A drawing of a cartoon character

Description automatically generated

Jeff Levesque

https://github.com/jeff1evesque/ist-736 | Final Project

IST-736: Market Sentiment

PRofessor gates

Table of Contents

[Introduction 2](#_Toc20320117)

[Data Preparation 2](#_Toc20320118)

[Twitter API 2](#_Toc20320119)

[Quandl API 3](#_Toc20320120)

[Amazon Mechanical Turk 4](#_Toc20320121)

[Joining Data 4](#_Toc20320122)

[Exploratory 5](#_Toc20320123)

[Stop Words 5](#_Toc20320124)

[Topic Model 5](#_Toc20320125)

[Latent Dirichlet Allocation 6](#_Toc20320126)

[Selected Topics 7](#_Toc20320127)

[Sentiment Analysis 9](#_Toc20320128)

[Analysis 11](#_Toc20320129)

[Baseline Results 15](#_Toc20320130)

[Classification 15](#_Toc20320131)

[Time series 21](#_Toc20320132)

[Granger Causality 27](#_Toc20320133)

[Select Results 28](#_Toc20320134)

[Granger Causality 28](#_Toc20320135)

[Classification 29](#_Toc20320136)

[Time series 29](#_Toc20320137)

[Compute Benchmark 29](#_Toc20320138)

[Conclusions 31](#_Toc20320139)

[References 32](#_Toc20320140)

[Appendix A 33](#_Toc20320141)

# Introduction

Do the opinions of Financial Analysts on Twitter impact market sentiment and volatility? There are many financial analysts on twitter offering investment advice to millions of people daily. These analysts also have other mediums outside of Twitter, including TV shows, blogs and newspaper columns. However, Twitter can provide twitter sentiment towards the market that analysts are portraying daily. For example, Jim Cramer is arguably the most famous financial analyst today, having over 1.1 million followers on Twitter, and host of the popular Mad Money television series. While audience members often look to financial analysts such as Jim Cramer, one may question whether their advice impacts the investment decisions of followers, and in turn, impacting market volatility?

Market volatility is a statistical measure of dispersion relative to security or market index. Generally, higher volatility is a sign of greater risk since the range of possible values vary relatively more[[1]](#footnote-1). Market volatility can be tracked by using the CBOE Volatility Index (VIX). The VIX is a market index that represents the market's expectation of 30-day forward-looking volatility. It is derived from the price inputs of the S&P 500 index options; it provides a measure of market risk and investors' sentiments. The VIX is considered a reflection of investor sentiment; the higher the VIX is up, the higher the levels of investor anxiety and market volatility.

If there is a correlation between the sentiment being portrayed by these analysts towards the market and the level of market volatility for that day, then this information can be a valuable tool for investors to incorporate into their investment strategy. Having advanced knowledge of when a market is about to increase in volatility would be invaluable to investors allowing them to adjust their portfolios preempting the market risk. Tweets have been known to impact the market before, on May 5th 2019, President Trump tweeted negatively about a Trade War with China and the VIX rose by as much as 46.1% intra-day the next market day, while the Dow Jones Industrial Average plunged by as much as 471 points .

# Data Preparation

## Twitter API

The Twitter API[[2]](#footnote-2) was implemented using an approved Twitter developer[[3]](#footnote-3) account. A python template file (config--TEMPLATE.py) was created containing dummy text representing the secret key and tokens provided with the Twitter developer account. This file was copied as config.py, with values properly substituted. Changes were a requirement of the general application implementing the Twython[[4]](#footnote-4) package. Within the application, two main twitter functionalities were streamlined. The first allowed general querying through a set of parameters[[5]](#footnote-5), while the second allowed querying content for specified twitter screen names. This functionality was provided using the user timeline component[[6]](#footnote-6).

Five screen names were queried:

* Jimcramer
* ReformedBroker
* TheStalwart
* LizAnnSonders
* SJosephBurns

Corresponding code generated dataframe structures, for each of the above screen names, then outputted to an associated csv file[[7]](#footnote-7). On future executions, if the corresponding csv file already exists, then the twitter api did not duplicate exiting files.

Furthermore, the parameters collected from the twitter accounts were screen\_name, created\_at, and full\_text. Each account was collected using a rate\_limit=900[[8]](#footnote-8). This ensured that the maximum number of tweets could be collected per screen name. However, due to the request limit, roughly 15 minutes needed to transpire before re-executing, to obtain the maximum content for the successive screen name. Thus, a little over 1.5 hours was required to initially generate local csv files.

Finally, a default start\_date = datetime(3000, 12, 25) and end\_date = datetime(1000, 12, 25) was defined. This definition was created to represent the datetime range for a given twitter screen name. Specifically, the initial start\_date was compared to each tweet for a given user. If a tweet exists with an earlier datetime, this was set as the new start\_date. This type of logic was extended similarly for the end\_date. This maximized value allows the functional tweet domain to accurately map to the quandl historical range.

