# Market Price Sensitivity to News

Jason Sawyer

March 16, 2020

**Abstract**

In the age of information, we are flooded with news articles every day that can be accessed virtually anywhere and at anytime given the proliferation of smartphones, laptops, and traditional media. Investors pay close attention to the news. When recent developments impact their economic forecasts, they make trading decisions that directly impact stock prices. Therefore, to some extent stock prices can be thought of as a function of the news. Is it possible, then, to forgoe fundamental analysis of the economy and instead directly model this relationship to predict market movements? In this paper, we will consider this question from a mathematical perspective. We will leverage techniques for natural language processing, dimension reduction, and machine learning classification to attempt to describe this relationship.

https://github.com/sawy0056/AMATH582/tree/master/FinalProject

## 1 Introduction and Overview

Theoretically, the price of a stock represents the true value of a company driven by its fundamental financial health. Market participants constantly update their assumptions and reevaluate their positions. For example, suppose that a major news outlet reports that holiday sales are significantly lower than in previous years. Stock traders would likely react by lowering their expected earnings forecasts for major retailers which would in turn drive their stock prices lower.

As another example, consider the uncertainty surrounding the novel coronavirus. As news coverage pours out on a daily basis, market participants are unsure how to forecast the full economic impact. Surely, there will be disruption to international supply chains and likely a domestic slowdown if more extreme quarantine measures become necessary in the U.S. The anxiety surrounding this has manifested in drastic market corrections and significant volatility.

These are two examples of how analysts leverage news to recalculate their outlook on fundamental valuations of companies. In addition to analysts, of course there is a growing portion of trade volume being driven by algorithms. Many of these algorithms trade on momentum and, at times, serve to amplify market swings.

All of the above begs the question; it is possible to construct a model that predicts the market's reaction to the latest news? In this paper, we will consider this question with the following approach. First, we will construct a web crawler to gather raw text from news articles over a historical window. Second, we will analyze this text data by leveraging concepts from natural language processing. Finally, we will use the singular value decomposition coupled with machine learning techniques in an effort to predict market movements.

## 2 Theoretical Background

The two primary tools employed in this research are the singular value decomposition (SVD) and the Linear Discriminant Analysis (LDA).

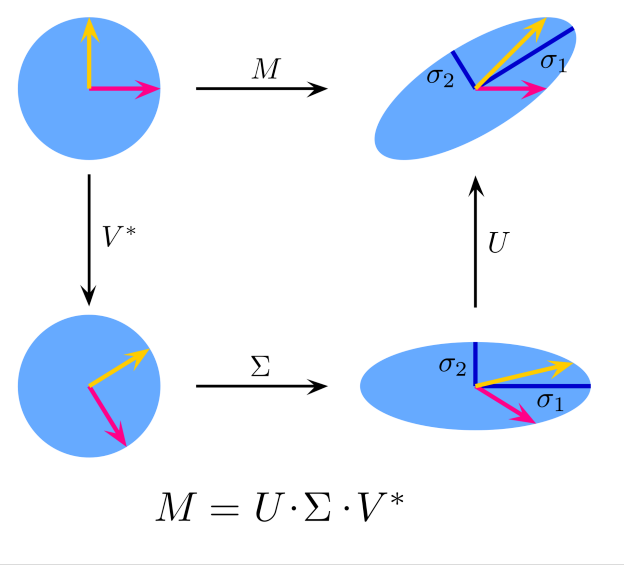



Figure 1: A geometrical interpretation of the SVD. Credit to Georg-Johann for producing this image and making it available on wikipedia. (https://commons.wikimedia.org/wiki/User:Georg-Johann)

## 2.1 Singular Value Decomposition

The singular value decomposition (SVD) is one of the most powerful results in linear algebra. Given a matrix of data, the SVD will provide information about both the direction and magnitude of its covariance.

$$M = USV^* \tag{1}$$

Geometrically, the SVD is a rotation (U), a stretch (S), and another rotation (V). Both U and V are unitary real or complex matricies that need not be square while S is a diagonal matrix containing the singular values.

The goal of the SVD is to take the matrix M into a new coordinate system in which the variance of the system is orthogonalized. In other words, all correlations have been accounted for and only the truly distinct directions of variance remain. Suppose an array of 10 points over time were perfectly correlated. The SVD would determine that all variation to this system can be explained with a single mode representing parallel shifts. The dimension of this system would thus be reduced from 10 to 1.

## 2.2 Linear Discriminant Analysis

Given a data set representing two or more classes, linear discriminant analysis seeks a basis that accomplishes two things. First, the variance within classes are minimized. In other words, in this new basis the data for each class cluster near each other to the maximum extent possible. Second, the variance between classes are maximized. Said another way, while the classes are clustered tightly together with the first condition, the distinct clusters are maximally seperated.

Mathematically, LDA is defined as

$$w = arg\max_{w} \frac{w^T S_{\mathrm{B}} w}{w^T S_{\mathrm{W}} w} \tag{2}$$

Having $S_{\mathrm{W}}$ in equation 2 defined as

$$S_{\mathrm{W}} = \sum_{i=1}^{C} S_{\mathrm{i}} \tag{3}$$

With $S_\mathrm{i}$ as

$$S_\mathrm{i} = \sum_{z \in w_\mathrm{i}} (x - \mu_\mathrm{i})(x - \mu_\mathrm{i})^T \tag{4}$$

$$\mu_\mathrm{i} = \frac{1}{N_\mathrm{i}} \sum_{z \in w_\mathrm{i}} x \tag{5}$$

$S_\mathrm{B}$ in equation 2 is defined as follows

$$S_\mathrm{B} = \sum_{i=1}^{C} N_\mathrm{i}(\mu_\mathrm{i} - \mu)(\mu_\mathrm{i} - \mu)^T \tag{6}$$

Having $\mu$ and $\mu_\mathrm{i}$ defined as

$$\mu = \frac{1}{N} \sum_{\forall x} x = \frac{1}{N} \sum_{\forall x} N_\mathrm{i} \mu_\mathrm{i} \tag{7}$$

$$\mu_\mathrm{i} = \frac{1}{N_\mathrm{i}} \sum_{x \in \omega_\mathrm{i}} x \tag{8}$$

Given the projection of our data onto this new basis, it should be the case that our data is maximally separated. With this maximal separation, class boundaries can be drawn such that data within each separate interval represent a distinct class. Certainly, when working with noisy 'real world' data, the effectiveness of this approach will depend completely on the presence of distinct classes. If there is significant overlap in data, no matter which basis we project onto it will be difficult to achieve satisfactory accuracy in classification predictions.

