Stock Price Movement Prediction Using StockTwits and Topic-Sentiment Neural Network

Lee, Woo Jung
Department of Operations Research and Financial Engineering,
Princeton University
wool@princeton.edu

Supervised by

Professor Frank Fabozzi
Department of Operations Research and Financial Engineering,
Princeton University
fabozzi321@aol.com

Submitted in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Engineering
Department of Operations Research and Financial Engineering
Princeton University

June 2016

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Woo Jung Lee

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Woo Jung Lee

Contents

1 Introduction **10**

2 Literature Review **12**

2.1 Topic Modeling of Social Network 12

2.2 Modeling Topic and Sentiment Simultaneously 13

2.3 Use of Social Network Sentiment for Stock Price Prediction 16

2.4 Latent Dirichlet Allocation Parallelization 18

2.5 Stock Prediction Using Neural Network 18

3 Data **21**

3.1 StockTwits Data 21

3.2 Stock Price Data 21

3.3 Exploratory Analysis 22

3.4 Preprocessing 27

4 Methodology **31**

4.1 Model Overview 31

4.2 Topic-Sentiment Latent Dirichlet Allocation 31

4.2.1 Model 31

4.2.2 Collapsed Gibbs Sampling 35

4.2.3 Optimization 45

4.2.4 Parameters 51

4.3 Echo State Network 52

4.3.1 Overview 52

4.3.2 Model 53

4.3.3 Parameter Selection 55

4.3.4 Data 58

5 Results **59**

5.1 Topic and Sentiment Assignment 59

5.2 Stock Price Movement Prediction 61

6 Conclusion **65**

List of Figures

1 Tree kernel for a synthesized tweet: @Fernando this isnt a great day for playing the HARP! :) . . . 13
2 The generation process of the topic- sentiment mixture model . . . 14
3 The Hidden Markov Model to extract topic life cycles and sentiment dynamics . . . 15
4 Graphical Model Representation of TSLDA . . . 15
5 Comparison of prediction accuracy of up/down stock index on S&P 100 index for different training window sizes. . . . 17
6 Diagram outlining 3 phases of methodology and corresponding data sets 18
7 Model structures of feed-forward ANN and RNN . . . 19
8 Comprehensive bar chart for each month in (a) 2011 and (b) 2012 . . . 22
9 Comprehensive bar chart for each month between 2011/01/01 and 2013/12/31 22
10 Pie chart of stock mentions in (a) 2011 and (b) 2012 . . . 23
11 Pie chart of stock mentions between 2011/01/01 and 2013/12/31 . . . 24
12 Plots of user information for each user seen in tweets from 2011 . . . 25
13 Plots of user information for each user seen in tweets from 2012 . . . 25
14 Plots of user information for each user seen in tweets between 2011/01/01 and 2013/12/31 . . . 26
15 Plots of stock price in 2013 for top 4 most mentioned stocks on StockTwits 26
16 Part-of-speech tags returned by ark-tweet-nlp . . . 29
17 Flowchart representation of TSLDA-ESN model . . . 32
18 Echo State Network structure . . . 53
19 (a) Using $\boldsymbol{Y}^{target}$ instead of the true value $\boldsymbol{Y}$ to train ESN with feedback matrix (b) Generating outputs using ESN trained using teacher forcing . 55
20 Prediction of $GOOG and $BAC . . . 62
21 Prediction of $GE and $KO . . . 62
22 Prediction of $MSFT and $YHOO . . . 62
23 Prediction of $NFLX . . . 63

24 Heatmaps of topic-sentiment features of $GE 64

List of Tables

1 Top 10 most mentioned tickers for each time frame 23

2 List of most mentioned words for each category in 2013 24

3 List of most mentioned stocks, hashtags, and users 27

4 Notations of TSLDA model 33

5 $GOOG Price Prediction 59

6 $GE Price Prediction 59

7 $EURUSD Price Prediction 60

8 $KO Price Prediction 60

9 $BAC Price Prediction 60

10 $MSFT Price Prediction 61

11 Prediction Performance Comparison 63

12 Values of W^{out} for each financial feature 64

List of Algorithms

1 TSLDA Generative Process 34
2 TSLDA Collapsed Gibbs Sampling 45
3 Parallelized TSLDA Collapsed Gibbs Sampling 50
4 ESN algorithm 57

Abstract

In this thesis, StockTwits - a social network of investors and traders that has similar structures to Twitter - is analyzed to extract topics and sentiments about a certain stock during a fixed length of time using an Topic Sentiment Latent Dirichlet Allocation (TSLDA) and use the extracted topic-sentiment as features of a neural network to make a prediction of stock price movement. TSLDA is specifically studied in detail to develop a parallelization of the Collapsed Gibbs Sampling process.

1 Introduction

There are many ongoing researches in the field of machine learning that aim to analyze social network services such as Twitter to study the user dynamics. These user dynamics learned from an algorithm are very useful, and they can even be used to make inferences about stock price movement. There exist many text mining algorithms that use information sources such as Wall Street Journal[23] or Twitter[6][21] to make predictions on whether a given stock will move up or down in price.

With rapid advancement of text mining algorithms, researchers are able to study the characteristics of social network service users by studying trending topics, sentiments represented in users' statements, and opinions about a certain topic.[19] Combining these powerful text mining methods with social network service data allows one to infer stock price movements; Twitter became a popular choice of data source for such researchers, and there are many papers on the subject.[6][21][23] Similar approaches can be applied to data gathered from StockTwits[10], which is a social network system in which users share information about certain stocks. Each tweet is tagged with the name of stock that it is about, so we have a direct mapping from each tweet to a certain stock. By analyzing the content of the tweets, we can find out what the overall opinion of StockTwits users about certain stocks is, and hopefully use this information to infer the future values of stock price.

We aim to use tweet data from StockTwits to predict stock movements for the next day by combining two algorithms: Topic Sentiment Latent Dirichlet Allocation (TSLDA)[17], which is an extension of the famous Latent Dirichlet Allocation[5], and Echo State Network (ESN). TSLDA is a probabilistic model that uses latent variables to model topics and sentiments present in a group of documents comprised of sentences, which is comprised of words. LDA has been widely used to study social network services, specifically Twitter[8], due to its effectiveness in topic modeling. TSLDA shares similar characteristics as LDA, and therefore should be able to model topic and sentiments of a group of documents.

The topic and sentiment assignment of each sentence allows one to create a feature

matrix that holds information about the given corpus. These features are then used as inputs of ESN, which is a type of Recurrent Neural Network that has been gathering attention due to its ability to model and predict stock price to a great accuracy.

The overall model to be examined in this thesis involves two parts. First, the StockTwits corpus is translated into a lemmatized corpus to be used as an input for TSLDA. Then TSLDA infers the distributions of topics, sentiments, and words in the document. This information is then combined with other information about the specified stock, such as price history or moving average, and inputted into ESN, which ultimately calculates the predicted price of the stock on the next day. Each model is implemented and optimized using Python; specifically, the Collapsed Gibbs Sampling of TSLDA fitting process is extended to a parallelized version which allows much faster calculation. Because Gibbs Sampling is a very slow algorithm and the data to be manipulated is text data, TSLDA will be very slow and optimization is crucial.

2 Literature Review

2.1 Topic Modeling of Social Network

Topic modeling allows one to model topics that are inherent in a group of documents, which reveals information about trending topic, main subject of documents, and vocabularies that are part of a certain topic. When this is applied to social network "documents" such as StockTwits, Twitter, or even a simple online message board, one can study the dynamics of topics in each "document". When applied to a collection of tweets from StockTwits, topic modeling methods allow us to extract the key topics regarding stocks on a given frame of time. These topics are what many traders read about on StockTwits network, and these topics are likely to affect the dynamics of stock market, and we aim to predict stock price movement using these topics.

Many studies have been done on applying different kinds of topic models to Twitter, which is very similar to StockTwits. Hong [8] applies and compares different topic modeling approaches to a collection of tweets. He focuses on two algorithms: LDA and author-topic model. Author-topic model is an extension of LDA in which each document also has an attribute called 'author'. Different schemes of applying these models are proposed, and each model/scheme is compared with each other in terms of popular messages prediction, retweet prediction, and classifying users and topics into their own topical categories. Similar approaches can be applied to StockTwits data to extract top topics of users on a day and classify them according to their topics, which is the first part of our algorithm. However, different approaches could perform better in this data, so they should also be considered.

Another popular sub-field of text mining is sentiment analysis. When applied to social network service entries such as tweets, sentiment analysis algorithms compute how positive or negative each tweet is. Agarwal [1] analyzes Twitter data to predict sentiments for each tweet. Many types of preprocessing steps are done, such as emoticon polarity tagging, hashtag target tagging, and stopwords removal. Afterwards, words are tokenized and combined into a data structure that they call kernel trees. Each tweet is converted to this structure according to a set of rules; an example is shown in figure 1. To the

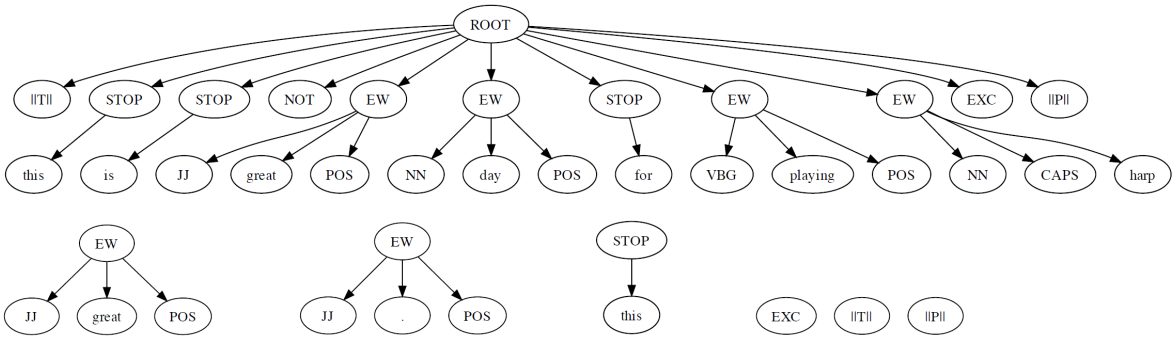


Figure 1: Tree kernel for a synthesized tweet: @Fernando this isnt a great day for playing the HARP! :)

ROOT node of a tweet, different tokens are attached: STOP (stopword), EW (English word), ||T|| (target user), ||P|| (positive emoticon) and NOT (n't) are examples of tokens in the case of tweet in figure 1. Then they use partial tree kernel to estimate similarity between two tweets in kernel tree structure. This allows them to study correlations between features to remove or add necessary features. They concluded that the following features return the best accuracy:

- # of negation, positive, negative words
- # of extremely-pos., extremely-neg., positive, negative emoticons
- # of pos./neg. hashtags, capitalized words, exclamation words
- sum of prior polarity scores of all words

Many of these works involving topic modeling of Twitter-style data has different types of preprocessing steps. Ones used in [1] can definitely be applied to StockTwits. Pang [19] surveys different approaches, problems, and different models in opinion mining and sentiment analysis.

2.2 Modeling Topic and Sentiment Simultaneously

There exist researches which attempt to model topic and sentiment of documents simultaneously, in one comprehensive model. Mei [14] proposes a model called Topic Sentiment

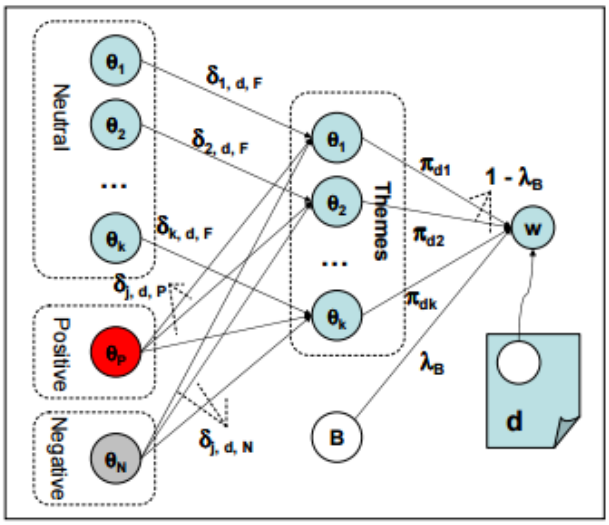


Figure 2: The generation process of the topic- sentiment mixture model

Mixture model. This model is based on topic modeling, sentiment analysis, and temporal topic analysis [15]. Figure 2 summarizes this algorithm; a word w in a document d is generated from a mixture model that combines themes $\theta_1, \cdots, \theta_k$ and base word B. Each theme θ_i is generated from a mixture model of neutral words of that topic, θ_i, positive words of that topic, θ_P, and negative words of that topic, θ_N. This model allows one to study sentiments on a certain topic of a set of documents. If trained on a set of StockTweets, it can model sentiments of users about certain event regarding a stock, which can be used as a proxy to predict stock prices.

Mei also studies what they call *topic life cycle* and *sentiment dynamics* by using Hidden Markov Model. Figure 3 illustrates HMM model that keeps track of dynamics of themed sentiment (a mix of topic and sentiment). T_i is theme i, and it can only transit to other theme T_j through state E. Each theme T_i is another HMM that models dynamics of words that move from base words to positive/sentiment to neutral words. Similar model can be applied in the case of StockTwits to estimate theme transitions to catch changes in sentiments about a certain stock, which could be useful in stock price movement prediction.

Topic Sentiment Latent Dirichlet Allocation (TSLDA) is proposed by Nguyen in [17]. It is an extension of LDA, except that it models topic and sentiment simultaneously.

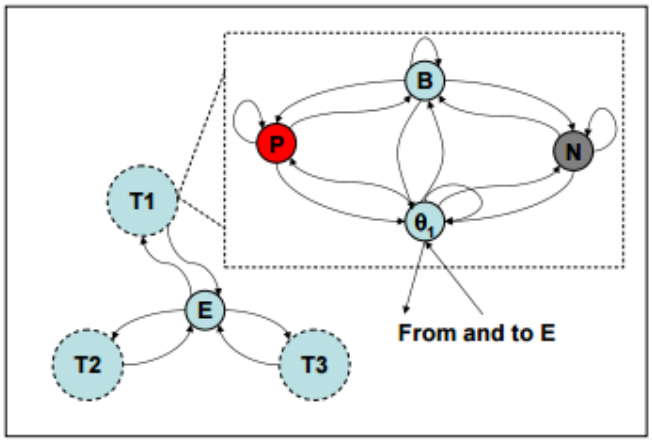


Figure 3: The Hidden Markov Model to extract topic life cycles and sentiment dynamics

Figure 4 shows a plate notation of generation process for a word w. c is a set of word categories: topic word, opinion word, others. Φ is a word distribution for base words, topic words, and opinion words. Each z^t, θ^t and z^o, θ^o is a per-document word distribution and topic distribution among documents, for topic and opinion words, respectively. This model assumes that each document has one topic and one opinion about it, which is ideal in our goal in aiming sentiments about events surrounding a stock. It was combined

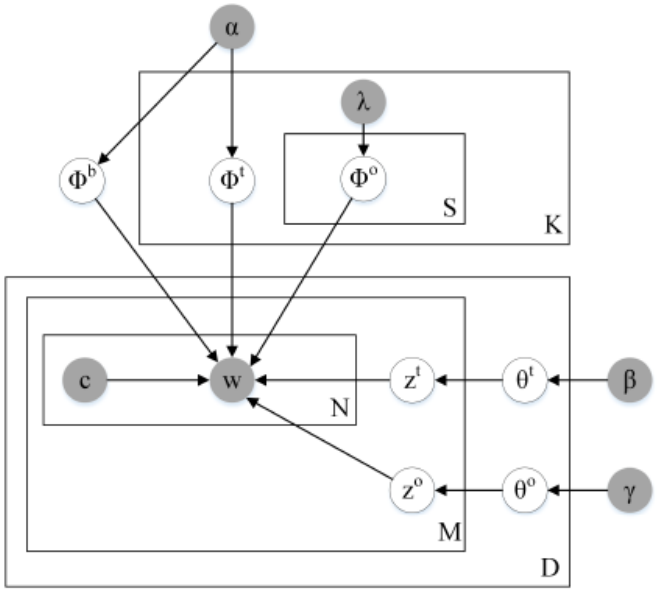


Figure 4: Graphical Model Representation of TSLDA

with Support Vector Machine to make binary stock price prediction (up/down). When compared to regular LDA and JST, which is a model that assigns multiple sentiments to a document, TSLDA performed better than both. These models were trained using Yahoo! finance data and a finance-related message board posts. TSLDA performed 6.07% better in prediction accuracy than using just historical prices and JST, and 6.43% better than regular LDA. Addition of sentiment seemed to have improved stock price movement prediction than simply just modeling topics.

2.3 Use of Social Network Sentiment for Stock Price Prediction

There are other works on using social network services and topic modeling to estimate a general sentiment and use it to predict stock price. Si [21] attempts to make stock price predictions using Twitter data. They pick the topics of each day using Dirichlet Process Mixture. Because DPM is a non-parametric method, we do not need to pre-set the number of topics, which may be hard to set. Afterwards, each word is distinguished into opinion word and non-opinion word. Using opinion word and topic distribution found from DPM, they infer sentiments of tweets about each topic using pre-trained opinion lexicon. Then the evaluated sentiment time series is fitted with S&P500 time series using time series regression. Their model predicts stock movement with accuracy of 68% on average. Figure 5 shows prediction accuracy of this model (cDPM) versus just using index values and AR model (Index) and using simple opinion-tagged lexicon (Raw), for different training window size and prediction lag. We can use similar approach and try to adjust our training window and lag length to find the most optimal set of arguments; some news may affect stock price immediately, while others might take longer time to affect the price.

Wong [23] proposes a new model that is able to leverage correlations a) among stock prices, b) among news articles, and c) between stock prices and news articles. This model is able to do this, while having a pretty low computational complexity and cost, by applying Alternating Direction Method of Multipliers algorithm in doing a sparse matrix factorization. It is able to predict and model more than 500 stocks simultaneously. It is trained and tested using Wall Street Journal articles to make stock price predictions; it

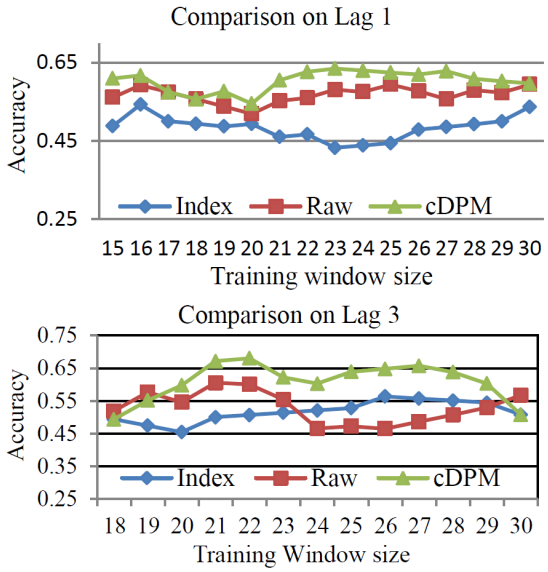


Figure 5: Comparison of prediction accuracy of up/down stock index on S&P 100 index for different training window sizes.

performs substantially better than many widely used algorithms. Although this paper does not attempt to apply a topic model to documents, it shows indication that sources of news, such as StockTwits, have predictability power in stock price movement.

