JOYCE, BRANDON, M.S. Sentiment Analysis and Review Ranking Using Naive Bayes with Weights in Online Social Networks. (2019)
Directed by Dr. Jing Deng. 51 pp.

Online reviews are critical in many aspects, for business as well as customers. Yet the accuracy and trustworthiness of these reviews are usually unsubstantiated and little research has been performed to investigate them. For the 2016 US Presidential election, many people expressed their likes or dislikes for a particular presidential candidate. Our aim was to calculate the sentiment expressed by these tweets, and then compare this sentiment with polling data to see how much correlation they share. We used a lexicon and Naive Bayes Machine Learning Algorithm to calculate the sentiment of political tweets collected one-hundred days before the election. We used manually labeled tweets as well as automatically labeled tweets based on hashtag content/topic. Our results suggest that Twitter is becoming a more reliable platform in comparison to previous work. Furthermore, we use a set of Yelp reviews on various topics (food, hotel, etc.) as an example to perform sentiment analysis and investigate the correlation between review comment sentiment and its numeric rating. We used feature selection techniques to statistically remove redundant words from reviews, thus improving run time and accuracy. Our method gives higher weight to those terms/words appearing in reviews with more useful votes. These techniques combined with Naive Bayes approach achieves an overall accuracy of 75%. More interestingly, our method is shown to perform well in 1-star and 5-star reviews, with 92% accuracy for the latter. With such a strong accuracy, we argue that the proposed sentiment analysis technique can be used to shed light on all online comments, especially those without numerical ratings.

SENTIMENT ANALYSIS AND REVIEW RANKING USING NAIVE BAYES WITH WEIGHTS IN ONLINE SOCIAL NETWORKS

by

Brandon Joyce

A Thesis Submitted to
the Faculty of The Graduate School at
The University of North Carolina at Greensboro
in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Greensboro
2019

Approved by

Committee Chair

To my parents for their continual support of my academic career.

APPROVAL PAGE

This thesis written by Brandon Joyce has been approved by the following committee of the Faculty of The Graduate School at The University of North Carolina at Greensboro.

Committee Chair _____
Jing Deng

Committee Members _____
Nancy Green

_____
Minjeong Kim

_____
Date of Acceptance by Committee

_____
Date of Final Oral Examination

ACKNOWLEDGMENTS

I would like to thank my advisor, Jing Deng, for helping me to prepare this Thesis. His experience and advise were invaluable; this allowed me to also submit a conference paper for peer review.

PREFACE

Much of the researched I surveyed for sentiment analysis was complicated and not intuitive (for a novice at least). I was curious if I could simplify the methodology and achieve similar or better results.

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER

I. INTRODUCTION 1

I.1. Overview 1

I.2. Development of Social Networks 2

I.3. Early Text Sentiment Research 3

II. RELATED WORK 4

II.1. Twitter Social Network 4

II.1.1. Sentiment Analysis for Previous Elections. 4

II.1.2. Sentiment Analysis using Emoticons and Hashtags 4

II.1.3. Other Sentiment Analysis Techniques 5

II.2. Yelp Social Network 6

II.2.1. Previous Research vs. Our Research 7

III. TWITTER SOCIAL NETWORK 9

III.1. Overview 9

III.2. Methodology 10

III.2.1. Lexicon 10

III.2.2. Naive Bayes Algorithm. 11

III.2.3. Sentiment Scoring 11

III.3. Experiments and Results. 12

III.3.1. Hashtag Sentiment 14

III.3.2. Correlation to Opinion Polls 15

IV. YELP SOCIAL NETWORK 19
IV.1. Overview 19
IV.2. Sentiment Analysis Methodology 19
IV.2.1. Data Preparation 19
IV.2.2. Naive Bayes Algorithm. 20
IV.2.3. Negating Words. 21
IV.2.4. Global and Local Words. 21
IV.2.5. Information Gain (IG) 23
IV.2.6. Simplifying IG for a Two-Class Dataset. 23
IV.2.7. Weighting Reviews 25
IV.2.8. Tested Techniques. 27
IV.3. Performance Evaluation. 27
IV.3.1. Algorithm Complexity and Runtime 33
V. CONCLUSIONS AND FUTURE DIRECTIONS 34
BIBLIOGRAPHY 36
APPENDIX A. NUMERIC RESULTS OF SELECTED TESTS 40
APPENDIX B. SELECTED CODE SNIPPETS. 46

LIST OF TABLES

Page

Table IV.1. Word Sentiment Magnitude: Top 10 Pos. & Neg. Words 28

Table IV.2. List of Negation Words 28

Table IV.3. Information Gain Words: Top 20 29

Table IV.4. Average Useful Votes & Total Reviews with Each Star Rating 29

Table IV.5. Comparing Our Design with Delta TFIDF 32

Table A.1. Classification Results 40

Table A.2. Test 7 (Non-Weighted Reviews) with Different Values of η_0 43

Table A.3. Test 11 (Weighted Reviews) with Different Values of η_0 43

Table A.4. Delta TFIDF Results 43

Table A.5. Unbalanced Tests 44

Table A.6. Accuracy Per Star Review 44

Table A.7. Accuracy Per Star using WSM 45

Table A.8. Accuracy Per Star using wWSM 45

Table A.9. Accuracy Per Star Review (Delta TFIDF) 45

Table A.10. Accuracy Per Star Review (Unbalanced Reviews) 45

LIST OF FIGURES

Page

Figure III.1. Lexicon Sentiment for Clinton Hashtags. 13
Figure III.2. NLTK Sentiment for Clinton Hashtags. 13
Figure III.3. Lexicon Sentiment for Trump Hashtags 13
Figure III.4. NLTK Sentiment for Trump Hashtags 14
Figure III.5. Polling Data 15
Figure III.6. Trump Tweet Sentiment Scores by Day 16
Figure III.7. Clinton Tweet Sentiment Scores by Day. 16
Figure III.8. Correlation Coefficients Using a Moving Average 17
Figure III.9. Tweet Volume Chart 17
Figure IV.1. Word Cloud for All Yelp Reviews 22
Figure IV.2. Word Cloud for a Local Company. 22
Figure IV.3. Comparison of Different η_0 for WSM & wWSM 30
Figure IV.4. Accuracy for Reviews Per Star Ratings with wWSM & η_0 31
Figure IV.5. Accuracy Results Per Star with wWSM, $\eta_0 = 1$ 32

CHAPTER I

INTRODUCTION

I.1. Overview

Online social networks play increasingly important roles in our daily lives. We "tweet" how we feel on a daily basis. This sentiment can express how we feel about certain topics, ideas, people, etc. Such sentiment could be analyzed in regards to a presidential election to see how users in the social network feels about a particular candidate (usually using "hashtags" to label their tweet's sentiment or category). Furthermore, people use Yelp to choose restaurants and lodging based on user reviews and we also try to provide helpful reviews for others to decide. However, it is unclear how such reviews' sentiments correlate with the review ratings accompanying them. For instance, do all 5-star reviews have the same levels of negative or positive sentiment? Should people disregard all 3-star reviews or maybe make selections solely based on these? When facing several detailed reviews, which one or ones should be trusted more?

These are the questions that motivated us to investigate user reviews and we focused on Yelp reviews due to a number of reasons. Yelp is a popular social media site where users can post reviews about companies, such as restaurants and hotels. Other Yelp users can respond to these reviews, for example, by indicating if the review was helpful or funny. However, before we divulge our research methodology and results for both Twitter and Yelp social networks, we will briefly examine some history of social networks and sentiment analysis.

I.2. Development of Social Networks

Before "online" social networks, people could communicate in an "offline" or real-life social network. An example would be at a workplace. The area of interest is the "pathways" that form in this network. Farace and Danowski [1] described these pathways in an organization. People who communicate more frequently in the organization become "cliques" or "clusters" (example, coworkers in the same department). Other employees become "bridges" between these cliques (example mid-level manager with employees and upper-management). By examining this flow of information and patterns of interactions, a social network could be constructed. Such a network could be a useful tool for management purposes.

However, Farace and Danowski [1] concluded that such "real life" social networks could be hard to construct. With the advent online social networking sites (SNSs), such social networks can be constructed or accessed more easily, at least for academic research purposes. The first major social networking site launched in 1997, called "Six Degrees.com " [2]. Other popular SNSs would eventually follow: MySpace & LinkedIn (2003), Facebook (2004 - Harvard, 2006 - Everyone), YouTube (2005), Twitter (2006). SNSs can be defined by the way they allow users to interact with each other. For example, messaging, publishing content, reacting to content, follow/friend other users, etc. Not all SNSs have these features. For example, some SNSs started as instant messaging or forum services. Furthermore, SNSs can be differentiated by the way they allow users to access information. Some SNSs guard against public access and only allow a user's friends to see their information (e.g. Facebook, MySpace), others give public access to anyone – even someone not in the social network (e.g. Friendster), and some offer paid access to the social network (e.g. LinkedIn).

In addition, a user may be able to customize the audience that can see their content/profile information. Facebook is a popular example of this..

SNSs allow users to meet strangers, but this is not usually the goal (with the exception of dating sites). Boyd and Ellison [2] noted that participants of SNSs (especially the larger sites) may just be re-enforcing "real-life" connections instead of meeting new people, suggesting a link between our online social network and real life social network. However, Boyd and Ellison [2] also noted that many connections on a SNSs may just be "latent ties" with individuals that do not usually network with each other.

