Machine Learning Engineer Nanodegree Capstone Project

Can you out-predict an Index Fund?

Eric Westerkamp
December 31, 2017

Contents

I. Definition

Project Overview

The ability to predict the stock market is a compelling goal. The perceived ability to predict the future movement of stocks and bonds is the foundation of a multi-trillion dollar industry. Portfolio managers, mutual fund advisors, financial planners – 10's if not 100's of thousands of people across the world get paid to predict the future movement of the markets.

Recently, though, there has been a shift toward index investing. The theory is that over time the extra expense investors pay for active investing causes active portfolio's to under-perform low cost investments in stock index funds. Conventional U.S. stock mutual funds and ETF's that invest passively now hold $3.6 trillion in value and account for 42% of all U.S. stock fund assets. An increase from 12% in 2000 to 24% in 2010 (Petruna, 2017). This increase has come at the expense of actively traded funds, in 2016 alone $264.5 billion flowed out (Braham, 2017).

The question is - can markets be predicted in a way that can then be leveraged to outperform passive index investing? I have always been drawn to this question; with enough information can active equity management out-perform an Index investing?

This goal of this project is to see if machine learning, with access to similar information as a portfolio manager, can predict whether or not the overall stock market is going to go up or down in the near term. Is there enough information publically available to do a better job of predicting the market than just a buy and hold model? This project will leverage existing information, in the form of news headlines, to try and predict if a given basket of index funds is going to be up or down for the next day. Reliably being able to do this would be an indicator that there is public information available that would enable an active investor to beat passive investing. An inability to do so would indicate that financial news headlines are not a reliable source of information that would enable better performance by active investors.

Problem Statement

According to Vanguard and Morningstar 67 percent of all active equity funds have underperformed their benchmarks in the decade through Dec. 31, 2016. If you account for liquidated funds that number jumps to 86% (Braham, 2017). This is an astoundingly bad track record and indicates that your typical active equity fund cannot out-perform index funds by enough margin to justify their extra expense ratio. Given how much information is broadly available and how efficient and fast the markets are, can any individual or organization pull together enough information that would enable them to predict the market at an increased percentage over the indexes?

The problem I would like to solve is to understand if using financial news headings would have the ability to predict the markets at a rate better than passive index investing. Can the next day's market movement, up or down, be predicted? The *Efficient Markets Hypothesis* states that market prices immediately reflect all available information. Pablo Daniel Azar's paper, *Sentiment Analysis in Financial News*, specifically addresses this Efficient Market Hypothesis (Azar, 2009). He talks about how the Hypothesis states how difficult it is to make a profit on the markets using information, unless an individual has access to information before the market does. This theory would then indicate that it would be difficult, if not impossible to use market based sentiment to gain insight into the market. This would also indicate that active investors, without having access to hidden information, would not be able to outperform the market.

Pablo's paper also went on to explain his approach to solving this exact problem. In it he merged a corpus of news from Reuters from the years 1998-2009, a dataset of quarterly financials, a dataset of earnings surprises, and an indication of abnormal returns. The abnormal returns are essentially the data-point he is trying to predict. His

general approach for sentiment analysis was to count the number of pos/neg words in a given article as it relates to a specific stock. Pablo determined in his paper that Pos and Neg sentiment was able to predict outcomes on day 0 (the day of the news story) but not on the following day (day+1). However, Pablo only used statistical analysis and not Machine Learning to test this assumption. Can Supervised Learning learn patterns that would enable better results?

The strategy I am going to use is to pull financial news headlines for a period of time going back at least five years. Each day's set of headlines will be used to predict whether or not the closing value (Adj. Close) of a set of indexes will be either higher or lower than the current days. At the end of the project we should have predictions against three different indexes (S&P 500, Dow Jones Industrial Index, MSCI World Index) for a period of approximately five years. We can then use these predictions to determine if a portfolio using the best set of predictions can out-perform a passive investment in the same index.

Metrics

The goal of this project is to eventually determine if there is enough information contained within the news headings to make predictions on whether or not the next day's market index is going to be up or down. This is not a simple task and is compounded by the fact that the markets themselves have trended upwards for the past 7 years, creating imbalanced data. For this reason the model should be able to predict which days are going to be down at a reliable rate. This means that the accuracy metric is not a good indication and we should be looking at Precision and Recall. Specifically the balance between the two indicated by the F1 metric.

Our eventual model will look at each day and predict if the next days' market is up or down. Then it will be a simple task of counting how many days it got correct (up or down) and creating a simple confusion matrix. That will then tell us the Precision, Recall and F1 scores.

Note that we will be are looking very hard at the number of True Negatives as they are the indication that the next day's market is going to "buck the trend" and be down.

The final metric we will use is to find the best model of all run and use it to simulate buying and selling the index based on the predictions and compare it back to a model where we just bought and held. Can we actually build predictions that enable us to generate a higher return than just buy and hold?

The Core Metric chosen is F1 because it balances between precision and recall. This balance helps us focus on models that choose both True Positives and True Negatives. We are eventually looking for models that do better than index investing, which are effectively predicting "up" every day (as you buy and hold the index indefinitely). So a model that blends in picking "down" without sacrificing too much recall could potentially beat standard index investing.

II. Analysis

Data Exploration

For this problem we are going to require two different data sets. Once is a corpus of news headers for the selected time period. The second is the daily data for each selected stock index for the same period. The index information is easy to acquire. It can be pulled directly from the finance.yahoo.com web-site using standard python APIs.

The corpus of news headers is much harder to locate. There are a number of data-sets in existence, including a Kaggle Dataset (Aaron7Sun, 2016) and the Reuters data set included in the NLTK toolkit. However, there are a couple of issues with each. Neither contains information up to 2017 nor is either focused on financial news. For this project I want to pull headlines that are indicative of market movements, as that news is more likely to be correlated with changes in the market.

The data-set I finally chose is the daily news headlines from Bloomberg news. This data-set was chosen because it ends to be focused on news affecting the financial markets and is available going all the way back to 2007 on Archive.org.

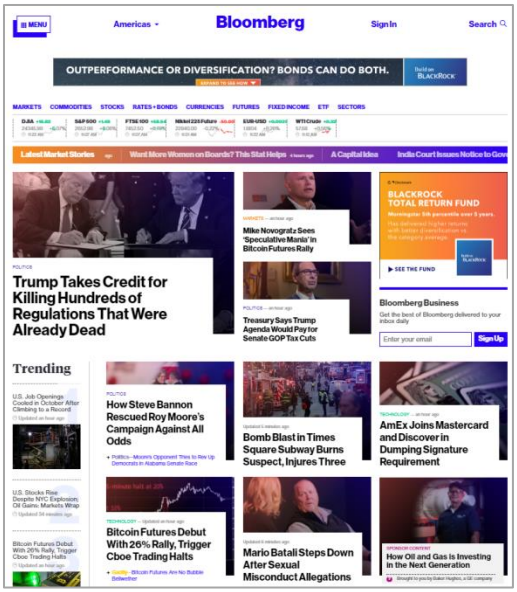
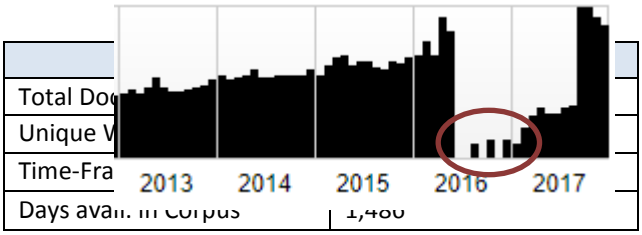


Figure 1 Bloomberg Website

Details

The ***Scrapy API*** was used to create a web scraper that collects all Bloomberg pages going back to 1/1/2013. During that time period of 1,795 days there is information available for 1,486 days. One outlier is that for the period of time between June to December 2016 there are limited pages available. This reduces the over-all size of the data-set.

For the same period of time, 1/1/2013 – 12/1/2017, the daily trading information was pulled for the S&P500, Dow Jones Industrial Average and the MSCI world index from finance.yahoo.com. This reading is accomplished using the ***pandas_datareader*** API.