# 3 Algorithm Implementation and Development

Code has been developed in python for this research and is broken into two primary sections. First, a web crawler has been constructed to gather text data. Second, given a populated database of text data there is a distinct code base for analysis, plotting, and classification purposes.

## 3.1 Web Crawler

Text data is central to this analysis. In python, a web crawler has been developed to retrieve every news article from Yahoo Finance for a given day. This web crawler can be run over many days to accumulate a time series of text data. While New York Times, Bloomberg, and other publications might also be desired, Yahoo Finance was selected because of simplicity.

### 3.1.1 Workflow

First, we gather an inventory of article URLs for a news day. To accomlish this, the Yahoo Finance site map is traversed using the urllib3 and BeautifulSoup libraries. The raw html from the site map is parsed and only valid news articles are extracted.

Second, given the inventory of article URLs for a day, we extract the article text for each. Within a loop, the raw html for each article URL is parsed to locate the article text. This exercise is surprisingly non-trivial given the overwhelming number of advertisements, script, and style elements embedded throughout the raw html of a typical article URL.

Third, by leveraging the Natural Language Toolkit (NLTK), the raw text is normalized. Normalization consists of removing punctuation, capitalization, and stop words such as 'the', 'and', and 'of'. A current list of English stop words are available for download via the NLTK. After this step, we should be left with a clean list of meaningful words in a given article. For example, the words 'rally', 'buy', and 'optimism' might remain giving a clear positive sentiment for this particular article.

Fourth, we condense our normalized data into a unique word list paired with the frequency of occurrence. In other words, which words occurred in this article (the dictionary) and how many times did each word occur (the frequency)? This vectorized form of text data is ultimately the final form for this analysis. In future research, the tokenization of word groups and word coincidence would likely prove fruitful.

Finally, to more scientifically assign sentiment to words, the SentiWords resource has been leveraged. This resource contains more than 150,000 words and word combinations with an assigned sentiment polarity between -1 and 1. Positive values are associated with positive sentiment and negative values with negative sentiment. Greater absolute values represent stronger sentiments.

This workflow is iterated for every article in a news day. The results of this iteration are combined into one vector per day. Said another way, we will leverage this crawler to construct a matrix that is (number of distinct words) x (number of dates) in size where each element in the matrix is the frequency of occurrence.

#### 3.1.2 Crawling Efficiency

There are an astounding number of articles published in a single news day. Throughout 2019 and into 2020, it is common for more than 1500 articles, and in some cases well over 2000 articles to be published in a single news day. Realistically, if leveraging a single threaded approach, the processing time required to gather this data over a meaningful length of history is several months or more. It is for this reason that a multithreaded approach has been considered.

It is essential, however, to attempt this multithreaded approach in the most respectful manner possible. While Yahoo Finance typically fields 150 million site requests per month, hammering their site with several hundred thousand additional requests might put undue burden on their servers. It is for this reason that a time window from 2013 to 2015 was selected. Far fewer articles were published per day during this time period which allows us to obtain a fairly long time series to work with for this research while minimizing the additional traffic to Yahoo.

#### 3.1.3 SQL Database and Threading

A local instance of Microsoft SQL Express was installed for the purpose of storing and accessing the word dictionary, frequencies, and sentiment score data. Primarily, the need for SQL arose from the threading approach. As each web crawling thread synchronously retrieves data, a tool that can properly handle the potential need for simultaneous access is required. For example, a raw text file with the windows operating system does not have the concept of queueing. SQL Server is designed precisely for this purpose.

Although threading primarily drove the use of SQL, it additionally performs well as a data transformation and retrieval mechanism for the subsequent analysis.

### 3.2 Market Data Sourcing

Historical market data was downloaded from Yahoo Finance by leveraging their web interface manually. The csv files were subsequently loaded into the SQL database noted in section 3.1.3.

### 3.3 Classification

The classification algorithm implements the core theoretical foundation of this paper. Specifically, we are classifying a news date as either 'buy' or 'sell'. In the 'buy' case, we expect that today's daily return will be positive. In the 'sell' case, we expect that today's daily return will be negative.

In order to accomplish this, first the singular value decomposition (SVD) is computed on the text frequency data. Secondly, given a specified number of principal components to use in the 'num features' variable, the modes for each class are extracted. Supposing a clear difference exists, these modes should capture the unique component signature between days when the market increases versus when it decreases.

The next step is to implement linear discriminant analysis. This consists of finding the interclass variance, which we'd like to minimize, and the between-class variance which we'd like to maximize. Given this we can calculate the eigendecomposition by using the $eig()$ function in SciPy's linear algebra library. The result of this step should provide us with the linear operation that optimally projects our data into a space where classes are most separated.