A paper that popularized using social network services to predict financial time series is done by Bollen [6]. They used a set of tweets and topic modeling tools to study topics of tweets. The two models were OpinionFinder, which measures positive vs. negative mood, and Google-Profile of Mood States, which measures mood in terms of 6 dimensions (Calm, Alert, Sure, Vital, Kind, and Happy). Then these sentiments are used as inputs in a Self-Organizing Fuzzy Neural Network to make a prediction of DJIA closing values. They achieved prediction accuracy of 87.6%, which is very high in financial time series prediction using language models. Figure 6 illustrates the structure of this model. This online algorithm accepts tweets and DJIA index. By analyzing tweet feed, time series of opinion and mood are extracted. Relationship between time series of opinion and DJIA is studied using Granger causality test, which measures ability of a time series to predict another. Mood and DJIA time series are used as inputs of self organizing fuzzy neural network (SOFNN) to make prediction of DJIA movement. The F-statistic p-value returned by Granger test and mean absolute percentage error are used together to tune

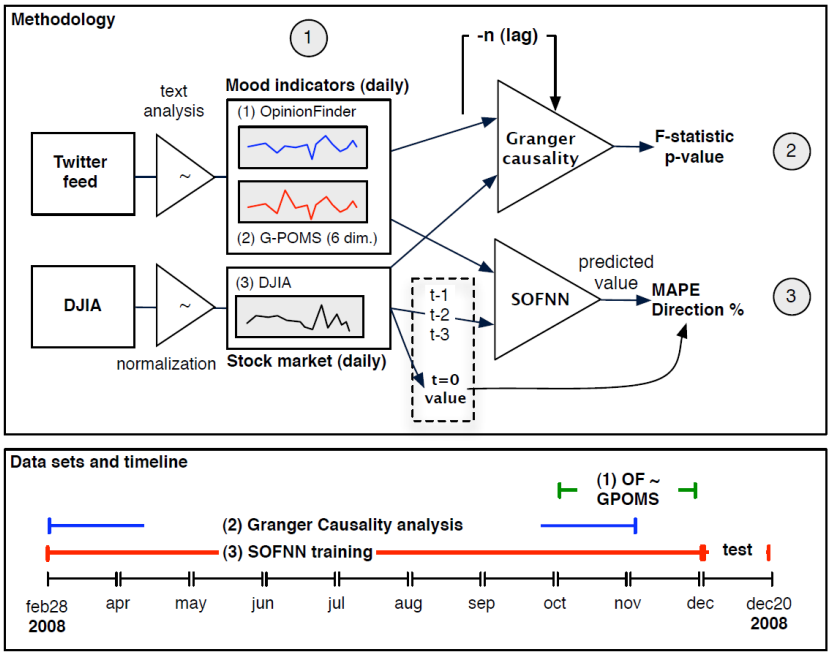


Figure 6: Diagram outlining 3 phases of methodology and corresponding data sets

and validate the model.

2.4 Latent Dirichlet Allocation Parallelization

Despite its simplicity and ability to capture topics fairly easily, LDA is usually computationally expensive because Gibbs Sampling is often used. In the case of TSLDA, a special type of Gibbs Sampling called Collapsed Gibbs Sampling is used. Although there are few papers discussing parallelization of LDA algorithm such as [24], [16] and [7], there is none for TSLDA. Its probabilistic structure is very similar to that of LDA, and thus it leads one to wonder if TSLDA can also be parallelized. Its derivation is explored in detail in section 4.

2.5 Stock Prediction Using Neural Network

With rapid advancement of neural network methods, many researchers are using neural network as a way of modeling financial data. In this thesis, we attempt to model the

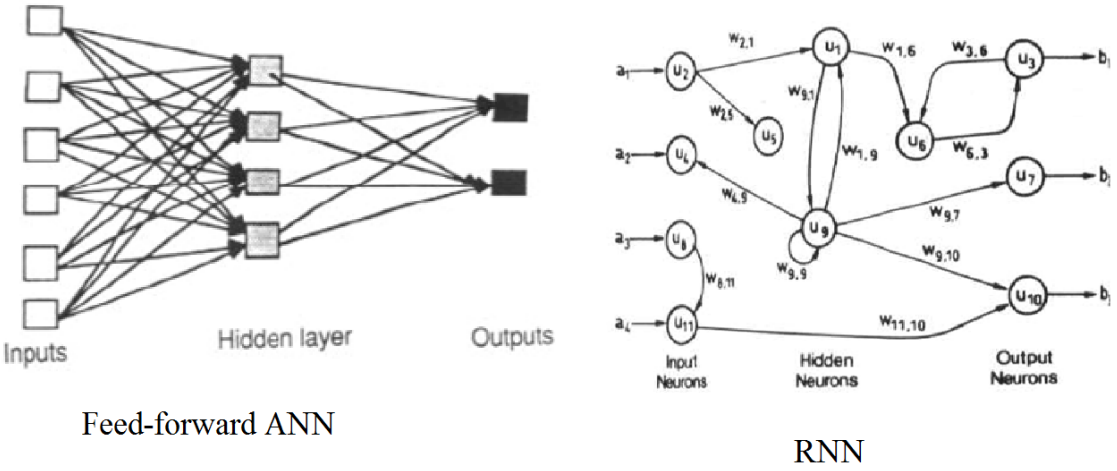


Figure 7: Model structures of feed-forward ANN and RNN

decision process of financial market. It takes data extracted from TSLDA, as discussed in section 4, then attempts to predict the market behavior on decisions of stock buys and sells. *Neural Networks in Finance Gaining Predictive Edge in the Market* written by McNelis [13] is a great literature that explains the procedures of applying a neural network for different types of data. The method sections that are relevant to this thesis are Recurrent Neural Network (RNN), Neural Network Smooth-Transition Regime Switching Models, and Neural Networks and Discrete Choice.

RNN is more suitable in modeling a complex structure of a system than feedforward Artificial Neural Network (ANN). ANN is most widely used neural network, but its hidden state structure does not allow backward movement in different layers. RNN is more flexible in that it allows hidden states in different layers to interact with each other. Figure 7 illustrates the different structures of feed-forward ANN and RNN. Financial market is a complex structure where many different feedbacks can occur, and therefore it is more suitable to fit a RNN than a feed-forward ANN.

A particular type of RNN called Echo State Network (ESN) is the model used to predict stock price using topic-sentiment information. Complex nature of RNN makes it a model that is hard to fit, despite its ability to capture wide range of behavior in data. However, ESN has relatively simple implementation and fitting process. ESN is developed in an emerging field called reservoir computing; in reservoir computing system,

the recurrent connections of the network are viewed as a fixed reservoir used to map inputs into a high dimensional, dynamical space. This is a similar approach to that of a support vector machine.

Bernal [4] uses ESN to predict prices of S&P 500. The features used as input are the current and 5-day history of the stock price, the 5, 10, 15, and 20 day moving averages, volume, and the S&P500 index. These variables will also be used in the model discussed in this thesis. ESN performed better than Kalman filtering method, predicting more of the higher frequency of fluctuations that Kalman filtering was not able to. Lukosevicius [12] provides a comprehensive guide to fitting an ESN along with its mathematical model and parameter fitting methods.

3 Data

3.1 StockTwits Data

The main dataset that will be used is a collection of .json files that have tweets on StockTwits from 2011/01 to 2015/08, a total of 56 months. It has been provided by Michael Lachanski '15 [10]. Our main focus is to make a future prediction of stock price movement using these tweets. For a given day t, we have $\mathcal{D}_{t_0,t_f}$, which is a collection of N_{t_0,t_f} tweets from t_0 to t_f. Each tweet $d_i \in \mathcal{D}_{t_0,t_f}$ is comprised of words w_{ij}, where $i \in \{1, \cdots, N_{t_0,t_f}\}$ and $j \in \{1, \cdots, N_i\}$, where N_i is length of tweet i.

Features other than the main body of a tweet in each json file are information of user who posted the tweet, and "entities", which is a collection of extra information provided by StockTwits. Entities include charts, relevant stock symbols, and sentiment. This sentiment feature is of our interest, as we aim to model opinions of users on certain stocks. Nevertheless, these sentiment estimates only exist for only a subset of tweets.

Data processing and model fitting are done using Python. Scikit-learn package [20] is used heavily as it contains many machine learning and data handling packages that can be used. The two main algorithms of this thesis, Topic Sentiment Latent Dirichlet Allocation (TSLDA) and Echo State Network (ESN), are also both implemented in Python.

3.2 Stock Price Data

Using Python allows us to take advantage of many packages that provide finance information. A module that uses Yahoo! finance database to retrieve data regarding a stock[3] is used to fetch stock price data. Closing, opening, high, and low historical prices of any stock can be accessed for each trading day. We can also retrieve volume of stocks on a given day, and this is used as a feature in ESN, which will be discussed later.

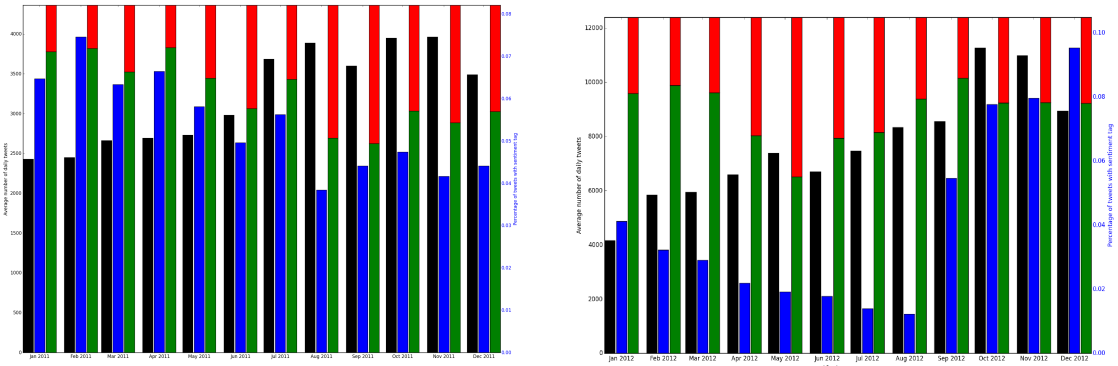
Because StockTwits data is stored in daily interval instead of trading day interval, mapping from a given day to its corresponding trading day must be calculated. This is done using the Yahoo! finance package. It returns stock prices of all trading day in an interval, and these dates can be used to map tweets to their correct trading days. For

each trading day, all tweets before 4pm, which is the time when the stock market closes, are mapped to that day, and all tweets after 4pm are mapped to the next trading day.

3.3 Exploratory Analysis

Before we apply anything to the data, we will explore the data by visualizing tweets so that we can get a sense of how StockTwits is structured. Three different plots are used to visualize a given set of tweets. We will present these plots generated from tweets from 2011, 2012, and 2011-2013.

First plot is a mixture of three bar charts. Black bar indicates total number of tweets, blue bar indicates % of tweets with sentiment tag, and green/red portions of last bar indicate Bullish/Bearish sentiment, among the tagged tweets. (All three bars are on



(a) (b)

Figure 8: Comprehensive bar chart for each month in (a) 2011 and (b) 2012

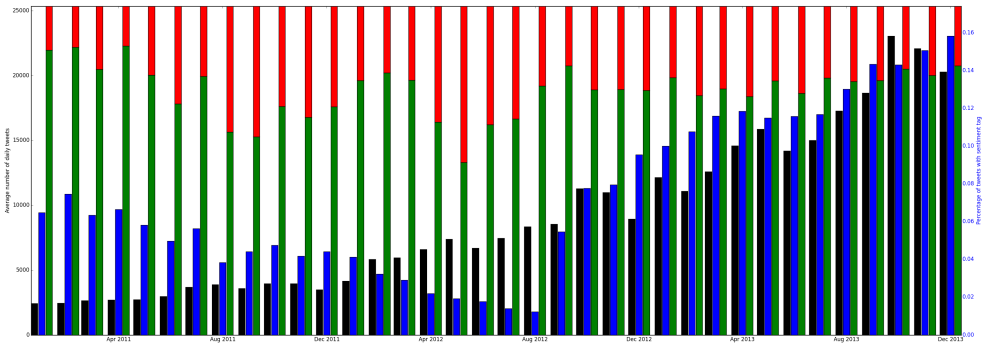


Figure 9: Comprehensive bar chart for each month between 2011/01/01 and 2013/12/31

Table 1: Top 10 most mentioned tickers for each time frame

2011	$spy, $aapl, $es_f, $eurusd, $spx, $cl, $slv, $nflx, $gld, $goog
2012	$aapl, $spy, $es_f, $vrng, $amrn, $arna, $eurusd, $fb, $spx, $goog
2011-2013	$aapl, $es_f, $eurusd, $spy, $spx, $nflx, $goog,

different scales) This is shown in figure 8 and figure 9. We see that the number of tweets steadily increase, as more users join StockTwits. The percentage of tweets tagged with sentiment slowly decreases, but we see a sudden peak around 3rd quarter of 2012. This surge in sentiment tags is also indicated in figure 9. The composition of Bearish/Bullish sentiments seem to also fluctuate up and down.

Second group of plots has pie charts showing the top ten most mentioned stocks in a given frame of time. Figure 10 and 11 show these pie charts. Table 1 summarizes top 10 most mentioned tickers for three different training time frame. As expected, popular stocks such as $aapl, $es_f, and $s&p500 are mentioned frequently.

The last set of plots is a collection of four plots that indicate characteristics of StockTwits users. A user's number of followers, number of users following, number of stocks following, and number of tweets are plotted on a same x-axis (user). They are shown in

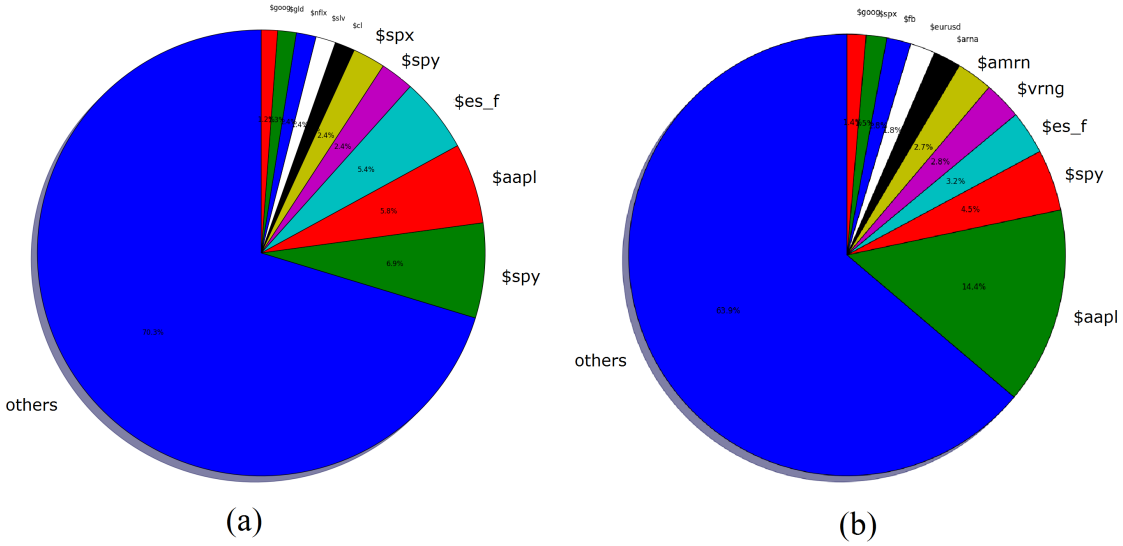


Figure 10: Pie chart of stock mentions in (a) 2011 and (b) 2012

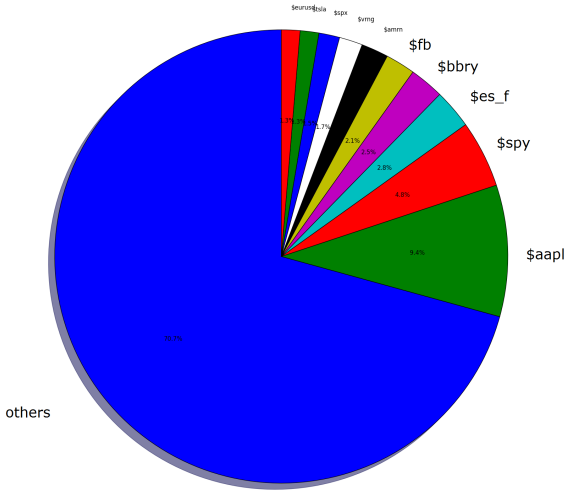


Figure 11: Pie chart of stock mentions between 2011/01/01 and 2013/12/31

figures 12, 13, and 14.

As discussed in later sections, TSLDA is an algorithm that is computationally very expensive. Therefore, No computer was able to handle memory requirement for processing all 5 years at once. Therefore, this thesis focuses on data from 2013. This year was selected by examining figure 9, which implies that data from 2013 is more balanced and clean, unlike data from 2012.