## Quandl API

Like the Twitter API, the python Quandl API[[9]](#footnote-9) was utilized to acquire market data, including the Nasdaq index. An account was needed to obtain the associated API key, and the same config.py was utilized, respectively. Moreover, the date range was maximized[[10]](#footnote-10) in order to obtain the largest possible dataset. While obtaining data was not as restricted by the same rate limit as Twitter, a local csv file was created. This ensures integrity and optimization in case a future study extends with additional datasets. While five different columns were returned, only the Index Value was utilized for successive calculations, described in sections below:

* Trade Date
* Index Value
* High
* Low
* Total Market Value
* Dividend Market Value

Finally, as described earlier, market data range, was predicated on the maximized tweet domain.

|  |
| --- |
|  |
| Figure 1: domain mapping from Twitter API (x) and Quandl Data f(x)[[11]](#footnote-11). |

## Amazon Mechanical Turk

An attempt was made to acquire additional data, using the Amazon Mechanical Turk (MTurk) crowd sourcing platform[[12]](#footnote-12). Specifically, MTurk templates were created, with the goal of attaining additional 1600 tweets for each financial analyst starting on the date 05/08/2019[[13]](#footnote-13). However, obtained results were not desirable for many of the earlier completed workers[[14]](#footnote-14). Thus, remaining jobs were immediately canceled. In addition to poor results, the general data collection[[15]](#footnote-15) task was not optimal. Specifically, any number of workers can overload data collection for a given date, quickly exhausting the total requested amount. Thus, even in the ideal case of accurate reporting, it possible nearly all participants report on the first start date, causing the last N tweets/dates not to be reported. The ideal situation would restrict a single worker per published job. However, the platform allows number of users to participate.

## Joining Data

To simplify processing, tweets were aggregated by created\_at and screen\_name. If an account tweeted multiple times a given day, each full\_text instance was concatenated to a single string. This allowed sentiment measure to be computed as a time series. Furthermore, each twitter account data was merged on Trade Date = created\_at column. Moreover, later classification tasks were predicated on comparing the current day market value with the previous day. Thus, some edge cases needed to be considered:

1. If the first Index Value is nan, drop the instance
2. If successive (n+1) index has a previous step nan, skip and do nothing
3. If successive (n+1) index is nan, set market values to previous day and concatenate current full\_text with previous day.

Additionally, if a given day contained an empty string for full\_text, this instance was dropped, and the dataframe index was reset.

# Exploratory

Initial exploration was performed for each twitter screen name, and overall aggregation. Specifically, word clouds, vader sentiment, and topic modeling was determined for each twitter screen name. Finally, word clouds and sentiment measures were repeated on the overall dataset.

## Stop Words

Two set of stop words[[16]](#footnote-16) were utilized during exploration, and later analysis:

* stopwords: general stop words for topic modeling and vectorization
* stopwords\_topics: combined with general stopwords and for topic modeling

## Topic Model

Topic Modeling (TL)[[17]](#footnote-17) was implemented against the five financial analysts mentioned above. Specifically, the corresponding twitter accounts were fed into the twitter API[[18]](#footnote-18) using the Twython[[19]](#footnote-19) python package. Then, the collected data was tokenized[[20]](#footnote-20) using CountVectorizer[[21]](#footnote-21) to obtain the term frequency (TF):

 **(equation 1)**

Specifically, the TF is the ratio of word occurrences divided by the total number of terms in the given document, inputted into the LatentDirichletAllocation[[22]](#footnote-22) method.

### Latent Dirichlet Allocation

The implemented codebase[[23]](#footnote-23) provides the ability to utilize a deterministic Non-Negative Matrix Factorization (NMF), and the probabilistic Latent Dirichlet Allocation (LDA) for topic modeling. While a comparison of the approaches could be analyzed, the benefits would not significantly outweigh additional computing. Without loss of generality, only the LDA was used. Furthermore, since latent variables are inferred (rather than observed) through iteration and maximization steps, the overall model follows a generative pattern.

 **(equation 2)**

However, to better understand the generative process, consider a simple case – predicting the topic of a token (x). This can be expressed as the joint probability of the word and topic:

 **(equation 3)**

In the above (equation 3), θ represents the per document topic distribution, and β the per corpus topic distribution. Furthermore, the likelihood component denotes the distribution of words for a given topic, while the prior signifies the number of topics for a given document. Moreover, the product of all token probabilities represents the probability a given document belongs to a specific topic[[24]](#footnote-24). LDA can be generalized and expressed as a distribution:

 **(equation 4)**

However, both θ and z are unknown hyperparameters, so approaches such as maximum likelihood estimation (MLE) cannot be performed directly. Instead, the Expectation-Maximization (EM) LDA[[25]](#footnote-25) was utilized. This process involves iteratively computing latent variables of the posterior distribution from equation 4. Since derivation of the posterior cannot be computed easily, an alternative posterior is used in place:

 **(equation 5)**

In this step, inferential statistic is used to approximate the best γ, and φ, minimizing the difference with the true posterior distribution[[26]](#footnote-26).

### Selected Topics

While the number of topics was chosen to be 10 for each financial analyst, an elbow method could be implemented to dynamically determine an appropriate number of topics. However, due to limited compute resources and timeline, this is left for future enhancements. Results from the below exploratory step, determined the associated quandl[[27]](#footnote-27) code for later analysis:

* FNYX\_QQQ
* FNSQ\_SPY
* BATS\_AMZN
* BATS\_GOOGL
* BATS\_AAPL
* BATS\_NFLX
* BATS\_MMT
* FNYX\_MMM
* PET\_RWTC\_D
* PR\_CON\_15YFIXED\_IR
* PR\_CON\_30YFIXED\_APR

Furthermore, for completeness additional codes were later analyzed:

* CBOE\_VX1
* COMP-NASDAQ

|  |  |
| --- | --- |
|  |  |
| Figure 2: LDA for JimCramer. | **Figure 3:** LDA for JimCramer. |

|  |  |
| --- | --- |
|  |  |
| Figure 4: word cloud for LizAnnSonders. | **Figure 5:** word cloud for LizAnnSonders. |

|  |  |
| --- | --- |
|  |  |
| Figure 6: word cloud for ReformedBroker. | **Figure 7:** word cloud for ReformedBroker. |

|  |  |
| --- | --- |
|  |  |
| Figure 8: LDA for SJosephBurns. | **Figure 9:** LDA for SJosephBurns. |

|  |  |
| --- | --- |
|  |  |
| Figure 10: LDA for TheStalwart. | **Figure 11:** LDA for TheStalwart. |