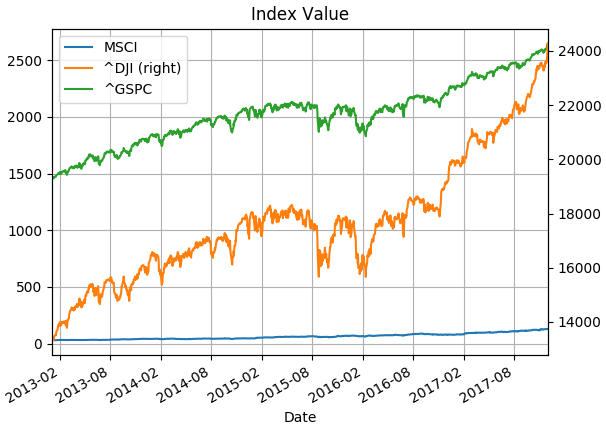
Unique Data Notes

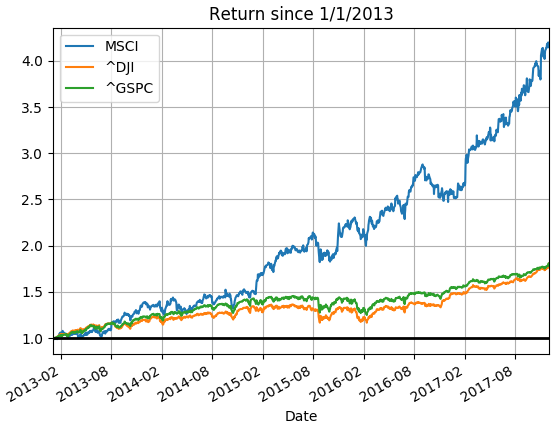
1. The Bloomberg data-set contains a large amount of days where there are multiple web-pages pulled for each day. Often as many as 10-20. This means there could be huge duplication in the article headings for specific days. To handle

Key Stats for Index's			
	^GSPC	**^DJI**	**MSCI**
1/1/2013 Adj. Close	1,462.42	13,412	30.80
12/1/2017 Adj. Close	2,642.21	24,231	128.95
% Change in Value/Growth	1,179/80.67%	10,819/80.66%	98.14/318.5%
Compound Annual Growth Rate (CAGR)	12.6%	12.6%	33%
Trading Days	1,240	1,240	1,240
Avg. Daily Trading Volume	3,542,242,862	147,022,830	598,826

this only the first web-page for each day will be used. This should separate page pulls by approximately 24 hours.

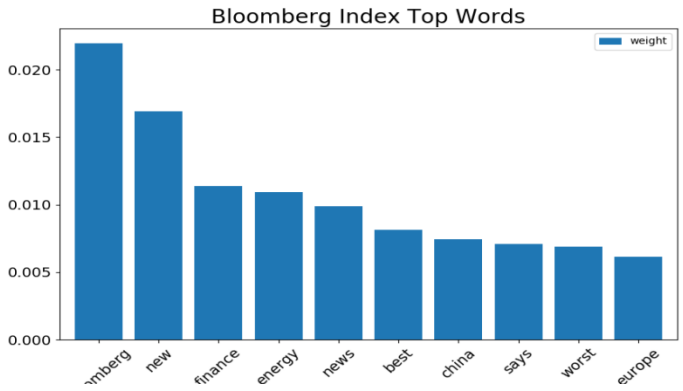
2. The ^GSPC and ^DJI indexes are highly correlated. They both have almost exactly the same CAGR and total change in index value over the period. It is assumed that this is caused by a large number of companies being the same across the two indexes.
3. Due to the time-range selected, 2013-2017, all indexes have had material positive growth. We will need to take that into account when determining and evaluating appropriate success metrics.





Two charts show us what is going on with these stocks. The one on the left shows us the value of the stocks since 1/1/2013. The right shows the return since 1/1/2013 for each stock. As you can see this highlight the fact that MSCI world index has increased by a much larger percentage during the period, even though the price is a fraction of that of the Dow Jones Industrial Average. It also reinforces how closely tied the S&P 500 and DJI indexes are. Their returns are almost entirely in sync. The return is calculated as $\boldsymbol{return\ t_0} = \frac{\boldsymbol{Price_t}}{\boldsymbol{Price_0}}$

An initial analysis of the Bloomberg corpus was performed to gain some level of understanding as to what words are driving the highest variance ratio. The corpus was fit to a TF-ID Vectorizer. From that initial analysis it was determined that there is a total of 161,692 distinct headings and 36,105 words. The top 10 words were Bloomberg, new, finance, energy, news, best, china, says, worst and Europe. This initial analysis indicates that there may well be headers in the corpus that are not really news headers, but are names/links to other web pages.



Grouping the data by the heading column produced 92,081 unique headings in the corpus. The top 20 headings by count are indicative of headers that are links to other sections of the web-site.

Outliers

The header information seems to contain a significant number of items that are not true headings. Due to that, all headers that appear over 200 times within the corpus will be removed from the data-set. This will remove approximately 41 Headers accounting for 19,842 rows in the header table, or approximately 12.3%.

The resulting corpus contains 141,433 entries and 36,101 words.

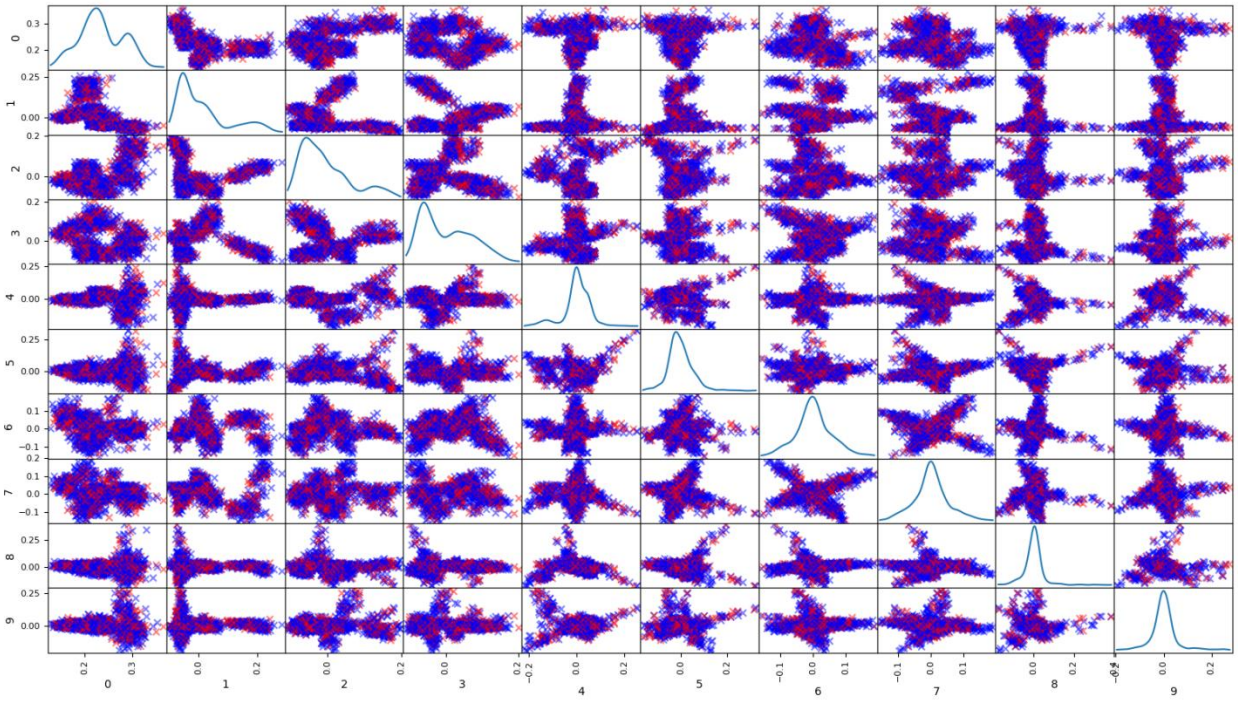
Based on this information we will remove any headers that appear within the corpus more than 200 times.

Exploratory Visualization

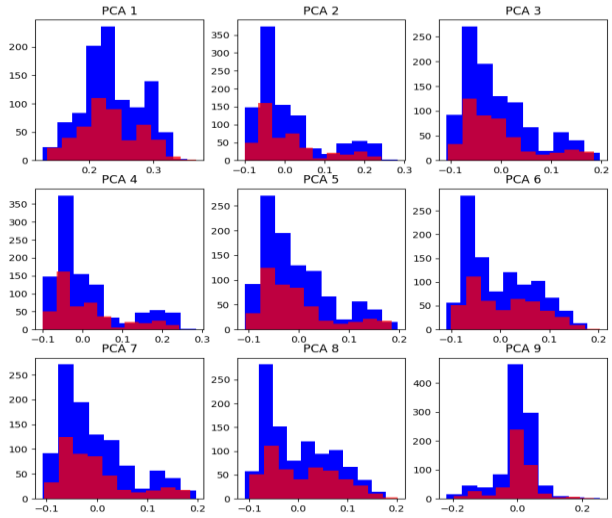
A number of visualizations were created with the goal of understanding the relationship between the features and the labels. In these visualizations we look at both the Scatterplot of the top 10 components showing both positive (1) and negative (0) relationships. We also take a look at the histograms created by the components for both LSA and Semantic to see if there is correlation between the features and their labels.