I.3. Early Text Sentiment Research

In [3], Graziotin and Kuutila examine the history of sentiment analysis. Before computers, sentiment analysis was restricted to surveys that had to be manually analyzed for content. In addition, these surveys "...were focused on public or expert opinions rather than users or customers' opinions." In 1995, a paper was published using a computer for opinion analysis. In this paper, Sandri and Dubois [4] research experts providing probability distributions and then trying to produce one probability distribution that represents the opinion of the group. They also look at weighting each distributions based on the experts' reliability. The Association for Computational Linguistics also published papers that focused on the textual aspect of sentiment analysis, for example research was published about " ...methods to detected subjective sentences from a narrative..." [3]. Machine learning would eventually be used for sentiment analysis. In 2002, Pang et al. published a paper entitled "Thumbs up? Sentiment Classification using Machine Learning Techniques" that used machine learning algorithms such as Naive Bayes and Support Vector Machine to classify movie reviews [5].

CHAPTER II

RELATED WORK

II.1. Twitter Social Network

II.1.1. Sentiment Analysis for Previous Elections

In [6], O'Connor et al. gathered tweets from the 2008 presidential election that contained the phrase "McCain" or "Obama." They applied a lexicon sentiment analysis technique to the gathered election tweets and compared their results to polling data. They used the subjectivity lexicon from OpinionFinder, which has approximately 1,600 positive words and 1,200 negative words [7]. To compare sentiment scores for tweets to polling data, O'Connor et al. used the ratio of positive to negative tweets for a particular topic (i.e presidential candidate). They achieved a correlation factor of about 44% using this particular method. Similar research using a lexicon classification method was done in [8].

In [9], Jahanbakhsh and Moon used a Naive Bayes Algorithm to classify tweets relating to the 2012 presidential election. They manually labeled 989 tweets to train the classifier. When they compared their results to polling data, they were "mostly in match with [their] Twitter results but with some latency."

II.1.2. Sentiment Analysis using Emoticons and Hashtags

In [10], Pak and Paroubek used the Naive Bayes Algorithm to classify tweets with emoticons (e.g ":)" would be a positive label and ":(" would be a negative label) They achieved 81% accuracy using two classes (positive and negative), instead of three (positive, negative, and neutral). They also removed words with a low entropy value (or words that did not occur uniformly in each dataset set). Go et al. in [11]

had a similar approach with Naive Bayes using emoticons to construct a training dataset, but also preprocessed the data by removing usernames, URL's and repeated letters (so "hungry" and "huuuuuungry" would be used as the same word).

In [12], Nielsen construct a custom lexicon by examine several tweets. They used these words to construct a lexicon that he compared to other popular lexicons, such as ANEW. They employed Amazon Mechanical Turk (AMT, a type of crowd sourcing) to label 1,000 tweets for the comparison. Their results showed that the custom lexicon might have performed slightly better than the ANEW lexicon.

In [13], Wang et al. performed a sentiment analysis on the 2012 presidential tweets using a Naive Bayes algorithm. To create their training dataset, they employed AMT to label tweets for them. They used four categories (positive, negative, neutral, and unsure) and achieved 59% accuracy.

In [14], Wang et al. tried to automatically label tweets based on emotion. They used hashtags labels such as "#happy" and "#sorrow" to train a classier to identify tweets that expressed joy or sadness. They achieved an accuracy as high as 65.65%. In [15], Davidov et al. performed a variety of sentiment analyses using tweets. This included using hashtags as labels to train a classier to identify "focused" sentiment. For example, a hashtag that includes an emotion and a target such as "#tmobilesucks" could be used to calculate the sentiment that users express toward T-Mobile.

II.1.3. Other Sentiment Analysis Techniques

In [16], Bollen et al. use over 9 million tweets from the second half of 2008 to determine the "mood" for that day. They scored 6 different modes for each day by mapping words in a tweet to words associated with different moods. Each tweet would be represented by a mood vector. These vectors would be aggregated to

calculate values for each mood each day. The time series of aggregated mood vectors for each day also took into account the different frequency of tweets for each day. Their research showed for example an increase in the mood "Tension" on election day.

In [17], Amolik et al. performed a sentiment analysis of movie reviews using machine learning algorithms such as Naive Bayes and Support Vector Machine. They used 600 positive, 600 neutral, and 600 negative tweets in their experiment. They also replaced usernames with a generic marker "AT_USER."

In [18], Kolchyna et al. combined lexicon and machine learning techniques by using a lexicon scoring scheme as the input for the machine learning algorithm. A manually labeled set of tweets was used to construct a lexicon. This was accomplished by determine the number of times a word appeared in a positively or negatively labeled sentence. Each word was given a positive and negative sentiment score between 0 and 1. This input was used for experiments with Naive Bayes and Support Vector Machine Algorithms.

II.2. Yelp Social Network

There have been some work in different review sentiment analysis, such as aspect identification in reviews [19], frequently co-occurring entropy [20], opinion mining [21], word vector analysis [22], dimension impacts [23], and big data analytics [24]. More specific works are discussed below:

In [25], Dey et al. used 10,000 movie reviews and 10,000 hotel reviews for two separate experiments. Each dataset had an equal distribution of positive and negative reviews. A Naive Bayes Algorithm was used with a top performance accuracy of about 82% for the movie reviews and 55% for the hotel reviews.

In [26], Bakhshi et al. examined the social evaluation of Yelp Reviews. A Yelp user can interact with a review and vote for the review being Useful, Funny, or Cool.

They found that active and regular members were the highest contributors to high-quality reviews. A strong relationship with the number of reviews a user wrote and the number of useful votes was identified, suggesting that more experienced users write more reviews that are deemed helpful.

In addition, Lei et al. [27] relied on users' social circle/network to calculate the interpersonal sentimental influence of their friends in order to make better predictions for them. Thus, a user's preferences may be related to their social network friends.

In *Feature Selection*, Naive Bayes Classifier assumes that each feature/word is independent. In [28], Uysal first performed global feature selection by implementing either information gain, Gini index, and Distinguishing feature selector on their text dataset. Afterwards, they would perform a local feature selection process. This was done by partitioning the data by class, and using an odds ration metric to determine the top features in each class. These two feature sets were combined and Naive Bayes or Support Vector Machine (SVM) would be used for text classification.

In [29], Salinca achieved an accuracy of about 90% using a Naive Bayes Classifier. Further improvement was achieved with negating words that were preceded/succeeded by a negation word, e.g. "not". Three-star reviews were dropped in order to improve accuracy.

II.2.1. Previous Research vs. Our Research

Compared to these works, our approach differ from the following perspectives. We apply a *linear* feature selection process (no exponents, logarithms, or division of feature frequencies). For example, information gain applies a logarithmic function. We instead argue that by not applying a logarithmic function, we can take advantage of the exaggerated differences of word frequencies. In essence, a word that has a higher frequency appears in more reviews, so even if a less frequent word contains

slightly more information, we will not be able to use it in more reviews. Furthermore, we take advantage of the contextual information in Yelp reviews, such as the number of user votes for a review. Finally, we treat different Yelp categories as local groups. Thus, reviews for a specific category should contain local words that are unique for that category and may contain sentiment that is unique for that local group/category. When they are treated differently within the local groups, more accurate results can be achieved.

The goal of this research is to predict the positive or negative sentiment of Yelp reviews. Our approach is to use a Naive Bayes classifier. We use feature selection techniques to statistically remove redundant words from reviews, thus improving run time and accuracy. We also weight reviews with large useful votes more than those with small or none useful votes.

CHAPTER III

TWITTER SOCIAL NETWORK

III.1. Overview

The US Presidential Election of 2016 was historic for many reasons. The Washington Post called it the "most negative presidential election of our lives" [30]. Many people expressed their feelings for each candidate on different social networks including Twitter, a popular micro-blogging site. Each micro-blog is referred to as a "Tweet" and can be no more than one-hundred and forty characters long. Many tweets also include a label for other Twitter users they are referencing (e.g. @username), and a "hashtag" that usually indicates the topic of the tweet (e.g. #election2016). The growth of Twitter users since the last election suggests that it may have become a more accurate polling tool since the 2012 election. For example, according to Statista, the number of monthly active Twitter users worldwide from 4th quarter 2012 to 4th quarter 2016 grew from 185 million to 328 million [31].

Tweets carry sentiments from their senders and it is important to understand such sentiments. Major events such as presidential elections and users reactions on social networks can be used to understand how users express themselves. Presidential elections are unique in the sense that voters' opinions are usually carefully polled and published. With such data, it is then possible to understand how tweeter sentiments and voter opinions correlate.

Furthermore, twitter users can express their sentiment of a candidate using a hashtag. Since the presidential election is largely dominated by two parties, a hashtag that is used to convey positive sentiment for one candidate might express negative

sentiment for the other candidate and vice versa. For our work, we assume that the majority of tweets that express positive "candidate focused" hashtags plan to vote for that candidate on the day the tweet was posted. Likewise, we assume that the majority of tweets that express negative "candidate focused" hashtags plan to vote for the complementary candidate on the day when the tweet was posted (note, we are ignoring third-party candidates for this analysis in order to assume that our hashtag sentiment is "binary").

III.2. Methodology

III.2.1. Lexicon

We used the OpinionFinder Lexicon [7]. This lexicon contains roughly 1,600 and 1,200 positive and negative words. As in [6], we did not utilize the weak/strong labels the lexicon provided for each word. We combined this lexicon with the lexicon first used in [32] by Hu et al., since this lexicon accounts for some misspellings, which seem to be frequent on social media. After combining both lists, we checked if any words were labeled as positive and negative. If this was the case, we labeled the word as only positive (there were less than one percent of such words). Then, any duplicate words in the list were removed. We also added a few explicit words to this list and labeled them as negative and removed the words "trump" and "trumpet" for obvious reasons. To calculate a sentiment score for a tweet, we counted the number of positive words and subtracted them from the number of negative words. If the result was negative, we labeled the tweet as negative. If the result was positive, we labeled the tweet as positive. Otherwise, the tweet was labeled as neutral and was not used in our lexicon sentiment analysis [10].