In 2013, there were 4,054,710 tweets and total of 4,261,736 sentences. That is about 1.05 sentences per tweet; the highest number of sentences in a tweet was 13. There were total of 1,017,050 words; these vocabularies are divided into three not necessarily disjoint categories: background, topic, and sentiment words. 911,460 were background words, 172,550 topic words, and 12,621 sentiment words. Categorization methods are discussed in section 4. Table 2 shows the top 20 most frequent words for each category. We can see

Table 2: List of most mentioned words for each category in 2013

Top 20 words with $c = 0$	Top 20 words with $c = 1$	Top 20 words with $c = 2$
. , be the to @ a of and ” in on for this ... i at it - ?	$aapl $spy today $bbry high stock market price $es_f $fb earnings $spx day chart report $tsla $gld call week	like good nice break well right lol great bullish strong bad :) wow top love miss beneficial work lose bearish

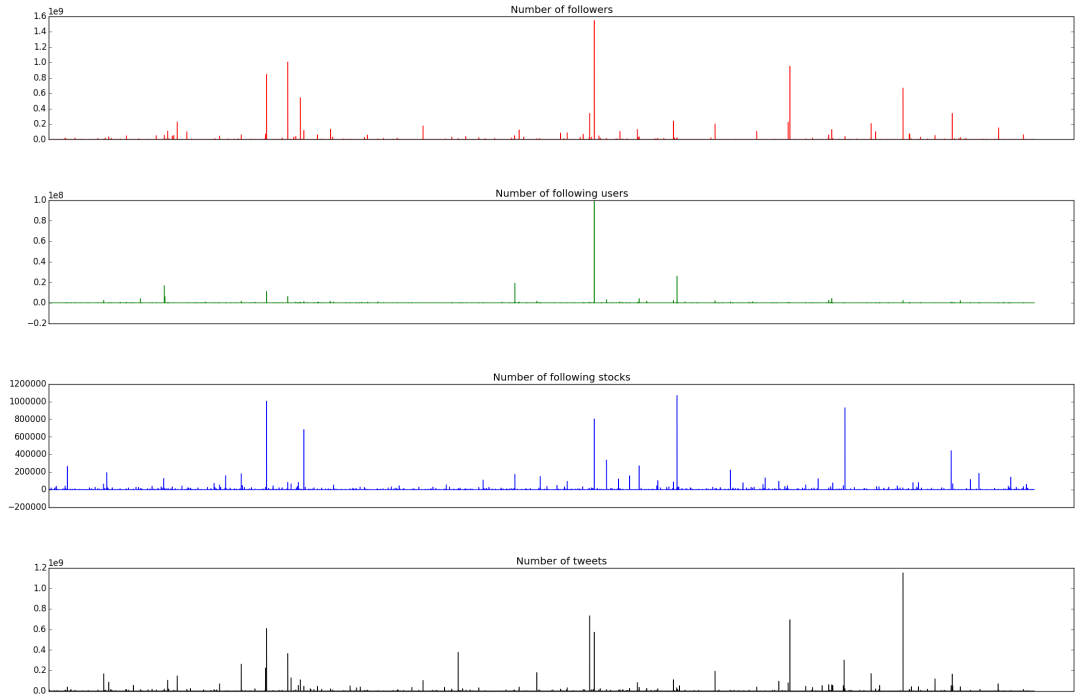


Figure 12: Plots of user information for each user seen in tweets from 2011

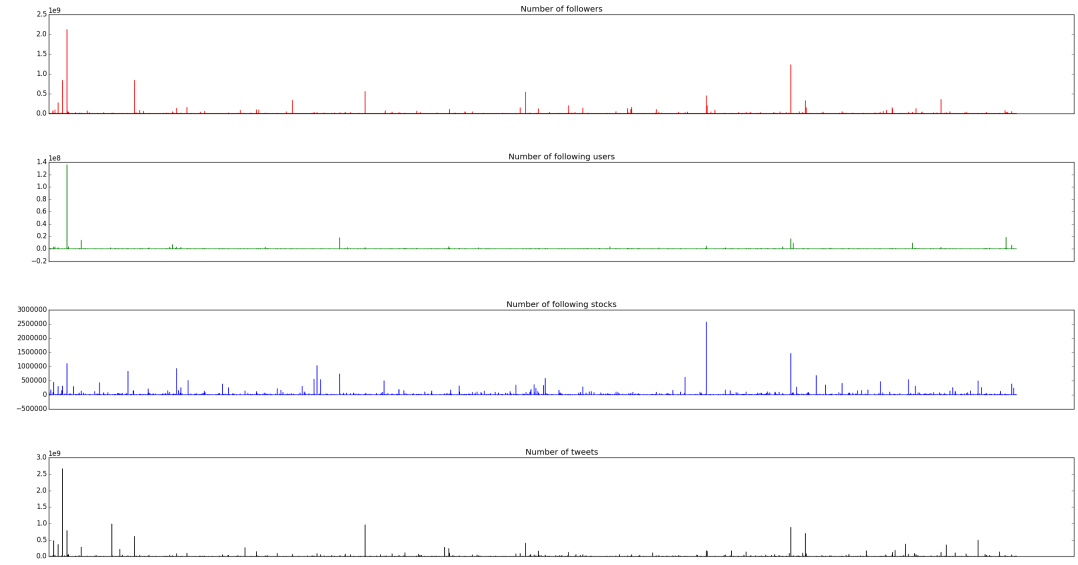


Figure 13: Plots of user information for each user seen in tweets from 2012

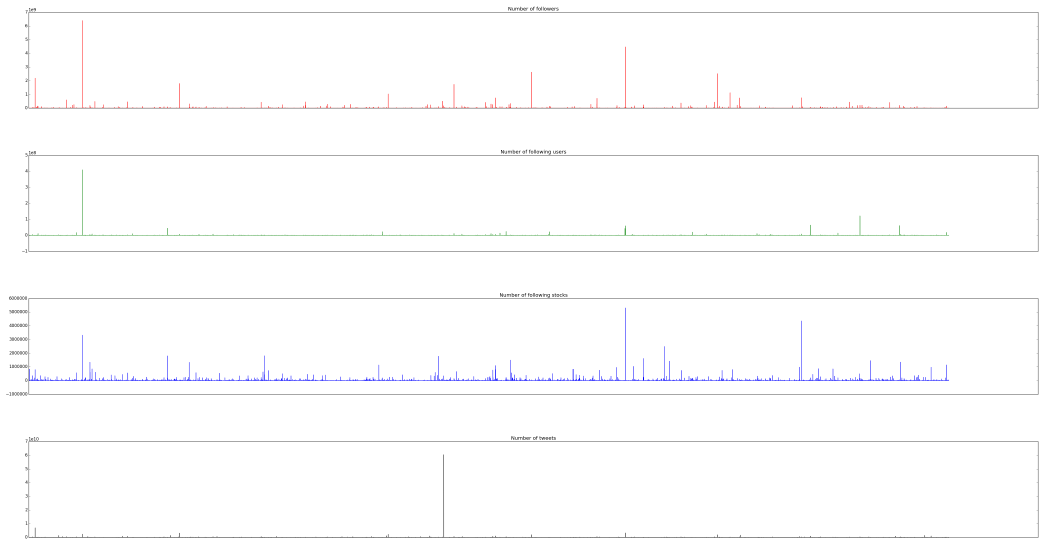


Figure 14: Plots of user information for each user seen in tweets between 2011/01/01 and 2013/12/31

that words with no information such as "the" or punctuation marks are classified into background words, while words that have positive or negative connotations are classified together. The topic words are noun words that occur in succession. The top 4 most mentioned stocks in 2013 were $aapl, $bbry, $spy, and $fb, as shown in table 3. Their

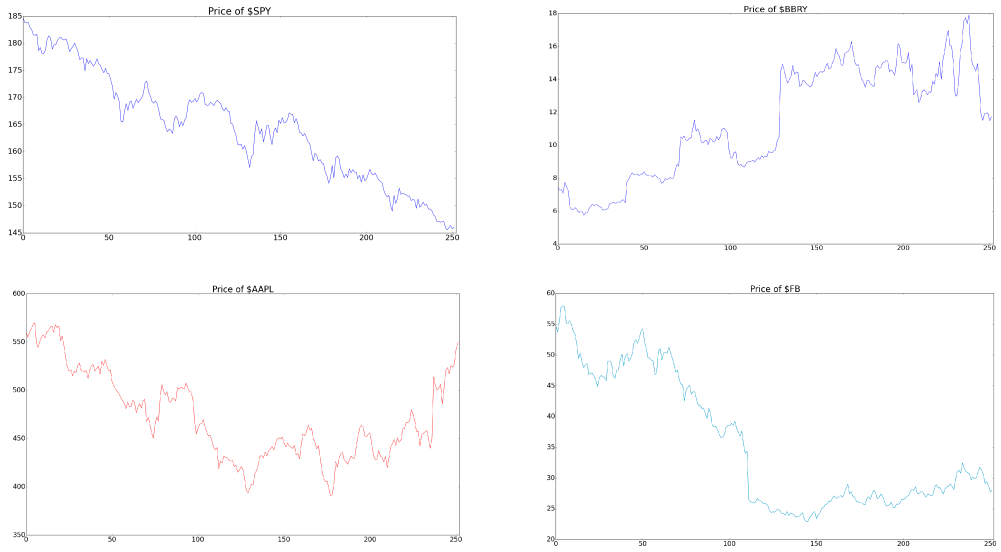


Figure 15: Plots of stock price in 2013 for top 4 most mentioned stocks on StockTwits

Table 3: List of most mentioned stocks, hashtags, and users

Top 20 stocks	Top 20 hashtags	Top 20 users referred
$aapl $bbry $spy $fb $tsla $amrn $vrng $spx $goog $gld $nflx $nugt $uvxy $eurusd $nq $amd $study $twtr $jcp $amzn	#stocks #trading #forex #1 #elliottwave #s #gold #futures #highyield #alsalamanty #2 #spcapiq #wti #mkt #market #p #trader #fx #es_f #spx	@insiderbuysuperstocks @technicaloptiontrader @mastertrader_consultant @iamastockgeniusjustaskme @firstcorinthiansfifteen @flowerpoweredqe2infinity @conviction_trader2012 @blackholesblackscholes @rickjamesstockoperator @theothersideofthetrade @optionsriskmanagement @superstocktradesmadeeasy @riskybusinesshighprofits @analystratingsnetwork @dtrainer_newconstructs @flavisam_letstradeshares @globalmacrostrategist @warren_theoracle_buffett @tradeagainstthemachines @intelligent_speculator

stock prices are plotted in figure 15. Hashtags in the dataset are mostly about financial instruments such as stocks, futures, or forex. StockTwits users refer to each other very frequently when they talk about a certain stock, and therefore there are many user references that needed to be handled.

3.4 Preprocessing

TSLDA is a topic modeling algorithm, which is a type of natural language processing, which implies that preprocessing is crucial in obtaining a nicely fitted model. There are many preprocessing methods that are applied to text data when probabilistic model is applied to it. The main steps that are used in this thesis are sentence division, part-of-speech tagging, categorizing, and lemmatizing each word.

First preprocessing applied to tweets is removing repeated characters that happen more than 3 times in a row. This is a widely used method in natural language processing, because there are not that many words that have three consecutive characters in a

row; however, it eliminates many strings that could cause problems, such as "!!!!!!!!!!" or "hellooooo". Although removing consecutive characters do not return a correct word, it creates a word in which all words of same structure get mapped to. This is a similar idea to regular expression word stemming, in which regular expression is used to find the stem representation of same but different forms of words; for instance, 'running' and 'run' both get mapped to a certain word determined according to type of text data, such as 'run' or 'ru'.

After reducing lengths of repeated characters, each tweet is divided into sentences. This is done by using nltk[11] module that is available for Python. It builds a model of abbreviation words, collocations, and sentence-starting words using unsupervised algorithm. It is first trained on a corpus of English words, which is also provided by nltk.

The next step is one of the two important steps of preprocessing. For each sentence, we use ark-tweet-nlp, which is described in detail in [18]. This is a natural language processing program that specializes in pos-tagging twitter-style texts. Because tweets are restricted by number of characters, they are usually not in complete sentences. Furthermore, many other internet specific words, such as emoticons, are used frequently and disrupt the grammatical structure that other pos-tagging algorithms try to estimate. Ark-tweet-nlp was specially developed for pos-tagging tweets. It includes part-of-speeches for special words such as emoticons, interjections or URLs, which makes it an appealing choice for pos-tagging StockTwits data. Another advantage of ark-tweet-nlp is that it is able to distinguish and tag words that are not English, such as meaningless collection of characters or a number. Figure 16 shows all the tags returned by ark-tweet-nlp.

Once part-of-speeches of words are estimated, we are able to use them to categorize each word. The category assignment is represented by $c_{d,m,n}$ for nth word of sentence m in document d. Because topics are usually represented by noun words, consecutive noun words are assigned as topic words. ($c = 1$) If a word is not topic word and is one of opinion words in SentiWordNet, [2] it is a sentiment word. ($c = 2$) All other words are background words. ($c = 0$)

To have a one-to-one mapping from different representations of a word to its root

N	common noun
O	pronoun (personal/WH; not possessive)
^	proper noun
S	nominal + possessive
Z	proper noun + possessive
V	verb including copula, auxiliaries
L	nominal + verbal (e.g. *i'm*), verbal + nominal (*let's*)
M	proper noun + verbal
A	adjective
R	adverb
!	interjection
D	determiner
P	pre- or postposition, or subordinating conjunction
&	coordinating conjunction
T	verb particle
X	existential *there*, predeterminers
Y	X + verbal
#	hashtag (indicates topic/category for tweet)
@	at-mention (indicates a user as a recipient of a tweet)
~	discourse marker, indications of continuation across multiple tweets
U	URL or email address
E	emoticon
$	numeral
,	punctuation
G	other abbreviations, foreign words, possessive endings, symbols, garbage

Figure 16: Part-of-speech tags returned by ark-tweet-nlp

word, each word is lemmatized using WordNetLemmatizer. [22] When provided with a word and its part-of-speech, WordNetLemmatizer returns the estimated lemmatized form of the input word. This is the other crucial step of preprocessing, because otherwise words of same meaning can be recognized as two different words.

For each word, its part-of-speech and category must be stored. Therefore, a new data structure is created to store the word itself, its part-of-speech, and its category. This

allows for simple and fast storage and access of each attributes of a word.

Because we are interested in estimating the topic and sentiment about a certain stock from text data, we only need to look at tweets that directly mention the stock. Due to the shortness of tweets, most tweets that do not talk about the input stock have very little relationships to the topic and sentiment assignment that we are looking for. Therefore, from the set of lemmatized tweets, only tweets that talk about the stock of interest are used in analysis for that stock.

4 Methodology

4.1 Model Overview

We are interested in using corpus of tweets from StockTwits to extract topics and sentiments about these topics to make a prediction of stock price movement. Traders do this naturally when they collect information about a certain stock; when new information about a company is released, traders are able to use this information to guess whether the stock price of the company will increase or decrease depending on the positivity or negativity of the news.

This procedure is modeled using two models: Topic-Sentiment Latent Dirichlet Allocation (TSLDA) and Echo State Network (ESN). TSLDA is an extension of Latent Dirichlet Allocation (LDA), which is a generative probabilistic model that extracts topics from a corpus of documents. Unlike LDA, TSLDA is able to extract topics and sentiments for each topic simultaneously. Therefore, when tweets about a certain stock are given as input to this model, major topics and sentiments about them portrayed in the tweets can be extracted. This output can then be supplied as input to train ESN, a type of neural network, to fit a network that learns the non-linear relationship between the topic-sentiment assignments and stock price. Figure 17 summarizes the TSLDA-ESN model.

4.2 Topic-Sentiment Latent Dirichlet Allocation

4.2.1 Model

TSLDA is an extension of LDA formulated by Nguyen and Shirai [17] that is able to model sentiments along with topics. It takes corpus of documents of lemmatized words as input, and finds parameters that are probability distributions of topics, sentiments, and words. Using the given input and Collapsed Gibbs Sampling, model parameters are estimated in Bayesian fashion.

The input corpus is composed of D documents, with each document d having m_d sentences. We assume that each sentence has one topic $k \in \{1, \cdots, K\}$ and one sentiment

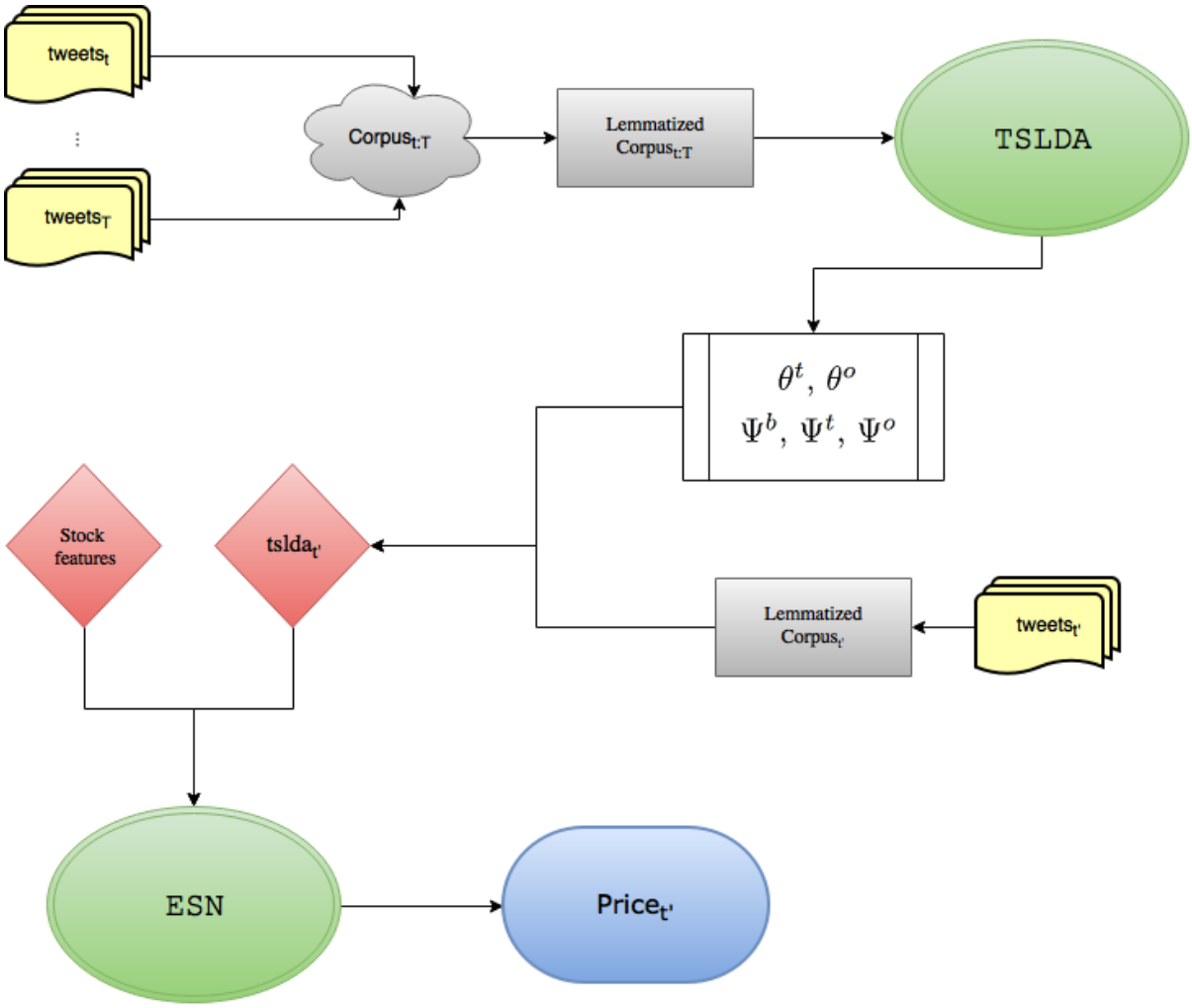


Figure 17: Flowchart representation of TSLDA-ESN model

$s \in \{1, \cdots, S\}$; we set $S = 3$ to represent positive, negative, and neutral sentiment. For each pair (d, m) that represents sentence m in document d, $z^t_{d,m}$ and $z^o_{d,m}$ are topic and sentiment of the sentence, respectively. Furthermore, each word is categorized into 3 categories: background word, topic word, or sentiment word. This category assignment is represented by $c_{d,m,n}$ for nth word of sentence m in document d. Because topics are usually represented by noun words, consecutive noun words are assigned as topic words. $(c = 1)$ If a word is not topic word and is one of opinion words in SentiWordNet, [2] it is a sentiment word. $(c = 2)$ All other words are background words. $(c = 0)$ To have a one-to-one mapping from given word to actual word, the input words must be lemmatized. As discussed in Chapter 3, each tweet is pos-tagged using ark-tweet-nlp,

Table 4: Notations of TSLDA model

Notation	Definition
α, β, γ, λ	Dirichlet distribution priors
K	number of topics
S	number of sentiments
Φ^b	background words distribution
Φ^t	topic words distribution
Φ^o	sentiment words distribution
D	number of documents
m_d	number of sentences in document d
$N_{d,m}$	number of words in sentence m in document d
V	set of all words
θ^t	topic distribution
θ^o	sentiment distribution
$z^t_{d,m}$	topic of sentence m in document d
$z^o_{d,m}$	sentiment of sentence m in document d
$w_{d,m,n}$	nth word of sentence m in document d
$c_{d,m,n}$	category of $w_{d,m,n}$

then using these part of speeches, all words are lemmatized into their corresponding base words. This allows us to consider words of different forms as the same word. For instance, the word 'running' and 'run' are considered as same word instead of two different words. All notations used in this model are listed in Table 4.