LSA

One indicator that would be important to understand is if any of the components created through the LSA analysis has a stronger likelihood of predicting the market movement for the following day. A Scatterplot of the top 10 components from the LSA analysis with the next days close showing as a specific color (blue=up, red=down) is plotted below. As you can see the red and blue seem to be evenly distributed along the same axis. This does NOT show me that any of top ten components is more predictive of the next days close than others. As a matter of fact it indicates that the components are well distributed.



Looking at it another way we take the histograms of the top 9 components. This shows us the distribution of documents that are affected by each component. We divide it into 2 histograms, one for days that where the next closing day is up (blue) or down (red). To see if any PCA's do a good job of separating out the predictions of up or down we would look for a right-left separation on the red/blue histograms. However the histograms are almost 100% reflections of each other, showing that it is not readily apparent that the components predict the next day's adj. closing price.

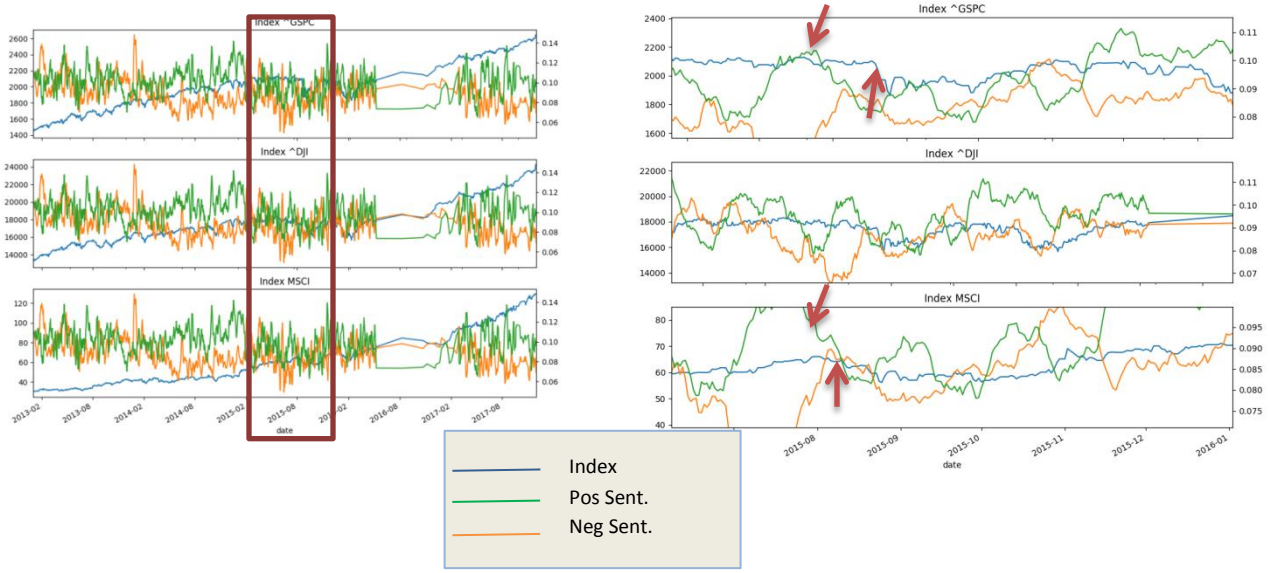


This visualization indicates that it may be difficult to find large predictions in the results based on these features.

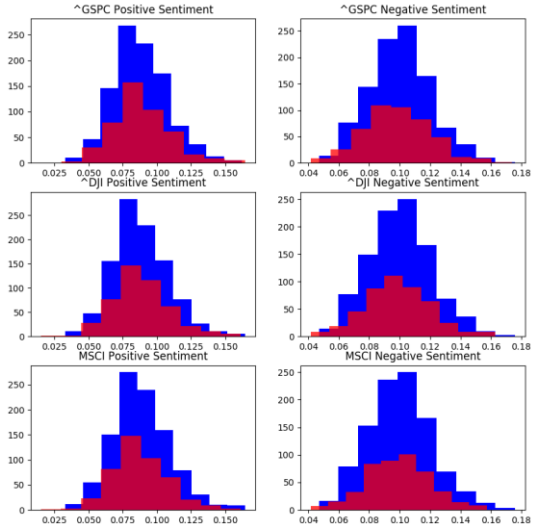
Sentiment Analysis

There are a number of ways of visualizing the effect of the sentiment analysis against the stock index Adj. Closing price. Here we show the positive and negative sentiment over time against the index daily closing value. Looking at the entire period makes it difficult to see any patterns (left image below). However, zooming in on the period in 2015 when the market dropped across the 3 indexes shows us what looks like a bit of a correlation.

There does seem to be some correlation between an increase in the negative sentiment (green line) and the market moving into a down cycle.



Finally we can look at the histograms to determine if there is a break between predictions for the next day's close. We show the histogram for each index for the positive sentiment and the negative sentiment. It is broken into days where the next days close is up (blue) or down (red). If the sentiment was highly correlated to predicting an up or down day then we would expect to see some deviation in the histograms. However, as can be seen here the histograms show almost complete alignment between the up and down days as compared to the sentiment. I would hope to see some separation indicating that sentiment correlated more to the red or blue.



Algorithms and Techniques

Benchmark

One of the challenges in picking an appropriate benchmark for this model is that our labels are imbalanced. There are many more positive (up) days than there are negative (down) days. One way of handling this imbalance is to look more closely at metrics like Precision and Recall than Accuracy.

A core thesis of this document is to try and beat a buy and hold strategy for the indexes. Therefore using a benchmark that represents the buy and hold strategy would be appropriate. This maps very nicely to the concept of a Naïve Predictor. Essentially, can we create a model that is able to perform better for any of the given indexes than always choosing the next day to be up?

The first benchmark "model" that we then create is a model that always predicts up (or 1) on our test Data. This can be found in the file *Benchmark_Model.py.* Taking the results of this prediction set we can run it through a confusion matrix and come up with a benchmark set of metrics for the Naïve Predictor. That resulted in the matrix here.

Benchmarks	MSCI	^DJI	^GSPC
Accuracy	0.685	0.685	0.681
Precision	0.685	0.685	0.681
Recall	1	1	1
Total	295	295	295
False Pos.	93	93	94
False Neg.	0	0	0
True Neg.	0	0	0
True Pos.	202	202	201
f1	0.813	0.813	0.810
f2	0.916	0.916	0.914

Now with this benchmark set of metrics the goal is to see if we can create a model that can beat it, specifically against the Precision and F1 metrics.

In the Naïve Predictor we have an average Precision of 68%, matching the accuracy score, and a Recall of 100%. This is because ALL up days (which are classified as 1) are accounted for. That said by increasing the Precision without sacrificing too much recall, we may have a model that can do better at predicting the market. Using the F1 score for this is a good way of making our measurement. That is because the F1 score is the harmonic mean between Precision and Recall. We want a model that predicts some of the down days (classified as 0) without giving up too much recall.

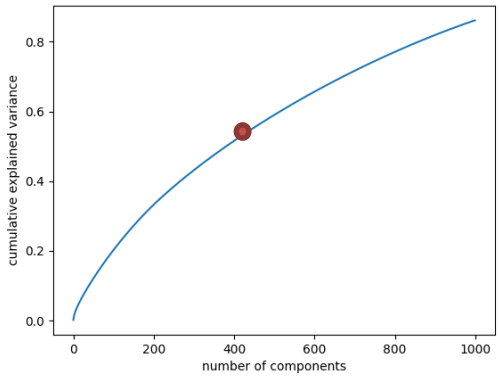
Dimensionality Reduction

Given the high level of dimensionality of the resulting data-set it is important to reduce the dimensions. This can be done in a number of ways including LSA, PCA or by creating a sentiment graph of the header information. This project will use both Latent Semantic Analysis (LSA) and Sentiment Analysis (SA) to reduce dimensions and create a feature set that can be used for training and testing. Each one will be tested independently to discover the final fit for predictability.

There is a core problem that needs to be addressed within each method of dimensionality reduction. That is the asymmetry between the number of headers per day and the index information. We have (n) number of headers per day but only 1 entry for index information (Open, Low, High, Adj. Close). Also since we are going to be training our system to predict the change between today's Adj. Close and the Adj. close of the next day we cannot use a 1 to 1 relationship between the headers and labels.

LSA Analysis

For the asymmetry problem resulting between (n) headers per day to 1 label we will aggregate the daily headers into a single document. This means that each day will contain a single document containing the aggregate set of headers for the entire day. This aggregate set will be run through *TFID* to create a sparse-matrix. *TruncatedSVD* is then run on the resulting data-set to create a more manageable feature list that can be used in the machine learning and training.