III.2.2. Naive Bayes Algorithm

For the Naive Bayes Algorithm (see IV.1 for equation), we labeled 500 negative and 500 positive tweets for both Donald Trump and Hillary Clinton. We labeled tweets from our positive, negative, and neutral tweets calculated during our lexicon analysis. During labeling, we tried to correct any obvious misspellings, as well as isolate key terms from hyperlinks (e.g. "WikiLeaks," "imwithher," etc.). We then used the Naive Bayes Algorithm to classify the tweets we had collected.

However, manually labeling tweets is a very time-consuming process. We attempted to automate the process by using "candidate specific" hashtags that we felt expressed a strong sentiment for a particular candidate. We used the hashtags "#imwithher", "#strongertogether", and "#nevertrump" as positive labels for Clinton and negative labels for Trump. Similarly, we used the hashtags "#draintheswamp", "#lockherup", and "#makeamericagreatagain" as positive labels for Trump and negative labels for Clinton. We randomly labeled twenty thousand tweets in total that contained at least one of the aforementioned hashtags (so duplicate tweets were possibly labeled) and the candidate's name. Thus, there were five thousand positive and negative tweets for each candidate.

III.2.3. Sentiment Scoring

In [6], O'Connor et al. calculated the sentiment score x for a particular day ℓ to be:

$$x_\ell = \frac{\text{count}_\ell(\text{pos. tweets} \wedge \text{topic word})}{\text{count}_\ell(\text{neg. tweets} \wedge \text{topic word})}, \tag{III.1}$$

where the topic word is either "Donald Trump" or "Hillary Clinton." A similar formula is given by O'Connor et al.[6], except that they essentially counted

positive/negative words instead of positive/negative tweets. Thus, a tweet could be labeled as positive and negative. We only label a tweet as positive or negative.

However, this method does not generate very smooth data. O'Connor et al. used a moving average to smooth their data [6]. In essence, a moving average (MA) uses the average of the past k days to calculate a sentiment score on a particular day ℓ. The formula for this moving average is:

$$MA_\ell = \frac{1}{k}(x_{\ell-k+1} + x_{\ell-k+2} + ... + x_\ell) \tag{III.2}$$

This smoothing method helps us to compare our data to opinion polls, since they use similar smoothing techniques.

III.3. Experiments and Results

We collected tweets from July 31st through November 7th (the day before the election) that contained the keywords "Hillary Clinton" or "Donald Trump." For brevity, those tweets containing "Hillary Clinton" will be called "the Clinton tweets" hereafter, and those containing "Donald Trump" are called "the Trump tweets." The script we used to collect tweets utilized the Twitter Search Engine [33]. The script does not guarantee that every public tweet from the specified date range/query is gathered. The fields for each Tweet include id, username, text, date, etc. Unfortunately, the tweets we collected did not include location data. We collected roughly 3,068,000 tweets mentioning Donald Trump, and roughly 4,603,000 mentioning Hillary Clinton (with some possible overlaps) from July 31st-November 7th, 2016. Like O'Connor et al., we used polling data from Pollster.com. [34]. This Poll combines 315 polls from 38 pollsters. The last 100 days of this poll are shown in Figure III.5.

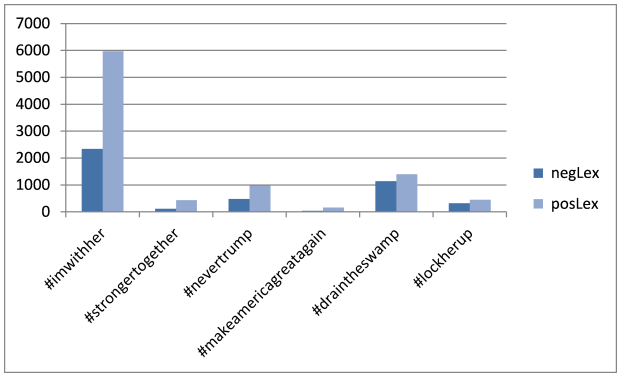

Figure III.1. Lexicon Sentiment for Clinton Hashtags

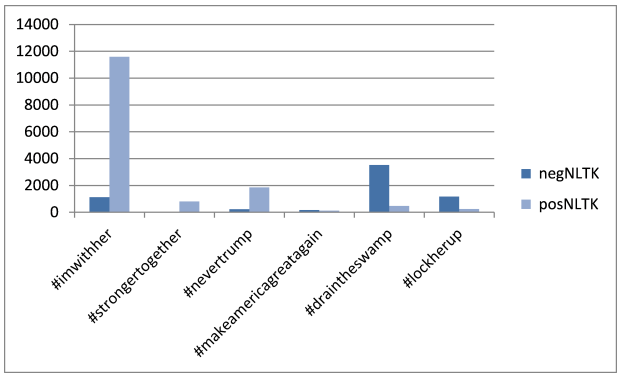

Figure III.2. NLTK Sentiment for Clinton Hashtags

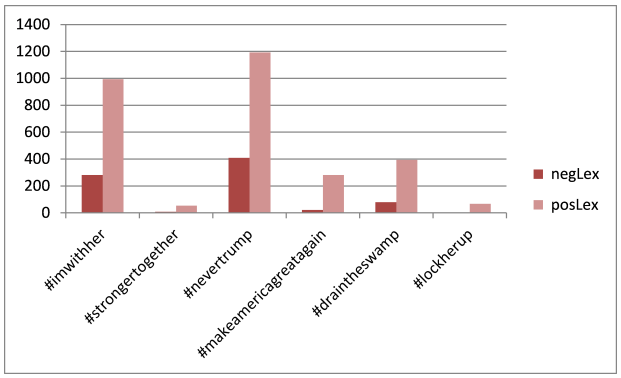

Figure III.3. Lexicon Sentiment for Trump Hashtags

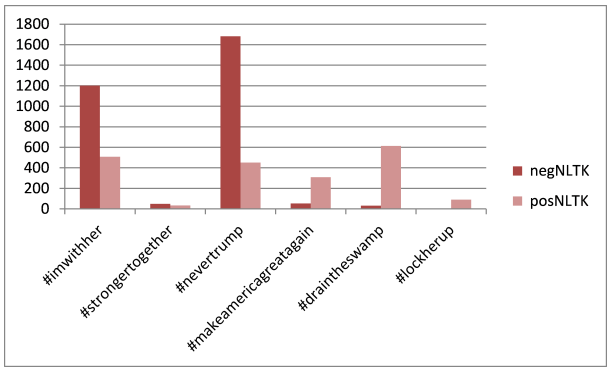

Figure III.4. NLTK Sentiment for Trump Hashtags

III.3.1. Hashtag Sentiment

To help us see how well our Naive Bayes Algorithm was working, we examined tweets that contained popular hashtags. These hashtags were special in that we expected the sentiment of the hashtag to either be positive or negative. For example, we expect the hashtags "imwithher" and "nevertrump" to express positive sentiment for Clinton, but negative sentiment for Trump. However, in Figure III.3, these hashtags in the Trump tweets were mostly labeled as positive instead of negative during our lexicon analysis. We can see that in Figure III.1 the Clinton tweets with these hashtags were mostly labeled as positive. So we can conclude that our lexicon analysis could not identify positive Clinton sentiment as negative Trump sentiment. However, in Figure III.4, the Naive Bayes Algorithm seems to be able to identify positive Clinton hashtags as expressing negative sentiment towards Trump.

It is also interesting to note how the sentiment expressed by the tweets we collected respond to current events. For example, in Figure III.7, there is a large drop in Clinton's sentiment score on October 28th, when former FBI Director James Comey announced that new emails had been uncovered in the Clinton investigation [35].

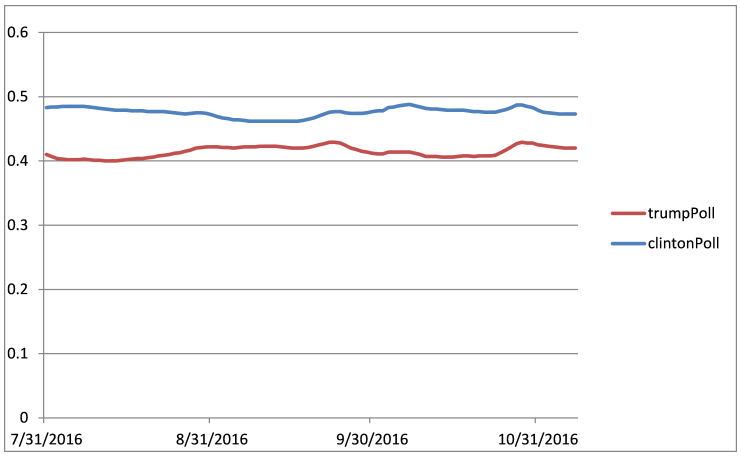

Figure III.5. Polling Data

In Figure III.6, there is a large drop in Trump's sentiment score on October 7th, when a tape leaked regarding Trump's so-called "Locker Room" conversation [36].

III.3.2. Correlation to Opinion Polls

We tested our method for all of the tweets we gathered 100 days before the election. We found a correlation coefficient around 40%-60% for most of our methods (with the exception of the automatically labeled Clinton tweets, which had a negative correlation for $k < 14$). However, 100 days before the election is a "long" time in terms of social media. For example, there were no debates in August 2016, so one can assume that social media was more "quiet" in August than in late September or October. If we focus on the date ranges with more popular events (i.e debates), we should have more tweets (see Figure III.9) and hopefully more accurate results. If we look at all the tweets beginning on the date of the first debate through the day before the election (43 days total), we get very different results.