## Sentiment Analysis

Tweets associated with financial analysts were measured for positive, negative, and neutral sentiments. Using the python Vader package[[28]](#footnote-28), tweets were found more neutral, then positive, and negative, respectively. Next, aggregated words by topic from LDA, were used for sentiment measures for each financial analyst[[29]](#footnote-29). Since 10 topic models were generated, each x-label corresponds to a computed topic, with three associated measures of sentiment varying along the y-axis. Like the twitter sentiments, the overall topic models have similar Vader sentiment measures – neutral, positive, and negative.

|  |  |
| --- | --- |
|  |  |
| Figure 12: twitter sentiment for JimCramer. | **Figure 13:** lda sentiment for JimCramer. |

|  |  |
| --- | --- |
|  |  |
| Figure 14: twitter sentiment for LizAnnSonders. | **Figure 15:** lda sentiment for LizAnnSonders. |

|  |  |
| --- | --- |
|  |  |
| Figure 16: twitter sentiment for ReformedBroker. | **Figure 17:** lda sentiment for ReformedBroker. |

|  |  |
| --- | --- |
|  |  |
| Figure 18: twitter sentiment for SJosephBurns. | **Figure 19:** lda sentiment for SJosephBurns. |

|  |  |
| --- | --- |
|  |  |
| Figure 20: twitter sentiment for TheStalwart. | **Figure 21:** lda sentiment for TheStalwart. |

# Analysis

A baseline exploration was conducted on the Nasdaq and VIX index, then expanded to other indices to determine – “Can market sentiment from financial analysts predict stock prices”? To begin, time series data consisting of stock index or volume (xi) needed to be converted to set of classes used as the target vector during classification. First, the upper threshold limit (l) was determined for stock index or volume:

 **(equation 6)**

Once the upper limit (l) is determined, the associated threshold(s) are computed:

 **(equation 7)**

Finally, a moving average is taken with several considerations. First, a lag parameter (L) is used to smooth the data, possibly eliminating trend. This value is alternatively known as a moving average (MA) or sliding window. To begin, the filter average (favg) and filter standard deviation (fstd) are computed for each threshold (T) on a given financial analyst (A):

 **(equation 8)**

 **(equation 9)**

A threshold parameter (T) defines the number of standard deviations above MA, which a point (Ai) is classified as a signal (sj)[[30]](#footnote-30).

 **(equation 10)**

 **(equation 11)**

Using equation 10, when Bj > Cj, the influence parameter (I) provides context whether past signals exceeding a z-score threshold, should be used to determine successive threshold values. Values range between [0, 1], where zero values have no influence on calculating successive threshold values, while a value of one is weighted with greater importance:



Conversely, when Bj <= Cj, the same values update using the current value:



The average and standard deviation filters from equation 8 and 9 are updated as follows:

 **(equation 12)**

Moreover, if no threshold is computed, a default behavior is applied. Specifically, stock prices or volume measures (xi) were binned into classes when individual values exceed a previous defined step, namely (up, down) 🡪 (0, 1). Thus, “up” was assigned if the current day Index Value was greater than the previous day; otherwise, assigned down:

 **(equation 13)**

In this study, an initial of TA = [0.5, 2] was utilized, rather than using the provided equation 6 and equation 7. However, not many signals exceed beyond two standard deviations. Therefore, the assignment was further reduced TA = [0.5]. Without loss of generality, an update function has been coded in python[[31]](#footnote-31), allowing the earlier system of equations to update the corresponding filters in order to better discriminate future signals against noise. Furthermore, the array of points classified as either signal or noise (see equation 13), becomes the target vector (y) to the vectorized tweets (X) from financial analysts (A):



 **(equation 14)**

Specifically, the term frequency-inverse document frequency (TFIDF) was applied to each set of tweets (xi) for a given financial analyst (A). Since the overall data distribution sometimes was unbalanced, two rules governed whether a given class (ci) has an insufficient amount of data. If the first condition is true, then each matching row from X, and target value (ci) from the target vector (y) are removed. Additionally, if the second rule is satisfied, then the first matching row from X, and associated target value (ci) from the target vector (y) is iteratively removed until the second rules becomes false:

|  |  |
| --- | --- |
| 1. |  |
| 2. |  |

Once completed, data was split into 80% train, and 20% test. Bernoulli Naïve Bayes (BNB), Multinomial Naïve Bayes (MNB), and Support Vector Machine (SVM) were trained equation 14.

Additionally, timeseries methodologies, including, ARIMA, and Long Short-Term Memory (LSTM) were applied on earlier Vader sentiment scores. Using the grangercausalitytests[[32]](#footnote-32) method, the causality test provides insight whether sentiment scores could forecast time series – stock Index or Volume measures. While the Granger test is ideal for the ARIMA model, autocorrelation (ACF) and partial autocorrelation (PACF) were not utilized in determining appropriate (p,q,d) components for stationarity. A future study may utilize both ACF and PACF to find optimal hyperparameters. However, a custom grid search[[33]](#footnote-33) method could provide an automated method, implicitly resolving stationarity.

## Baseline Results

Baseline analysis was conducted using the volatility index (VIX), serving as a benchmark for successive measurements. While results were computed for jimcramer, LizAnnSonders, and ReformedBroker, only results for LizAnnSonders[[34]](#footnote-34) was considered. This determination was largely made, since corresponding confusion matrices were only slightly less biased.