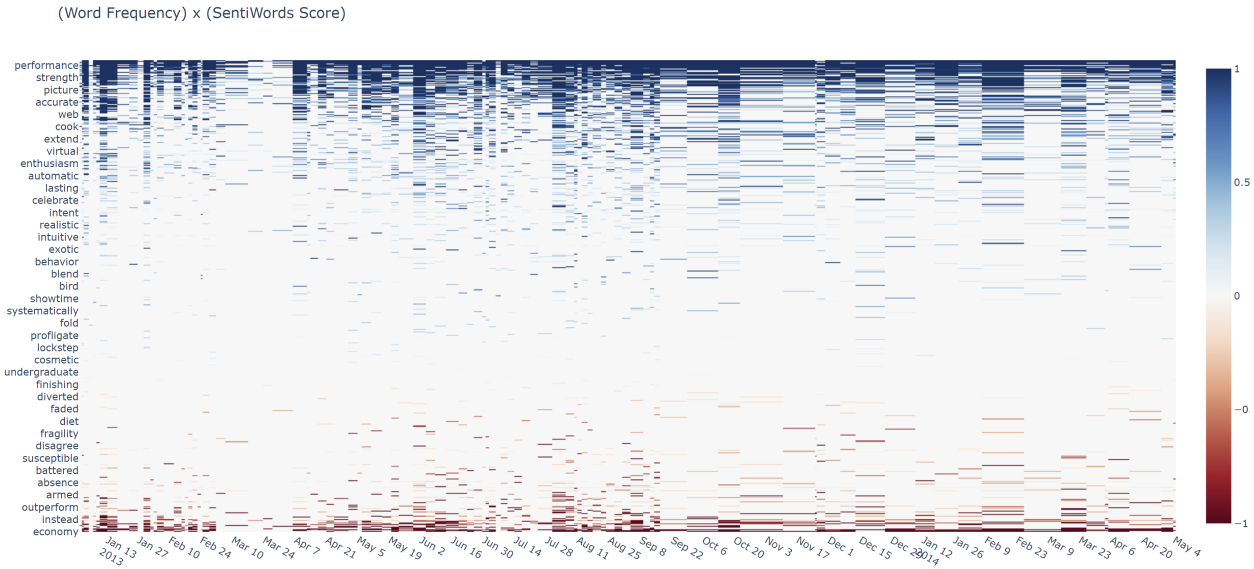



Figure 2: This heat map represents the dictionary of distinct words that occurred over the historical period. The color of each cell represents the percent of total frequency per day multiplied by its SentiWords score. Using this approach, the sentiment of a particular word is captured and it is further weighted by the number of occurrences. Red represents negative sentiment while blue represents positive sentiment.

The final step is to then draw the boundaries between the separated classes. These boundaries represent the result of our supervised learning. With these boundaries we can find our training accuracy to determine how well this approach is performing on the training data.

# 4 Computational Results

A random sample of 98 news days spanning from January 1st, 2013 to May 4th, 2014 has been scraped from Yahoo Finance. A representation of the word frequencies and sentiments can be seen in figure 2. In total, after text normalization 23,956 distinct words occurred more than once per day during this time period. Words occurring only 1 time per day were filtered out due to the increased computing power required to manage the larger data set coupled with the relatively low level of information provided by them.

## 4.1 Market Data

Market data for 6 major indexes and stocks were retrieved from Yahoo Finance and converted to buy versus sell signals. A representation of these classes can be found in figure 4.

- `FB` is Facebook stock. Selected because it was referenced often during this time period.
- `MSFT` is Microsoft stock. Selected because it was referenced often during this time period.
- `EURUSD` is the Euro to US Dollar exchange rate.
- `VIX` is an index capturing equity market volatility.
- `S&P` is the S&P500 stock market index.
- `DJI` is the Dow Jones Industrial Average.

## 4.2 Test and Training Data

The text and market data were split into training sets (80%) and test sets (20%) for purposes of avoiding data snooping while building the machine learning classifier.

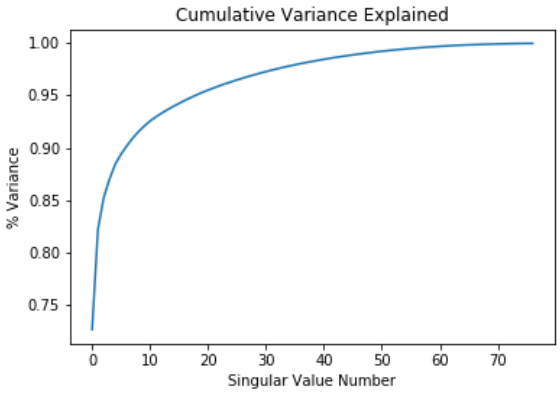



Figure 3: 95% of the variance in the text data, which is comprised of 23,956 distinct words over 98 days, can be represented by 20 principal components.

### 4.3 Singular Value Decomposition

Performing the SVD on the training text data, in figure 3 we find that the text data is well represented with as few as 20 modes. This is a significant reduction in overall dimension of our text data..

### 4.4 Classification

In terms of classification results, the LDA performed surprisingly well on the training data given that a sufficient number of principal components were used in the SVD. In figure 5, we find that given a single principal component, the classification is no better than random guessing hovering around 50% for all indexes and stocks.

However, on the other end of the spectrum, if leveraging at least 40 modes in the SVD, the Dow Jones Industrial Average, S&P500, VIX Volatility indicator, and Microsoft stock are all predicted with greater than 90% accuracy. In fact, above 60 modes all are predicted with greater than 90% accuracy while the S&P500 is predicted with a surprising 100% accuracy. A more detailed view of the classification performance for the Dow Jones Industrial Average can be seen in figure 7.

### 4.5 Test Performance

On the test data, out of sample performance was less impressive. As seen in figure 6, no index exceeded 80% accuracy while the majority hovered slightly above guessing at 50% to 60%. One possible reason for this is that the test data set is just too small. With only 20 days of frequency data represented, we have a very small sample from which to draw conlclusions. In subsequent work, it will be interesting to allow the web crawler to gather a much longer time series of text data. Perhaps this will allow for better out of sample performance.

Another possibile explanation for the poor test performance is that this analysis requires a much larger training data set in order to capture the full variation that exists in the universe of news. In other words, an incredibly wide variety of topics may be covered week to week and month to month. If we have not included a sufficient sample of this variety in our training, we may not be surprised to see that the algorithm struggles to achieve peak accuracy. Interpolation given a large training set can often be very successful while extraplation can be much less predictable.

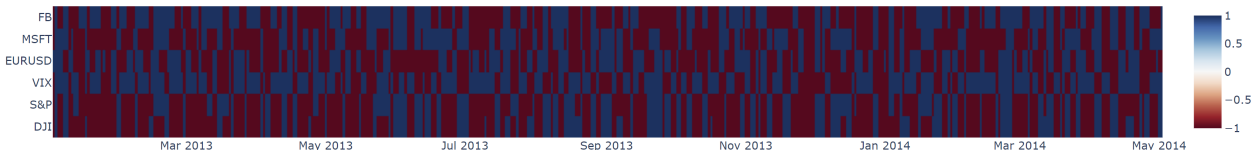



Figure 4: This heat map represents the buy (blue) and sell (red) signal for 6 market indexes stocks over the historical window. Buy signals represent positive forward looking daily returns. Sell signals represent negative forward looking daily returns. Note that weekends and holidays have been carried forward.