The difference between TSLDA and LDA is that TSLDA fits multiple 'sentiments' for each topic; we use the term 'sentiment' but the fitting process is very similar to that of topics. In a way, we can call it a sub-topic Latent Dirichlet Allocation because for each topic k, S sub-topics (sentiments) are fitted. This structure is crucial because some sentiment words could be positive and negative depending on the context. For instance, the word 'low' is positive when used in phrase 'low cost', but it is negative when used in phrase 'low salary'.

TSLDA is a generative process with observed variables α, β, γ, λ, $\mathbf{w}$ and $\mathbf{c}$, and hidden variables $\boldsymbol{z^t}$, $\boldsymbol{z^o}$, θ^t, θ^o, Φ^b, Φ^t and Φ^o. They are represented as black and white nodes in figure 4, respectively. The generative process is shown below.

Algorithm 1 TSLDA Generative Process

Require: α, β, γ, λ, $\boldsymbol{w}$, $\boldsymbol{c}$

Choose $\Phi^b \sim Dirichlet(\alpha)$
for $k \in (1, K)$ **do**
  Choose $\Phi^t_k \sim Dirichlet(\alpha)$
  for $s \in (1, S)$ **do**
    Choose $\Phi^o_{k,s} \sim Dirichlet(\lambda)$
  end for
end for
for $d \in (1, D)$ **do**
  Choose $\theta^t_d \sim Dirichlet(\beta)$
  Choose $\theta^o_d \sim Dirichlet(\gamma)$
  for $m \in (1, d_m)$ **do**
    Choose $z^t_{d,m} \sim Multinomial(\theta^t_d)$
    Choose $z^o_{d,m} \sim Multinomial(\theta^o_d)$
    for $w_{d,m,n} \in V_{d,m}$ **do**
      Choose $w_{d,m,n}$: $w_{d,m,n} \sim \begin{cases} Multinomial(\Phi^b) & \text{if } c_{d,m,n} = 0 \\ Multinomial(\Phi^t_{z^t_{d,m}}) & \text{if } c_{d,m,n} = 1 \\ Multinomial(\Phi^o_{z^t_{d,m}, z^o_{d,m}}) & \text{if } c_{d,m,n} = 2 \end{cases}$
    end for
  end for
end for

4.2.2 Collapsed Gibbs Sampling

Variables of our interest are the word and topic/sentiment distributions, θ^t, θ^o, Φ^b, Φ^t, and Φ^o. Because they are multinomial distributions over $\boldsymbol{z}^t$, $\boldsymbol{z}^o$ and $\boldsymbol{w}$, (along with $\boldsymbol{c}$) the aforementioned variables can be found by calculating proportions of $\boldsymbol{z}^t$, $\boldsymbol{z}^o$ and $\boldsymbol{w}$. Furthermore, we only need to find $\boldsymbol{z}^t$ and $\boldsymbol{z}^o$ as $\boldsymbol{w}$ and $\boldsymbol{c}$ are given. To estimate these hidden variables, collapsed Gibbs Sampling is used to find the conditional distribution of $\boldsymbol{z}^t$ and $\boldsymbol{z}^o$.

Gibbs Sampling is widely used in Bayesian inference to obtain a sequence of observations because it can be applied to problems in which the joint probability distribution of our interest is intractable. Collapsed Gibbs Sampling (CGS) is a type of Gibbs Sampling in which some variables are marginalized over (collapsed) to sample from simpler distribution. CGS is efficient because the marginalization of variables greatly reduce variance; it is also useful in context of TSLDA because we are able to generate simulated values of $\boldsymbol{z}^t$ and $\boldsymbol{z}^o$ after marginalizing over other hidden variables. More notations must be introduced to examine this sampling procedure.

Let $Z_d^{k,s}$ denote the total number of sentences in document d that has topic k and sentiment s; let $W_{d,m,v,c}^{k,s}$ denote the total number of times the word v with category c appears in sentence m in document d, where sentence m has topic k and sentiment s. For these two values, if * is placed instead of an index, it denotes the sum of values summed over that index. For instance, $Z_d^{k,*}$ denotes the total number of sentences in document d that has topic k; $W_{*,*,v,c}^{k,s}$ denotes the total number of times the word v with category c appears in all sentences that have topic k and sentiment s. Similarly, $\hat{}$ is used to represent a vector of values over the specified index. For example, $W_{*,*,v,1}^{\wedge,s} = (W_{*,*,v,1}^{1,s}, \cdots, W_{*,*,v,1}^{k,s})$. Another notation is addition of $-(d,m)$, which simply means excluding sentence m in document d. For example, $W_{*,*,v,2}^{k,*-(d,m)}$ denotes the total number of times the word v with category 2 appears in all sentences that have topic k, excluding sentence m in document d.

We are interested in finding the value of $p(z_{d,m}^t = a, z_{d,m}^o = b | \boldsymbol{z}_{-(\boldsymbol{d,m})}^{\boldsymbol{t}}, \boldsymbol{z}_{-(\boldsymbol{d,m})}^{\boldsymbol{o}}, \boldsymbol{w}, \boldsymbol{c}, \alpha, \beta, \gamma, \lambda)$. We will repeatedly use the fact that values that do not depend on d, m, a and

b are constant and can be removed to get expression that is proportional to the previous expression. Because we can normalize the posterior after all calculation is done without changing the probability structure, we are only concerned with proportionality instead of equality. Bayes rule can be applied to simplify these distributions:

$$\begin{aligned}
&p(z^t_{d,m}, z^o_{d,m}|\boldsymbol{z^t_{-(d,m)}}, \boldsymbol{z^o_{-(d,m)}}, \boldsymbol{w}, \boldsymbol{c}, \alpha, \beta, \gamma, \lambda) \\
=&\frac{p(z^t_{d,m}, z^o_{d,m}, \boldsymbol{z^t_{-(d,m)}}, \boldsymbol{z^o_{-(d,m)}}, \boldsymbol{w}, \boldsymbol{c}|\alpha, \beta, \gamma, \lambda)}{p(\boldsymbol{z^t_{-(d,m)}}, \boldsymbol{z^o_{-(d,m)}}, \boldsymbol{w}, \boldsymbol{c}|\alpha, \beta, \gamma, \lambda)} \\
=&\frac{p(\boldsymbol{z^t_{d,m}}, \boldsymbol{z^o_{d,m}}, \boldsymbol{w}, \boldsymbol{c}|\alpha, \beta, \gamma, \lambda)}{p(\boldsymbol{z^t_{-(d,m)}}, \boldsymbol{z^o_{-(d,m)}}, \boldsymbol{w}, \boldsymbol{c}|\alpha, \beta, \gamma, \lambda)} \\
\propto& p(\boldsymbol{z^t}, \boldsymbol{z^o}, \boldsymbol{w}, \boldsymbol{c}|\alpha, \beta, \gamma, \lambda)
\end{aligned}$$

This is joint distribution of $\boldsymbol{z^t}$ and $\boldsymbol{z^o}$ marginalized over other hidden variables. It can be expressed as:

$$\begin{aligned}
&\int\int\int\int\int p(\boldsymbol{z^t}, \boldsymbol{z^o}, \boldsymbol{w}, \boldsymbol{c}, \theta^t, \theta^o, \Phi^b, \Phi^t, \Phi^o|\alpha, \beta, \gamma, \lambda) d\theta^t d\theta^o d\Phi^b d\Phi^t d\Phi^o \\
=&\int\int\int\int\int p(\boldsymbol{z^t}, \theta^t|\alpha, \beta, \gamma, \lambda) d\theta^t \\
&\qquad \times p(\boldsymbol{z^o}, \theta^o|\alpha, \beta, \gamma, \lambda) d\theta^o p(\boldsymbol{w}, \boldsymbol{c}, \Phi^b, \Phi^t, \Phi^o|\alpha, \beta, \gamma, \lambda) d\Phi^b d\Phi^t d\Phi^o \\
=&\int p(\boldsymbol{z^t}, \theta^t|\beta) d\theta^t \int p(\boldsymbol{z^o}, \theta^o|\gamma) d\theta^o \int\int\int p(\boldsymbol{w}, \boldsymbol{c}, \Phi^b, \Phi^t, \Phi^o|\alpha, \gamma) d\Phi^b d\Phi^t d\Phi^o \\
=&\int p(\boldsymbol{z^t}, \theta^t|\beta) d\theta^t \int p(\boldsymbol{z^o}, \theta^o|\gamma) d\theta^o \\
&\qquad \times \int\int\int p(\boldsymbol{w}|\boldsymbol{c}, \Phi^b, \Phi^t, \Phi^o, \alpha, \gamma) p(\boldsymbol{c}, \Phi^b, \Phi^t, \Phi^o|\alpha, \gamma) d\Phi^b d\Phi^t d\Phi^o \\
=&\int p(\boldsymbol{z^t}, \theta^t|\beta) d\theta^t \int p(\boldsymbol{z^o}, \theta^o|\gamma) d\theta^o \\
&\qquad \int\int\int p(\boldsymbol{w}|\boldsymbol{c}, \Phi^b, \Phi^t, \Phi^o) p(\boldsymbol{c}) p(\Phi^b|\alpha) p(\Phi^t|\alpha) p(\Phi^o|\lambda) d\Phi^b d\Phi^t d\Phi^o \\
\propto&\int p(\boldsymbol{z^t}, \theta^t|\beta) d\theta^t \int p(\boldsymbol{z^o}, \theta^o|\gamma) d\theta^o \\
&\qquad \times \int\int\int \prod_{i=1}^{3} p(\boldsymbol{w}|c=i, \Phi^b, \Phi^t, \Phi^o) p(\boldsymbol{c}) p(\Phi^b|\alpha) p(\Phi^t|\alpha) p(\Phi^o|\lambda) d\Phi^b d\Phi^t d\Phi^o \\
=&\int p(\boldsymbol{z^t}, \theta^t|\beta) d\theta^t \int p(\boldsymbol{z^o}, \theta^o|\gamma) d\theta^o
\end{aligned}$$

$$
\begin{aligned}
&\times \int\int\int p(\boldsymbol{w}|c=0,\Phi^b)p(\Phi^b|\alpha)p(\boldsymbol{w}|c=1,\Phi^t)p(\Phi^t|\alpha) \\
&\quad \times p(\boldsymbol{w}|c=2,\Phi^o)p(\Phi^o|\lambda)d\Phi^b d\Phi^t d\Phi^o \\
&= \int p(\boldsymbol{z^t},\theta^t|\beta)d\theta^t \int p(\boldsymbol{z^o},\theta^o|\gamma)d\theta^o \\
&\quad \times \int p(\boldsymbol{w}|\Phi^b)p(\Phi^b|\alpha)d\Phi^b \int p(\boldsymbol{w}|\Phi^t)p(\Phi^t|\alpha)d\Phi^t \int p(\boldsymbol{w}|\Phi^o)p(\Phi^o|\lambda)d\Phi^o \qquad (1)
\end{aligned}
$$

where the first and fourth equalities hold because $\boldsymbol{c}$, θ^t, θ^o, Φ^b, Φ^t and Φ^o are all independent. Dirichlet prior parameters that are not required are filtered out for each variables. (for instance, γ is required only for the second probability) The proportionality holds as $p(\boldsymbol{c})$ is constant for all (d,m).

Each integral can be simplified to products of beta functions. This step portrays the usefulness of Multinomial-Dirichlet conjugacy. Dirichlet distribution is the conjugate prior of Multinomial function, and therefore the posterior distribution is also Dirichlet and can be simplified significantly. The first integral is:

$$
\begin{aligned}
\int p(\boldsymbol{z^t},\theta^t|\beta)d\theta^t &= \int p(\boldsymbol{z^t}|\theta^t,\beta)p(\theta^t|\beta)d\theta^t \\
&= \int \prod_d^D p(\boldsymbol{z_d^t}|\theta_d^t)p(\theta_d^t|\beta)d\theta_d^t \\
&= \prod_d^D \int \prod_m^{d_m} p(z_{d,m}^t|\theta_d^t)p(\theta_d^t|\beta)d\theta_d^t \\
&= \prod_d^D \int \prod_m^{d_m} \theta_{d,z_{d,m}^t}^t \frac{1}{B(\beta)} \prod_k^K \theta_{d,k}^{t,\beta_k-1} d\theta_d^t
\end{aligned}
$$

The last equality holds from the fact that $\boldsymbol{z_{d,m}^t} \sim Multinomial(\theta_d^t)$ and $\theta_d^t \sim Dirichlet(\beta)$. $B(\cdot)$ is the Beta function, and β_k denotes kth value of β.The above integral simplifies to:

$$
\begin{aligned}
\int p(\boldsymbol{z^t},\theta^t|\beta)d\theta^t &= \prod_d^D \int \frac{1}{B(\beta)} \prod_k^K \theta_{d,k}^{t,\beta_k+Z_d^{k,*}-1} d\theta_d^t \\
&= \prod_d^D \frac{B(\beta+Z_d^{\wedge,*})}{B(\beta)} \int \frac{1}{B(\beta+Z_d^{\wedge,*})} \prod_k^K \theta_{d,k}^{t,\beta_k+Z_d^{k,*}-1} d\theta_d^t
\end{aligned}
$$

$$= \prod_d^D \frac{B(\beta + Z_d^{\wedge,*})}{B(\beta)}$$

where the last equality holds because the expression inside the integral is simply the support of Dirichlet distribution with parameter $\beta + Z_d^{\wedge,*}$, and thus integrates to 1. We can apply similar methods of simplification to other integrals:

$$\begin{aligned}
\int p(\boldsymbol{z^o}, \theta^o|\gamma)d\theta^o &= \int p(\boldsymbol{z^o}|\theta^o, \gamma)p(\theta^o|\gamma)d\theta^o \\
&= \int \prod_d^D p(\boldsymbol{z_d^o}|\theta_d^o)p(\theta_d^o|\gamma)d\theta_d^o \\
&= \prod_d^D \int \prod_m^{d_m} p(z_{d,m}^o|\theta_d^o)p(\theta_d^o|\gamma)d\theta_d^o \\
&= \prod_d^D \int \prod_m^{d_m} \theta_{d,z_{d,m}^o}^o \frac{1}{B(\gamma)} \prod_s^S \theta_{d,s}^{o,\gamma_s-1} d\theta_d^o \\
&= \prod_d^D \int \frac{1}{B(\gamma)} \prod_s^S \theta_{d,s}^{o,\gamma_s+Z_d^{*,s}-1} d\theta_d^o \\
&= \prod_d^D \frac{B(\gamma + Z_d^{*,\wedge})}{B(\gamma)} \int \frac{1}{B(\gamma + Z_d^{*,\wedge})} \prod_s^S \theta_{d,s}^{o,\gamma_s+Z_d^{*,s}-1} d\theta_d^o \\
&= \prod_d^D \frac{B(\gamma + Z_d^{*,\wedge})}{B(\gamma)}
\end{aligned}$$

$$\begin{aligned}
\int p(\boldsymbol{w}|\Phi^b)p(\Phi^b|\alpha)d\Phi^b &= \int \prod_d^D \prod_m^{d_m} \prod_n^{N_{d,m}} \Phi_{w_{d,m,n}}^b \frac{1}{B(\alpha)} \prod_v^V \Phi_v^{b,\alpha_v-1} d\Phi^b \\
&= \int \frac{1}{B(\alpha)} \prod_v^V \Phi_v^{b,\alpha_v+W_{*,*,v,0}^{*,*}-1} d\Phi^b \\
&= \frac{B(\alpha + W_{*,*,\wedge,0}^{*,*})}{B(\alpha)} \int \prod_v^V \Phi_v^{b,\alpha_v+W_{*,*,v,0}^{*,*}-1} d\Phi^b \\
&= \frac{B(\alpha + W_{*,*,\wedge,0}^{*,*})}{B(\alpha)}
\end{aligned}$$

$$\int p(\boldsymbol{w}|\Phi^t)p(\Phi^t|\alpha)d\Phi^t = \int \prod_k^K p(\boldsymbol{w}|\Phi_k^t)p(\Phi_k^t|\alpha)d\Phi_k^t$$

$$
\begin{aligned}
&= \prod_k^K \int \prod_d^D \prod_m^{d_m} \prod_n^{N_{d,m}} \Phi^t_{k,w_{d,m,n}} \frac{1}{B(\alpha)} \prod_v^V \Phi^{t,\alpha_v - 1}_{k,v} d\Phi^t_k \\
&= \prod_k^K \frac{B(\alpha + W^{k,*}_{*,*,\wedge,1})}{B(\alpha)} \int \prod_v^V \Phi^{t,\alpha_v + W^{k,*}_{*,*,v,1}}_{k,v} d\Phi^t_k \\
&= \prod_k^K \frac{B(\alpha + W^{k,*}_{*,*,\wedge,1})}{B(\alpha)}
\end{aligned}
$$

$$
\begin{aligned}
\int p(\boldsymbol{w}|\Phi^o) p(\Phi^o|\lambda) d\Phi^o &= \int \prod_k^K \prod_s^S p(\boldsymbol{w}|\Phi^o_{k,s}) p(\Phi^o_{k,s}|\lambda) d\Phi^o_{k,s} \\
&= \prod_k^K \prod_s^S \int \prod_d^D \prod_m^{d_m} \prod_n^{N_{d,m}} \Phi^o_{k,s,w_{d,m,n}} \frac{1}{B(\lambda)} \prod_v^V \Phi^{o,\lambda_v - 1}_{k,s,v} d\Phi^o_{k,s} \\
&= \prod_k^K \prod_s^S \frac{B(\lambda + W^{k,s}_{*,*,\wedge,2})}{B(\lambda)} \int \prod_v^V \Phi^{o,\lambda_v + W^{k,s}_{*,*,v,2}}_{k,s,v} d\Phi^o_{k,s} \\
&= \prod_k^K \prod_s^S \frac{B(\lambda + W^{k,s}_{*,*,\wedge,2})}{B(\lambda)}
\end{aligned}
$$