### Classification

Various classification algorithms were implemented, using a custom stop word list[[35]](#footnote-35), and standard TF-IDF implementation:

* Support Vector Machines (SVM)
* Bernoulli Naïve Bayes (BNB)
* Multinomial Naïve Bayes (MNB)

The classification task was repeated, with the remaining corpus being suffixed a part-of-speech (POS) tag before the TF-IDF. The intention of this approach was merely exploratory, since the resulting corpus increases sparsity:

* Support Vector Machines (SVM-POS)
* Bernoulli Naïve Bayes (BNB-POS)
* Multinomial Naïve Bayes (MNB-POS)

The daily VIX “Total Volume” was utilized with the peak z-score methodology[[36]](#footnote-36) (described above) to eliminate irrelevant data points. The remaining signal points were binned into classes based on a default threshold implementation (see equation 11 and equation 13):

|  |  |
| --- | --- |
| VIX Total Volume for LizAnnSonders | |
|  |  |
| Figure 22: overall classification results | **Figure 23:** z-score signal indicator |

However, the above classifiers only train if there are enough instances in the constructed dataframe, consisting of tweets in one column, with associated sentiment scores in separate columns. Specifically, the number of rows must satisfy the following constraint:



Furthermore, the dimensionality of tweets was reduced using the Porter stemmer[[37]](#footnote-37) and chi-squared[[38]](#footnote-38). Specifically, after applying the stemmer, followed by the TF-IDF vectorizer, the top 1000 words were selected using the chi-squared. The resulting corpus was trained against the provided target vector. The baseline results do not provide insight beyond a comparative benchmark. Furthermore, the below results were selected, primarily for being better balanced:

|  |  |  |
| --- | --- | --- |
| Classification: VIX Total Volume for LizAnnSonders | | |
|  |  |  |
| Figure 24: BNB | **Figure 25: B**NB (POS) | **Figure 26:** MNB |

To further check False-Positive with True-Positives, the precision, recall, and f-score are computed respectively. Specifically, precision was used to measure the ratio of correctly predicted positive labels against the entire positive labels.

 **(equation 15)**

Generally, high precision is related to low false positives[[39]](#footnote-39). Similarly, recall was calculated to measure the ratio of correctly positive labels against the entire labels for the given class.

 **(equation 16)**

Finally, the f-score combined the former scores to produce a harmonic mean[[40]](#footnote-40).

 **(equation 17)**

|  |  |  |
| --- | --- | --- |
| Check: VIX Total Volume for LizAnnSonders | | |
|  |  |  |
| Figure 27: BNB | **Figure 28:** BNB (POS) | **Figure 29:** MNB |

As shown in Figure 27 – Figure 29, BNB and BNB (POS) have the best combination of precision, recall and f-score. However, depending on a single execution may rely too much on chance. Thus, a k-fold[[41]](#footnote-41) implementation was utilized having n\_splits=5, repeated 750 times.

|  |  |  |
| --- | --- | --- |
| K-fold: VIX Total Volume for LizAnnSonders | | |
|  |  |  |
| Figure 30: BNB | **Figure 31: B**NB (POS) | **Figure 32:** MNB |

While the remaining k-fold model iterations did not exceedingly outperform the above combinations, performance of the remaining models can be reviewed separately[[42]](#footnote-42). The train and test distribution were reviewed for Figure 21 - Figure 23:

|  |  |  |
| --- | --- | --- |
| Train Distribution: VIX Total Volume for LizAnnSonders | | |
|  |  |  | |
| Figure 33: BNB | **Figure 34: B**NB (POS) | **Figure 35:** MNB | |

|  |  |  |
| --- | --- | --- |
| Test Distribution: VIX Total Volume for LizAnnSonders | | |
|  |  |  |
| Figure 36: BNB | **Figure 37:** BNB (POS) | **Figure 38:** MNB |

The train distributions generally show balance between the target classes, providing the model a balanced learning opportunity. However, both BNB and MNB test data, were of increasing unbalanced spread.

Additionally, the corpus utilized TF-IDF before train, with coding facilities to determine the top N weighted words used during classification. This functionality was available for all classifiers, as well as the MNB – capable of attaining word scores associated with negative, or positive sentiments. However, the selected words were often characterized by numerical values from the corpus. Moreover, the implemented chi-squared provided similar yet more general facilities to determine the top N chi-squared selected words:

|  |
| --- |
| Coding snippet: get\_top\_chi2() |
| self.wscores = pd.DataFrame(  list(zip(  self.get\_feature\_names(),  self.chi2.scores\_,  self.chi2.pvalues\_  )),  columns=['feature', 'score', 'pval']  )  self.wscores = self.wscores.sort\_values(  by='score',  ascending=False  ).head(top\_words) |
| Figure 39: brain/algorithm/text\_classifier.py[[43]](#footnote-43) |

Furthermore, the chi-squared was applied after the TF-IDF vectorizer, to reduce dimensionality to the top 1000 words:

|  |  |  |
| --- | --- | --- |
| Top 25 chi2: VIX Total Volume for LizAnnSonders | | |
|  |  |  |
| Figure 40: BNB | **Figure 41: B**NB (POS) | **Figure 42:** MNB |

Due to limitation of the port stemmer, the over-generalized words are more difficult to discern. Some commonalities between models (including those not shown above) indicate the following top stemmed words:

* jan
* twitterback
* 267B
* Uneasi
* wow
* 8662
* ack
* short

Overall, the top words for VIX are related to President Trump’s proposed $267B tariffs on all goods from China early September 2018[[44]](#footnote-44). While market drop follow, general threats on American Chinese economic tension caused general uneasiness in the global market. Counter tariffs largely impacted U.S. farm products the following year:

|  |
| --- |
| Farm products impacted by U.S. China trade war |
| Figure 43: Effects of Chinese Tariffs on U.S. Agriculture. |

Additional retaliatory threats include stopping export of rare earth materials to the U.S., which impacts the U.S. tech and manufacturing. Continued tension between the American Chinese economies has caused much of the U.S. stock market to be shorted in the past year.