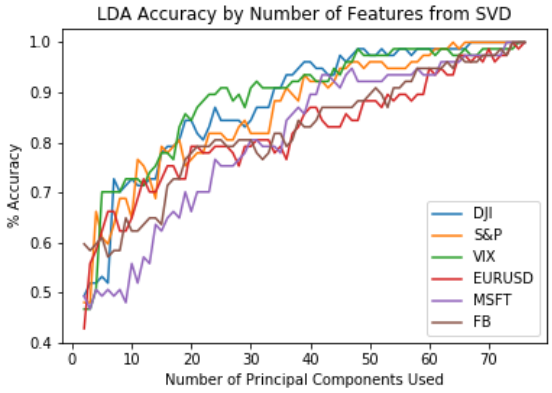



Figure 5: A representation of LDA **training** accuracy performance for each index and stock spanning a range of principal components used in the SVD.

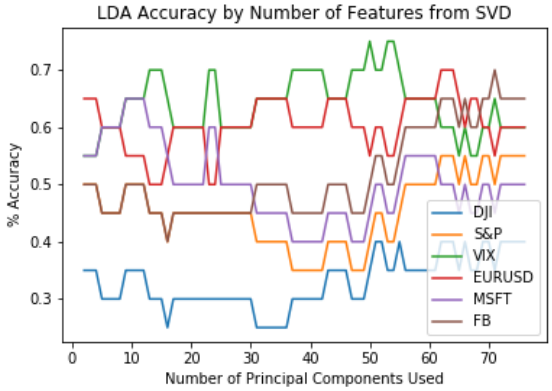
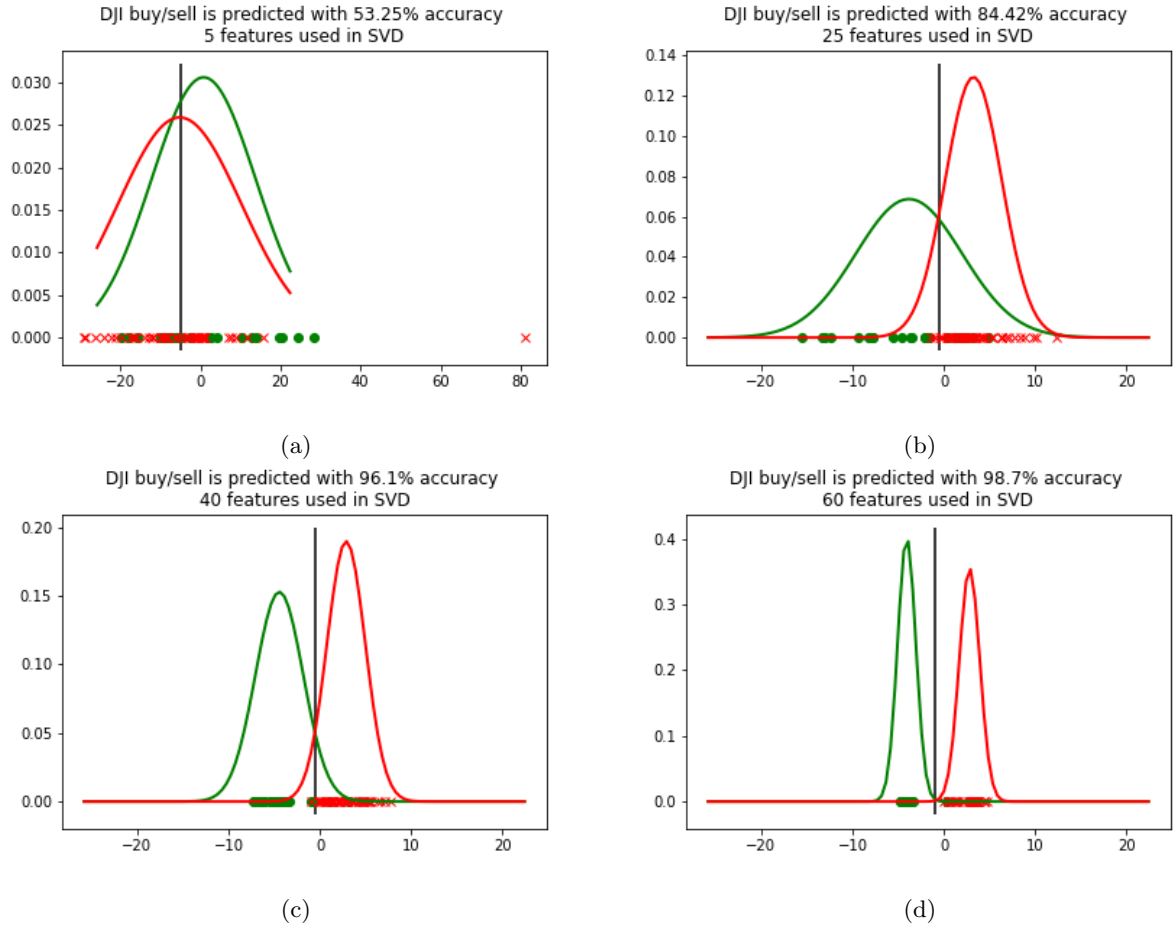



Figure 6: A representation of LDA **test** data accuracy for each index and stock spanning a range of principal components used in the SVD.




(a) (b) (c) (d)

Figure 7: A visual representation of the LDA applied to the Dow Jones Industrial Average classification. Green represents the 'buy' class while red represents the 'sell' class. The LDA struggles with fewer SVD features but performs exceptionally well in separating the classes when greater than 25 features are used.

# 5 Summary and Conclusions

There appears to be a promising relationship present in this research. Intuitively, the stock market can be thought of as a function of the news and the results presented in this paper suggest that this relationship can likely be quantified. The linear discriminant analysis (LDA) performed exceptionally well on the training data given a sufficient-rank approximation of the text data. Less than rank 25 does not produce acceptable results while greater than 50 seems to produce an impressive accuracy.

Test performance paled in comparison to the training performance. Likely this was driven, at least in part, by a relatively small number of days available for validation.