The result of aggregating data together by day and running through *TFID* creates a sparse matrix with 1,475 documents and 36,060 features. Running *TruncatedSVD* with 1000 components generates an almost linear progression of *n_components* to the explained variance. I chose 450 components as the final number to use in the LSA analysis as it describes over 54% of the variance in our data.

Sentiment Analysis

Another way of reducing the dimensionality of the data is to perform a sentiment analysis upon each header, and use the resulting sentiments in our final predictions. Sentiment analysis is a well-studied field and is often sited in documents attempting to predict stock markets. For instance in the paper *Sentiment Analysis of Financial News* (Azar, 2009) does exactly that. In that analysis he uses positive and negative word counting to create the sentiment for his corpus. Here I am going to use NLTK's Vader toolkit to rank the positive, negative and neutral values for every header. Again we have the challenge of aggregating the rankings of many headers and their scores for a given day and comparing to a single value, the classification on whether or not the Adj. Close for the next trading day is up or down. This aggregation is done by taking the mean of the values for each day. I.E. the positive value for a given day $posMean = \frac{1}{n}\sum_1^n p$ where n is the number of headers in the given day and p is the positive value from NLTK Vader for that header. So for every day we create a final data set with (X) rows each containing pos, neg, neu, compound, class. Where X is the number of days in which we have header information and class is a 1 or 0. 1 if the next day's adjusted close is up or equal, and 0 if it is down.

III. Methodology

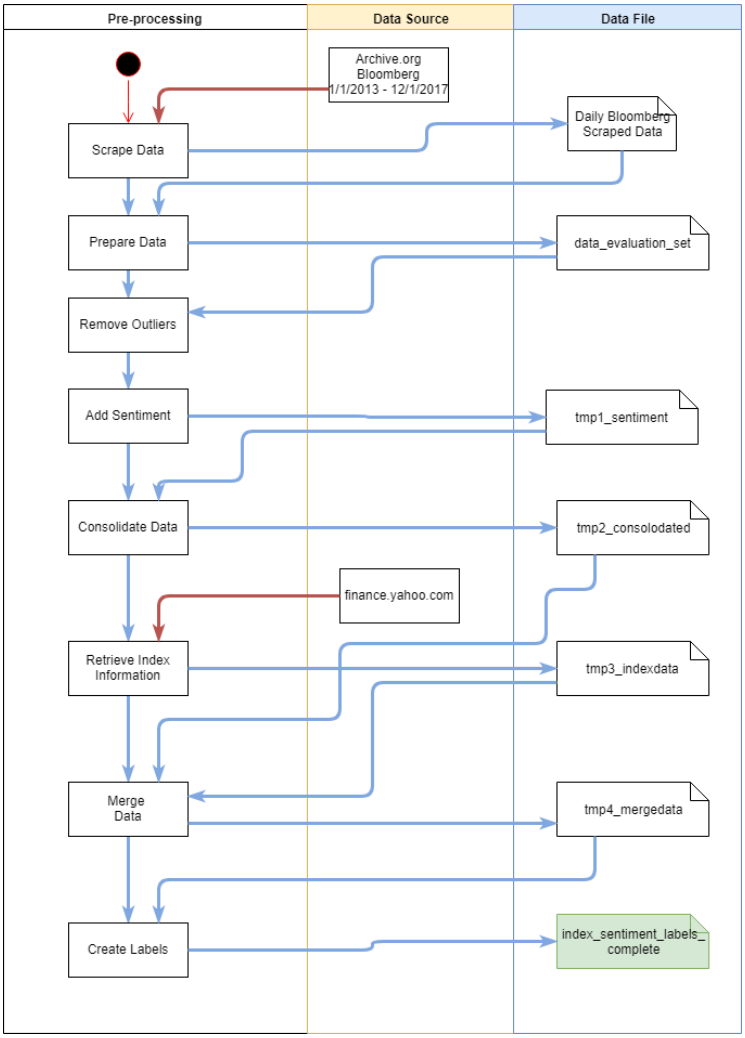
Data Preprocessing

Data preprocessing for this project involves multiple steps to create a final output data-set that can be used for the machine learning analysis. The following preprocessing steps are designed to create a master data set that can be used for both LSA and Semantic analysis. For data sources it utilizes the scraped Bloomberg data and financial information for each selected index from Yahoo.

Python Class – Master_Data.py (function createData)

Step 1 Scrape Data – Run the following search query against Archive.org.
web.archive.org/cdx/search/cdx
?url=www.bloomberg.com
&fl=timestamp,original,statuscode
&from=2013&to=2017
&filter=!statuscode:301
&filter=!statuscode:307
&filter=!statuscode:302

Take the resulting list and aggregate to 1 per day and pull those pages into the folder ./data/scrape_data.



Step 2 Prep Raw Data – Look in the directory where all of the scraped data is located. In that folder is 1 file per day where each file contains two columns, [cdx] [data]. Where cdx is the date-timestamp from Archive.org for the specific scraped page and data contains the header. The first step is to read in every file and consolidate into a single raw-data file. Additionally the cdx data has date and time, we don't care about the time and want to turn the cdx format into a standard python date function.

Output is a file with 2 columns, [date][data] where date is in the format mm/dd/yyyy and data is the header. File contains is 161,692 rows.

File ./tmp/data_evaluation_set.csv

Step 3 Remove Outliers – During the data evaluation stage we identified that many headers appeared within the data set many times, some as often as 500+. All headers within the data-set that appear more than 200 times are removed from the data. This results in a new data-set that is 141,433 rows long. This is a reduction of 20,259 rows.

Output -data set with 2 columns, [date][data] where date is in the format mm/dd/yyyy and data is the header information. Data-set contains 141,433 rows.

Step 4 Add Sentiment Information – For each row in the resulting data set we calculate the sentiment features. Sentiment is created by having each header sent to NLTK's Vader sentiment analysis. This results in 4 numbers between 0 and 1 representing the % that the toolkit believes a given sentiment is present within the header. The final output is [neg][neu][pos][compound] for each row. This information is stored within the data with each header. Compound is an aggregate of the other 3.

Output – data set with 6 columns [date][data][neg][neu][pos][compound] where data is in the form mm/dd/yyyy, data is the header and the sentiment for each value is present in neg, neu, pos, compound. Results in a file with 141,433 rows.

File – ./tmp/tmp1_sentiment.csv

Step 5 Consolidate Data – At this point we have a data-set that still represents 1 row per header. We need to turn this into 1 row per day, so that we can add in market index information and labels. Step 4 consolidates the data into 1 row per day. For each row 2 transformations happen. First the new [data] column contains the concatenated headers of all the headers for that day. So if there were 100 headers the new data cell will contain all the headers concatenated together as a single string.

Additionally the sentiment is also transformed. For each sentiment for the given day the new sentiment is the mean of all the previous sentiments. . I.E. the positive value for a given day is $posMean = \frac{1}{n}\sum_1^n p$ where n is the number of headers in the given day and p is the positive value from NLTK Vader for that header. This is done independently for each value, so neg, nue, pos, compound each result in a new number that is the mean of their values for that day.

Output – data set with 6 columns. [date] [data] [neg] [neu] [pos] [compound] where data is in the format mm/dd/yyyy, data is the concatenated headers, and neg/neu/pos/compound are the means of the daily sentiment. There are 1,476 rows in the resulting file.

File - ./tmp/tmp2_consolidated.csv

Step 6 Retrieve index information – In order to create our labels we are going to need the adjusted closing values for every day for each index. In this step we loop through each index in our list and make a call to yahoo finance. We pull down the daily trading information for each index. Note that a real issue is that the header information and trading data don't match up 100%. The yahoo information contains only data for each trading day. This means that there is no information for weekends or holidays. In order to make sure that our header data and index information can match up this gap in the data needs to be fixed. For this project the index information is *resampled* for each day and forward filled. This means that every day in the range will be present, and for a day where there is no data the most recent days' information is filled in.

Ex. There is a data gap between 1/4/2013 and 1/7/2013 (1/5 and 1/6 are not present) the resulting data would look like the following, where 1/5 and 1/6 are resampled and forward filled.