Putting them all together, equation (1) becomes

$$
\begin{aligned}
&p(z^t_{d,m}, z^o_{d,m} | \boldsymbol{z^t_{-(d,m)}}, \boldsymbol{z^o_{-(d,m)}}, \boldsymbol{w}, \boldsymbol{c}, \alpha, \beta, \gamma, \lambda) \\
\propto &\Big[\prod_d^D \frac{B(\beta + Z^{\wedge,*}_d)}{B(\beta)} \Big] \Big[\prod_d^D \frac{B(\gamma + Z^{*,\wedge}_d)}{B(\gamma)} \Big] \\
&\times \Big[\frac{B(\alpha + W^{*,*}_{*,*,\wedge,0})}{B(\alpha)} \Big] \Big[\prod_k^K \frac{B(\alpha + W^{k,*}_{*,*,\wedge,1})}{B(\alpha)} \Big] \Big[\prod_k^K \prod_s^S \frac{B(\lambda + W^{k,s}_{*,*,\wedge,2})}{B(\lambda)} \Big] \qquad (2)
\end{aligned}
$$

$z^t_{d,m}$ and $z^o_{d,m}$ are not dependent on the third fraction of (2), thus we can eliminate this part. Beta functions in denominators are constant, and hence can be eliminated as well. Beta function is defined as:

$$
B(x = \{x_1, \cdots, x_n\}) = \frac{\prod_{i=1}^n \Gamma(x_i)}{\Gamma(\sum_{i=1}^n x_i)}
$$

Therefore, (2) can be simplified in proportionality:

$$
\begin{aligned}
&\propto \Big[\prod_d^D \frac{B(\beta + Z_d^{\wedge,*})}{B(\beta)} \frac{B(\gamma + Z_d^{*,\wedge})}{B(\gamma)}\Big] \Big[\prod_k^K \frac{B(\alpha + W_{*,*,\wedge,1}^{k,*})}{B(\alpha)}\Big] \Big[\prod_k^K \prod_s^S \frac{B(\lambda + W_{*,*,\wedge,2}^{k,s})}{B(\lambda)}\Big] \\
&\propto \prod_d^D B(\beta + Z_d^{\wedge,*}) B(\gamma + Z_d^{*,\wedge}) \prod_k^K B(\alpha + W_{*,*,\wedge,1}^{k,*}) \prod_k^K \prod_s^S B(\lambda + W_{*,*,\wedge,2}^{k,s}) \\
&= \prod_d^D \frac{\prod_k^K \Gamma(\beta_k + Z_d^{k,*})}{\Gamma(\sum_{k'}[\beta_{k'} + Z_d^{k',*}])} \frac{\prod_s^S \Gamma(\gamma_s + Z_d^{*,s})}{\Gamma(\sum_{s'}[\gamma_{s'} + Z_d^{*,s'}])} \\
&\quad \times \prod_k^K \frac{\prod_v^V \Gamma(\alpha_v + W_{*,*,v,1}^{k,*})}{\Gamma(\sum_{v'}[\alpha_{v'} + W_{*,*,v',1}^{k,*}])} \prod_k^K \prod_s^S \frac{\prod_v^V \Gamma(\lambda_v + W_{*,*,v,2}^{k,s})}{\Gamma(\sum_{v'}[\lambda_{v'} + W_{*,*,v',2}^{k,s}])}
\end{aligned}
$$

Each of the product terms include some terms that are not constant in d and v in (d, m):

$$
\begin{aligned}
&= \prod_{d' \neq d} \left[\frac{\prod_k^K \Gamma(\beta_k + Z_{d'}^{k,*})}{\Gamma(\sum_{k'}[\beta_{k'} + Z_{d'}^{k',*}])} \frac{\prod_s^S \Gamma(\gamma_s + Z_{d'}^{*,s})}{\Gamma(\sum_{s'}[\gamma_{s'} + Z_{d'}^{*,s'}])}\right] \\
&\quad \times \frac{\prod_k^K \Gamma(\beta_k + Z_d^{k,*})}{\Gamma(\sum_{k'}[\beta_{k'} + Z_d^{k',*}])} \frac{\prod_s^S \Gamma(\gamma_s + Z_d^{*,s})}{\Gamma(\sum_{s'}[\gamma_{s'} + Z_d^{*,s'}])} \\
&\quad \times \prod_k^K \frac{\prod_{v' \notin V_{d,m}} \Gamma(\alpha_{v'} + W_{*,*,v',1}^{k,*}) \prod_{v \in V_{d,m}} \Gamma(\alpha_v + W_{*,*,v,1}^{k,*})}{\Gamma(\sum_{v'}[\alpha_{v'} + W_{*,*,v',1}^{k,*}])} \\
&\quad \times \prod_k^K \prod_s^S \frac{\prod_{v \notin V_{d,m}} \Gamma(\lambda_{v'} + W_{*,*,v',2}^{k,s}) \prod_{v \in V_{d,m}} \Gamma(\lambda_v + W_{*,*,v,2}^{k,s})}{\Gamma(\sum_{v'}[\lambda_{v'} + W_{*,*,v',2}^{k,s}])} \\
&\propto \frac{\prod_k^K \Gamma(\beta_k + Z_d^{k,*})}{\Gamma(\sum_{k'}[\beta_{k'} + Z_d^{k',*}])} \frac{\prod_s^S \Gamma(\gamma_s + Z_d^{*,s})}{\Gamma(\sum_{s'}[\gamma_{s'} + Z_d^{*,s'}])} \\
&\quad \times \prod_k^K \frac{\prod_{v \in V_{d,m}} \Gamma(\alpha_v + W_{*,*,v,1}^{k,*})}{\Gamma(\sum_{v'}[\alpha_{v'} + W_{*,*,v',1}^{k,*}])} \prod_k^K \prod_s^S \frac{\prod_{v \in V_{d,m}} \Gamma(\lambda_v + W_{*,*,v,2}^{k,s})}{\Gamma(\sum_{v'}[\lambda_{v'} + W_{*,*,v',2}^{k,s}])} \\
&\propto \prod_k^K \Gamma(\beta_k + Z_d^{k,*}) \prod_s^S \Gamma(\gamma_s + Z_d^{*,s}) \\
&\quad \times \prod_k^K \frac{\prod_{v \in V_{d,m}} \Gamma(\alpha_v + W_{*,*,v,1}^{k,*})}{\Gamma(\sum_{v'} \alpha_{v'} + W_{*,*,*,1}^{k,*})} \prod_k^K \prod_s^S \frac{\prod_{v \in V_{d,m}} \Gamma(\lambda_v + W_{*,*,v,2}^{k,s})}{\Gamma(\sum_{v'} \lambda_{v'} + W_{*,*,*,2}^{k,s})}
\end{aligned} \tag{3}
$$

The count variables can be divided into two different components:

$$Z_d^{k,*} = \begin{cases} Z_d^{k,*-(d,m)} + 1 & \text{if } k = z_{d,m}^t = a \\ Z_d^{k,*-(d,m)} & \text{if } k \neq z_{d,m}^t = a \end{cases}$$

$$Z_d^{*,s} = \begin{cases} Z_d^{*,s-(d,m)} + 1 & \text{if } s = z_{d,m}^o = b \\ Z_d^{*,s-(d,m)} & \text{if } s \neq z_{d,m}^o = b \end{cases}$$

$$W_{*,*,v,1}^{k,*} = \begin{cases} W_{*,*,v,1}^{k,*-(d,m)} + W_{d,m,v,1}^{*,*} & \text{if } k = z_{d,m}^t = a \\ W_{*,*,v,1}^{k,*-(d,m)} & \text{if } k \neq z_{d,m}^t = a \end{cases}$$

$$W_{*,*,v,2}^{k,s} = \begin{cases} W_{*,*,v,2}^{k,s-(d,m)} + W_{d,m,v,2}^{*,*} & \text{if } k = z_{d,m}^t = a \text{ and } s = z_{d,m}^o = b \\ W_{*,*,v,2}^{k,s-(d,m)} & \text{otherwise} \end{cases}$$

Plugging in these into (3) and dividing the product into separate terms, (3) can be rewritten as:

$$\begin{aligned} &= \Gamma(\beta_a + Z_d^{a,*-(d,m)} + 1) \prod_{k \neq a} \Gamma(\beta_k + Z_d^{k,*-(d,m)}) \\ &\quad \times \Gamma(\gamma_b + Z_d^{*,b-(d,m)} + 1) \prod_{s \neq b} \Gamma(\gamma_s + Z_d^{*,s-(d,m)}) \\ &\quad \times \frac{\prod_{v \in V_{d,m}} \Gamma(\alpha_v + W_{*,*,v,1}^{a,*-(d,m)} + W_{d,m,v,1}^{*,*})}{\Gamma(\sum_{v'} \alpha_{v'} + W_{*,*,*,1}^{a,*-(d,m)} + W_{d,m,*,1}^{*,*})} \prod_{k \neq a} \frac{\prod_{v \in V_{d,m}} \Gamma(\alpha_v + W_{*,*,v,1}^{k,*-(d,m)})}{\Gamma(\sum_{v'} \alpha_{v'} + W_{*,*,*,1}^{k,*-(d,m)})} \\ &\quad \times \frac{\prod_{v \in V_{d,m}} \Gamma(\lambda_v + W_{*,*,v,2}^{a,b-(d,m)} + W_{d,m,v,2}^{*,*})}{\Gamma(\sum_{v'} \lambda_{v'} + W_{*,*,*,2}^{a,b-(d,m)} + W_{d,m,*,2}^{*,*})} \prod_{k \neq a} \prod_{s \neq b} \frac{\prod_{v \in V_{d,m}} \Gamma(\lambda_v + W_{*,*,v,2}^{k,s-(d,m)})}{\Gamma(\sum_{v'} \lambda_{v'} + W_{*,*,*,2}^{k,s-(d,m)})} \end{aligned} \tag{4}$$

Now we can use a property of Gamma function for further simplification. Gamma function is defined as

$$\Gamma(x) = \begin{cases} \int_0^\infty t^{x-1} e^{-t} dt & \text{if } x \notin \mathbb{Z}^- \\ \text{undefined} & \text{otherwise} \end{cases}$$

For $x \in \mathbb{Z}^+$, the integral simplifies to $(x-1)!$. The property of our interest is stated by the following theorem.

Theorem 1. *For $x > 1$:*

$$\Gamma(x) = (x-1)\Gamma(x-1)$$

Proof. Let $u = t^{x-1}$ and $v = -e^{-t}$, which implies that $du = (x-1)t^{x-2}dt$ and $dv = e^{-t}dt$. Using integration by parts,

$$\begin{aligned}
\Gamma(x) &= \int_0^\infty t^{x-1}e^{-t}dt \\
&= \lim_{n\to\infty} -t^{x-1}e^{-t}|_{t=0}^{t=n} + (x-1)\int_0^\infty t^{x-2}e^{-t}dt \\
&= \lim_{n\to\infty} -\frac{n^{x-1}}{e^n} + (x-1)\Gamma(x-1) \\
&= \lim_{n\to\infty} -e^{(x-1)\ln n - n} + (x-1)\Gamma(x-1) \\
&= \lim_{n\to\infty} -e^{(x-1)(\frac{\ln n}{n}-1)n} + (x-1)\Gamma(x-1)
\end{aligned}$$

Using L'Hospital's rule,

$$\begin{aligned}
\lim_{n\to\infty} \frac{\ln n}{n} &= \lim_{n\to\infty} \frac{1/n}{1} \\
&= 0
\end{aligned}$$

Therefore, the exponents of e goes to $-\infty$, and the limit term becomes 0. We are left with

$$\Gamma(x) = (x-1)\Gamma(x-1)$$

□

This property can be repeatedly applied to (4):

$$\begin{aligned}
&= (\beta_a + Z_d^{a,*-(d,m)})\Gamma(\beta_a + Z_d^{a,*-(d,m)}) \prod_{k\neq a} \Gamma(\beta_k + Z_d^{k,*-(d,m)}) \\
&\quad \times (\gamma_b + Z_d^{*,b-(d,m)})\Gamma(\gamma_b + Z_d^{*,b-(d,m)}) \prod_{s\neq b} \Gamma(\gamma_s + Z_d^{*,s-(d,m)}) \\
&\quad \times \frac{\prod_{v\in V_{d,m}} \Gamma(\alpha_v + W_{*,*,v,1}^{a,*-(d,m)}) \prod_i^{W_{d,m,v,1}^{*,*}} (\alpha_v + W_{*,*,v,1}^{a,*-(d,m)} + i - 1)}{\Gamma(\sum_{v'} \alpha_{v'} + W_{*,*,*,1}^{a,*-(d,m)}) \prod_j^{W_{d,m,*,1}^{*,*}} (\sum_{v'} \alpha_{v'} + W_{*,*,*,1}^{a,*-(d,m)} + j - 1)}
\end{aligned}$$

$$\times \frac{\prod_{v\in V_{d,m}} \Gamma(\lambda_v + W^{a,b-(d,m)}_{*,*,v,2}) \prod_i^{W^{*,*}_{d,m,v,2}} (\lambda_v + W^{a,b-(d,m)}_{*,*,v,2} + i - 1)}{\Gamma(\sum_{v'} \lambda_{v'} + W^{a,b-(d,m)}_{*,*,*,2}) \prod_j^{W^{*,*}_{d,m,*,2}} (\sum_{v'} \lambda_{v'} + W^{a,b-(d,m)}_{*,*,*,2} + j - 1)}$$
$$\times \prod_{k\neq a} \frac{\prod_{v\in V_{d,m}} \Gamma(\alpha_v + W^{k,*-(d,m)}_{*,*,v,1})}{\Gamma(\sum_{v'} \alpha_{v'} + W^{k,*-(d,m)}_{*,*,*,1})} \prod_{k\neq a} \prod_{s\neq b} \frac{\prod_{v\in V_{d,m}} \Gamma(\lambda_v + W^{k,s-(d,m)}_{*,*,v,2})}{\Gamma(\sum_{v'} \lambda_{v'} + W^{k,s-(d,m)}_{*,*,*,2})}$$

Combining respective Gamma functions,

$$= (\beta_a + Z^{a,*-(d,m)}_d) \prod_k \Gamma(\beta_k + Z^{k,*-(d,m)}_d)$$
$$\times (\gamma_b + Z^{*,b-(d,m)}_d) \prod_s \Gamma(\gamma_s + Z^{*,s-(d,m)}_d)$$
$$\times \frac{\prod_{v\in V_{d,m}} \prod_i^{W^{*,*}_{d,m,v,1}} (\alpha_v + W^{a,*-(d,m)}_{*,*,v,1} + i - 1)}{\prod_j^{W^{*,*}_{d,m,*,1}} (\sum_{v'} \alpha_{v'} + W^{a,*-(d,m)}_{*,*,*,1} + j - 1)} \prod_k \frac{\prod_{v\in V_{d,m}} \Gamma(\alpha_v + W^{k,*-(d,m)}_{*,*,v,1})}{\Gamma(\sum_{v'} \alpha_{v'} + W^{k,*-(d,m)}_{*,*,*,1})}$$
$$\times \frac{\prod_{v\in V_{d,m}} \prod_i^{W^{*,*}_{d,m,v,2}} (\lambda_v + W^{a,b-(d,m)}_{*,*,v,2} + i - 1)}{\prod_j^{W^{*,*}_{d,m,*,2}} (\sum_{v'} \lambda_{v'} + W^{a,b-(d,m)}_{*,*,*,2} + j - 1)} \prod_k \prod_s \frac{\prod_{v\in V_{d,m}} \Gamma(\lambda_v + W^{k,s-(d,m)}_{*,*,v,2})}{\Gamma(\sum_{v'} \lambda_{v'} + W^{k,s-(d,m)}_{*,*,*,2})}$$

Terms not dependent on a and b are separated and can be eliminated. Furthermore, by definition of Gamma function:

$$\propto (\beta_a + Z^{a,*-(d,m)}_d)(\gamma_b + Z^{*,b-(d,m)}_d)$$
$$\times \frac{\prod_{v\in V_{d,m}} \prod_i^{W^{*,*}_{d,m,v,1}} (\alpha_v + W^{a,*-(d,m)}_{*,*,v,1} + i - 1)}{\prod_j^{W^{*,*}_{d,m,*,1}} (\sum_{v'} [\alpha_{v'} + W^{a,*-(d,m)}_{*,*,v',1}] + j - 1)}$$
$$\times \frac{\prod_{v\in V_{d,m}} \prod_i^{W^{*,*}_{d,m,v,2}} (\lambda_v + W^{a,b-(d,m)}_{*,*,v,2} + i - 1)}{\prod_j^{W^{*,*}_{d,m,*,2}} (\sum_{v'} [\lambda_{v'} + W^{a,b-(d,m)}_{*,*,v',2}] + j - 1)}$$
$$= (\beta_a + Z^{a,*-(d,m)}_d)(\gamma_b + Z^{*,b-(d,m)}_d)$$
$$\times \frac{\prod_{v\in V_{d,m}} \Gamma(\alpha_v + W^{a,*-(d,m)}_{*,*,v,1} + W^{*,*}_{d,m,v,1}) / \Gamma(\alpha_v + W^{a,*-(d,m)}_{*,*,v,1})}{\Gamma(\sum_{v'} \alpha_{v'} + W^{a,*-(d,m)}_{*,*,*,1} + W^{*,*}_{d,m,*,1}) / \Gamma(\sum_{v'} \alpha_{v'} + W^{a,*-(d,m)}_{*,*,*,1})}$$
$$\times \frac{\prod_{v\in V_{d,m}} \Gamma(\lambda_v + W^{a,b-(d,m)}_{*,*,v,2} + W^{*,*}_{d,m,v,2}) / \Gamma(\lambda_v + W^{a,b-(d,m)}_{*,*,v,2})}{\Gamma(\sum_{v'} \lambda_{v'} + W^{a,b-(d,m)}_{*,*,*,2} + W^{*,*}_{d,m,*,2}) / \Gamma(\sum_{v'} \lambda_{v'} + W^{a,b-(d,m)}_{*,*,*,2})}$$

$rf(x,n) = \Gamma(x+n)/\Gamma(x)$, known as Pochhammer's polynomial or rising factorial function,

is a widely used function, and there are Python methods to calculate this. Although Gamma function representation is used in code due to its faster computational speed, rf can be used to further simplify the probability. To prevent overflow, we calculate log conditional probability by applying log-transformation to all count variables to get:

$$\begin{aligned}\log p(z^t_{d,m}, z^o_{d,m}|\cdot) \propto & \log(\beta_a + Z_d^{a,*-(d,m)}) + \log(\gamma_b + Z_d^{*,b-(d,m)}) \\ & + \sum_{v \in V_{d,m}} \big[\log rf(\alpha_v + W^{a,*-(d,m)}_{*,*,v,1}, W^{*,*}_{d,m,v,1}) \\ & \quad + \log rf(\lambda_v + W^{a,b-(d,m)}_{*,*,v,2}, W^{*,*}_{d,m,v,2})\big] \\ & - \log rf(\sum_{v'} \alpha_{v'} + W^{a,*-(d,m)}_{*,*,*,1}, W^{*,*}_{d,m,*,1}) \\ & - \log rf(\sum_{v'} \lambda_{v'} + W^{a,b-(d,m)}_{*,*,*,2}, W^{*,*}_{d,m,*,2}) \end{aligned} \tag{5}$$

This conditional probability is calculated for each pair (a, b), and a sample is drawn for each (d, m) in every iteration of the fitting algorithm. The sampling process is the most computationally expensive part of the model, and thus requires optimization, which will be discussed in the next section.