### Time series

Time series models were created using the Vader sentiment scores derived from each financial analyst tweet corpus. Additionally, an overall timeseries model for a given stock index/volume was created, independent of any financial analyst. Trained models include the ARIMA and the LSTM neural network variant. These two different models were trained, with the intention of providing a level of comparison, like the variety of models utilized for classification. Moreover, the mean squared error (MSE) provides a basis of comparison between the two models.

In general, the ARIMA model is a regression methodology which includes three components (p,q,d) to help predict successive future values:

* AR (p): autoregression – previous N weighted time series value(s) regressed to predict current or future observation.
* I (q): integrated – differencing term to ensure constant mean and variance (stationary)
* MA (d): moving average – previous N weighted error terms averaged to predict current or future observation

This modeling technique often requires stationarity, to ensure predicted value(s) are relevant and not caused by trends or seasonal time dependencies. More generally, if stationarity is assumed, then the distribution is time independent, or roughly the same at the different times. This permits the use of statistical inferencing from the stochastic distribution[[45]](#footnote-45). To induce stationarity, the ACF and PACF plots can be used to determine optimal MA or AR terms, respectively. Some suggested rules can be reviewed in Appendix A below.

In this study, rather than utilizing the ACF or PACF, a grid-search implementation was deployed with the Dickey-Fuller test[[46]](#footnote-46). Specifically, the (p,q,d) parameters were allowed to vary with a range(0, 4). The null hypothesis assumes non-stationarity, while p <= 0.05 assumes the given combination is stationary. Furthermore, an associated mean squared error (MSE) was computed for each combination. Only significant models were given consideration, while the lowest MSE model for a given analyst/sentiment case was selected. This approach was also implemented for the general case – stock value/volume. Furthermore, each value in the time series was log transformed:



In general, the recommended minimum observations for the ARIMA model is between 50 – 100[[47]](#footnote-47). However, the size of the VIX dataset was sufficiently long[[48]](#footnote-48). This indicates the trained model characterizes reliable predictions for the general VIX volume:

|  |  |  |
| --- | --- | --- |
| ARIMA Distribution: VIX Total Volume | | |
|  |  |  |
| Figure 44: train (3,2,0) | **Figure 45:** test (3,2,0) | **Figure 46:** overall MSE |

**Note:** Figure 44 represents data after being log transformed. See Figure 59 for original scale.

Additionally, since the sentiment models aggregated data with the same join\_data[[49]](#footnote-49) method, the number of train and test distribution follow identically as Figure 33-38. Now, when training the vader sentiment scores for the corresponding twitter corpus:

|  |  |  |
| --- | --- | --- |
| ARIMA Train Distribution: LizAnnSonders Sentiment | | |
|  |  |  |
| Figure 47: negative (3,1,0) | **Figure 48:** neutral (3,2,0) | **Figure 49:** positive (0,2,0) |

The model still depicts moderate accuracy, with neutral and positive models performing best:

|  |  |  |
| --- | --- | --- |
| ARIMA Test Distribution: LizAnnSonders Sentiment | | |
|  |  |  |
| Figure 50: negative (3,1,0) | **Figure 51:** neutral (3,2,0) | **Figure 52:** positive (0,2,0) |

Similarly, the LSTM model was trained against both the corresponding VIX volume and Vader sentiment scores. Unlike the ARIMA implementation, there is no general guideline on the size of the train data. Often, the convergence of the neural network is a function of the data complexity. In general, the LSTM framework, a variant of the recurrent neural network (RNN), greatly differs from traditional regression models. Rather than requiring stationarity through time series differencing, LSTM focuses on persisting information through a gated system. This gated system was intended to solve the challenges of the vanishing gradient[[50]](#footnote-50), a characteristic of uncoiling N + 1 RNN layers, each consisting of a single tangent activation function.

|  |  |
| --- | --- |
|  |  |
| Figure 53: coiled RNN | **Figure 54:** uncoiled RNN cells |

In this study, since four cells were utilized with large number of epochs (initially 750, then increased to 1500), cell states are susceptible to converge to zero. As a result, the LSTM architecture is better suited, providing the three-gate alternative, instead of one[[51]](#footnote-51):

|  |  |
| --- | --- |
|  |  |
| Figure 55: coiled LSTM | **Figure 56:** uncoiled LSTM cells |

1. Forget gate: sigmoid function multiplies cell states (Ct-1) in the given input matrix zero (forget) or one to persist the given data
2. Input gate: sigmoid function decides what values will be updated
3. Tanh layer: creates new cell state vector (Ct)
4. Output gate: sigmoid function decides which cell state (squashed from neighboring tanh layer) to persist

|  |  |
| --- | --- |
|  |  |
| Figure 57: previous (Ct-1), w/updated cell state (Ct) | **Figure 58:** previous hidden state supplied into current cell |

In this study, each of the four cells consists of the same attributes:

* 50 units: number of neurons in each cell
* Sigmoid activation: output zero for negative values, else one for positive values
  + replaces the generalized tanh function
* Return sequences: return the hidden state to the next cell
* Dropout layer: reduce overfitting by ignoring 0-100% of neurons in each cell

For the base study, the dropout was fixed at zero, ignoring the potential for overfit. Once a baseline was known, the both the number of neurons and dropout was varied. In general, the more cells that are used in a stacked network, the greater the amount of data that is needed for train. Though, no specific rules exist, a deeper network is often more complicated, requiring both more train data, and greater dropout to reduce overfitting. Conversely, simpler datasets often suffice using 1-2 cells. In this study, rather than using a suggested 2 cell network, four cells were stacked, initially trained with 750 epochs, then increased.

