (with 5 as the maximum number of values to be taken). The last grid specifies the number of random features Random Forest should pick. Usually it is the square root of the total number of features ($p = \sqrt{m}$).

Once we have the grid search configured, we are now on to the next step of configuring the cross-validation evaluator. Cross-validation evaluator runs the grid search for different combinations of hyperparameters and verifies the result against one another. In the end it results a best model with the best combination of hyperparameters. The following figure shows the configuration of a cross-validator evaluator.

```
#Configuring Crossvalidator with RandomForest Pipeline, Grid Searchm and Regression Performance evaluator to get the best model
from pyspark.ml.tuning import CrossValidator
from pyspark.ml.evaluation import RegressionEvaluator

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator(),
                          numFolds=3)
```

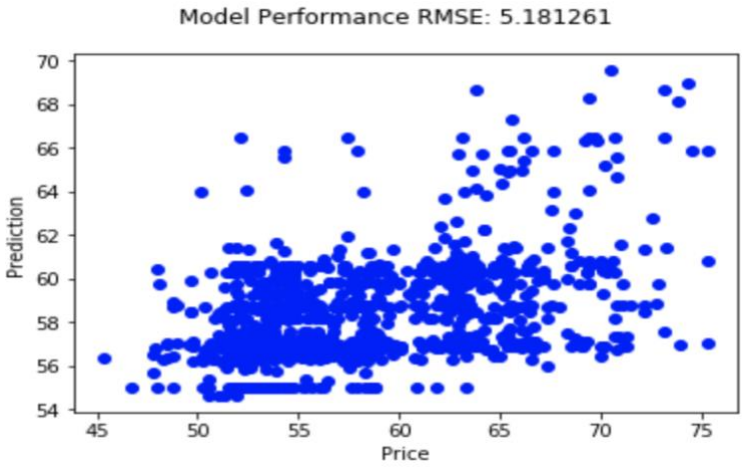
The parameters of cross-validation evaluator are the randomforest pipeline, the grid search build and the evaluator. The whole process is run 3 times to come up with the best model. The whole dataset is split into two halves, training data and testing data. Usually a random split is done in the ratio of 80% to 20%.

The final best model is run with the test data and the a predictor model is successfully created with an error of approximately $5.

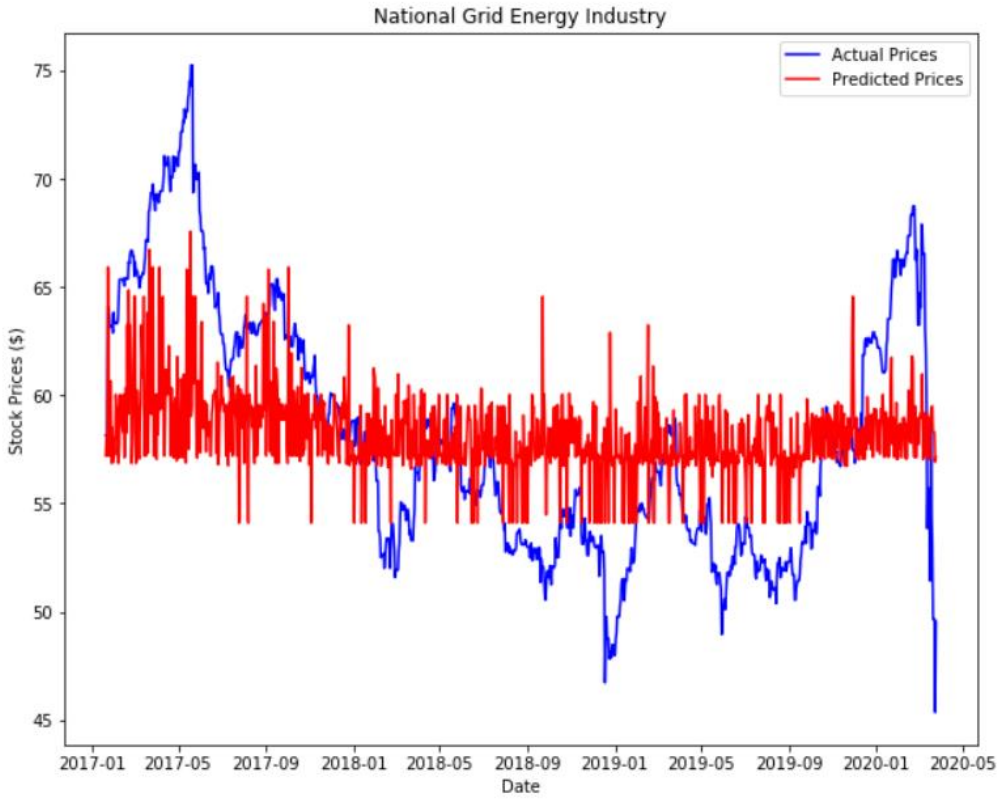
Best Model Presentation after implementing Random Forest Regression

We received a best model after implementing grid search in random forest regressor. This model is best model provided parameters such as Num of Trees as 32 and Max Depth as 6. The best model is then fitted on the whole dataset consisting of 1148 datapoints i.e. tweets and a root mean squared error (RMSE value) is calculated. RMSE value is used to check the accuracy of the regression model and our model achieved a RMSE of 5.1.