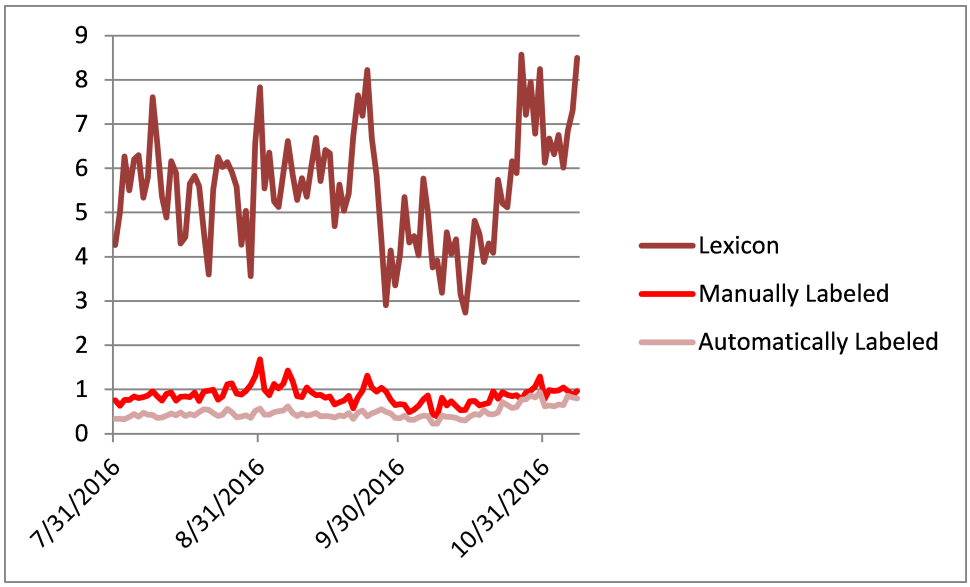

Figure III.6. Trump Tweet Sentiment Scores by Day

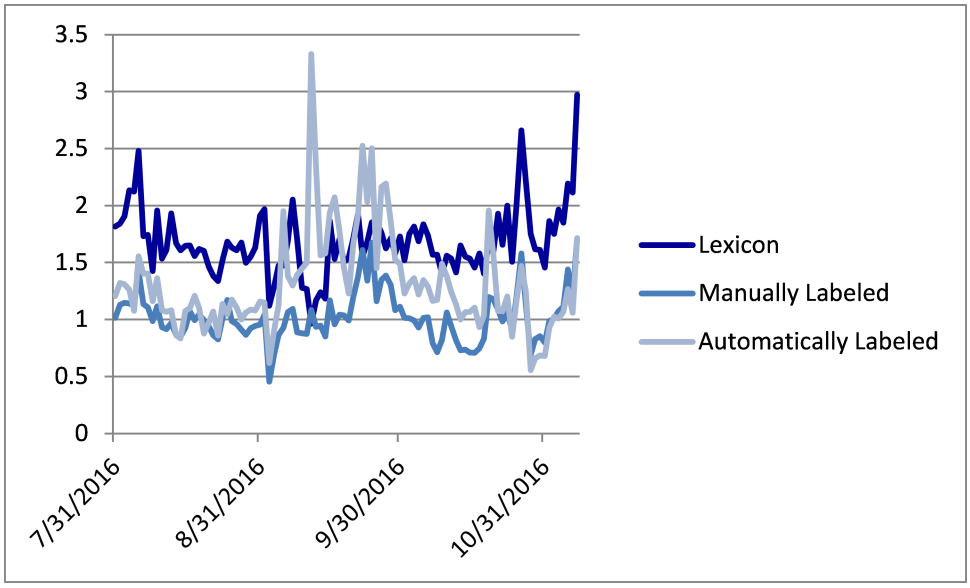

Figure III.7. Clinton Tweet Sentiment Scores by Day

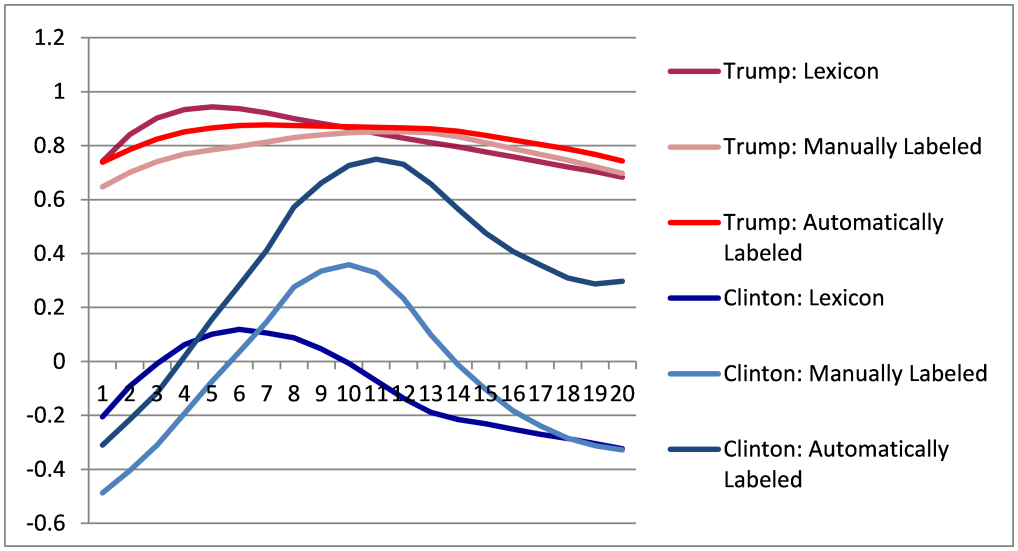

Figure III.8. Correlation Coefficients Using a Moving Average

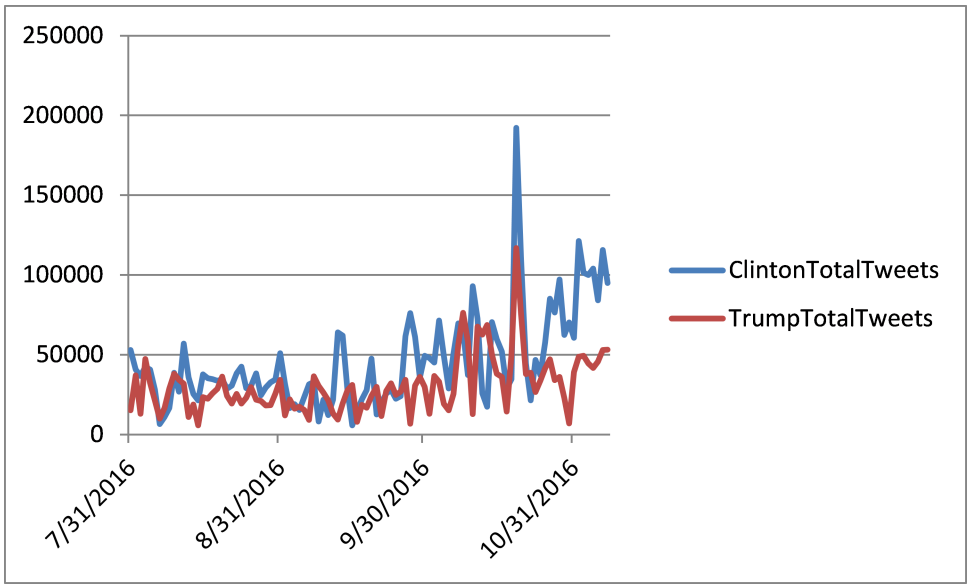

Figure III.9. Tweet Volume Chart

For the Clinton tweets, our Naive Bayes Algorithm had a surprising negative correlation of 48.77% using a window of $k = 1$ and a positive correlation of 35.85% using a window of $k = 10$ (See Figure III.8). For the Trump tweets, our Naive Bayes Algorithm had a positive correlation as high as 85.19% using a window of $k = 12$ (See Figure III.8). For our lexicon analysis, the Trump tweets had a correlation of 94.42% using a window of $k = 5$, and the Clinton tweets had a negative correlation of 20.53% using a window of $k = 1$. Thus, the Trump tweets appear to correlate very well to the polling data we used. The automatically labeled tweets using the hashtags we selected seem to perform better than the manually labeled tweets. The Trump tweets had a correlation of 87.68% using a window of $k = 7$. The Clinton tweets had a correlation of 74.98% using a window of $k = 11$ and this correlation was not negative for $k > 3$ unlike the other two methods. In general, a window of 11-14 seems to produce better results.

CHAPTER IV

YELP SOCIAL NETWORK

IV.1. Overview

Our work with the sentiment analysis of presidential election tweets lead us to ask new questions. Could we reduce the size of our data with feature selection and still achieve similar results? If so, how could we intelligently make our dataset smaller? For these questions, our dataset of presidential election tweets seemed insufficient, since the size of our labeled dataset based on hashtags was very small compared to the entire dataset. Hence, we could use a new labeled text dataset: Yelp Reviews. This dataset also provides the advantage of categories that we could also utilize in our research.

IV.2. Sentiment Analysis Methodology

IV.2.1. Data Preparation

In this work, we used the Yelp challenge dataset, round 13 [37]. First we need to decide what review ratings should be considered as "negative" and "positive". There are altogether 5 simple different ratings, 1-5. Since most likely costumers leaving a rating of 3 is unlikely to have liked or enjoyed the service, we map all reviews with ratings of 1-3 as "negative" and 4-5 as "positive".

For our experiments, a 70/30 data split was used for training/testing purposes. There are 6,685,900 reviews in the Yelp Challenge Dataset, only a small number of which were removed due to bad formatting during the importing process. Then we removed any graphical charters and converted the text encoding to UTF-8. We replaced all special characters with a space, except for "space" and "apostrophe." We

left spaces alone, and replaced "apostrophes" with the empty string. This was done in order to preserve negative words such as "don't" and "won't." All review texts were then converted to lowercase.