The deployed LSTM is much like the earlier autoregressive ARIMA component. Specifically, the previous N=4 steps was used to predict the next M=1th position. While the LSTM class[[52]](#footnote-52) allowed the prediction of the next (M + x)th step, this was not formally explored. Moreover, each time series train data was scaled using MinMaxScaler[[53]](#footnote-53). The intention was to downscale potential large values, to better facilitate LSTM convergence.

The trained LSTM VIX volume model show significant performance, underperforming against the earlier ARIMA variant, as shown by corresponding MSE scores:

|  |  |  |
| --- | --- | --- |
| LSTM Distribution: VIX Total Volume | | |
|  |  |  |
| Figure 59: train (750 epochs) | **Figure 60:** test (750 epochs) | **Figure 61:** overall MSE |

The sentiment models still aggregated data with the same join\_data[[54]](#footnote-54) method:

|  |  |  |
| --- | --- | --- |
| LSTM Train Distribution: LizAnnSonders Sentiment | | |
|  |  |  |
| Figure 59: negative | **Figure 60:** neutral | **Figure 61:** positive |

While the trained neural network appears to perform moderately well:

|  |  |  |
| --- | --- | --- |
| LSTM Test Distribution: LizAnnSonders Sentiment | | |
|  |  |  |
| Figure 60: negative (750 epochs) | **Figure 61:** neutral (750 epochs) | **Figure 62:** positive (750 epochs) |

The associated MSE comparison indicates that ARIMA outperforms LSTM at 750 epochs:

|  |  |  |
| --- | --- | --- |
| ARIMA Sentiment MSE | | |
|  |  |  |
| Figure 63: negative | **Figure 64:** neutral | **Figure 65:** positive (750 epochs) |

|  |  |  |
| --- | --- | --- |
| LSTM Sentiment MSE | | |
|  |  |  |
| Figure 66: negative | **Figure 67:** neutral | **Figure 68:** positive (750 epochs) |

While the ARIMA outperformed the LSTM at low epochs, the neural network models were not bounded with the restriction of only creating models at a defined significance level. Differences will become more noticeable as the number of epochs is significantly increased, indicating a favor for neural network over traditional regression methods. Thus, it will later be seen that the benefits of the grid-search hyperparameter optimization will become less desirable, when compared against neural networks with higher epoch order.

### Granger Causality

The granger causality is a hypothesis testing construct helping to identify whether one time series can forecast another. While the premise involves stationarity, future studies should take better care ensuring the comparison of vader sentiment scores (negative, neutral, positive) and stock index/volume are each stationary. In the immediate successive section, it will be seen that the Granger Causality test will be the driving force, used to identify which time series models to consider, under the assumption that a given sentiment univariate series “granger causes” the associated paired stock index/volume.

Since earlier consideration assigned VIX/LizAnnSonders as the base case, various testing approaches with associated p-value are shown:

|  |  |  |  |
| --- | --- | --- | --- |
| P-Value: Negative Sentiment “Granger Causes” VIX Volume | | | |
|  |  |  |  |
| Figure 69: chi-test | **Figure 70:** f-test | **Figure 71:** params ftest | **Figure 72:** ratio test |

|  |  |  |  |
| --- | --- | --- | --- |
| P-Value: Neutral Sentiment “Granger Causes” VIX Volume | | | |
|  |  |  |  |
| Figure 73: chi-test | **Figure 74:** f-test | **Figure 75:** params ftest | **Figure 76:** ratio test |

|  |  |  |  |
| --- | --- | --- | --- |
| P-Value: Positive Sentiment “Granger Causes” VIX Volume | | | |
|  |  |  |  |
| Figure 77: chi-test | **Figure 78:** f-test | **Figure 79:** params ftest | **Figure 80:** ratio test |

At a 95% confidence, the negative LizAnnSonders sentiment scores can be said to “granger cause” the associated VIX index. In the next section, paired time series with significant granger scores (p <= 0.05), will decide which timeseries and classification models to consider.

## Select Results

### Granger Causality

### Classification

### Time series

Earlier, the dropout rate was fixed at zero for the base study. However, successive models varied this parameter, as well as the number of units (i.e. neurons) in each cell. The MSE was utilized again to determine the performance on the test data, providing the ability to compare the different models. Since various literature has found 0.4 as an optimal dropout (Srivastava N., 2014), successive LSTM models utilized this recommendation as well as 0.2[[55]](#footnote-55).