This indicates that our model predicts the values of stock prices based on the President's tweet with an error of $5.1. A scatter plot is plotted to display the relation between the actual and the predicted value of stock prices. The scatter plot displayed below reiterates the fact that our model gives an approximately linear relation between actual and predicted stock with some outliers due to RMSE value.



Interpreting relation between Actual and Predicted Stock Prices



The actual and predicted stock prices for the National Grid stock is plotted using a Time Series plot as shown above. We can provide various conclusions based on the above plot. Firstly, we can interpret that during the period of January to May, 2017 when Donald Trump was in his nascent stage of presidential rule, a peak in the National grid stock prices was observed. Our machine predicts this peak in prices with minimum error possible. This peak also strengthens our topic of discussion that there is an effect of Trump's tweets on energy stock prices.

Secondly, we observe very less volatility during the remaining period of the plot shown. This is caused as the stock prices have a stationary distribution unless there has been a major impact that has hit the market. The initial peak was seen as an effect of Trump's tweets. Therefore, we can state that there are some other lurking variables responsible for volatility shown in stock prices during the remaining period of consideration.

Demo of the Working Model and Conclusion

The screenshot above showcases the full demonstration of the working model i.e. stock price prediction after topic modeling. For demonstration, President's tweet was taken for the day of March 10, 2020 and was fed into the machine. Initially, topic modelling was applied to this inputted tweet and was divided into various topics with appropriate weights. Then, the topic with the highest weight was printed. This is the topic with which the tweets relate the most.

Moreover, the words comprised in that topic is printed and the demo clearly shows that the input tweet relates to the printed topic. Further, the concluded topic predicts the stock price for that day. The stock price for the said day is predicted to be $62.5 whereas it was actually $62.08. Therefore, our machine predicts the stock price for National Grid stock with a minimal error.

Our machine captures the volatility caused in Energy Stock Prices by President Trump's tweets. Hence, predicting future stock prices of the energy industry. This gives us an idea that in current times one of the major influencers of the stock market is sentiments of the Political personalities. This prediction would most certainly help Energy industry corporate organizations take wise business decisions in the area of stock investments.

Now we agree to the fact that there are other lurking variables that directly or indirectly effect the stock market prices. Hence, this model could be further developed in the future by taking other influential variables into account.

References:

1. Chen, Y. (2019, February 26). How to build a LDA Topic Model using from text. Retrieved from https://medium.com/@yanlinc/how-to-build-a-lda-topic-model-using-from-text-601cdcbfd3a6
2. Li, S. (2018, June 1). Topic Modeling and Latent Dirichlet Allocation (LDA) in Python. Retrieved from https://towardsdatascience.com/topic-modeling-and-latent-dirichlet-allocation-in-python-9bf156893c24
3. sklearn.model_selection.GridSearchCV¶. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV
4. Brems, M. (2019, June 10). A One-Stop Shop for Principal Component Analysis. Retrieved from https://towardsdatascience.com/a-one-stop-shop-for-principal-component-analysis-5582fb7e0a9c
5. Singh, S. (2018, October 9). Understanding the Bias-Variance Tradeoff. Retrieved from https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229
6. Koehrsen, W. (2018, January 10). Hyperparameter Tuning the Random Forest in Python. Retrieved from https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
7. Classification and regression. (n.d.). Retrieved from https://spark.apache.org/docs/2.1.0/ml-classification-regression.html#random-forest-regression
8. Lorberfeld, A. (2019, March 26). Machine Learning Algorithms In Layman's Terms, Part 2. Retrieved from https://towardsdatascience.com/machine-learning-algorithms-in-laymans-terms-part-2-a0a74df9a9ac
9. Singman, B. (2020, April 22). Trump says he's instructed Navy to 'destroy' any Iranian gunboats harassing US ships. Retrieved from https://www.foxnews.com/politics/trump-says-hes-instructed-navy-to-destroy-any-iranian-gunboats-harassing-us-ships
10. Prabhu. (2019, November 21). Understanding NLP Word Embeddings - Text Vectorization. Retrieved from https://towardsdatascience.com/understanding-nlp-word-embeddings-text-vectorization-1a23744f7223
11. Wu, H. C., Luk, R. W. P., Wong, K. F., & Kwok, K. L. (2008). Interpreting TF-IDF term weights as making relevance decisions. ACM Transactions on Information Systems, 26(3), 1–37. doi: 10.1145/1361684.1361686
12. National Grid Transco, PLC Nati (NGG) Stock Historical Prices & Data. (2020, May 3). Retrieved from https://finance.yahoo.com/quote/NGG/history?p=NGG
13. Trump Twitter Archive. (n.d.). Retrieved from http://www.trumptwitterarchive.com/
14. D'Souza, J. (2018, April 4). An Introduction to Bag-of-Words in NLP. Retrieved from https://medium.com/greyatom/an-introduction-to-bag-of-words-in-nlp-ac967d43b428
15. Sharma, R. (2019, July 30). Skewed Data: A problem to your statistical model. Retrieved from https://towardsdatascience.com/skewed-data-a-problem-to-your-statistical-model-9a6b5bb74e37