We used the R tm package [38] to remove stopwords ("and," "or" etc.), and we also performed stemming, so words like "love," "loving" and "loved" would all be represented by the same word "love."

We used reviews from the top 1000 businesses, for a total of 1,127,333 reviews. When needed to make the number of positive/negative labels equal, we would have 681,414 reviews (there were more positive reviews overall).

IV.2.2. Naive Bayes Algorithm

The Naive Bayes Algorithm is based on the following formula:

$$P(\text{label}|\text{feature}) = \frac{P(\text{label})P(\text{feature}|\text{label})}{P(\text{feature})} \tag{IV.1}$$

Naive Bayes algorithm is used as the baseline in this work. There are two labels in the reviews: positive and negative. The features are the words in the Yelp reviews. Note that we removed stopwords from Yelp reviews. The Naive Bayes Algorithm makes the "naive" assumption that any given word/feature has an independent probability from another word/feature. Thus, the $P(\text{label}|\text{feature})$ for a Yelp review containing n words can be calculated as:

$$P(\text{label}|\text{review}) = \frac{P(\text{label})P(t_1|\text{label})...P(t_n|\text{label})}{P(\text{review})} \tag{IV.2}$$

We used the Natural Language Toolkit (NLTK) to implement the Naive Bayes Algorithm [39].

IV.2.3. Negating Words

Since the Naive Bayes Algorithm assumes that features are independent, it cannot easily distinguish the word "good" in the contexts of "that was very good" and "that was not good." We look at several words preceded by "negation words" (not, no, didn't, won't, etc.) and negate them, so in the previous example "that was not good" would be converted to "that was not NOTgood." This is similar to [29], but we do not (explicitly) negate any words before a negation word, rather directly after a negation word. We also applied this negation as the last step of our feature selection process. We converted words to lowercase, so the uppercase negation would not be found naturally in the dataset. Furthermore, we kept the original negation word to aid in the review being classified as negative, while at the same time the negated word should help prevent the review from being labeled incorrectly, i.e., as positive.

IV.2.4. Global and Local Words

We implemented a similar global and local features selection process as in [28], but instead of partitioning our data by class, we created subsets of "similar reviews," in essence reviews from the same category (food, hotel, transportation, retail, etc.). With the aforementioned process we calculated a global word list N with the entire dataset. We then calculated several local word lists m by performing feature selection on a subset of reviews from a particular category. We then combined these local words lists into a single (unique) word list M that contained all relevant local words from every subset. Finally, we would remove all words from reviews with the exception of $N \cup M$.

To illustrate this, consider the word clouds in Figures IV.1 and IV.2. In the global word cloud of Figure IV.1, we can see words that would be common to all reviews such as "good" and "great." However, let's look at the local word cloud.

Figure IV.1. Word Cloud for All Yelp Reviews

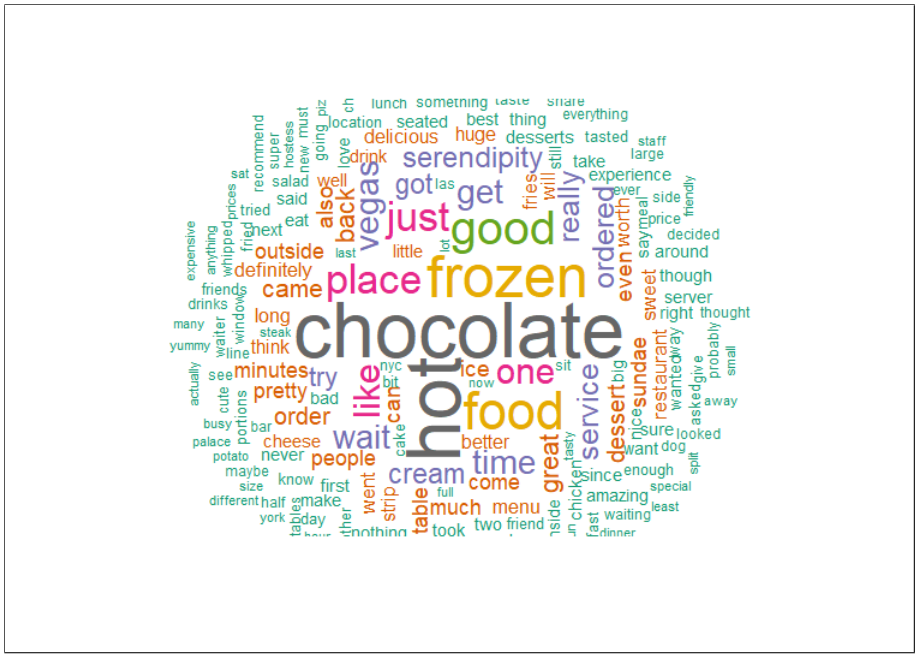

Figure IV.2. Word Cloud for a Local Company

The word "serendipity" (near the top of Figure IV.2 in purple) might be overlooked or not chosen to be one of the N words (for example, N could be the square root of the total number of words). However, it would be included the M word list and thus might improve our sentiment analysis results.

The results for this part of our research were not substantial and have been included in the appendix A.5.

IV.2.5. Information Gain (IG)

We used the information gain formula presented in [28]. The information gain (IG) of a word/term t is defined as the following:

$$IG(t) = P(t)\sum_{i=1}^{M}(P(C_i|t)\log P(C_i|t)) + P(\bar{t})\sum_{i=1}^{M}(P(C_i|\bar{t})\log P(C_i|\bar{t})) \quad \text{(IV.3)}$$

where M is the number of classes (2 in our analysis), C_i represents a particular class (just positive and negative in our study), and $\bar{t}$ represents the probability of the absence of t.

To calculate the information gain, we separated the data into two sets: positive labels and negative labels. We then converted each review into a unique array/vector of words. Then we used R's table function [40] to count the number of times a word appeared in the positive set and the negative set. Using these frequencies, we were able to calculate information gain, IG.

IV.2.6. Simplifying IG for a Two-Class Dataset

The information gain formula as in (IV.3) could be further simplified by just looking at the difference of the number of times a word appears in a positive review

and the number of time a word appears in a negative review. Thus, if a word appeared equally in each class, the difference would be zero and we would know this word possessed no relevant information that could help us distinguish reviews containing them between classes. Likewise, if a word only appeared in positive reviews, then it would have a higher score. However, a word that only appeared 50 times in positive reviews would be weighted less than a word that appeared 1,000 times in positive reviews and only 50 times in negative reviews. We call this difference "word sentiment magnitude", termed η. The difference of this with Delta TFIDF [41] is that we do not use division instead of subtraction, apply a logarithmic function to our quotient/difference of positive and negative words, and we do not scale our results by the total frequency of the word (this occurs naturally in the author's opinion). Furthermore, the Delta TFIDF results presented in [41] assumed balanced datasets, which are not always true. We can also scale the positive or negative class with η, so we do not have to assume that each class contains equally informative word frequencies.

By calculating the word sentiment magnitude, we could rank words more intelligently than just by ranking words by their frequencies. Also, unlike the information gain metric used in [28], we could calculate a list of positive and negative words to form a better distribution of words than information gain that might favor one class of words.

For a given term/word t in a 2-class dataset with an equal number of classes, its word sentiment magnitude is calculated with the following formula:

$$\eta(t) = f^{+}(t) - \eta_0 f^{-}(t) \tag{IV.4}$$

Notice $f^+(t)$ and $f^-(t)$ are the number of times term/word t appears in positive and negative labeled Yelp reviews, respectively, and η_0 is a constant that can be adjusted for different weights between $f^+(t)$ and $f^-(t)$.

We also extended this process to work with uneven classes by scaling the frequency of words that appeared in the smaller class. Assume that all reviews are classified into two different sets, positive set $\mathcal{R}^+$ and negative set $\mathcal{R}^-$, with L^+ is the total number of positive review, $L^+ = \|\mathcal{R}^+\|$, and L^- is the total number of negative reviews, $L^- = \|\mathcal{R}^-\|$, then we could calculate word sentiment magnitude with the following formula:[1]

$$\eta(t) = f^+(t) - \frac{L^+}{L^-}\eta_0 f^-(t) \tag{IV.5}$$

We could calculate the word with the most "positive" impact by looking at the words with the highest word sentiment magnitude. We could calculate the most "negative" impact words by looking at the words with the most negative word sentiment magnitude.

IV.2.7. Weighting Reviews

Furthermore, we could take advantage of useful votes, as in [26] and [27] to support our use of weighting reviews. In [26], their research shows that useful reviews may have higher quality, since they are probably written by active users (as opposed to a non-active, bot, or fake user). In addition, we expect that any user on Yelp might have some friends who interact with him/her on reviews, in addition to other traditional followers, forming a type of social network (the useful votes tell us how many people are in the network, but not who). In essence, we are using an approach

[1] We would ignore the trivial case of $L^- = 0$.

that is reverse of that in [27], in which Lei et al. used the sentiment of friends to make a prediction/suggestion. We take "useful votes" (someone who found the review useful) as an indicator of sentiment for that friend/follower. So, if someone found the review useful, they might share a similar sentiment for that business, and we can weight this review higher (as if several people wrote the same review). Thus, instead of constructing a social friend network (which may be computational expensive), we simply use the useful votes as indicators for that hidden/uncalculated social network.