# Compute Benchmark

Exploratory, classification, and timeseries modeling performance was generally compared. Each modeling variation generally involved a select stock index:

* AAPL
* AMZN
* GOOGL
* MMT
* NFLX
* VX1
* NASDAQ
* MMM
* SPY
* QQQ
* RWTC\_D
* 15YFIXED\_IR
* 30YFIXED\_APR

Paired with a corpus of financial analyst tweets:

* Jimcramer
* ReformedBroker
* TheStalwart
* LizAnnSonders
* SJosephBurns

Each variation of the performance benchmark includes the time to complete the join\_data[[56]](#footnote-56) aggregation. Accounting the time to join each stock index against the matrix of financial analyst, can be considered a constant scaled factor across the different benchmarks:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Table 1: Compute Performance (min) | | | | | | | |
| Resources | **TS Stock** | | **TS Sent\*\*** | | **Class Sent\*\*** | **Granger\*\*** | |
| 3.0GHz (2 core), 8GB RAM | 42 | (p,q,d) = autoscale  epochs = 750  cells = 4  units = 50  dropout = 0 | ~13 | (p,q,d) = range(0,4)  epochs=750  cells = 4  units = 50  dropout = 0 | ~4 | ~3 | range(0,4) |
| p2.xlarge: 1GPU, 4vCPU 61GB RAM | 43 | (p,q,d) = autoscale  epochs = 3,000  cells = 4  units = 50  dropout = 0 | ~20 | (p,q,d) = range(0,4)  epochs = 3,000  cells = 4  units = 50  dropout = 0 |  |  | range(0,4) |
| p2.xlarge: 1GPU, 4vCPU 61GB RAM |  | (p,q,d) = autoscale  epochs = 1,500  cells = 4  units = 50  dropout = 0.2 |  | (p,q,d) = range(0,4)  epochs = 1,500  cells = 4  units = 50  dropout = 0.2 | N/A | N/A | range(0,4) |
| p2.8xlarge: 8GPU, 32 vCPU, 488 GB RAM |  | (pq,d) = autoscale  epochs = 2,000  cells = 4  units = 100  dropout = 0.4 |  | (p,q,d) = range(0,4)  epochs = 2,000  cells = 4  units = 100  dropout = 0.4 | N/A | N/A | range(0,4) |

**Note:** p2 architecture was implemented based on 2019 resource attributes[[57]](#footnote-57).

**Note:** \*\*scores were estimated per stock index/volume with all combinations of financial analyst Vader sentiment scores.

Furthermore, each modeling technique depicted in Table 1, was executed independent of other model(s). However, in the case of the granger, stock time series, and classification analysis, each stock code was executed manually one at a time. Except for the stock time series, each analysis joined against all variations of the financial analyst twitter corpus. The motivation of this approach was largely due to limitations of the codebase. Specifically, the original codebase iterated each quandl dataset, and joined each step against every combinations of financial analyst twitter corpus. While on a higher level, the iterative analysis appears to be some O((x1+x2+x3+x4)2) factor, realistically, each xnth factor decomposes further into a higher non-linear order. A simple depiction would be to allow x1 to represent the TS Sent analysis. This includes both ARIMA and LSTM modeling, which encompasses the earlier grid-search optimization, with high LSTM epoch order.

In general, future optimizations include better garbage collection, possibly applying existing tensorflow constructs[[58]](#footnote-58). Moreover, rather than running the codebase as a script, it could be converted to an application with distributed and scaled resources. This was the original intention, and already partially written with the flask app-factory[[59]](#footnote-59) in mind.

# Conclusions

# References

Srivastava N., H. G. (2014). Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn*, 1929-1958.

# Appendix A

To determine the optimal AR, or MA terms for a given (possibly differenced) series, first construct the corresponding ACF or PACF plots. The following rules serve as guidelines to determine ARMA terms[[60]](#footnote-60):

* **Rule 6:** If the PACF of the differenced series displays a sharp cutoff and/or the lag-1 autocorrelation is positive--i.e., if the series appears slightly "underdifferenced"--then consider adding an AR term to the model. The lag at which the PACF cuts off is the indicated number of AR terms.
* **Rule 7:** If the ACF of the differenced series displays a sharp cutoff and/or the lag-1 autocorrelation is negative--i.e., if the series appears slightly "overdifferenced"--then consider adding an MA term to the model. The lag at which the ACF cuts off is the indicated number of MA terms.
* **Rule 8:** It is possible for an AR term and an MA term to cancel each other's effects, so if a mixed AR-MA model seems to fit the data, also try a model with one fewer AR term and one fewer MA term--particularly if the parameter estimates in the original model require more than 10 iterations to converge.
* **Rule 9:** If there is a unit root in the AR part of the model--i.e., if the sum of the AR coefficients is almost exactly 1--you should reduce the number of AR terms by one and increase the order of differencing by one.
* **Rule 10:** If there is a unit root in the MA part of the model--i.e., if the sum of the MA coefficients is almost exactly 1--you should reduce the number of MA terms by one and reduce the order of differencing by one.