The full Collapsed Gibbs Sampling algorithm for TSLDA is shown in Algorithm 2. It involves initiation, in which count variables are initialized to random values, and sampling, in which new z^t and z^o are sampled. Initialization involves setting random labels for topic and sentiment for each document and updating count variables correspondingly. For every iteration, new topic and sentiment assignments are sampled for each sentence from the conditional probability (5). Count variables are updated before and after the sampling process.

After the algorithm is complete, we can use $W^{\wedge,\wedge}_{\wedge,\wedge,\wedge,\wedge}$ and $Z^{\wedge,\wedge}_{\wedge}$ as empirical distribution to estimate other hidden variables:

$$\theta^t_d[k] = \frac{Z_d^{a,*} + \beta[k]}{\sum_{k'}^K Z_d^{k',*} + \beta[k']}, \theta^o_d[s] = \frac{Z_d^{*,b} + \gamma[s]}{\sum_{s'}^S Z_d^{*,s'} + \gamma[s']}$$

$$\Phi^b[v] = \frac{W^{*,*}_{*,*,v,0} + \alpha[v]}{\sum_{v'}^V W^{*,*}_{*,*,v',0} + \alpha[v']}, \Phi^t_k[v] = \frac{W^{k,*}_{*,*,v,1} + \alpha[v]}{\sum_{v'}^V W^{k,*}_{*,*,v',1} + \alpha[v']}, \Phi^o_{k,s}[v] = \frac{W^{k,s}_{*,*,v,2} + \lambda[v]}{\sum_{v'}^V W^{k,s}_{*,*,v',2} + \lambda[v']}$$

Algorithm 2 TSLDA Collapsed Gibbs Sampling

Require: K, α, β, γ, λ, $\boldsymbol{w}$, $\boldsymbol{c}$

Initialization
for each pair (d, m) **do**
  Sample two hidden variables: $z^t_{d,m} = a \sim Multinomial(1/K)$
  $z^o_{d,m} = b \sim Multinomial(1/S)$
  $Z^{a,b}_d$ += 1
  for $n \in N_{d,m}$ **do**
    $W^{a,b}_{d,m,w_{d,m,n},c_{d,m,n}}$ += 1
  end for
end for

Sampling
for $i \in (1, \text{n\_iter})$ **do**
  for each pair (d, m) **do**
    $\hat{a} = z^t_{d,m}$, $\hat{b} = z^o_{d,m}$
    $Z^{\hat{a},\hat{b}}_d$ −= 1
    for $n \in N_{d,m}$ **do**
      $W^{\hat{a},\hat{b}}_{d,m,w_{d,m,n},c_{d,m,n}}$ −= 1
    end for
    Sample new $z^t_{d,m} = a$ and $z^o_{d,m} = b$ using the conditional probability (5)
    $Z^{a,b}_d$ += 1
    for $n \in N_{d,m}$ **do**
      $W^{a,b}_{d,m,w_{d,m,n},c_{d,m,n}}$ += 1
    end for
  end for
end for

Although CGS is very simple and returns accurate estimate of the true distribution of a variable, its disadvantage is that it is very time consuming; furthermore, topic modeling involving text data is also computationally expensive. Therefore, optimizations such as parallelization are important.

4.2.3 Optimization

Two optimization methods are implemented in TSLDA model: data structure optimization and parallelization. Each optimization is discussed in details below.

There are many possible methods of storing count variables $Z^{\wedge,\wedge}_{\wedge}$ and $W^{\wedge,\wedge}_{\wedge,\wedge,\wedge,\wedge}$, but it must allow fast access and addition of individual variables. In the TSLDA algorithm (Algorithm 2), the most computationally expensive step is the calculation of probabilities

in equation (5). Sufficient statistics required for conditional probability calculation is

$$W^{*,*}_{d,m,v,1}, \quad W^{*,*}_{d,m,v,2}, \quad W^{*,*}_{d,m,*,1}, \quad W^{*,*}_{d,m,*,2}$$

$$Z^{a,*-(d,m)}_{d} = Z^{a,*}_{d} - Z^{a,*}_{d,m}, \quad Z^{*,b-(d,m)}_{d} = Z^{*,b}_{d} - Z^{*,b}_{d,m}$$

$$W^{a,*-(d,m)}_{*,*,v,1} = W^{a,*}_{*,*,v,1} - W^{a,*}_{d,m,v,1}, \quad W^{a,*-(d,m)}_{*,*,*,1} = W^{a,*}_{*,*,*,1} - W^{a,*}_{d,m,*,1}$$

$$W^{a,b-(d,m)}_{*,*,v,2} = W^{a,b}_{*,*,v,2} - W^{a,b}_{d,m,v,2}, \quad W^{a,b-(d,m)}_{*,*,*,2} = W^{a,b}_{*,*,*,2} - W^{a,b}_{d,m,*,2}$$

Because $V = |\{v_1, \cdots, v_V\}|$ and $M = |\{(d_1, m_1), \cdots, (d_D, m_{d_D})\}|$, which are number of words and sentences, are very large, summing over word or sentence index is slow, and therefore these summed values should be cached for faster access. Furthermore, d and m should be indexed together for W because there is no difference between indexing with (d, m) and d and m separately.

Naive brute force method would be simply storing all variables in matrices. Nonetheless, due to immense size of the data, doing so will involve creating matrices Z and W with $K \times S \times M$ and $K \times S \times M \times V \times 3$ dimensions each. With M and V very large, these matrices are too large to be stored in memory. Therefore a perfect compromise between memory and caching must be selected.

Because many documents do not have most words, most entries will be zero. Although this naturally leads to a data structure that uses sparse matrices, the dimensions are still too large to fit the matrices in memory comfortably. Another method of storing sparse data in a smaller structure is to use hashtables. Prioritizing the sufficient statistics above, the following structures allow fast access of required variables:

$$Z^{T}_{d} = \begin{bmatrix} Z^{1,*}_{1} & \cdots & Z^{K,*}_{1} \\ \vdots & \ddots & \vdots \\ Z^{1,*}_{D} & \cdots & Z^{K,*}_{D} \end{bmatrix}, \quad Z^{O}_{d} = \begin{bmatrix} Z^{*,1}_{1} & \cdots & Z^{*,S}_{1} \\ \vdots & \ddots & \vdots \\ Z^{*,1}_{D} & \cdots & Z^{*,S}_{D} \end{bmatrix}$$

$$Z^T_{dm} = \begin{bmatrix} Z^{1,*}_{(d_1,m_1)} & \cdots & Z^{K,*}_{(d_1,m_1)} \\ \vdots & \ddots & \vdots \\ Z^{1,*}_{(D,m_D)} & \cdots & Z^{K,*}_{(D,m_D)} \end{bmatrix}, \quad Z^O_{dm} = \begin{bmatrix} Z^{*,1}_{(d_1,m_1)} & \cdots & Z^{*,S}_{(d_1,m_1)} \\ \vdots & \ddots & \vdots \\ Z^{*,1}_{(D,m_D)} & \cdots & Z^{*,S}_{(D,m_D)} \end{bmatrix}$$

$$\begin{aligned} W^T_{dmv} = \{(d_1,m_1) : \{v_1 : [W^{1,*}_{d_1,m_1,v_1,1}, \cdots, W^{K,*}_{d_1,m_1,v_1,1}], \cdots \\ v_{N_{d_1,m_1}} : [W^{1,*}_{d_1,m_1,v_{N_{d_1,m_1}},1}, \cdots, W^{K,*}_{d_1,m_1,v_{N_{d_1,m_1}},1}]\}, \\ \vdots \\ (D,m_D) : \{v_1 : [W^{1,*}_{D,m_D,v_1,1}, \cdots, W^{K,*}_{D,m_D,v_1,1}], \cdots \\ v_{N_{D,m_D}} : [W^{1,*}_{D,m_D,v_{N_{D,m_D}},1}, \cdots, W^{K,*}_{D,m_D,v_{N_{D,m_D}},1}]\}\} \end{aligned}$$

$$\begin{aligned} W^T_v = \{v_1 : & [W^{1,*}_{*,*,v_1,1}, \cdots, W^{K,*}_{*,*,v_1,1}], \\ & \vdots \\ v_V : & [W^{1,*}_{*,*,v_V,1}, \cdots, W^{K,*}_{*,*,v_V,1}], \\ allv : & [W^{1,*}_{*,*,*,1}, \cdots, W^{K,*}_{*,*,*,1}]\} \end{aligned}$$

$$\begin{aligned} W^T_{dm} = \{(d_1,m_1) : & [W^{1,*}_{d_1,m_1,*,1}, \cdots, W^{K,*}_{d_1,m_1,*,1}], \\ & \vdots \\ (D,m_D) : & [W^{1,*}_{D,m_D,*,1}, \cdots, W^{K,*}_{D,m_D,*,1}]\} \end{aligned}$$

W^O_{dmv}, W^O_v, and W^O_{dm} are similarly defined, but because these variables also index S, the final entries will be a $S \times K$ matrix instead of $1 \times K$ vectors shown above. The three data structures are used for different sufficient statistics. W^T_{dmv} is for accessing variables that are not summed over any index, such as $W^{a,*}_{d,m,v,1}$. The hash table format of the structure allows us to store only present vocabularies for each sentence, therefore minimizing data structure size. W^T_v is used to access variables summed over sentences such as $W^{a,*}_{*,*,v,1}$. Sum of all vocabularies is also stored and counted concurrently to avoid summation. Similarly, W^T_{dm} is used to access variables summed over vocabularies, such

as $W^{a,*}_{d,m,*,1}$. Although $W^{*,*}_{\wedge,\wedge,\wedge,0}$ is not used in probability calculation, it is needed to find background word distribution. The sufficient statistics for calculating Φ_b is $W^{*,*}_{*,*,v,0}$, and thus we store

$$W^B_v = \{v_1 : W^{*,*}_{*,*,v_1,0}, \cdots, v_{V_0} : W^{*,*}_{*,*,v_{V_0},0}\}$$

where V_0 denotes total number of words with category $c = 0$. Topic and sentiment assignments also need to be stored in two vectors of length M, $\boldsymbol{z^t}$ and $\boldsymbol{z^o}$; $z^t[(d,m)]$ is topic (between 0 and $K-1$) and $z^o[(d,m)]$ is topic (between 0 and $K-1$) of sentence m in document d, respectively. Finally, a hash table, *tdict*, that maps sentences to documents is stored.

Using the above data structure, sufficient statistics can be calculated with minimal summation and access:

$$W^{*,*}_{d,m,v,1} = \sum W^T_{dmv}[(d,m)][v], \quad W^{*,*}_{d,m,v,2} = \sum W^O_{dmv}[(d,m)][v]$$

$$W^{*,*}_{d,m,*,1} = \sum W^T_{dm}[(d,m)], \quad W^{*,*}_{d,m,*,2} = \sum W^O_{dm}[(d,m)]$$

$$Z^{a,*-(d,m)}_d = Z^T_D[d][a] - Z^T_{dm}[(d,m)][a], \quad Z^{*,b-(d,m)}_d = Z^O_D[d][b,a] - Z^O_{dm}[(d,m)][b,a]$$

$$W^{a,*-(d,m)}_{*,*,v,1} = W^T_v[v][a] - W^T_{dmv}[(d,m)][v][a]$$

$$W^{a,*-(d,m)}_{*,*,*,1} = W^T_v[allv][a] - W^T_{dm}[(d,m)][a]$$

$$W^{a,b-(d,m)}_{*,*,v,2} = W^O_v[v][b,a] - W^T_{dmv}[(d,m)][v][b,a]$$

$$W^{a,b-(d,m)}_{*,*,*,2} = W^O_v[allv][b,a] - W^O_{dm}[(d,m)][b,a]$$

For the first four sufficient statistics, we sum over all topic and sentiment, but this is only $K \times S$, which is at most a few hundred; optimization does not improve the performance too much at this level. Another optimization that can be applied is calculating the above sufficient statistics using matrices instead of calculating probabilities for each pair (a,b). Using NumPy package on Python, one can perform much faster calculation by doing matrices manipulation rather than element-wise manipulation. Because we will do a log transformation, matrix multiplications become matrix addition/subtraction.

Other major optimization applied to TSLDA is parallelization. By using a group of processes to iterate through documents faster, parallel Collapsed Gibbs Sampling (pCGS) is able to significantly decrease the runtime while making only negligible accuracy error. Due to its long training time, LDA is analyzed frequently to create a parallelization such as pCGS that is fast and accurate.[16] Because TSLDA is not a widely known probabilistic model, there exists no parallel algorithm for fitting TSLDA model; it is presented below.

The most computationally expensive section of Algorithm 2 is the sampling process. Initialization and counting processes only involve initializing, accessing, and incrementing/decrementing by 1, and therefore do not take as much time as sampling process, in which $K \times S$ posterior probabilities need to be calculated for each sentence in each iteration. This means that a fitting process for TSLDA runs in $O(N_{iter} \cdot M \cdot K \cdot S)$, with large M and V.

M is usually the largest value for a given set of data. Furthermore, the sampling process is iterated through all pairs (d, m), and in each iteration, the only variables that require count variables of all (d, m) are W_v^T and W_v^O used in calculating $W_{*,*,v,1}^{a,*-(d,m)}$ and $W_{*,*,v,2}^{a,b-(d,m)}$. Therefore, the input data can be divided into P partitions so that P processes can simultaneously sample new topic and sentiment for each sentence in partitions using approximate W_v^T and W_v^O without losing too much accuracy. (and exact values for other variables) This reduces the runtime to $O([N_{iter} \cdot M \cdot K \cdot S]/P)$.

The approximation using this parallelization with LDA performed well without sacrificing accuracy in [16]. They classified each sufficient statistics as global or local variables; global variable is shared among processes and synchronized after every iteration, and local variables are stored in each process. This parallelization can be extended to TSLDA by using W_v^T and W_v^O as global variables. The set of documents $\{d_1, \cdots, d_D\}$ is divided into P partitions, each with size of D/P. Then P separate processes iterate through documents and sentences in P partitions concurrently. Each process is passed local variables and global variables in each iteration. After all processes iterate through all documents, the resulting global variables are synchronized because they are not updated correctly, but approximately. This parallelization reduces the running time by P-fold while affecting accuracy minimally. Algorithm 3 summarizes pCGS:

Algorithm 3 Parallelized TSLDA Collapsed Gibbs Sampling

Require: K, α, β, γ, λ, $\boldsymbol{w}$, $\boldsymbol{c}$

Initialization
Partition tweets into P disjoint sets $\{D^1, \cdots, D^P\}$
for each partition D^p **do**
  for each sentence $(d, m) \in D^p$ **do**
    Sample two hidden variables: $z_{d,m}^{t,p} = a \sim Multinomial(1/K)$
    $z_{d,m}^{o,p} = b \sim Multinomial(1/S)$
    $Z_d^{a,b,p}$ += 1
    for $n \in N_{d,m}^p$ **do**
      $W_{d,m,w_{d,m,n},c_{d,m,n}}^{a,b,p}$ += 1
    end for
  end for
end for

Sampling
for $i \in (1, \text{n\_iter})$ **do**
  Concurrently run P processes for:

  for each pair $(d, m) \in D^p$ **do**
    $\hat{a} = z_{d,m}^{t,p}$, $\hat{b} = z_{d,m}^{o,p}$
    $Z_d^{\hat{a},\hat{b},p}$ −= 1
    for $n \in N_{d,m}^p$ **do**
      $W_{d,m,w_{d,m,n},c_{d,m,n}}^{\hat{a},\hat{b},p}$ −= 1
    end for

    Sample new $z_{d,m}^{t,p} = a$ and $z_{d,m}^{o,p} = b$ using the conditional probability (5)
    $Z_d^{a,b,p}$ += 1
    for $n \in N_{d,m}^p$ **do**
      $W_{d,m,w_{d,m,n},c_{d,m,n}}^{a,b,p}$ += 1
    end for
  end for
  Synchronize all $W_v^{T,p}$ and $W_v^{O,p}$:
  $W_{*,*,v,1}^{k,*} \leftarrow W_{*,*,v,1}^{k,*} + \sum_p (W_{*,*,v,1}^{k,*,p} - W_{*,*,v,1}^{k,*}), \forall p : W_{*,*,v,1}^{k,*,p} \leftarrow W_{*,*,v,1}^{k,*}$
  $W_{*,*,v,2}^{k,s} \leftarrow W_{*,*,v,2}^{k,s} + \sum_p (W_{*,*,v,2}^{k,s,p} - W_{*,*,v,2}^{k,s}), \forall p : W_{*,*,v,2}^{k,s,p} \leftarrow W_{*,*,v,2}^{k,s}$
end for

Every part of the algorithm is the same except for partitioning and synchronization. The synchronization step of global variables is necessary because in each process they are updated differently; after every iteration, they should be synchronized to minimize error.