Web crawling is a non-trivial exercise. Developing an algorithm to properly scrape article text from an array of URLs proved more time consuming than anticipated. In addition, the run time of the web crawler proved significantly longer than expected. As a result, while a fairly large volume of articles were retrieved in order to form the basis of this analysis, more text data is likely required for a truly robust evaluation. Since the crawler is fully automated, it can be run perpetually in the background accumulating data to be prepared for future analysis.

# Appendix A Python Functions

The following Python libraries and functions were used for the implementation of the solution to this problem

- `pandas.DataFrame` provided the standard data structure for this analysis. Additionally, there is seamless integration with pyodbc to retrieve data from SQL directly into DataFrame objects.
- `plotly` provides an array of graphical tools such as heatmaps that were useful in illustrating the data.
- `U,S,V = numpy.linalg.svd()` is numpy's singular value decomposition implementation.
- `np.dot()` is numpy's dot product implementation. This also handles 2d matrix multiplication.
- `np.outer()` is numpy's outer product implementation. This was required when computing between class variance in the LDA routine.
- `np.argsort()` is a function in numpy that provides indexes of a sorted array. This was used to group 'buy' and 'sell' signals together in the LDA routine.
- `sp.linalg.eig()` is SciPy's eigenvalue decomposition implementation which was used in constructing the LDA routine.
- `sp.stats.norm()` is SciPy's normal distribution fitting function. While no strict distributional assumption was made in the LDA analysis computation, this routine was useful for visualization purposes.
- `manager = urllib3.PoolManager()` creates an object capable of making http and https requests.
- `manager = urllib3.ProxyManager()` functions identical to PoolManager but with the option to optionally route requests through proxy servers.
- `soup = BeautifulSoup()` parses raw html into an object that is easier to extract information from.
- `nltk.download('stopwords')` retrieves a current list of english stop words useful for text normalization.
- `pyodbc` provides functionality required to connect with and send requests to an instance of SQL server.
- `joblib.parallel_backend` provides threading functionality. This was useful to expedite web crawling.