It follows that we can weight each review by increasing the number of times in appears in the sample. We choose the Yelp review feature "useful votes" as the weight (an integer representing how many people voted for the review as useful). Again, the motivation behind this is that people who found the review useful can "relate" to the review (i.e. they wrote a similar review, had a similar experience, or they believed the sentiment expressed in the review matched the star rating). This should help mitigate fake or irrelevant reviews by increasing the more relevant words that appear in highly useful reviews.

Given a review r with $u(r)$ useful votes, we duplicate this datapoint $\log u(r)$ times. In essence, we are increasing the probability of words that appear in useful reviews. Our assumption is that these words are more meaningful than words that appear in a review without/fewer "useful" votes. Reviews with more "useful" votes should contain words that accurately describe the sentiment of the view (positive or negative). We choose a log function to prevent reviews with extremely high useful scores from dominating the training dataset, which could lead to overfitting and/or extraneous results.

Given a set $\mathcal{R}$ of n reviews that can be partitioned into a set of positive reviews $\mathcal{R}^+$ and negative reviews $\mathcal{R}^-$. We will also use an indicator variables t_r which

will be 1 if a term t appears in review r and zero otherwise. Combining weighting reviews with (IV.5), the weighted Word Sentiment Magnitude becomes:

$$\eta(t) = \sum_{r \in \mathcal{R}^+} [(1 + \ln(1 + u(r))t_r] - \frac{L^+}{L^-}\eta_0 \sum_{r \in \mathcal{R}^-} [(1 + \ln(1 + u(r))t_r] \quad \text{(IV.6)}$$

IV.2.8. Tested Techniques

In this work, we focus on experimenting different combinations of the above techniques. These are discussed below:

First of all, Negating Words are performed through pre-processing. This is important such that word sentiments will not be mis-classified. Word Sentiment Magnitude (WSM) is then investigated because of its potential. This technique takes care of words showing up in highly positive or highly negative reviews and such will be counted. Finally, Useful Votes are used in addition to evaluate ratings on reviews and how these ratings are reflected on review content quality. Combining these techniques, the new sentiment analysis technique is called Weighted Word Sentiment Magnitude (wWSM).

The baseline technique is Naive Bayes with little feature selection. We did remove words that occurred in fewer than 50 reviews (we went from 38,848 total words to 11,059).

IV.3. Performance Evaluation

First, we present the list of words with the highest and lowest Word Sentiment Magnitude, η, as defined in (IV.5). Table IV.1 lists all top-10 positive and top-10 negative words that we identified in the Yelp dataset. We used $\eta_0 = 1$ and

unstemmed words for clarity in this table. While the list of these positive words looks normal, the negative word list does have some surprises: the word "just" is the top negative word and even some of the positive or neutral words are on the list, e.g., "get", "like", and "better."

Table IV.1. Word Sentiment Magnitude: Top 10 Pos. & Neg. Words

positive word	η	**negative word**	η
great	44,194	just	-31,083
delicious	33,956	didnt	-28,984
love	33,290	like	-27,471
amazing	32,906	ask	-26,379
best	25,239	bad	-23,685
perfect	22,194	dont	-23,589
definite	21,372	get	-23,320
favorite	18,179	better	-22,929
awesome	16,899	us	-22,329
recommend	15,235	said	-21,623

Next, the list of popular negation words is shown in Table IV.2. When the negation technique is used, any word preceded by any of these words would have a reversed sentiment (positive to negative, or negative to positive).

Table IV.2. List of Negation Words

wont	wasnt	wouldnt	werent	no
not	never	doesnt	cant	cannot

Information gain has been used in some of the techniques and we only kept words with the highest information gain score (top 1,000 words). Note that an information gain score does not distinguish between positive/negative words. We show the top 20 words with the highest information gain value in Table IV.3.

Table IV.3. Information Gain Words: Top 20

word	**IG**	**word**	**IG**	**word**	**IG**
delicious	0.6035	worst	0.5966	said	0.5947
amazing	0.6023	ask	0.5961	terrible	0.5938
great	0.6009	bad	0.5961	minute	0.5936
love	0.5984	didnt	0.5960	favorite	0.5935
ok	0.5974	nothing	0.5956	disappoint	0.5933
perfect	0.5967	horrible	0.5954	best	0.5932
told	0.5966	rude	0.5947		

In Table IV.4, we summarized the average number of useful votes for reviews with different star ratings. It is interesting to see reviews that gave out lower ratings such as 1-star or 2-star received more useful votes. Five-star reviews received the fewest useful votes. This could have been caused by the fact, when people submitted strong reviews, they simply wrote a few praising sentences. While those reviewers giving out lower star ratings might have felt that more explanations were warranted and thus provided more details, helping them to receive more useful votes. Overall, the average of useful votes for all reviews is about 1.26. Also included on Table IV.4 are the number of reviews in different star ratings.

Table IV.4. Average Useful Votes & Total Reviews with Each Star Rating

	1-Star	2-Star	3-Star	4-Star	5-Star	All
mean	1.67	1.41	1.28	1.27	0.97	1.26
#rev.	73,362	64,429	100,569	90,341	148,019	476,720

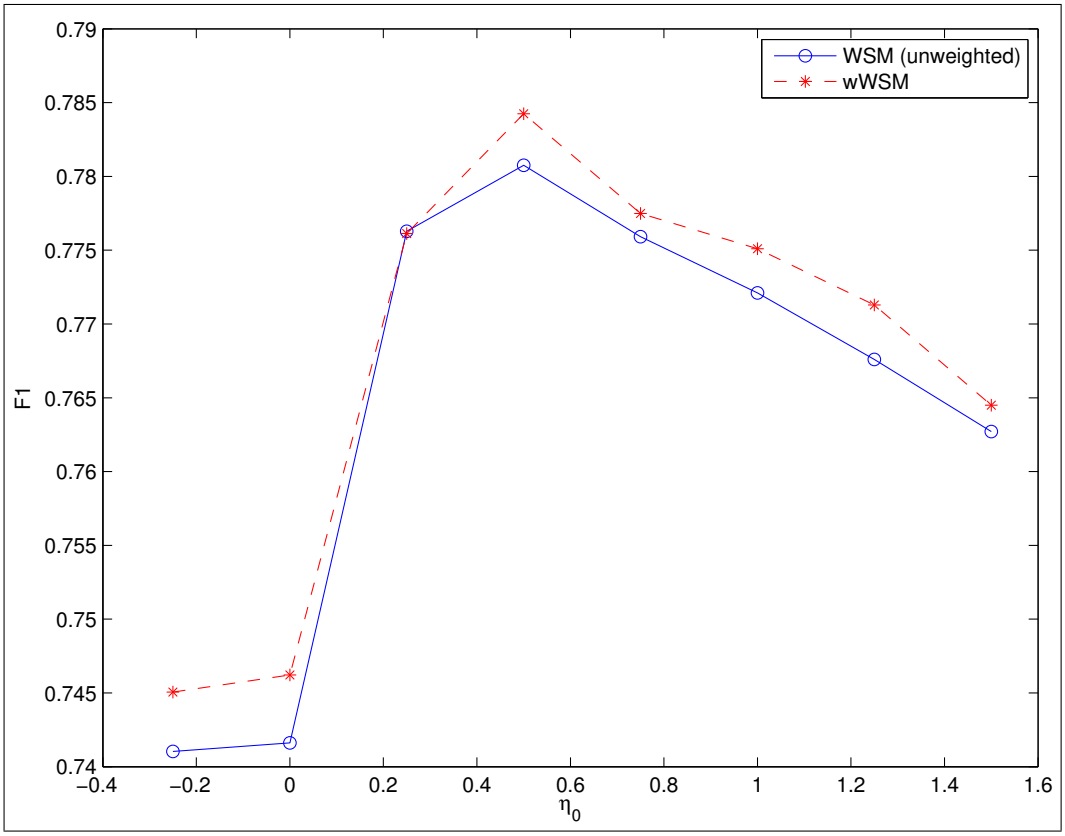

Figure IV.3. Comparison of Different η_0 for WSM & wWSM

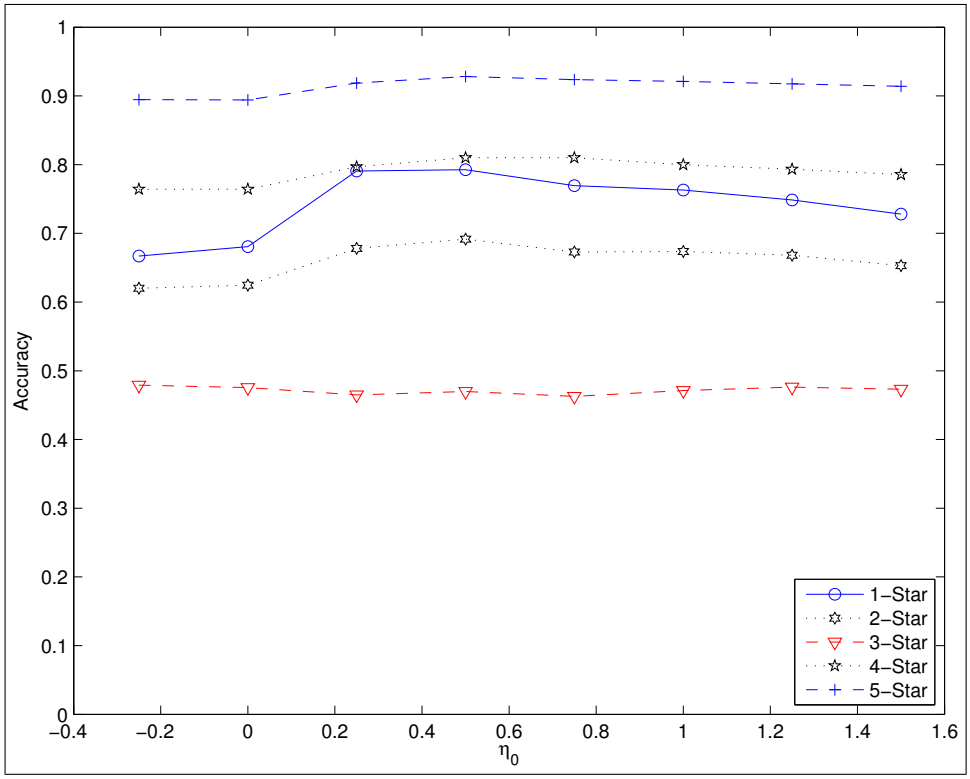

Figure IV.4. Accuracy for Reviews Per Star Ratings with wWSM & η_0

Next, we investigate the impact of different η_0 values. We tried $\eta_0 = -0.25$ to 1.5 with 0.25 gap for two of the techniques that we developed, unweighted WSM and weighted WSM (wWSM). The results are shown in Figure IV.3 (note the small range of y-axis for the purpose of clarity.). The F1 scores of both schemes improve as η_0 increases, until $\eta_0 = 0.5$, at which point the F1 scores peak then decrease with further increase of η_0. Such a change could have been caused by the impact of words showing up in negative reviews with marginal meanings. We will focus on $\eta_0 = 0.5$ for the rest of our investigations.