CGS is especially easy to parallelize because many of the variables are marginalized over. In TSLDA, each sampling step depends only on count variables related to the document it is sampling for, excluding the two global count variables. These count variables $W^{k,*}_{*,*,v,1}$ and $W^{k,s}_{*,*,v,2}$ are also very big because they are summed over all sentences. Therefore, the difference between the actual conditional probabilities calculated from single process and the approximate probabilities calculated from multiple processes is minuscule.

4.2.4 Parameters

There are multiple hyperparameters for TSLDA: $\alpha, \beta, \gamma, \lambda$, and K. Similar to LDA, Dirichlet hyperparameters $\alpha, \beta, \gamma, \lambda$ determine how topics, sentiments, and words are distributed in the given corpus. They can be symmetric or asymmetric; if $\boldsymbol{\alpha} = [\alpha, \cdots, \alpha]$ for some α, it is a symmetric Dirichlet prior. (same applies to other parameters) If the parameters are symmetric, the size of parameters determine how concentrated a topic/sentiment/word distribution is. For higher values of parameters, the corresponding distribution is broader; lower values lead to topic/sentiment/word distributions that are more focused on a specific entry. For example, higher β creates a model in which each document is likely to have a mixture of many topics; lower β leads to a model in which each document focuses on a single or very few topics. In this model, symmetric Dirichlet hyperparameters are assumed; the values used are $\alpha = 0.1$, $\beta = 0.01$, $\gamma = 0.01$, and $\lambda = 0.1$, as noted in [17].

In the case of asymmetric hyperparameters, index with higher parameter is focused more. For instance, if $\boldsymbol{\gamma} = [10, 1, 0.1]$, documents are more likely to have sentiment 1, which has highest γ value. Although asymmetric hyperparameters lead to models with more flexibility, symmetric parameters are more suitable to use for StockTwits data, which is a collection of very short messages rather than long documents.

K is another hyperparameter to be selected. The best way of picking the best K is to use a validation set to pick the K that returned the best result. Because we are interested in using the output topic-sentiment distributions to make a prediction using a Echo State Network, the complete model is a predictive model and thus we can validate the performance of the model. After TSLDA infers topic-sentiment distributions, the input corpus is divided into training, validation, and test set. Training set is used to train predictive model, (ESN) and test set is used as input for the predictive model to see how well the model predicts stock price movement. Because test set is used to evaluate the performance of the model, it is not the best set to use to determine K. Therefore, validation set is used to pick the K that performed the best when tested in validation set.

4.3 Echo State Network

4.3.1 Overview

Echo State Network (ESN) is a special type of Recurrent Neural Network in which certain nodes of the network function as 'Echo States', which was developed in [9]. The overall structure of ESN is illustrated in figure 18. It is specified by a network of nodes, or reservoirs. Input variable $\boldsymbol{U} = \{\boldsymbol{u}(1), \cdots, \boldsymbol{u}(T)\}$, indexed in time. Similarly, the output variable is also a time series: $\boldsymbol{Y} = \{\boldsymbol{y}(1), \cdots, \boldsymbol{y}(T)\}$. There are four matrices that determine how every node, including the input and output node, is related to others: W^{in}, W, W^{fb}, and W^{out}. W^{in} characterizes relationship between $\boldsymbol{U}$ and $\boldsymbol{X}$. Similarly, W affects how $\boldsymbol{x}$ affects its future values, and W^{fb} affects how $\boldsymbol{y}$ affects $\boldsymbol{x}$. Figure 18 shows which aspects of the network the four matrices control. W^{in}, W, and W^{out} are initialized with random numbers. Then, weight of each reservoir is calculated, or 'activated', by $\boldsymbol{U}$ and the three fixed matrices. After all reservoirs have been calculated, W^{out} is estimated using a least squares method. ESN is a very simple type of Recurrent Neural Network, but it can easily represent and capture complex relationships that exist between different nodes.

ESN is used widely in signal processing, where the future values of a signal is interpo-

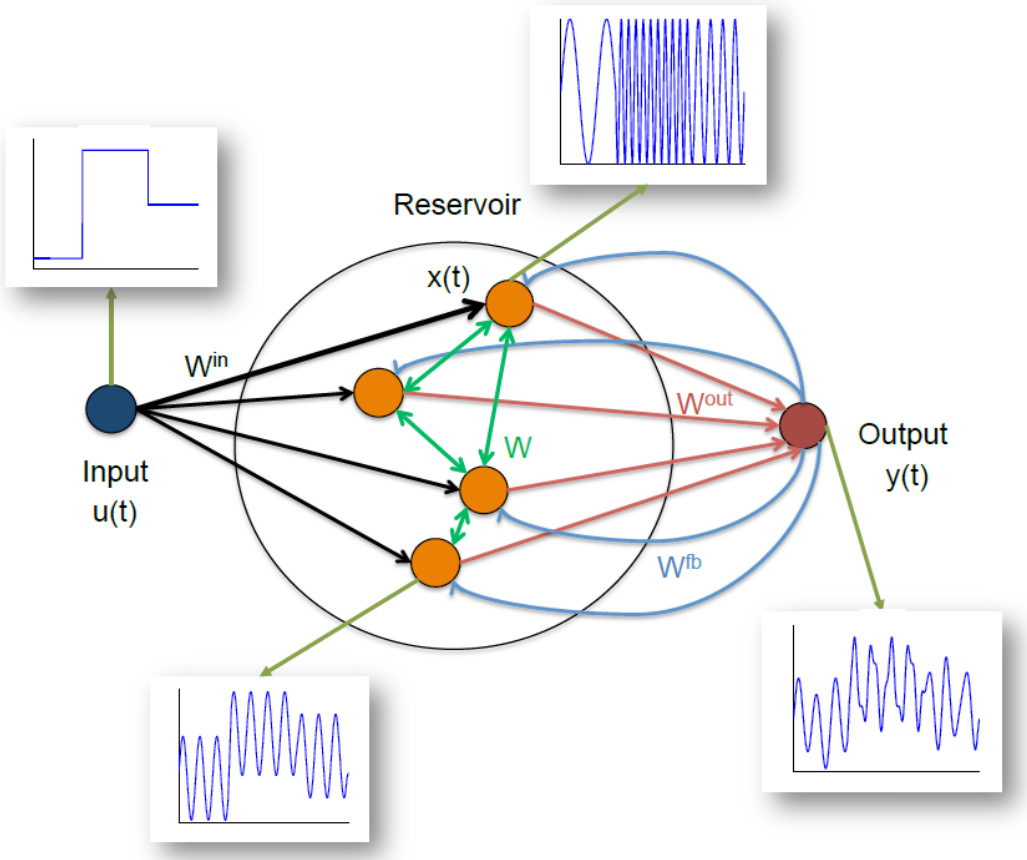


Figure 18: Echo State Network structure

lated from previous values. Furthermore, its inherent ability to model a complex system makes it a suitable prediction model for stock price movement.[4]

4.3.2 Model

The most important equations for this model are the relationships between the matrices and nodes:

$$\boldsymbol{x}(t) = (1-\alpha)\boldsymbol{x}(t-1) + \alpha\overline{\boldsymbol{x}}(t) \tag{6}$$

$$\overline{\boldsymbol{x}}(t) = f(W^{in}\boldsymbol{u}^1(t) + W\boldsymbol{x}(t-1) + W^{fb}y(t-1)) \tag{7}$$

$$\boldsymbol{y}(t) = W^{out}\boldsymbol{z}(t) \tag{8}$$

$\boldsymbol{X} = \{\boldsymbol{x}(1), \cdots, \boldsymbol{x}(T)\}$ are the reservoir weights; $\boldsymbol{u}^1(t) = [\mathbf{1}, \boldsymbol{u(t)}]^T$ is $\boldsymbol{u}$ augmented with 1 to include a bias term; $\boldsymbol{z}(t) = [\mathbf{1}, \boldsymbol{u(t)}, \boldsymbol{x(t)}]^T$ is the fully augmented vector that holds every node information; $f(\cdot)$ is an activation function, usually a sigmoid or tanh function. In this paper, $f(\cdot) = \tanh(\cdot)$.

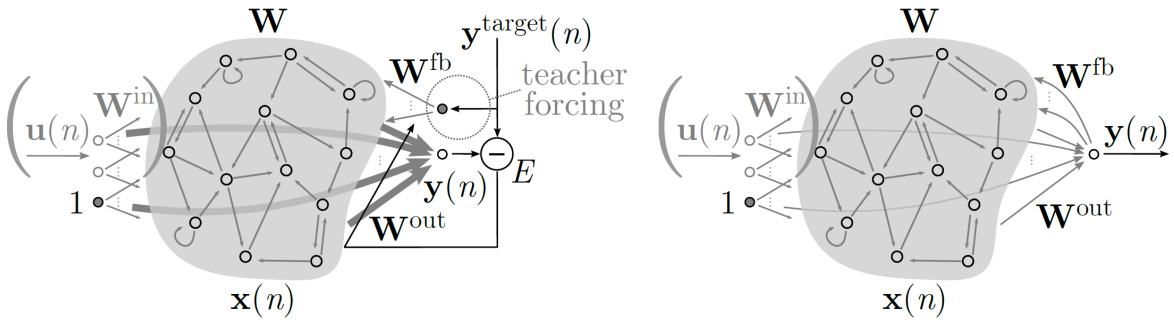
The fitting process for ESN is straightforward. First, W^{in}, W, W^{fb}, and W^{out} are initialized to matrices with random numbers. With these matrices and input variable $\boldsymbol{U}$, reservoir weights $\boldsymbol{X}$ are calculated to obtain fully augmented matrix $\boldsymbol{Z}$. Then a least squares method is applied to $\boldsymbol{y}$ and $\boldsymbol{Z}$ to estimate W^{out}. This can be done by using penalty functions such as mean squared error to minimize the distance between predicted values and true values:

$$MSE(\hat{\boldsymbol{Y}}) = \frac{1}{T}\sum_{t=1}^{T}(\hat{y}(t) - y(t))^2$$

where $\hat{\boldsymbol{Y}} = \{\hat{y}(1), \cdots, \hat{y}(T)\}$ is the predicted value and $\boldsymbol{Y} = \{y(1), \cdots, y(T)\}$ is the true value. Usually ridge regression is the preferred least squares method, because it controls the number of variables to prevent W^{out} from becoming unstable. Despite its simple fitting process, ESN is able to capture a complex relationships, such as those observed in stock markets.

ESN captures non-linear relationship between the input and output variables by letting reservoir variable $\boldsymbol{X}$ and output variable $\boldsymbol{Y}$ to affect the temporal evolution of the predicted values. Specifically, $\boldsymbol{X}$ plays a big role in ESN. First of all, it allows the model to use a non-linear expansion, $\boldsymbol{X}$, of the input variable $\boldsymbol{U}$. This expansion is a higher-dimension representation of the input variable, which may be linearly separable even when $\boldsymbol{U}$ is not. This allows us to model the movement of input variable using a hidden non-linear pattern that is hard to capture using just $\boldsymbol{U}$. Furthermore, because $\boldsymbol{x}(t)$ depends on $\boldsymbol{u}(t)$, $\boldsymbol{y}(t-1)$ and $\boldsymbol{x}(t-1)$, it contains temporal evolutionary information of $\boldsymbol{U}$. In time series predictions such as stock price movement prediction, temporal information is very helpful in making predictions about future values, and $\boldsymbol{X}$ enables us to do so.

W^{fb}, called feedback matrix, determines how much $\boldsymbol{Y}$ affects the reservoirs. This matrix is not always included, because it can easily make the model very convoluted and unstable; however, in some cases addition of a feedback matrix increases the prediction



(a) Training with teacher forcing (b) Running in a generative mode

Figure 19: (a) Using $\boldsymbol{Y}^{target}$ instead of the true value $\boldsymbol{Y}$ to train ESN with feedback matrix (b) Generating outputs using ESN trained using teacher forcing

accuracy if the inherent data structure has some type of feedback loop from the past realized values of a variable. In stock markets, change in stock price strongly and rapidly influences how the buyers and sellers react. Therefore, it is intuitive to include the feedback matrix in our model.

One caveat is that inclusion of feedback matrix forces one to calculate the estimated value of $\boldsymbol{y}$ from the past. This leads to a recursive algorithm, which can be expensive in terms of both memory and running time. One approximation method that bypasses this recursion is teacher forcing. Instead of estimated $\boldsymbol{Y}$, the true values $\hat{\boldsymbol{Y}}$ are used during training phase. Because $\hat{\boldsymbol{Y}}$ is close to $\boldsymbol{Y}$, teacher forcing enables one to train ESN with feedback loop in much faster manner without sacrificing accuracy. This method is shown in figure 19; here, $\boldsymbol{y}^{target} = \hat{\boldsymbol{Y}}$, and $\boldsymbol{y} = \boldsymbol{Y}$.

4.3.3 Parameter Selection

There are many different parameters that play roles in ESN. The most obvious and important parameter is N_x, which is the number of reservoirs. Having a large number of reservoirs improve the performance of the model because it is capable of holding more information among the reservoirs. However, it simultaneously entails computationally expensive calculation due to increase in dimension; furthermore, more parsimonious models are preferred in the field of machine learning. Therefore, finding a good compromise between the two is important. Usually, $N_x \approx 1000$ is used.

The random initialization of matrices need to be finely tuned, or otherwise this model can easily resort to erratic behaviors. The main parameters that need to be set are the range of weights, scaling of the matrices, and the mixing coefficient α. Each parameter plays a unique role in how $\boldsymbol{U}$, $\boldsymbol{X}$, and $\boldsymbol{Y}$ are related.

First of all, we set the random initialization of matrices to fall under $Uniform(-a, a)$ distribution centered around 0 for all matrices for some a. Although any distribution can be used, uniform distribution is the most practical choice. This weight initialization step is a common practice in neural network fitting process. Each matrix is then scaled to adjust how closely related each variable is. Because $f(\cdot) = \tanh(\cdot)$ is an increasing function, (7) shows that higher values of W^{in} causes $\boldsymbol{u}(t)$ to have a greater effect on the value of $\boldsymbol{x}(t)$. Similarly, W determines how strongly the past value $\boldsymbol{x}(t-1)$ affects the current value $\boldsymbol{x}(t)$. $1 - sp$ of the entries should be set to zeros, where $sp \in (0, 1]$, so that W is sparse; this improves algorithm speed while keeping performance the same. W^{fb} determines how strongly the past predicted values changes the output variable $\boldsymbol{x}$. Therefore, different scaling for each matrix can lead to various networks. W^{in} and W^{fb} are scaled using arbitrary numbers that are selected from the nature of data and our prior belief of how the model should function. Nonetheless, W should be scaled by its spectral radius; it hold information about how $\boldsymbol{x}(\cdot)$ behaves to its previous values, and is not unique to the given data. Scaling by spectral radius $\rho(W) = \max_{\lambda \in \sigma(W)} |\lambda|$, which is the maximum absolute eigenvalue of W, allows W to have a unit spectral radius. This decreases the breadth of non-zero distribution of values in W so that the desired echo state of nodes is satisfied; large spectral radius can lead to a chaotic system with many fixed points, which breaks the echo state assumptions.

The other important parameter is the mixing coefficient α. This is another parameter that determines how fast $\boldsymbol{x}$ is likely to diverge away from its older value. (6) shows that lower value of α makes temporal transition of $\boldsymbol{x}$ slower; that is, $\boldsymbol{x}$ reacts more slow to changes in other variables. On the other hand, higher value of α implies a system that reacts quicker to changes in other variables. This parameter therefore determines the temporal dynamics of the variables.

The full algorithm for fitting ESN is shown in algorithm 4. The inputs are N_x,

the number of reservoirs, α, the mixing coefficient, r_s, the new spectral radius, sp, the proportion of nonzero entries, s_{in}, the scale for W^{in}, s_{fb}, the scale for W^b, and reg, the regularization coefficient for ridge regression.

Algorithm 4 ESN algorithm

Require: N_x, α, r_s, sp, s_{in}, s_{fb}, reg

1: Initialize W_{in} so that each entry is distributed as $Uniform(-s_{in}, s_{in})$
2: Initialize W_{out} so that each entry is distributed as $Uniform(-s_{out}, s_{out})$
3: Initialize W as a sparse matrix whose $1 - sp$ of entries is 0
4: Scale W by its spectral radius, then scale it using r_s
5: **for** $t \in (1, T)$ **do**
6: Calculate $\boldsymbol{x}(t)$ according to (6) and (7)
7: **end for**
8: Get $\boldsymbol{Z}$ from $\boldsymbol{U}$ and $\boldsymbol{X}$ calculated above
9: Using ridge regression and reg, solve for W^{out} in (8)

To facilitate parameter selection process, grid search method is implemented to automatically find the best parameters. For a given set of parameters, the grid search algorithm checks the performance of the model using those parameters and a penalty function to find the best set of parameters. The ESN class was coded to suit scikit-learn environment to take advantage of GridSearchCV method, which does a grid search for best parameters, with an addition of cross-validation.

4.3.4 Data

We wish to use the topic-sentiment assignments extracted from TSLDA to make stock price movement prediction. For a given stock $\mathcal{S}$, The input matrix $\boldsymbol{U}$ is structured as:

$$\boldsymbol{U} = \begin{bmatrix} ts_{0,0}^{t_1} & \cdots & ts_{0,0}^{t_T} \\ \vdots & & \vdots \\ ts_{K,S}^{t_1} & \cdots & ts_{K,S}^{t_T} \\ ts_{0,0}^{t_0} & \cdots & ts_{K,S}^{t_0} \\ \vdots & & \vdots \\ ts_{K,S}^{t_0} & \cdots & ts_{K,S}^{t_T} \\ p_1 & & p_T \\ \vdots & & \vdots \\ p_{-4} & \cdots & p_{T-5} \\ ma_5^{t_0} & \cdots & ma_5^{t_T} \\ \vdots & & \vdots \\ ma_{20}^{t_0} & \cdots & ma_{20}^{t_T} \\ V_1 & \cdots & V_T \\ p_1^{s\&p} & \cdots & p_T^{s\&p} \end{bmatrix}$$

which is a $(K \times S \times 2 + 12) \times T$ matrix. $ts_{k,s}^t$ indicates the topic-sentiment variable for day t. For each sentence posted on a trading day t that has topic k and sentiment s, we let $ts_{k,s}^t = 1/N_{d,m}$. Then for each topic-sentiment pair (k, s) and a given trading day t, $ts_{k,s}^t \leftarrow ts_{k,s}^t / N^t$, where N^t is the total number of sentences of tweets posted on day t. Hence, $ts_{k,s}^t$ represents the weight of topic k and s on day t.