# Appendix B Python Code - Web Crawler

```
### Libraries
import time
import random
import urllib
import urllib3
from bs4 import BeautifulSoup
import datetime
import pandas as pd
import numpy as np
from pathlib import Path
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sqlalchemy import create_engine, MetaData, Table, select
import pyodbc
import re
import string
from collections import Counter
from joblib import Parallel, delayed
import multiprocessing


use_proxies = True
################################## PROXIES
proxy_file = 'C:\\Users\\sawy0\\Documents\\GitHub\\AMATH582\\FinalProject\\ProxyList
    .txt'
proxy_link = 'https://api.proxyscrape.com/?request=getproxies&proxytype=http&timeout
    =10000&country=all&ssl=all&anonymity=elite'

#Retrieve latest proxy list and store in dataframe
urllib.request.urlretrieve (proxy_link, proxy_file)
proxies = pd.read_csv(proxy_file, sep=" ", header=None)
proxies.columns = ["IP"]
proxies = proxies.sample(frac=1)


################################## USER AGENTS
agent_file = 'C:\\Users\\sawy0\\Documents\\GitHub\\AMATH582\\FinalProject\\
    UserAgents.txt'
agents = pd.read_csv(agent_file, sep="|", header=None)
agents = agents.drop(agents.columns[[1]], axis=1)
agents.columns = ["User_Agent"]
agents = agents.sample(frac=1)


################################## SETTINGS
#####The next two lines are needed to initialize your environment the first time
nltk.download('stopwords')
nltk.download('punkt')

#Parameters to connect to local SQL instance
server = 'DESKTOP-BR9P1E0\SQLEXPRESS'
database = 'TextAnalysis'
dbTable = 'MonthlyFreqData'
```

```
#Used in BeautifulSoup to locate news articles
key1 = 'https://finance.yahoo.com/news'
key2 = 'http://finance.yahoo.com/news'

#Location to output URL lists and raw article text to their respective subfolders
out_dir = "C://Users//sawy0//Documents//GitHub//AMATH582//FinalProject//"

#Finds URLs, scrapes text, normalizes the text, counts freq, outputs to SQL
def TextCrawler(start_date):
    failures = list()
    monthlyFreq = pd.DataFrame()
    df = pd.DataFrame(columns = ['Date','URL'])
    flag=0
    url = 'https://finance.yahoo.com/sitemap/' + str(start_date).replace("-","_") +'
        /'
    articles = [];

    while flag < 1:

        # Random Proxy
        Pidx = random.randint(0,len(proxies.index)-1)
        proxy = 'http://' + proxies.loc[Pidx][0] + '/'
        #Random User Agent
        Aidx = random.randint(0,len(agents.index)-1)
        user_agent = {'user-agent': agents.loc[Aidx][0]}

        r = None
        while r is None:
            try:
                if use_proxies:
                    pmanager = urllib3.ProxyManager(proxy, headers=user_agent)
                    r = pmanager.request('GET', url)
                else:
                    pmanager = urllib3.PoolManager()
                    r = pmanager.request('GET', url)
            except:
                failures.append(url)
                pass
        tmp = r.data.decode('utf-8')
        soup = BeautifulSoup(tmp)

        for a in soup.find_all('a', href=True):
            if (key1 in a['href'] or key2 in a['href']) and a['href']!='https://
                finance.yahoo.com/news/' and 'edited-transcript' not in a['href']:
                df = df.append(pd.Series([start_date, a['href']], index = df.columns
                    ), ignore_index=True)
                articles.append(a['href'])

        #Find the next URL (ieif you navigated to thenext 50 articles by clicking
            next)
        for a in soup.find_all('a', href=True):
            if 'start' in a['href']:
                next_url = a['href']
                url = next_url
                flag = 0
            else:
                flag = 1 #When flag is 1, move next day

    #Create Output Directory if Needed
```

```
Path(out_dir+'URL_Lists\\').mkdir(parents=True, exist_ok=True)
urlfile = out_dir+'URL_Lists\\'+str(start_date).replace("-","_")+"_URL_List.txt"
text_file = open(urlfile, "w")
text_file.write(str(articles))
text_file.close()

#############################################################################
####### At this point we have df containing dates and article URLs
####### Need to now loop and scrape text/freq for each
#############################################################################
text_data = list()
for i in range(len(df)):
    text_data = list()
    x = df['URL'].loc[i]
    r = None
    z=1
    while r is None:
        z=z+1
        t_end = time.time() + 30
        try:
            while time.time() < t_end:
                if use_proxies:
                    ### Connect to the URL
                    # Need a random Proxy
                    Pidx = random.randint(0,len(proxies.index)-1)
                    proxy = 'http://' + proxies.loc[Pidx][0] + '/'
                    # Need a random user agent
                    Aidx = random.randint(0,len(agents.index)-1)
                    user_agent = {'user-agent': agents.loc[Aidx][0]}
                    #Create the proxymanager pool object and retrieve
                    pmanager = urllib3.ProxyManager(proxy, headers=user_agent,
                        timeout=120)
                    r = pmanager.request('GET', x)
                else:
                    pmanager = urllib3.PoolManager()
                    r = pmanager.request('GET', x)
                t_end = time.time()
        except:
            failures.append(x)
            pass
        if z==10:
            r=1

    tmp = r.data.decode('utf-8')
    soup = BeautifulSoup(tmp)
    artsoup = soup.find('article')

    if artsoup is not None:
        #kill all script and style elements
        for script in artsoup(["script", "style"]):
            script.extract()    # rip it out
        #get text
        text = artsoup.get_text()
        #break into lines and remove leading and trailing space on each
        lines = (line.strip() for line in text.splitlines())
        #break multi-headlines into a line each
        chunks = (phrase.strip() for line in lines for phrase in line.split("  "
            ))
        #drop blank lines
```

```
text = '\n'.join(chunk for chunk in chunks if chunk)
#Append to text_data object
text_data.append(text)

#Write the raw text
filename = x
filename = filename.replace('http://finance.yahoo.com/news/','')
filename = filename.replace('https://finance.yahoo.com/news/','')
filename = filename.replace('.html','')
file = out_dir+'Articles\\'+str(start_date)+'\\'+filename+'.txt'

#Create Output Directory if Needed
Path(out_dir+'Articles\\'+str(start_date)+'\\').mkdir(parents=True,
    exist_ok=True)

#Write the RAW article to a text file
try:
    text_file = open(file, "w")
    text_file.write(text)
    text_file.close()
except:
    failures.append(x)
    pass

#########################################################
#Text Normalization, Corpus, and Frequency Block
#########################################################
#https://medium.com/@datamonsters/text-preprocessing-in-python-steps-
    tools-and-examples-bf025f872908
text2 = text.lower()                    #Make all lower case
text2 = re.sub(r'\d+', '', text2)       #Remove #s
text2 = text2.translate(str.maketrans("","", string.punctuation))   #
    Remove Punctuation
text2 = text2.replace("'", "")          #Remove apostrophes
text2 = text2.replace("'", "")          #Remove apostrophes
text2 = text2.replace('"', "")          #Remove quotes
text2 = text2.replace('"', "")          #Remove quotes
text2 = text2.replace("-", "")          #Remove dashes
text2 = text2.replace("-", "")          #Remove dashes
text2 = text2.replace("(", "")          #Remove dashes
text2 = text2.replace(")", "")          #Remove dashes
text2 = text2.strip()                   #Remove White Spaces

#Remove stopwords (and, the, but, etc...)
stop_words = set(stopwords.words('english'))
tokens = word_tokenize(text2)
text2 = [i for i in tokens if not i in stop_words]

if text2!=[]:

    #Get unique word list and frequencies
    dict_sum = Counter(text2)

    #Convert to dataframe
    out_df = pd.DataFrame(dict_sum.items(), columns=['Word', 'Freq'])
    out_df['Date'] = str(df['Date'].loc[i])
    out_df['UpdateTime'] = str(datetime.datetime.utcnow())
    out_df = out_df[['Date','Word','Freq','UpdateTime']]
```

```
                #Handle the monthly aggregates
                monthlyFreq = monthlyFreq.append(out_df)

    monthlyFreq = monthlyFreq.groupby(['Word']).sum()
    monthlyFreq['Date'] = str(df['Date'].loc[i])
    monthlyFreq['UpdateTime'] = str(datetime.datetime.utcnow())
    monthlyFreq = monthlyFreq.reset_index()
    monthlyFreq = monthlyFreq[['Date','Word','Freq','UpdateTime']]
    monthlyFreq = monthlyFreq.loc[monthlyFreq['Freq']>1]

    #Write dataframe to SQL
    params = urllib.parse.quote_plus("DRIVER={SQL Server};SERVER="+server+";DATABASE
        ="+database)
    engine = create_engine("mssql+pyodbc:///?odbc_connect=%s" % params)
    cnxn = engine.connect()
    monthlyFreq.to_sql(name=dbTable,con=cnxn, index=False, if_exists='append',
        method='multi', chunksize=100)
    cnxn.close()
    engine.dispose()

    del df
    return 1

##########################
###  THREADING
##########################

#Thread Settings
num_cores = 150
numdays = 150
base = datetime.date(2013,12,16)
date_list = [base + datetime.timedelta(days=x) for x in range(numdays)]


#Multi-Thread all of the above by date
from joblib import parallel_backend
with parallel_backend('threading', n_jobs=num_cores):
    results = Parallel(n_jobs=num_cores)(delayed(TextCrawler)(i) for i in date_list)
```