Similarly, Figure IV.4 shows the accuracy results for reviews with different star ratings when different η_0 values are used. Except for 3-star reviews, all reviews are predicted with better accuracy when $\eta_0 = 0.5$.

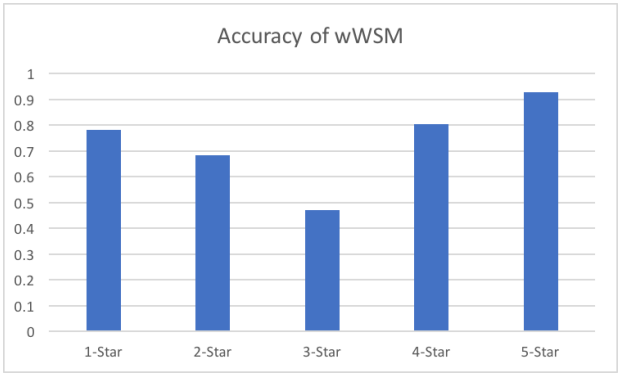

Figure IV.5. Accuracy Results Per Star with wWSM, $\eta_0 = 1$

Table IV.5 presents the comparison between Delta TFIDF and the weighted WSM scheme that we designed. Working on balanced datasets, the weighted WSM scheme does not show much improvement over delta TFIDF scheme. However, in unbalanced datasets, the weighted WSM scheme clearly outperforms Delta TFIDF scheme. For instance, in F1 scores, the weighted WSM scheme achieves about 85%, compared to the 80% performance of Delta TFIDF.

Table IV.5. Comparing Our Design with Delta TFIDF

Dataset	Alg.	FP	FN	TP	TN	F1
Bl'ed	wWSM	0.3707	0.1165	0.8835	0.6293	0.7842
Bl'ed	DT	0.3414	0.1503	0.8497	0.6586	0.7759
Unbl'ed	wWSM	0.3695	0.1198	0.8802	0.6305	0.8543
Unbl'ed	DT	0.9961	0.0010	0.9990	0.0039	0.8037

In Figure IV.5, we compare the Accuracy results for groups of reviews with different star ratings using the weighted WSM scheme. Therefore, all reviews with ratings of each of the 1-5 star ratings are grouped together for wWSM to analyze. Overall accuracy is about 75%. The accuracy for 3-star reviews is significantly lower than other reviews, indicating the vague meanings for such reviews. Therefore, a more accurate classification of 5 classes could raise the prediction accuracy for 3-star

reviews in the wWSM scheme (although some changes are needed for wWSM to function). We discuss this in our future research direction in Section V.

IV.3.1. Algorithm Complexity and Runtime

Overall, the wWSM scheme has a complexity of $O(NT \log U)$, where U is the largest number of useful votes among any review, N is the number of reviews, and T is the number of terms/words in the longest review. Using a computer server equipped with an Intel Xeon E5-2430 v2 2.5 GHz (6 cores) and 16G memory running Ubuntu 16.04.5 LTS, it took about 23 minutes to pre-process (about half of which was removing words with a frequency less than 50) and 54 minutes to run our wWSM scheme. When we increased the size of our training dataset from 476,720 reviews to 605,000 reviews (an increase of 27%), the runtime of wWSM increased from 54 minutes to 68 minutes (an increase of 17%.) In comparison, the baseline Naive Bayes technique needed 55 hours.

CHAPTER V

CONCLUSIONS AND FUTURE DIRECTIONS

We have investigated the accuracy of a lexicon sentiment analysis and a machine learning sentiment analysis when the results are compared to polling data. Even though the Naive Bayes Machine Learning Algorithm seemed to do well identifying sentiment associated with particular hashtags, it did not outperform the lexicon analysis as anticipated when compared with Trump polling data. However, the automatically labeled tweets outperformed the manually labeled tweets for both candidates and have better accuracy when compared to the Clinton lexicon analysis. Thus, the automatic method saves man-hours, improves accuracy, and removes any potential bias that could occur when the tweets are being manually labeled. The very high correlation coefficient with our trump tweets suggests that Twitter is becoming a larger and more diverse platform that is beginning to rival sophisticated polling techniques. Perhaps in the future, social media polls will become more incorporated into polling schemes.

As social media evolve, user experience and comments have become increasingly important. In this work, we have proposed a weighted Word Sentiment Magnitude (wWSM) scheme that is able to help us quantify user sentiment through simple training and the results can be used to predict review rating of new comments rather accurately.

Most interestingly, the wWSM scheme shows great improvements in classifying 1- and 5-star reviews even in datasets with unbalanced class data. An overall accuracy of 75% can be achieved for all reviews and 92% for 5-star reviews.

Furthermore, these user responses could help us weight datasets to increase their quality. We can use techniques to handle the resulting non-evenly weighted dataset, such as the word sentiment magnitude test we have developed. This test, as other feature selection processes prove, shows that "quality" is better than "quantity" and that we can process data more quickly and accurately by using such methods.

In our future work, we plan to investigate a 3-class (positive/negative/neutral) or 5-class (one for each star) classification problem instead of a 2-class positive/negative classification problem. We could also apply our feature selections methods to our Twitter dataset. Furthermore, we could use other review/tweet attributes that are user generated. For example, we could investigate applying a weight based on the number of likes or re-tweets, or even try to gather tweets with location data and analyze that additional information (e.g. could we predict which states would be won by a certain candidate?) . Perhaps we could examine the word in Yelp reviews marked as funny or cool to see if they could improve our analysis of 3-star reviews, or help us identify reviews to exclude in general.

BIBLIOGRAPHY

[1] R. V. Farace and J. A. Danowski, "Analyzing human communication networks in organizations: Applications to management problems." 1973.

[2] D. M. Boyd and N. B. Ellison, "Social network sites: Definition, history, and scholarship," *Journal of computer-mediated Communication*, vol. 13, no. 1, pp. 210–230, 2007.

[3] M. V. Mäntylä, D. Graziotin, and M. Kuutila, "The evolution of sentiment analysisâĂŤa review of research topics, venues, and top cited papers," *Computer Science Review*, vol. 27, pp. 16–32, 2018.

[4] S. A. Sandri, D. Dubois, and H. W. Kalfsbeek, "Elicitation, assessment, and pooling of expert judgments using possibility theory," *IEEE transactions on fuzzy systems*, vol. 3, no. 3, pp. 313–335, 1995.

[5] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, 2002, pp. 79–86.

[6] B. O'Connor, R. Balasubramanyan, B. R. Routledge, and N. A. Smith, "From tweets to polls: Linking text sentiment to public opinion time series." *ICWSM*, vol. 11, no. 122-129, pp. 1–2, 2010.

[7] T. Wilson, J. Wiebe, and P. Hoffmann, "Recognizing contextual polarity in phrase-level sentiment analysis," in *Proceedings of the conference on human language technology and empirical methods in natural language processing*. Association for Computational Linguistics, 2005, pp. 347–354.

[8] E. Kouloumpis, T. Wilson, and J. Moore, "Twitter sentiment analysis: The good the bad and the omg!" in *Fifth International AAAI conference on weblogs and social media*, 2011.

[9] K. Jahanbakhsh and Y. Moon, "The predictive power of social media: On the predictability of us presidential elections using twitter," *arXiv preprint arXiv:1407.0622*, 2014.

[10] A. Pak and P. Paroubek, "Twitter as a corpus for sentiment analysis and opinion mining." in *LREc*, vol. 10, no. 2010, 2010.

[11] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *CS224N Project Report, Stanford*, vol. 1, no. 12, p. 2009, 2009.

[12] F. Å. Nielsen, "A new anew: Evaluation of a word list for sentiment analysis in microblogs," *arXiv preprint arXiv:1103.2903*, 2011.

[13] H. Wang, D. Can, A. Kazemzadeh, F. Bar, and S. Narayanan, "A system for real-time twitter sentiment analysis of 2012 us presidential election cycle," in *Proceedings of the ACL 2012 System Demonstrations*. Association for Computational Linguistics, 2012, pp. 115–120.

[14] W. Wang, L. Chen, K. Thirunarayan, and A. P. Sheth, "Harnessing twitter" big data" for automatic emotion identification," in *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*. IEEE, 2012, pp. 587–592.

[15] D. Davidov, O. Tsur, and A. Rappoport, "Enhanced sentiment learning using twitter hashtags and smileys," in *Proceedings of the 23rd international conference on computational linguistics: posters*. Association for Computational Linguistics, 2010, pp. 241–249.