Date	^GSPC_Open	^GSPC_High	^GSPC_Low	^GSPC_Close	^GSPC_Adj Close	^GSPC_Volume
1/4/2013	1459.369995	1467.939941	1458.98999	1466.469971	1466.469971	3424290000
1/5/2013	1459.369995	1467.939941	1458.98999	1466.469971	1466.469971	3424290000

1/6/2013	1459.369995	1467.939941	1458.98999	1466.469971	1466.469971	3424290000
1/7/2013	1466.469971	1466.469971	1456.619995	1461.890015	1461.890015	3304970000

Output : [date] [index_info 1]… [index_info (n)] where each index_info looks like [index_Open][index_High][index_Low][index_Close][index_Adj Close][index_Volume]

File: ./tmp/tmp3_indexdata.csv

Step 7 Merge Index Information – Now we need to merge the index information with the previous set of headers and sentiment information. This will give us a full consolidated data-set where every day contains the date, headers, sentiment and index data for each index. This merge is done by day. Note that this is a join of the two data-sets and with the date field from the consolidated data from step 4 being used. This results in a data-set with no *NaN* values.

Output: [date] [index_info 1]… [index_info (n)] [neg][neu][pos][compound] where index_info = [index_Open][index_High][index_Low][index_Close][index_Adj Close][index_Volume]

Step 8 Create the labels – Finally we need to create the labels that will be used for training and test. In this step two labels are created, y1 and y2. These labels are the a binary representation on whether or not the Adj. Close for the given index was up or down, y1 represents the previous day vs. today close and y2 is today vs. the next day. So for a given index the pseudo-code would be:

For each row in data:
    if(index adj. close for day is greater than day-1 adj. close):
        index_y1=1 else 0
    if(index adj. close for day+1 is greater than day):
        index_y2=1 else 0

The result is y1 and y2 labels for each index within the set. Note that the y1 labels are in the data-set but not used in the final analysis. The final analysis only looks that the y2 values (tomorrows close vs. today). The final data-set also has its last row removed as the y2 values would not be present.

Output
[date] [index_info 1]… [index_info (n)] [neg][neu][pos][compound][labels] where [index_info]= [index_Open][index_High][index_Low][index_Close][index_Adj Close][index_Volume] and [labels] = [index1_y1][index2_y2]…[index(n)_y1][index(n)_y2]

Final data set contains 1,475 rows.

File: ./data/index_sentiment_labels_complete.csv

Implementation

The goal of the implementation is to create a process that can determine whether or not we can use the Bloomberg header information to predict the next day's close for different indexes. This will be done by using the data prepared in the data pre-processing section and stored in the data-set *index_sentiment_labels_complete.csv*. As a reminder this data set contains 4 different pieces of information for each day in the time range between 1/1/2013 and 12/1/2017. These are:

Index – Date in the form of mm/dd/yyyy (this is the DataFrame index, not stock index)

Data block – under the column [data], this is a large string field containing all of the Bloomberg headers for the front page for that day.

Sentiment Block – The prepared mean sentiment for all of the headers in the data-block for that day. Contains a numeric value between 0 and 1 for negative (neg), neutral (neu), positive (pos), and compound (compound).

Index Block – contains the Open, Low, High, Close, Adj. Close and Volume information for each of the three indexes S&P 500 (^GSPC), Dow Jones Industrial Average (^DJI) and MSCI World Index (MSCI).

Labels Block – For each index in the Index Block a y1 and y2 value. The value is an integer classification label of either 0 or 1 where 0 represents that the market went down and a 1 represents that the market stayed the same or went up. Y1 represents the current day vs. previous and y2 represents the next day vs. current.

Using the information specified in the data set we have the goal of determining if the y2 value can be predicted. We are going to run two different sets of data through the models for this prediction. First is a Latent Semantic Analysis (LSA) on the text corpus fed into both an SVC and RandomForest models. The second is Semantic Analysis on the text corpus also fed into the SVC and RandomForest models. The output is a set of metrics, including the Precision and F1 metric, for each model.

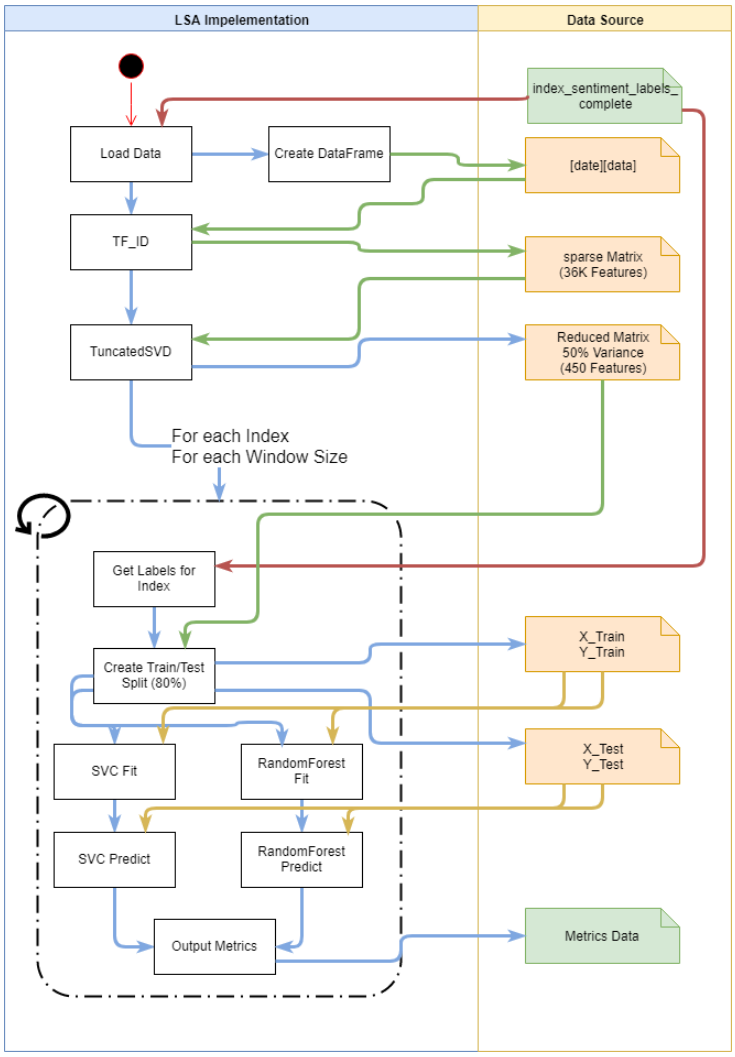
Because we are looking to predict information over time we are also going to need to setup and format our data as a Time-Series. For our tests we are going to look at three different windows, 1 day, 5 days and 20 days. So at the end of our tests we should have metrics showing if we can predict the y2 label for the following:

The end result will be 12 sets of metrics for each index (36 total); each metric set showing us the Accuracy, Precision, Recall, f1, f2 scores as well as the True Positives, False Positives, True Negatives, and False Negatives for each run. We will then look at the results to pick a best predictor model to perform our final evaluation upon.

Data Set	Concatenated Headers[data] + y2 Label		Semantic Data + y2 Label	
Dim. Reduction	LSA		Semantic	
Model	SVC	Random Forest	SVC	Random Forest
Time Series	1,5,20	1,5,20	1,5,20	1,5,20

LSA Implementation

Our LSA implementation first loads the field ***data*** from our master data set. That information is the same for all rows and so it is vectorized using TfidVectorizer. The result is a sparse matrix with approximately 36,000 features. The second step is to then reduce those features to something more manageable. This is performed using TruncatedSVD with n_components set to 450.



The number 450 was determined during the data evaluation stage as enabling over 54% of the variance in the original data set to be present.

Now we need to run this data through both SVC and RandomForest for each of the different indexes and each different time-series window.

Our loop first pulls out the list of indexes that are contained within the data-set. For each index in the data we perform a loop.

Our second loop then runs for each window size/sequence length we want to test for. In our code that is 1, 5 and 20. These are the number of days to look back when preparing each row of data.

Within our loops we run the training and testing data, then we run it through both SVC and RandomForest to create our output metrics.

Time-Series Training and Testing Data

Our system uses time-series analysis to make sure that we are not just trying to train on predicting the next day from just 1 days headlines, but training on a window of headlines going back in time a number of days equal to the window size. We use 3 different window sizes in this analysis (1, 5, 20). This was chosen so that we can see if we get better performance on an increasing amount of data.

Our data-set is created first by determining the window size. Then we move ahead in our data by that window size and that is the beginning of our train/test data set. One way to look at this is to say that each new training row is now a block containing the features for the current day and each day going back in time the length of the window size. Please reference Dr. Brownlee's work for more information on how to construct this type of Time Series network. (Brownlee, Time Series Prediction LSTM Recurrent Neural Network with Keras, 2016)

The returned feature data is a 3d *numpy* array for both training and testing which takes the shape of [(n) rows][window size][feature length].

After pulling out this data it is then flattened into a single 2D array of shape [(n) rows][window size*feature length]. So for example a 5 day window would result in n rows with 450*5 (2,250) features.