# Appendix C Python Code - Classification

```
### Libraries
import datetime
import pyodbc
import pandas as pd
import numpy as np
import scipy as sp
from scipy import linalg
from scipy.stats import norm
import plotly
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from plotly.offline import plot


###Plot Directory
plot_dir = 'C:\\Users\\sawy0\\Documents\\GitHub\\AMATH582\\FinalProject\\Figures\\'


### SQL Database Location
server = 'DESKTOP-BR9P1E0\SQLEXPRESS'
database = 'TextAnalysis'
dbTable = 'MonthlyFreqData'
cnxn = pyodbc.connect("DRIVER={SQL Server};SERVER="+server+";DATABASE="+database)


### If the web crawler has been run recently, update the Frequency Matrix in SQL
#sql = "EXEC [dbo].[PopulateFrequencyMatrix]"
#update_freq = pd.read_sql(sql,cnxn)


### Gather Frequency Data
sql = "SELECT * FROM [dbo].[FrequencyMatrix] WHERE abs(Score*All_Dates)>0 AND Score
    <> 0 ORDER BY Score*All_Dates"
raw_freq = pd.read_sql(sql,cnxn)
freq = raw_freq.copy()
del freq['Word']
del freq['Sentiment']
del freq['Score']
del freq['All_Dates']
freq = freq.fillna(0)
freq/freq.sum()
freq = freq.mul(raw_freq['Score'], axis=0)


### Gather Market Data
sql = """SELECT AsOfDate, DJI, [S&P], VIX, EURUSD, MSFT, FB FROM
(
        SELECT C.AsOfDate, A.Ticker, B.PriceAdjClose - A.PriceAdjClose AS
            DailyReturn
        FROM dbo.Calendar C
        LEFT JOIN dbo.vwMarketData A
                ON C.AsOfDate = A.AsOfDate
        LEFT JOIN (
                SELECT DATEADD(DAY, 1, AsOfDate) AS AsOfDate, PriceAdjClose, Ticker
                FROM [dbo].[vwMarketData]
```

```
                        ) B
        ON A.AsOfDate = B.AsOfDate
                AND A.Ticker = B.Ticker
        WHERE C.AsOfDate >= (SELECT MIN([Date]) FROM [dbo].[MonthlyFreqData])
                AND C.AsOfDate <= (SELECT MAX([Date]) FROM [dbo].[MonthlyFreqData])
) X
PIVOT
(
        Max(DailyReturn)
        FOR Ticker IN (DJI, [S&P], VIX, EURUSD, MSFT, FB)
)P
ORDER BY AsOfDate"""


raw_mkt = pd.read_sql(sql,cnxn)
raw_mkt = raw_mkt.fillna(method='ffill') #Forward fill to cover weekend values


### Create a Buy Flag for classification purposes -1 = sell, 1 = buy
#####################################################################
BuyFlag = raw_mkt.copy()
BuyFlag = BuyFlag.fillna(0)

for col in BuyFlag.columns[BuyFlag.columns!='AsOfDate']:
    BuyFlag[col][BuyFlag[col] < 0] = -1     #If negative, sell
    BuyFlag[col][BuyFlag[col] > 0] = 1      #If positive, buy
    BuyFlag[col][BuyFlag[col] == 0] = -1    #If 0, sell..... risk averse assumption


### Plot buy flag
fig = go.Figure(
        data = go.Heatmap(
        zmin = -1,
        zmax = 1,
        z=BuyFlag[BuyFlag.columns[BuyFlag.columns!='AsOfDate']].transpose(),
        x=list(BuyFlag['AsOfDate']),
        y=list(BuyFlag.columns[BuyFlag.columns!='AsOfDate']),
        colorscale='RdBu'
    )
)
plot(fig)


### 80 / 20 test train split
freq_tmp = freq.copy()
test_data = freq_tmp[freq_tmp.columns[78:98]]
train_data = freq_tmp[freq_tmp.columns[0:77]]


freq = train_data
#############################################################
### Analysis

df = pd.DataFrame(index=list(range(2,len(V))))
for ticker in raw_mkt.columns[raw_mkt.columns!='AsOfDate']:
    accuracies = list()
    for num_features in range(2,len(V)):

        date_idx = BuyFlag['AsOfDate'].isin(freq.columns)
```

```
mkt = BuyFlag[date_idx]
mkt = mkt.reset_index()

sort_idx = np.argsort(mkt[ticker])
ordered_freq = freq[:][freq.columns[sort_idx]]
ordered_signal = mkt.loc[sort_idx][ticker]

### SVD
#############################################################
U, sigma, V = np.linalg.svd(ordered_freq, full_matrices = False)
energy = np.power(sigma,1)/np.sum(np.power(sigma,1))
var = np.power(sigma,2)/np.sum(np.power(sigma,2))
np.cumsum(var)

###  Plot Cumulative Variance Explained
#############################################################
import matplotlib.pyplot as plt
%matplotlib qt
plt.plot(np.cumsum(var))
plt.title('Cumulative Variance Explained')
plt.ylabel('% Variance')
plt.xlabel('Singular Value Number')
#plt.show()
plt.savefig(plot_dir + 'SVD_Variance_Explained' + ticker + '.png')

### Linear Discriminant Analysis
#############################################################
SV = sigma*V.transpose()

### Locate buy and sell, truncate by num_features
buy_idx = ordered_signal==1
buy = SV[buy_idx,1:num_features]

sell_idx = ordered_signal==-1
sell = SV[sell_idx,1:num_features]

buy_mean = np.mean(buy,axis=0)
sell_mean = np.mean(sell,axis=0)

### Interclass Variance
Sw = np.dot((buy-buy_mean).transpose(),(buy-buy_mean))
Sw = Sw + np.dot((sell-sell_mean).transpose(),(sell-sell_mean))

### Between Class Variance
Sb = np.outer((buy_mean-sell_mean).transpose(),(buy_mean-sell_mean))

### Eigen Decomposition
e_val, e_vec = sp.linalg.eig(Sb, Sw)
max_eval = np.max(np.abs(e_val))
w = e_vec[:,e_val==max_eval]
w = w/np.linalg.norm(w)

#Project Buy and Sell onto new basis
v_buy  = np.dot(w.transpose(), buy.transpose())
v_sell = np.dot(w.transpose(), sell.transpose())

#Draw Boundaries and calculate accuracy
#############################################################
#Plot the new scatter
```

```
        %matplotlib qt
        val=0
        plt.plot(v_buy, np.zeros_like(v_buy) + val, 'o',color='green')
        plt.plot(v_sell, np.zeros_like(v_sell) + val, 'x',color='red')

        #Superimpose normal distribution fits
        buy_mu,  buy_std  = norm.fit(v_buy)
        sell_mu, sell_std = norm.fit(v_sell)
        x = np.linspace(xmin, xmax, 100)

        buy_p = norm.pdf(x, buy_mu, buy_std)
        sell_p = norm.pdf(x, sell_mu, sell_std)

        #Find Boundary
        pdf_sum = np.abs(buy_p-sell_p)
        if buy_mu<sell_mu:
            bound_idx = np.where(np.logical_and(x<sell_mu, x>buy_mu))