Other variables are numerical values more closely related to the stock itself. p_t is price of the stock on day t; ma_n^t is n-day moving average of stock price that ends at t; V_t is volume of the stock on day t; $p_t^{s\&p}$ is price of S&P 500 on day t.

This input matrix is used to predict the price of the stock p_{t+1} given topic-sentiment assignments and financial data on day t and $t-1$. Because $ts_{k,s}^t$ are very small compared to the financial data, all input variables are normalized prior to fitting.

5 Results

5.1 Topic and Sentiment Assignment

TSLDA model that uses parallelized Collapsed Gibbs Sampling was implemented using Python. Despite numerous optimizations and using research computer servers provided by Princeton University, (cycles.cs.princeton.edu) the training phase is very slow and memory-consuming. Therefore, only one year span of tweets in 2013 are considered.

Below are results returned by TSLDA algorithm for each stock. Each model was trained only on documents that mention the specified stock. In the following tables, top 10 topic words for topic with most sentences are shown, along with top 10 sentiment words for each sentiment of that topic. A sentence that has top topic assigned to is also presented.

Table 5: $GOOG Price Prediction

Topic words		
$pcln, $spy, comeback, $aapl, evening, star, fitzgerald, cantor, turd, pound		
Sentiment #1 words	Sentiment #2 words	Sentiment #3 words
heeelllooo, unsure, leak, omfggg, falsely, :-)), inevitable, hmmm, wowooo, pardon	freak, lean, constructive, excite, squeal, wa-wa-wa, ahaha, arg, dread, sexy	brutal, =(, ready, unreasonable, haggle, hi, funny, ease, precise, lean
"$aapl will turn low and $goog reverse high look forward to $goog earnings tonight."		

Table 6: $GE Price Prediction

Topic words		
john, crudele, chief, executive, $tts, rally, halfback, tomorrow, power, chairman		
Sentiment #1 words	Sentiment #2 words	Sentiment #3 words
popular, dominate, negative, stable, :\\, silly, strong, right, well, x.x	shucks, awwh, yeah, hell, upbeat, impressive, bloated, right, risky, beautiful	achievable, unlikely, protect, ok, concern, work,retreat, greedy, calm, fearful
"general electric co's senior vice president just pick up 5,135 share http://stks.co/ahbg $ge"		

Table 7: $EURUSD Price Prediction

Topic words		
phase, markdown, bdi, germany's, ad, hammer, va, yesterday's, midpoint, us$		
Sentiment #1 words	Sentiment #2 words	Sentiment #3 words
wedge, lethargic, recommend, hot, risk, durable, whatsa, popular, blatantly, trust	slump, dynamic, loool, noisy, recover, yeahh, :-)), broken, anyways, credible	well-known, phew, interesting, disagree, deceive, flatter, helpful, ready, mad, ignore
"if germany doesn't bend , europe may break $eurusd $macro http://stks.co/b01jd"		

Table 8: $KO Price Prediction

Topic words		
strategy, advertising, $nflx, barron's, $bidu, quicky, style, possibilities, mix, ice		
Sentiment #1 words	Sentiment #2 words	Sentiment #3 words
slow, failed, fav, good, ftw, nevermind, hefty, toxic, sink, kill	steal, glad, classic, lackluster, outperform, good, patiently, cool, explode, ahh	work, yawn, smile, well, ruin, warm, like, love, recommend, wrong
"some name that we be watch : $hca $bidu $sppi $ byi $ko $mpc $bery $hr $lqdt $ ctas"		

Table 9: $BAC Price Prediction

Topic words		
ashi, stanley, morgan, bush, folk, macro, cloud, life, data, union		
Sentiment #1 words	Sentiment #2 words	Sentiment #3 words
fabulous, manageable, threaten, risk, promising, brutal, patient, woot, beautiful, humble	desperate, dang, shatter, regard, comeon, clever, breach, bah, uh-oh, insane	hmm, joke, yeeehaaaw, panic, well, dark, welcome, <3, scratch, excuse
"$bac morgan stanley say : " $16 price target with rise bias towards our $20 bull case ""		

Table 10: $MSFT Price Prediction

Topic words		
signal, subscriber, mil, canadian, gov't, success, tablet/laptop, mister, $key, mode		
Sentiment #1 words	Sentiment #2 words	Sentiment #3 words
protect, limit, arghhh, rofl, favorable, calm, healthy, scary, competitive, mistakenly	envious, receptive, both, tank, spoil, quiet, exploit, loool, wrong, wow	expensive, um, undone, concern, aggressive, wrong, giddy, hmm, fresh, mess
and $msft have much deep pocket in the indian gov't .		

Although the correct topic and sentiment of a sentence are subjective, topic/sentiment word assignments shown in tables above seem accurate. We see many adjective and adverb words, as well as emoticons and interjections. Furthermore, sentiment words in same sentiment group share similar connotation. For instance, in tweets that talk about $GOOG, sentiment 1 generally contains positive words, while sentiment 3 contains more negative words. Some topic words indicate a major topic that many people have been talking about on StockTWits; there are topic words, such as 'strategy' and 'advertising' in tweets about $KO, that refer to certain information about the company of interest that may affect the stock price movement. Furthermore, there are many other stocks that are mentioned in tweets. For instance, $aapl is a topic word in tweets about $GOOG. The example sentence shows that StockTwits users like to talk about multiple stocks that are related at the same time. There were also some tweets such as the one shown in table for $KO that contains many stocks without adding too much detail.

5.2 Stock Price Movement Prediction

Using the estimated topic-, sentiment-, and word-distributions and other information about a given stock, ESN aims to predict the price of the stock on the next trading day. ESN has been trained and tested using the 7 stocks discussed in previous section, excluding $eurusd. The actual stock price time series and the predicted time series are shown in Figures 20 - 23. Each plot is created using the best set of hyperparameters

selected by grid search algorithm. We can notice that most of the predicted values have a small lag from the actual price, but the overall pattern of the price movement is very similar.

Performance of ESN was compared to Support Vector Regression (SVR). Table 11

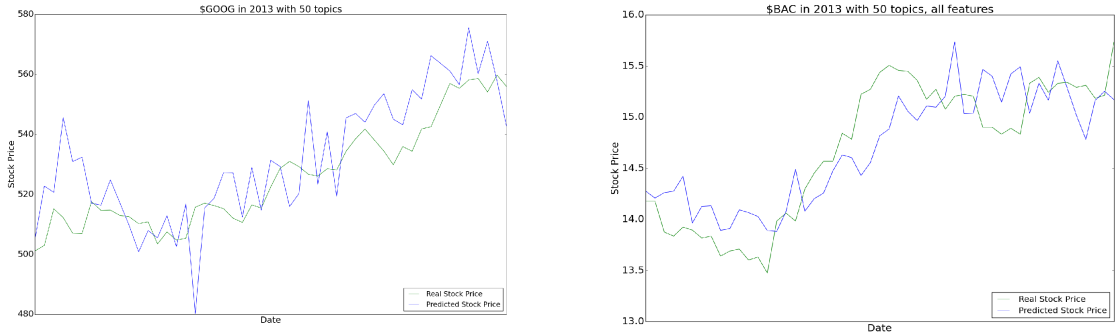


Figure 20: Prediction of $GOOG and $BAC

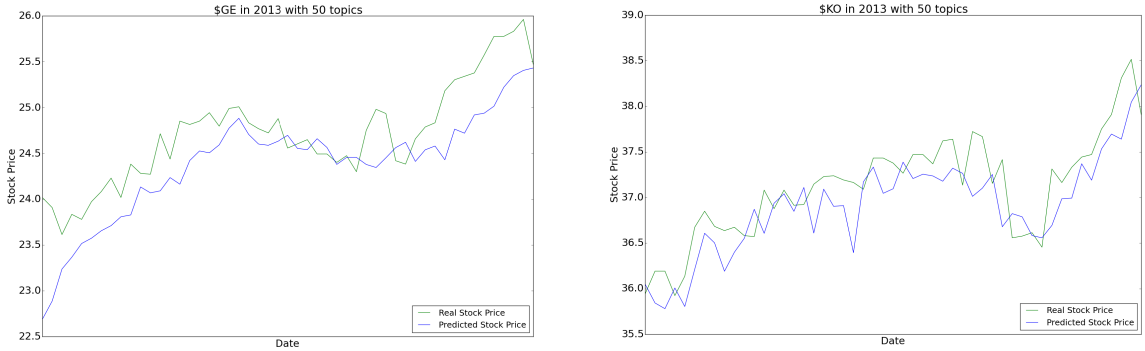


Figure 21: Prediction of $GE and $KO

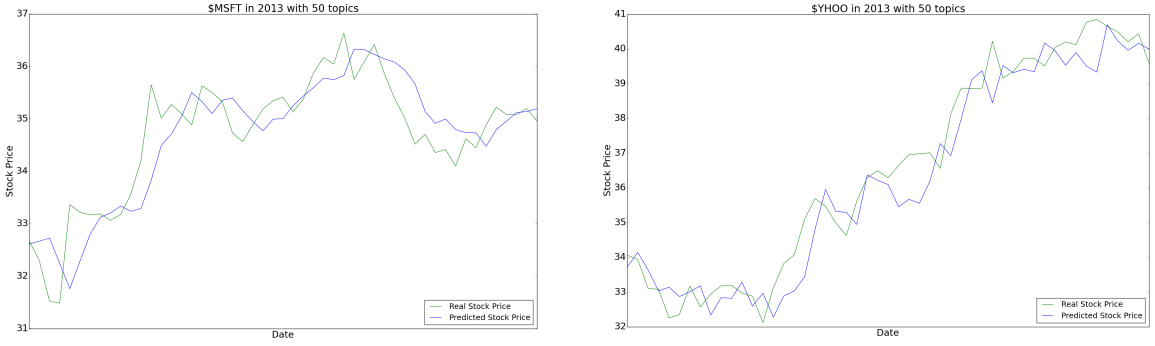


Figure 22: Prediction of $MSFT and $YHOO

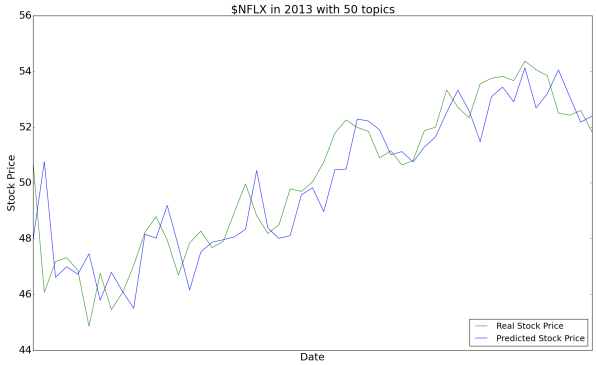


Figure 23: Prediction of $NFLX

Table 11: Prediction Performance Comparison

Stock	$GOOG	$BAC	$GE	$KO	$NFLX	$YHOO	$MSFT
MSE_t	10.51	0.371	0.449	0.347	1.251	0.748	0.603
MSE_s	94.89	2.077	3.533	1.016	18.94	11.14	4.99
P_{avg}	884.24	13.45	22.01	39.66	246.90	27.74	32.49
MSE_t/P_{avg}	0.0119	0.0276	0.0204	0.00875	0.00507	0.0270	0.0186
MSE_s/P_{avg}	0.107	0.154	0.147	0.0256	0.0767	0.402	0.154

shows the mean-squared error of values predicted by ESN, mean-squared error of valued predicted by SVR, average price of the stock over 2013, and ratios of MSE-to-average price. Because each stock has different price, ratio of MSE/ESN is a better method of evaluation than just MSE. MSE_t is MSE of ESN and MSE_s is MSE of SVR. $NFLX, $KO, and $GOOG were the stocks with top three lowest MSE-to-average price ratio for ESN. We see that ESN outperforms SVR for every stock. Although support vector machines are good at handling high-dimensional data, they are not able to capture the complex dynamics of stock prices.

Examining the structure of W^{out} produced by ESN allows us to understand which features are important and which ones are not. Figure 24 shows heatmaps of topic-sentiment features produced by TSLDA algorithm for $GE. White indicates high positive value, pink indicates zero value, and black indicates negative value with high absolute value. Row indexes are sentiment indexes and column indexes are topic indexes; the top heatmap is features for current date t and the bottom heatmap is features for previous day $t-1$. We see some interesting pattern in how the numbers are distributed. For

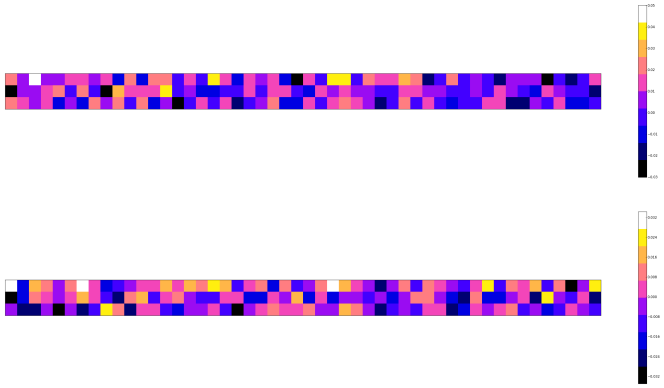


Figure 24: Heatmaps of topic-sentiment features of $GE

instance, the first topic in bottom heatmap shows a nice mix of positive value, zero, and negative value. This topic is likely to have a very strictly divided sentiments that correspond to positive, neutral, and negative. Furthermore, the top sentiment seems to be the positive sentiment, because it has many high positive values. On the other hand, the bottom sentiment has many negative values, implying that it represents the negative sentiments.

Table 12 shows values of W^{out} that correspond to the financial features of stocks. Current price p_t seems to have the biggest positive influence on the predicted price. Price of S&P seems to have a very small negative correlation with price of $GE. we also note that ma_5 has values of 0 in its corresponding index in W^{out}; this is because ridge regression was used in fitting W^{out}. Ridge regression is a type of regularized linear regression, and it automatically selects parameter space that is as sparse as possible. Therefore, ma_5 has been excluded by ridge regression, because all information that ma_5 has is already provided by $\{p_t, \cdots, p_{t-5}\}$.

Table 12: Values of W^{out} for each financial feature

p_t	p_{t-1}	p_{t-2}	p_{t-3}	p_{t-4}	p_{t-5}
0.154	0.121	0.0992	0.0859	0.0851	0.0732
ma_5	ma_{10}	ma_{15}	ma_{20}	V	$p_t^{s\&p}$
0	0.109	0.0839	0.0751	1.84e-22	-0.00342

6 Conclusion

In this thesis, TSLDA and ESN have been closely studied, and they have been combined to use text information from StockTwits to make prediction of stock price. The sampling equation for TSLDA has been derived, and a parallelized extension of the model has been developed. pCGS performed much faster while virtually having no effect on the values inferred. TSLDA with pCGS has been trained on tweets from 2013 that talk about $GOOG, $BAC, $GE, $EURUSD, $KO, $MSFT, $YHOO, and $NFLX. The topics and sentiments that were inferred seemed to represent the StockTwits users' sentiments correctly. These features were then conjoined with financial information about the stocks to be used as inputs of ESN. Price movement pattern predicted by ESN was very similar to that of the actual price, but there is a small lag that exist in many cases; however, ESN still outperformed SVR by almost ten-fold.

These two models were selected because we wanted to explore the relationship between textual information and stock price, while taking the complex and chaotic dynamics of stock market into account. Another improvement of this model can come from how we connect the two models; for instance, some type of feature engineering that creates the most optimal features for ESN from topic-sentiment assignments can be explored. Inter-stock dependency can also be incorporated into the model. Many tweets talk about more than one stock, and taking these into account could improve the model performance, because many stocks are correlated with each other in the market.

References

[1] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau. Sentiment analysis of twitter data. Department of Computer Science, Columbia University, New York, NY 10027 USA.

[2] S. Baccianella, A. Esuli, and F. Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *in Proc. of LREC*, 2010.

[3] L. Banasiak. yahoo-finance 1.2.1. PyPI.

[4] A. Bernal, S. Fok, and R. Pidaparthi. Financial market time series prediction with recurrent neural networks.

[5] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3, 2003.

[6] J. Bollen, H. Mao, and X. Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2011.

[7] W. Darling. A theoretical and practical implementation tutorial on topic modeling and gibbs sampling, dec 2011.

[8] L. Hong and B. Davison. Empirical study of topic modeling in twitter, 2010.

[9] H. Jaeger. The echo state approach to analysing and training recurrent neural networks - with an erratum note, jan 2010.

[10] M. Lachanski. Stocktwits dataset.

[11] E. Loper and S. Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[12] M. Lukosevicius. A practical guide to applying echo state networks. *Neural Networks: Tricks of the Trade, Reloaded*, 2012.

[13] P. McNelis. *Neural Networks in Finance: Gaining Predictive Edge in the Market*. Elsevier, 2005.

[14] Q. Mei, X. Ling, M. Wondra, H. Su, and C. Zhai. Topic sentiment mixture: Modeling facets and opinions in weblogs. In *WWW '07 Proceedings of the 16th international conference on World Wide Web*, pages 171–180. ACM, 2007.

[15] Q. Mei and C. Zhai. Discovering evolutionary theme patterns from text - an exploration of temporal text mining. Research Track Paper.

[16] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed inference for latent dirichlet allocation, 2008.

[17] T. Nguyen and K. Shirai. Topic modeling based sentiment analysis on social media for stock market prediction. School of Information Science, Japan Advanced Institute of Science and Technology.

[18] O. Owoputi, B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. Smith. Improved part-of-speech tagging for online conversational text with word clusters.

[19] B. Pang and L. Lee. *Opinion mining and sentiment analysis*, volume 2, chapter 1-2. now, 2008.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[21] J. Si, A. Mukherjee, B. Liu, Q. Li, H. Li, and X. Deng. Exploiting topic based twitter sentiment for stock prediction. *ACL*, 2013.

[22] Princeton University. About wordnet. http://wordnet.princeton.edu, jan 2010.

[23] F. Wong, Z. Liu, and M. Chiang. Stock market prediction from wsj: Text mining via sparse matrix factorization. *arXiv*, 2014. arXiv:1406.7330.

[24] F. Yan, N. Xu, and Y. Qi. Parallel inference for latent dirichlet allocation on graphics processing units.