Ex. if our initial data had 1 feature per row with 24 rows and looked like:

x1 label Y1
x2 label Y2
…
x24 with label Y24

then our resulting data would look like:

row1: x1, x2, x3, x4 label Y4
row2: x2, x3, x4, x5 label Y5
……
row21: x21, x22, x23, x24 label Y24

This same Time-series data creation is used for all models, the only difference is the size of the dimensions for the input rows depending on if it is LSA or Semantic.

Note that the label for each row is also returned and setup so that the label for a given row is the prediction of the next days adjusted close compared to the current day. So in the above example the y5 value is a 1 if the adjusted close for day 6 is above or equal to day 5 and 0 if it isn't.

Fit and Test Data

Once our data has been turned into time series information and flattened into a 2D numpy array it can then be fit into our different models to train them and then used to make predictions on the test data.

In order to optimize the predictions relative to each data set GridSearchCV is used to find the optimal parameters for both the SVC and RandomForest algorithms. The scoring metric used is F1 for all models. The resulting classifier information is saved out so that we can look at it later and see what metrics (C and Gamma for SVC, n_estimators and min_samples_leaf for RandomForest) were chosen.

We also set the class_weight to *balanced* as this helps handle the imbalanced nature of our resulting data.

Output Metrics

After each run is completed a set of output metrics is written. A file is created for each Model and each feature reduction method. So LSA_SVC_metrics, LSA_RandForest_metrics, SEM_SVC_metrics and SEM_RandForest_metrics.

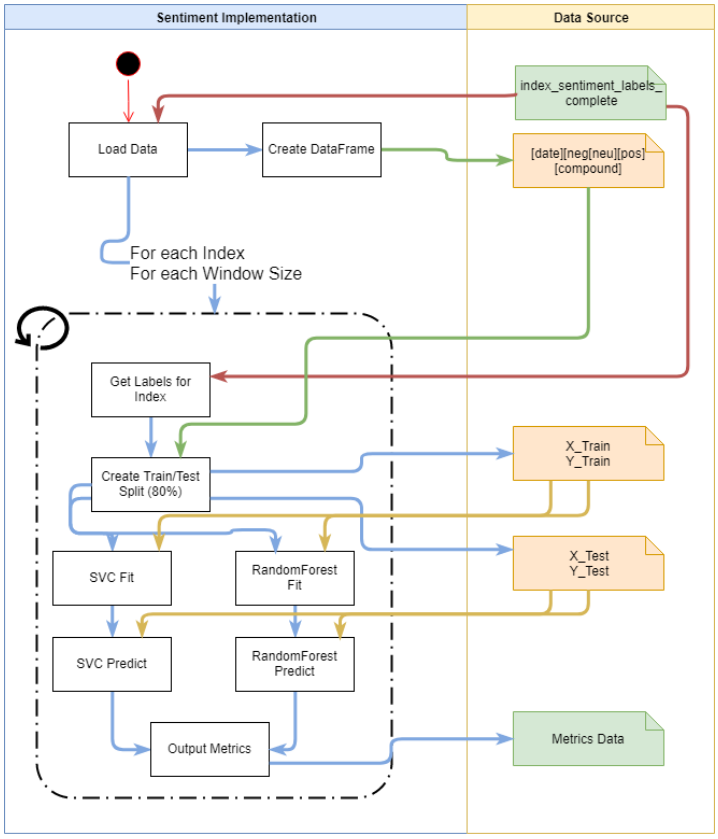
The following table is the raw output file from a run of the model. This one in particular is for LSA with SVC and shows the results for all indexes and window lengths.

	MSCI seq_len 1	**MSCI seq_len 20**	**MSCI seq_len 5**	**^DJI seq_len 1**	**^DJI seq_len 20**	**^DJI seq_len 5**	**^GSPC seq_len 1**	**^GSPC seq_len 20**	**^GSPC seq_len 5**
Accuracy	0.315254	0.690722	0.687075	0.508475	0.676976	0.687075	0.498305	0.666667	0.683673
False Neg	202	0	0	94	15	0	98	14	0
False Pos	0	90	92	51	79	92	50	83	93
Precision		0.690722	0.687075	0.679245	0.700758	0.687075	0.673203	0.690299	0.683673
Recall	0	1	1	0.534653	0.925	1	0.512438	0.929648	1
Total Predictions	295	291	294	295	291	294	295	291	294
True Neg	93	0	0	42	12	0	44	9	0
True Pos	0	201	202	108	185	202	103	185	201
f1	0	0.817073	0.814516	0.598338	0.797414	0.814516	0.581921	0.792291	0.812121
f2		0.917808	0.916515	0.558428	0.869361	0.916515	0.53814	0.869361	0.915301

This information is aggregated together across the different models in the file **metrics_summary.xlsx** and in the conclusion section below.

Semantic Implementation

Our Semantic Implementation is very similar to LSA, however it does not need to perform the dimensionality reduction through *TFID* and *TruncatedSVD* as the reduce feature set (semantic information) is already present within the master data set.



Within the Semantic section we run the same set of core steps. First the labels and Semantic information is retrieved from the *MasterData* set for each index.

Then training and testing data is created for each window size.

That information is then fed into the *SVC* and *RandomForest* classifiers using *GridSearchCV* and the scoring metric of F1.

The output for each run is then sent to the metrics file for later collation.

Other than the features set size (4 features vs. 450 in LSA) everything else was kept identical to the LSA analysis so that final review of the metrics would be an apples to apples comparison.

Again other than using NLTK Vader for creating the semantic information instead of LSA, everything is the same in the model including how the time-series data is created, the size of the train/test split, etc..

Refinement

During the project there were three distinct phases of refinement. The initial model was a sentiment only system using a basic *LSTM* Neural network. The intermediate model added in *SVC* and *RandomForest* and tested out precision as a metric. The final model refined the metric to F1, dropped *LSTM* adds in LSA and worked on balancing the data and used *GridSearchSVC*.

Initial Model - The first model built was a simple system that looked at the sentiment information coming from pythons *TextBlob* and running it through an *LSTM* Neural Network. Much of the complexity here was in creating the code that could build up time-series windows in a 3d shaped numpy array as required by the *LSTM* network. The core problem encountered here was the predictions. The network never predicted anything by 1's (up) for the next day. Additionally the *TextBlob* sentiment analysis only returned a single number rating the sentiment on a scale from -1 to 1. In this model I converted it to 3 numbers (neg, neu, pos) based on if the value was in the ranges of [-1 to -.025][-.0249 to .0249][.025 to 1].

- Issue 1 – I had concern that creating the sentiment in this way was a significant over-simplification and not really retaining any information for the system
- Issue 2 - Research indicated that the imbalance in the data (up days vs. down days) made the accuracy metric a suspect way of scoring the system
- Issue 3 – Deeper looks at LSTM and Neural Nets with imbalanced data made me concerned that it was going to be potentially very difficult to build a NN that managed the imbalanced data well.

Intermediate Model - Based on the issues encountered I decided to add in *SVC* and *RandomForest* classifiers and see if different results were obtained. Also as part of this change I added in *GridSearchCV* and a scoring metric of "precision". This enabled me to start to build out models that started to predict values other than just 1, "up". Finally I changed out the *TextBlob* mechanism for sentiment analysis for *NLTK* and *Vader*. The reason is that the *Vader* toolkit returns 4 different numbers, classifying how much neg/neu/pos and compound sentiment is in a given text string. I had success here in that the system started to predict different values for precision and recall. Essentially the recall was not always 1.0.

- Issue 1 – Still having problems with the model returning low precision vs. the benchmark.
- Issue 2 - Became concerned that sentiment analysis may be removing so much information from the system that it could not make meaningful predictions.
- Issue 3 – The imbalanced nature of the data was still having a real effect on the predictions. I started to research methods of handling imbalanced data with the different algorithms. It was at this time that I came across the *class_weight* parameters for the models and started playing around with different values.

Final Model – During the last phase I decided to add in a full LSA analysis in addition to sentiment. This was due to my concerns that sentiment analysis was removing huge amounts of information from the system, and may not be highly predictive. Research showed that LSA could potentially be a good way of retaining much of the information within the headers, and use the reduced dimensions for predictions. I also dropped the *LSTM* Neural Network as it never seemed to do much better than *SVC/RandomForest* and was harder to manage to the metrics.

Another change I made was to go to the F1 metric vs. precision. I felt that attempting to balance between Precision and Recall was the right goal for the system. This final model was then refined to be flexible enough to take different window sizes, switch easily between *SVC* and *RandomForest* and then run on both LSA and Sentiment.