1. <https://www.investopedia.com/terms/v/volatility.asp> [↑](#footnote-ref-1)
2. <https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets> [↑](#footnote-ref-2)
3. <https://developer.twitter.com/en/apps> [↑](#footnote-ref-3)
4. <https://twython.readthedocs.io/en/latest/> [↑](#footnote-ref-4)
5. <https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets.html> [↑](#footnote-ref-5)
6. <https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline.html> [↑](#footnote-ref-6)
7. <https://github.com/jeff1evesque/ist-736-hw/tree/master/data> [↑](#footnote-ref-7)
8. <https://developer.twitter.com/en/docs/basics/rate-limiting.html> [↑](#footnote-ref-8)
9. <https://www.quandl.com/tools/python> [↑](#footnote-ref-9)
10. <https://github.com/jeff1evesque/ist-736/blob/9652d7aa79dc576ca5ad671effbb76362beaa72a/app.py#L227> [↑](#footnote-ref-10)
11. <https://en.wikipedia.org/wiki/Domain_of_a_function> [↑](#footnote-ref-11)
12. <https://www.mturk.com/worker/how-it-works> [↑](#footnote-ref-12)
13. <https://github.com/jeff1evesque/ist-736/tree/master/data/mturk/data_collection_template> [↑](#footnote-ref-13)
14. <https://github.com/jeff1evesque/ist-736/tree/master/data/mturk/data_collection_results> [↑](#footnote-ref-14)
15. <https://blog.mturk.com/tagged/data-collection> [↑](#footnote-ref-15)
16. <https://github.com/jeff1evesque/ist-736/blob/master/utility/stopwords.py> [↑](#footnote-ref-16)
17. <https://github.com/jeff1evesque/ist-736/blob/master/resources/topic-modelling-with-scikitlearn.pdf> [↑](#footnote-ref-17)
18. <https://developer.twitter.com/en/docs/tweets/search/overview/standard.html> [↑](#footnote-ref-18)
19. <https://twython.readthedocs.io/en/latest/> [↑](#footnote-ref-19)
20. <https://github.com/jeff1evesque/ist-736/blob/master/brain/algorithm/topic_model.py#L110-L117> [↑](#footnote-ref-20)
21. <https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html> [↑](#footnote-ref-21)
22. <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html> [↑](#footnote-ref-22)
23. <https://github.com/jeff1evesque/ist-736/blob/master/brain/algorithm/topic_model.py> [↑](#footnote-ref-23)
24. <https://github.com/jeff1evesque/ist-736/blob/master/resources/nlp_lecture_12-04-13.pdf> [↑](#footnote-ref-24)
25. <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html> [↑](#footnote-ref-25)
26. <https://github.com/jeff1evesque/ist-736/blob/master/resources/research_exam09.pdf> [↑](#footnote-ref-26)
27. <https://www.quandl.com/> [↑](#footnote-ref-27)
28. <https://pypi.org/project/vaderSentiment/> [↑](#footnote-ref-28)
29. <https://github.com/jeff1evesque/ist-736/blob/master/brain/controller/topic_model.py> [↑](#footnote-ref-29)
30. <https://stackoverflow.com/a/22640362> [↑](#footnote-ref-30)
31. <https://github.com/jeff1evesque/ist-736/blob/master/brain/algorithm/peak_detection.py> [↑](#footnote-ref-31)
32. <http://www.statsmodels.org/stable/generated/statsmodels.tsa.stattools.grangercausalitytests.html> [↑](#footnote-ref-32)
33. <https://github.com/jeff1evesque/ist-736/issues/66> [↑](#footnote-ref-33)
34. <https://github.com/jeff1evesque/ist-736/tree/master/viz/analysis/chris--cboe_vx1/LizAnnSonders> [↑](#footnote-ref-34)
35. <https://github.com/jeff1evesque/ist-736/blob/master/brain/utility/stopwords.py> [↑](#footnote-ref-35)
36. <https://github.com/jeff1evesque/ist-736/blob/master/brain/algorithm/peak_detection.py> [↑](#footnote-ref-36)
37. <http://www.nltk.org/howto/stem.html> [↑](#footnote-ref-37)
38. <https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html> [↑](#footnote-ref-38)
39. <https://en.wikipedia.org/wiki/Precision_and_recall> [↑](#footnote-ref-39)
40. <https://www.youtube.com/watch?v=Clo-t9eeEwg> [↑](#footnote-ref-40)
41. <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html> [↑](#footnote-ref-41)
42. <https://github.com/jeff1evesque/ist-736/tree/master/viz/classification/chris--cboe_vx1/LizAnnSonders> [↑](#footnote-ref-42)
43. <https://github.com/jeff1evesque/ist-736/blob/master/brain/algorithm/text_classifier.py> [↑](#footnote-ref-43)
44. <https://www.cnbc.com/2018/09/07/trump-says-tariffs-on-another-267-billion-in-china-goods-ready-to-go.html> [↑](#footnote-ref-44)
45. <https://www.stat.berkeley.edu/~arturof/Teaching/STAT248/lab05_part2.html> [↑](#footnote-ref-45)
46. <https://en.wikipedia.org/wiki/Dickey%E2%80%93Fuller_test> [↑](#footnote-ref-46)
47. <https://www.researchgate.net/post/What_should_be_the_minimum_number_of_observations_for_a_time_series_model> [↑](#footnote-ref-47)
48. <https://github.com/jeff1evesque/ist-736/blob/master/data/quandl/CBOE_VX1.csv> [↑](#footnote-ref-48)
49. <https://github.com/jeff1evesque/ist-736/blob/master/app/join_data.py> [↑](#footnote-ref-49)
50. <https://en.wikipedia.org/wiki/Vanishing_gradient_problem> [↑](#footnote-ref-50)
51. <http://colah.github.io/posts/2015-08-Understanding-LSTMs/> [↑](#footnote-ref-51)
52. <https://github.com/jeff1evesque/ist-736/blob/master/brain/algorithm/lstm.py> [↑](#footnote-ref-52)
53. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html> [↑](#footnote-ref-53)
54. <https://github.com/jeff1evesque/ist-736/blob/master/app/join_data.py> [↑](#footnote-ref-54)
55. <https://machinelearningmastery.com/use-dropout-lstm-networks-time-series-forecasting/> [↑](#footnote-ref-55)
56. <https://github.com/jeff1evesque/ist-736/blob/master/app/join_data.py> [↑](#footnote-ref-56)
57. <https://aws.amazon.com/ec2/instance-types/p2/> [↑](#footnote-ref-57)
58. <https://github.com/jeff1evesque/ist-736/issues/125> [↑](#footnote-ref-58)
59. <https://flask.palletsprojects.com/en/1.1.x/patterns/appfactories/> [↑](#footnote-ref-59)
60. <http://people.duke.edu/~rnau/411arim3.htm> [↑](#footnote-ref-60)