        else:
            bound_idx = np.where(np.logical_and(x>sell_mu, x<buy_mu))
        pdf_sum_tmp = pdf_sum[bound_idx]
        x_tmp = x[bound_idx]
        x_tmp = x_tmp[pdf_sum_tmp==np.min(pdf_sum_tmp)]
        boundary = np.mean(x_tmp)

        plt.plot(x, buy_p, color='green', linewidth=2)
        plt.plot(x, sell_p, color='red', linewidth=2)

        ymin, ymax = plt.ylim()
        plt.vlines(boundary,ymin=ymin,ymax=ymax,color='black')

        if buy_mu<boundary:
            buy_correct     = np.sum(v_buy<boundary)
            sell_correct    = np.sum(v_sell>boundary)
            total           = len(buy)+len(sell)
            accuracy        = (buy_correct + sell_correct)/total
        else:
            buy_correct     = np.sum(v_buy>boundary)
            sell_correct    = np.sum(v_sell<boundary)
            total           = len(buy)+len(sell)
            accuracy        = (buy_correct + sell_correct)/total

        accuracies.append(accuracy)

        mng = plt.get_current_fig_manager()
        mng.window.showMaximized()
        plt.title(ticker + ' buy/sell is predicted with ' + str(np.round(accuracy
            *100,2)) + '% accuracy \n' + str(num_features) + ' features used in SVD')
        #plt.show()
        plt.savefig(plot_dir + 'LDA_Accuracy_' + ticker + '_' + str(num_features) +'
            .png')

    df[ticker] = pd.Series(accuracies, index=df.index)

#How does accuracy increase relative to num_features?
%matplotlib qt
plt.plot(df)
plt.title('LDA Accuracy by Number of Features from SVD')
plt.ylabel('% Accuracy')
plt.xlabel('Number of Principal Components Used')
```

```
plt.legend(df.columns)
plt.show()
plt.savefig(plot_dir + 'LDA_Accuracy_ALL_By_Num_Feature.png')


############################################################
### TEST DATA
############################################################

df_test = pd.DataFrame(index=list(range(2,len(V))))
for ticker in raw_mkt.columns[raw_mkt.columns!='AsOfDate']:
    test_accuracies = list()
    for num_features in range(2,len(V)):

        date_idx = BuyFlag['AsOfDate'].isin(test_data.columns)
        mkt = BuyFlag[date_idx]
        mkt = mkt.reset_index()

        sort_idx = np.argsort(mkt[ticker])
        ordered_freq = freq[:][freq.columns[sort_idx]]
        ordered_signal = mkt.loc[sort_idx][ticker]

        tmp = np.dot(U[:,0:(num_features-1)].transpose(), ordered_freq)
        pval = np.dot(w[0:(num_features-1)].transpose(),tmp)

        if buy_mu<boundary:
            buy_correct     = np.sum(pval[:,ordered_signal==1]<boundary)
            sell_correct    = np.sum(pval[:,ordered_signal==-1]>boundary)
            total           = len(ordered_signal)
            accuracy        = (buy_correct + sell_correct)/total
        else:
            buy_correct     = np.sum(pval[:,ordered_signal==1]>boundary)
            sell_correct    = np.sum(pval[:,ordered_signal==-1]<boundary)
            total           = len(ordered_signal)
            accuracy        = (buy_correct + sell_correct)/total

        test_accuracies.append(accuracy)
    df_test[ticker] = pd.Series(test_accuracies, index=df.index)

#How does accuracy increase relative to num_features?
%matplotlib qt
plt.plot(df_test)
plt.title('LDA Accuracy by Number of Features from SVD')
plt.ylabel('% Accuracy')
plt.xlabel('Number of Principal Components Used')
plt.legend(df_test.columns)
plt.show()
plt.savefig(plot_dir + 'Test_Accuracy_ALL_By_Num_Feature.png')
```

```
#############################################################
### Miscellaneous Plotting
#############################################################

### Heatmap of Raw Text and one indicator
#############################################################
fig = make_subplots(rows=2, cols=1, row_heights=[.1, .9]) #this a one cell subplot
fig.add_trace(
    go.Heatmap(
        zmin = -1,
        zmax = 1,
        z=ordered_freq,
        x=list(ordered_freq.columns),
        y=raw_freq['Word'],
        colorscale='RdBu'
    )
    ,row=2
    ,col=1
)
fig.add_trace(
    go.Scatter(
        mode='lines',
        #x=mkt.loc[sort_idx]['AsOfDate'],
        y=ordered_signal,
        line=dict(color='black')
    )
    ,row=1
    ,col=1
)
plot(fig)


#############################################################
### Heatmap of Principal Orthogonal Components and one indicator
fig = make_subplots(rows=2, cols=1, row_heights=[.1, .9]) #this a one cell subplot
fig.add_trace(
    go.Heatmap(
        zmin = -1,
        zmax = 1,
        z = SV[buy_idx,1:5].transpose(),
        colorscale='RdBu'
    )
    ,row=2
    ,col=1
)
fig.add_trace(
    go.Scatter(
        mode='lines',
        y=ordered_signal,
        line=dict(color='black')
    )
    ,row=1
    ,col=1
)
plot(fig)

#############################################################
### Heatmap and one indicator
fig = make_subplots(rows=2, cols=1, row_heights=[.1, .9]) #this a one cell subplot
```

```
fig.add_trace(
    go.Heatmap(
        zmin = -1,
        zmax = 1,
        z=freq,
        x=list(freq.columns),
        y=raw_freq['Word'],
        colorscale='RdBu'
    )
#     ,secondary_y=False
    ,row=2
    ,col=1
)

fig.add_trace(
    go.Scatter(
        mode='lines',
        x=raw_mkt['AsOfDate'],
        y=raw_mkt['DJI'],
        line=dict(color='black')
    )
#     ,secondary_y=True
    ,row=1
    ,col=1
)

plot(fig)


############################################################
### Heatmap only
fig = go.Figure(data=go.Heatmap(
        zmin = -1,
        zmax = 1,
        z=freq,
        x=list(freq.columns),
        y=raw_freq['Word'],
        colorscale='RdBu'))
        #https://plot.ly/python/builtin-colorscales/

fig.update_layout(
    title='(Word Frequency) x (SentiWords Score)',
    xaxis_nticks=36)

plot(fig)
```