In summary the core issues encountered during the process of building the models were:

1. **Imbalanced Data –** The data set is imbalanced with many more days classifying as a 1 (up) vs 0 (down). To handle this I played with the *class_weight* parameter for both models and found that setting it to *balanced* seemed to make the most difference.
2. **Not using proper metric** – Originally the accuracy metric was being used within the model. Trying to optimize on accuracy caused the models (*LSTM, SVC* and *RandomForest*) to create predictors that always predicted a 1 (up). Since we want to try to optimize both the Precision and Recall metrics the F1 metric was finally chosen as the scoring parameter for *GridSearchCV*.
3. **Sentiment vs. LSA** – I became concerned that the Sentiment analysis was reducing the data from our system to such an extent that it would not be able to predict. LSA was added to help solve potentially for this problem.
4. **LSTM Neural Network** – Originally this project was going to use an LSTM Neural Network for our predictions. However, after testing on the LSA data-set I discarded the idea and went with *SVC* and *RandomForest* instead. The reason is that the Neural Network was much lower on the F1 score, and seemed much more sensitive to the imbalanced nature of the data and the binary classification. While I was able to get the network to make predictions, it seemed to be much more sensitive to the setup and input variables. I therefore was not confident that the model was making predictions that were any better than the more simple Machine Learning algorithms.

IV. Results

Model Evaluation and Validation

Following is a summary of the final metrics output by the models. I am looking for four items within the metrics data set. The items we are looking for are which Index fund can be most predicted, which dimensionality reduction technique works best, which classification does the best job and which window/sequence length gives us the best data. To do this we compare the precision and F1 numbers vs. the benchmark.

			MSCI											
		Bench	LSA RandomForest			LSA SVC			Semantic RandomForest			Semantic SVC		
	Seq Len		1	5	20	1	5	20	1	5	20	1	5	20
MSCI	Accuracy	**0.685**	**0.671**	**0.684**	**0.691**	0.315	0.687	0.691	0.475	0.568	0.522	0.454	0.439	0.526
	Precision	**0.685**	**0.680**	**0.690**	**0.692**		0.687	0.691	0.644	0.685	0.694	0.650	0.680	0.723
	Prec. vs. Bench		-0.004	0.005	0.007	-0.685	0.002	0.006	-0.041	0.000	0.009	-0.035	-0.005	0.039
	Recall	1.000	0.980	0.980	0.995	0.000	1.000	1.000	0.520	0.688	0.552	0.441	0.347	0.507
	f1	0.813	0.803	0.810	0.816	0.000	0.815	0.817	0.575	0.686	0.615	0.525	0.459	0.596
	f1 vs. Bench		-0.010	-0.003	0.003	-0.813	0.002	0.004	-0.238	-0.126	-0.198	-0.288	-0.354	-0.216
^DJIA	Accuracy	0.685	0.681	0.690	0.687	0.508	0.687	0.677	0.519	0.486	0.581	0.492	0.520	0.491
	Precision	0.685	0.686	0.689	**0.703**	0.679	0.687	0.701	0.685	0.658	0.707	0.700	0.702	0.694
	Prec. vs. Bench		0.001	0.005	**0.018**	-0.006	0.002	0.016	0.000	-0.026	0.023	0.015	0.017	0.009
	Recall	1.000	0.985	1.000	0.945	0.535	1.000	0.925	0.550	0.525	0.665	0.450	0.525	0.465
	f1	0.813	0.809	0.816	**0.806**	0.598	0.815	0.797	0.610	0.584	0.686	0.548	0.601	0.557
	f1 vs. Bench		-0.004	0.003	**-0.007**	-0.215	0.002	-0.015	-0.203	-0.229	-0.127	-0.265	-0.212	-0.256
^GSPC	Accuracy	0.681	0.688	0.684	0.687	0.498	0.684	0.667	0.556	0.500	0.543	0.488	0.684	0.526
	Precision	0.681	0.686	0.685	0.688	0.673	0.684	0.690	0.680	0.636	0.696	0.676	0.684	0.707
	Prec. vs. Bench		0.005	0.004	0.006	-0.008	0.002	0.009	-0.001	-0.045	0.015	-0.005	0.002	0.026
	Recall	1.000	1.000	0.995	0.995	0.512	1.000	0.930	0.657	0.627	0.588	0.478	1.000	0.523
	f1	0.810	0.814	0.811	0.813	0.582	0.812	0.792	0.668	0.632	0.638	0.560	0.812	0.601
	f1 vs. Bench		0.003	0.001	0.003	-0.229	0.002	-0.018	-0.142	-0.179	-0.173	-0.251	0.002	-0.209

As can be seen the F1 number never gets much better than about .003 to .004 (.3% to .4%) than the benchmark F1 number. This is essentially insignificant. Precision does a bit better with a number of the models pushing above 1.5% of the benchmark. A number of models even get above 2%, specifically MSCI/Semantic/SVC/20 and ^GSPC/Semantic/SVC/20. Of course these models do so with a significant drop in recall and accuracy. Recall moving close to 50%.

The results here are very much in-line with the expectations. If any models had proven to be highly predictive of the index values that would have been a red-flag. As a matter of fact it is somewhat amazing how close to the benchmark model the different algorithms and data inputs come. Additionally having a larger time-series window does seem to enable the algorithms to come in closer to the benchmark, and even exceed it by a very small amount.

One concern though, is that the test data set is only 20% of the full data. When taking into account the window size our test set may be as small as 291 elements. This means that the change in just 1 data point can affect the results by almost 1/3 of a %. What this effectively means is that a positive .003 number may mean that the model only selected a single day as down vs. the benchmark.

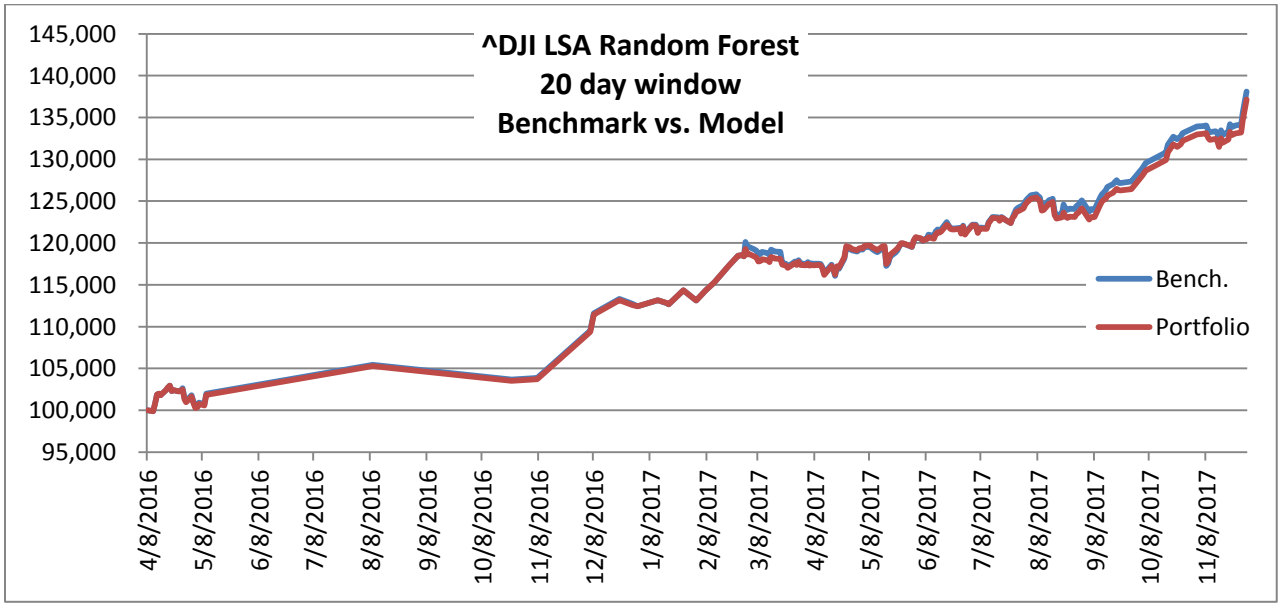
One way to solve this issue may be to train the model on a larger data-set (going back past 1/1/2013) and enabling for more testing data, however that is not within the scope of this project.

V. Conclusion

In order to do a final determination on whether or not the model would out-perform a buy and hold strategy one of the best performing models was chosen and a sample portfolio created. The model for **Dow Jones Industrial Average, LSA Dimension Reduction, Random Forest with a 20 day window** was chosen. The reason is that when looking at the results vs. the benchmark (see Results table above) it had a high precision score, good F1 score and did show a number of True Negatives, which would indicate a sell day. The scores for the chosen model were 0.703 precision and 0.806 f1 (vs. 0.685 precision and 0.813 f1).

To compare this model vs. a buy and hold strategy two portfolios were created on the test data set. Both models start with $100,000 in cash. On the buy and hold (Benchmark) the total dollar amount is invested in the index on the first day and held until the last. In the Portfolio model the total amount is invested on the first day. However, on any day that the model indicated that the NEXT DAY was going to be a down day 50% of the portfolio was then sold off and the proceeds kept as cash. On any day that there was an indication that the next day would be up any available cash was used to purchase shares.

The chart below shows the relative value of each portfolio over the time period.



As can be seen the buy and hold strategy outperformed the portfolio strategy by approximately $967 or about 2.6%. Note that this model does not take into account dividends, trading costs or any other fees. This means that this is the most optimistic model and any real world examples with those expenses included would only decrease the ending value of the Portfolio model.

	Buy and Hold	Portfolio
Initial Investment	100,000	100,000
Initial Shares	5.689	5.689
Ending Shares	5.689	5.65
Ending Portfolio Value	138,092	137,125
Net Gain/Loss	38,092	37,125
Difference		(967)

Final Conclusion

My final conclusion is there is not enough information or signals within financial news headings to enable a machine learning model to predict, with any accuracy, what the index funds next day's adjusted closing price will be. I can also conclude that it would be very difficult for a portfolio manager using the same data sources to do the same. Buying and holding an index fund over time will outperform a model trying to buy and sell based on signals coming from news headlines.

Note this doesn't mean that active investors may not have other data sources or deeper information about specific companies that would enable them to out-perform the market. However, using just news headlines would not be enough to make any meaningful predictions.

Reflection

My initial project idea was to take readily available article headings, run them through sentiment analysis, create a data-set for a Neural network and see how well it would do predicting the next day's stock market close. My assumption was that it would be a pretty straight-forward application of some readily available data and a simple Nerual Network. What I found out was that this project was significantly more complex than I expected.

First, I realized that there was NOT a good data-set to be used. I looked at Kaggle, went through 10-20 different data sites, and could not find one that met 2 criteria. That it contained article headings for financial news and that it covered at least a 5 year period of time ending close to the current date. This meant I had to create my own data set. That pulled me down a path of trying to locate the data on the web (which finally led me to Archive.org) and a way of creating the initial data-set (which let me to *Scrapy* and a web-site scraper).

The next difficulty I ran into was matching up the Bloomberg information against the stock data. It was pretty trivial converting the headers into sentiment using Textblob. However, there were multiple headers per day and only one stock value. I realized I had to mathematically combine this information so that it could be matched to the indexes. This also made me realize that even with 5 years worth of data, the actual number of trading days available were not very high (under 1,500).

Another challenge that was encountered at this point was the time-series nature of the data. When researching the best way to create a Neural Network for stock data I encountered *LSTM's*. However, this research made me realize that I could not use the standard data-preparation models we learned during the classes, as the data was time-sensitive. Much work was done to understand how to create window-based time series and format the data properly for an *LSTM*. I covered above why I eventually dropped *LSTM's* in favor of *SVC* and *RandomForest*, however the time-series nature of the data remained and had to be dealt with.

Along with the problems of finding the right model and creating time-series data came the realization that most of my time was being spent creating, sourcing, manipulating and fixing the data set. This is when I took a step back, looked at what I wanted the end data-set to look like and created the MasterData object. What I realized is that I was duplicating the same steps in many of the different tests and models I was performing. For instance creating the data, pulling down index information, building a test_set that matched to it, etc... Some of the steps took many minutes to perform (for instance running sentiment analysis on 161K rows of data) and I was doing it every time I ran models and tests. Creating a single MasterData object that performed many of the intensive steps once, storing the results for later loading was a breakthrough that sped up my model testing process immensely.

After solving the data manipulation issues I moved on to really trying out and testing the *SVC* and *RandomForest* models. The next issue encountered was an inability to get it to predict anything but 1's. My initial assumption was that the accuracy metric was the appropriate one as it would indicate if the model could do better than the benchmark. However, once the models kept returning 1 (up) for every day I began to try and figure out what is going on. Some of the reviewer notes on my proposal pointed me toward the imbalanced data set. In researching that I realized a couple of things. First, that I had to focus on a different set of metrics than accuracy. Second, that what I was really looking for was a model that could predict True Negatives while keeping as much recall as possible. This is where the real tweaking of the models started to take place and I focused in on the Precision and F1 metrics. It is also where I encountered the *class_weight* parameters and started to test them out.

Finally it was time to create the output. By this time most of the hard work was done, and creating the necessary output files was pretty fast. If I were to do this project different the two key aspects I would focus on to save time/energy is

1. Making sure to come up with what I want my final form data set to look like and then focus on building up to that FIRST! Not recreating the wheel many times and running the same code every time I ran a model.
2. Working more closely on the benchmark model and initial metrics. If I had done this earlier I would probably have gotten to my final solution much quick.

Improvements

There are three key improvements I would make to this project with more time. First I would not just scrape the header information from Bloomberg but also pull in the actual articles themselves. Second I would test against a much broader set of indexes, ETF's and mutual funds. Third and finally I would create a custom system for determining the positive or negative semantic meaning.

With enough time it would make sense to try and use a much larger set of data to try and get the meaning of a full article. The current project only uses the article header. However, pulling the full text of each article would be much more appropriate. The theory is that it would be much easier using the full text of an article, vs. just the header, to extract the general tone and meaning. This would expand the amount of information available to the algorithms 100 fold, and give it a much better chance on predicting the next day's market movement.

Testing against a larger set of Indexes and even ETF's and mutual funds would be extremely interesting. There may be certain indexes, stocks or funds that are much more predictable than the very general S&P500, Dow Jones and MSCI World funds. For instance do financial news stories do a better job of predicting the movement of Energy funds, or funds tied to specific countries - say BRIC. One could even run a different model whose goal is to find out which funds, indexes and ETF's are the most predictable.

Lastly I would like to create a custom system that determines the semantics of a financial article. The reason is that most of the systems I reviewed used very different data-sets to create their sentiment analysis. For instance

TextBlob used movie reviews. I think that a sentiment analysis API custom built from financial news would do a much better job of assigning the Positive, Negative and Neutral weights. Financial terms indicating positive or negative sentiment may not be the same as for a movie review. For instance the statement "The British Pound moved sharply lower today compared to the US Dollar". Would be potentially positive for British stocks and negative for US, but the opposite when looking at currency values.

VI. References

Aaron7Sun. (2016). *Kaggle*. Retrieved from https://www.kaggle.com/aaron7sun/stocknews/data

Azar, P. D. (2009). *Sentiment Analysis in Financial News.* Retrieved from http://people.csail.mit.edu/azar/wp-content/uploads/2011/09/thesis.pdf

Braham, L. (2017, 04 17). *CNBC - A Seismic Shift is happening and billions are pouring into these funds*. Retrieved from https://www.cnbc.com/2017/04/17/a-seismic-shift-is-happening-and-billions-are-pouring-into-these-funds.html

Brownlee, D. J. (2016, 7 21). *Time Series Prediction LSTM Recurrent Neural Network with Keras*. Retrieved from https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Brownlee, D. J. (n.d.). *Time Series Prediction LSTM Recurrent Neural Network with Keras*. Retrieved from https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Foundation, P. S. (n.d.). *textblob*. Retrieved from https://pypi.python.org/pypi/textblob

Landaur. (1998). *An Introduction to Latent Semantic Analysis.* Retrieved from http://lsa.colorado.edu/papers/dp1.LSAintro.pdf

Miller, C. (2016, 09 19). *An Introduction to Stock Market Data Analysis with Python*. Retrieved 2017, from https://ntguardian.wordpress.com/2016/09/19/introduction-stock-market-data-python-1/

NLTK. (n.d.). *nltk.sentiment.vader module*. Retrieved from http://www.nltk.org/api/nltk.sentiment.html#module-nltk.sentiment.vader

Ortega, F. M. (n.d.). *UO UA: Using Latent Semantic Analysis to Build a Domain-Dependent Sentiment Resource.* Retrieved from https://aclanthology.info/pdf/S/S14/S14-2137.pdf

Petruna, T. (2017, 04 09). Retrieved from LA Times: http://www.latimes.com/business/la-fi-investing-quarterly-index-funds-20170409-story.html

scikit-lear. (n.d.). *RandomForestClassifier*. Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

scikit-learn. (n.d.). *SVC*. Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

scikit-learn. (n.d.). *TruncatedSVD*. Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

Scrapy.org. (n.d.). *Scrapy*. Retrieved from https://scrapy.org/

TFID, s.-l. . (n.d.). *TfidVectorizer*. Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Wikipedia. (n.d.). *LSA*. Retrieved 2017, from https://en.wikipedia.org/wiki/Latent_semantic_analysis