[16] J. Bollen, H. Mao, and A. Pepe, "Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena," in *Fifth International AAAI Conference on Weblogs and Social Media*, 2011.

[17] A. Amolik, N. Jivane, M. Bhandari, and M. Venkatesan, "Twitter sentiment analysis of movie reviews using machine learning techniques," *international Journal of Engineering and Technology*, vol. 7, no. 6, pp. 1–7, 2016.

[18] O. Kolchyna, T. T. Souza, P. Treleaven, and T. Aste, "Twitter sentiment analysis: Lexicon method, machine learning method and their combination," *arXiv preprint arXiv:1507.00955*, 2015.

[19] Y. Jo and A. H. Oh, "Aspect and sentiment unification model for online review analysis," in *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, ser. WSDM '11. New York, NY, USA: ACM, 2011, pp. 815–824. [Online]. Available: http://doi.acm.org/10.1145/1935826.1935932

[20] S. Tan, X. Cheng, Y. Wang, and H. Xu, "Adapting naive bayes to domain adaptation for sentiment analysis," in *Advances in Information Retrieval*, M. Boughanem, C. Berrut, J. Mothe, and C. Soule-Dupuy, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 337–349.

[21] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Foundations and Trends in Information Retrieval*, vol. 2, no. 1-2, pp. 1–135, 2008. [Online]. Available: http://dx.doi.org/10.1561/1500000011

[22] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, ser. HLT '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 142–150. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002472.2002491

[23] W. Duan, Q. Cao, Y. Yu, and S. Levy, "Mining online user-generated content: Using sentiment analysis technique to study hotel service quality," in *2013 46th Hawaii International Conference on System Sciences*, Jan 2013, pp. 3119–3128.

[24] M. Salehan and D. J. Kim, "Predicting the performance of online consumer reviews: A sentiment mining approach to big data analytics," *Decision Support Systems*, vol. 81, pp. 30 – 40, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923615002006

[25] L. Dey, S. Chakraborty, A. Biswas, B. Bose, and S. Tiwari, "Sentiment analysis of review datasets using naive bayes and k-nn classifier," *arXiv preprint arXiv:1610.09982*, 2016.

[26] S. Bakhshi, P. Kanuparthy, and D. A. Shamma, "Understanding online reviews: Funny, cool or useful?" in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 2015, pp. 1270–1276.

[27] X. Lei, X. Qian, and G. Zhao, "Rating prediction based on social sentiment from textual reviews," *IEEE transactions on multimedia*, vol. 18, no. 9, pp. 1910–1921, 2016.

[28] A. K. Uysal, "An improved global feature selection scheme for text classification," *Expert systems with Applications*, vol. 43, pp. 82–92, 2016.

[29] A. Salinca, "Business reviews classification using sentiment analysis," in *2015 17th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2015, pp. 247–250.

[30] A. Blake, "Welcome to the next, most negative presidential election of our lives," http://wapo.st/2a9uWr8?tid=ss_mail, Washington Post, accessed: 2019-05-12.

[31] Statista, "Number of monthly active twitter users worldwide from 1st quarter 2010 to 1st quarter 2017 (in millions)," https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/, accessed: 2017-05-03.

[32] M. Hu and B. Liu, "Mining and summarizing customer reviews," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2004, pp. 168–177.

[33] Jefferson-Henrique, "Getoldtweets," https://github.com/Jefferson-Henrique/GetOldTweets-java, accessed: 2019-06-17.

[34] Pollster, "2016 presidential election," http://elections.huffingtonpost.com/pollster/2016-general-election-trump-vs-clinton, accessed: 209-06-17.

[35] CBS, "A james comey timeline," http://www.cbsnews.com/news/a-james-comey-timeline/, accessed: 2019-06-17.

[36] W. Post, "Election timeline," https://www.washingtonpost.com/news/wonk/wp/2016/10/07/the-real-issue-with-donald-trump-saying-a-man-can-do-anything-to-a-woman/?utm_term=.cdcf152a97fb, accessed: 2019-06-17.

[37] "Yelp challenge dataset," https://www.yelp.com/dataset, accessed: 2019-03-27.

[38] I. Feinerer and K. Hornik, *tm: Text Mining Package*, 2018, r package version 0.7-6. [Online]. Available: https://CRAN.R-project.org/package=tm

[39] S. Bird, E. Loper, and E. Klein, "Natural language processing with python," O'Reilly Media Inc., 2009.

[40] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2017. [Online]. Available: https://www.R-project.org/

[41] J. Martineau and T. W. Finin, "Delta tfidf: An improved feature space for sentiment analysis," in *ICWSM*, 2009.

[42] Jacob, "Text classification for sentiment analysis - naive bayes classifier," https://streamhacker.com/2010/05/10/text-classification-sentiment-analysis-naive-bayes-classifier/, accessed: 2019-05-13.

APPENDIX A

NUMERIC RESULTS OF SELECTED TESTS

A.1. Test 1: Control Group

For the first test, we used Naive Bayes with little feature selection. We did remove words of length 1 and words that occurred in fewer than 50 reviews (we went from 38,848 total words to 11,059).

Table A.1. Classification Results

Test #	**Accuracy**	**Time**	**FPR**	**FNR**	**TPR**	**TNR**	**PPV**	**F1**
Test 1	73.57%	12 hr. 9 min	37.76%	15.13%	84.87%	62.24%	69.27%	76.28%
Test 2	73.10%	24 min	37.39%	16.44%	83.56%	62.61%	69.14%	75.67%
Test 3	71.53%	31 min	45.24%	11.73%	88.27%	54.76%	66.17%	75.64%
Test 4	74.63%	23 min	35.09%	15.67%	84.33%	64.91%	70.67%	76.90%
Test 5	74.42%	1 hr. 16 min.	35.78%	15.40%	84.60%	64.22%	70.33%	76.81%
Test 6	73.93%	25 min	39.34%	12.83%	87.17%	60.66%	68.96%	77.00%
Test 7	75.41%	23 min	33.97%	15.23%	84.77%	66.03%	71.44%	77.54%
Test 9	73.59%	1 hr. 7 min	36.72%	16.12%	83.88%	63.28%	69.60%	76.08%
Test 10	75.07%	2 hr. 53min.	34.75%	15.15%	84.85%	65.25%	71.00%	77.31%
Test 11	74.58%	55 min.	38.40%	12.48%	87.52%	61.60%	69.56%	77.51%
Test 12	74.54%	3 hr. 22 min.	38.68%	12.28%	87.72%	61.32%	69.46%	77.53%

A.2. Test 2: Information Gain (IG)

We used information gain (IG) to filter words with low IG scores (only kept top 1,000 scores).

A.3. Test 3: Weighting Reviews Processed Using Information Gain

In our second, we applied a weight w_r a review r using its useful score r_u with the following formula:

$$w_r = \log_2(1 + \lceil (r_u + 1) \rceil) \tag{A.1}$$

The first "+1" ensures that the review will be replicated at least once, the second "+1" ensures we do not have an infinite value. After applying the weight, we

randomly choose an equal number of positive and negative reviews. This resulted in our training dataset being increased from around 476,000 reviews to over 800,000 reviews. Our results seem to indicate that choosing an equal number of classes negated the benefit of applying the weight in the first place. Furthermore, we suspected the higher percentage increase of our training dataset lead to overfitting. We remedied these issues with our second weighted test (Test 6).

A.4. Test 4: Word Sentiment Magnitude

In our fourth test, we used the word sentiment magnitude method for feature selection. We choose the top 500 positive words and the top 500 negative words.

A.5. Test 5: Adding Local Words

We calculated the top 25 review categories (food, hotel, etc.) and calculated the top 500 positive and 500 negative words for each category using the word sentiment magnitude feature selection process. We then unioned the local list of words together (to form a global list). We had a list of 3,024 words, so we added over 2,000 local words to our training set.

A.6. Test 6: Weighted Word Sentiment Magnitude

We used a similar weighting function as in Test 3, but we choose to use the floor function instead of the ceiling function, and we also used a natural logarithm to decrease the replication factor slightly:

$$w_r = \ln(1 + \lfloor (r_u + 1) \rfloor) \tag{A.2}$$

Again, The first "+1" ensures that the review will be replicated at least once, the second "+1" ensures we do not have an infinite value.

For this weighting scheme, we had over 600,000 instead of 800,000. We also did not choose and even number of classes: we had about 292,000 positive reviews and 313,000 negative reviews.

A.7. Test 7: Negating Words

For this test, we negated words that were preceded by a popular negation word. For example, "not good" would become "not NOTgood."

A.8. Tests 8-12: Adding Negation to Tests

Our final tests compare the performance and results of adding negation. Negation in general does improve results, but at the expense of runtime. In essence, adding negation increases the total number of words in the training/testing dataset, but the increase in accuracy may be worth this offset. Test 8 used negation for Test 1. Test 9 added negation for Test 2. Test 10 added negation for Test 5. Test 11 added negation for Test 6. And Test 12 combined the methodologies from Test 5 (local words), Test 6 (weighted reviews), and Test 7 (negating words).

By adding negation to words to Test 11, we were able to increase our 5 star accuracy by 3% in table A.6 for our weighted Test 7 in table A.1. Also notice that the star accuracy for reviews with 2-3 drops slightly, possibly signifying there may be more positive sentiment associated with 2 and 3 star reviews than originally assumed (especially for 3 star reviews).

In addition, we used a smaller set of categories for the local words. We choose the categories "Restaurants|Food", "Hotels&Travel", "Arts&Entertainment", "EventPlanning&Services", and "CarDealers" (notice we combined the restaurants and foods categories). These were chosen for their popularity and diversity. This gave us a smaller set of words to work with (2,234) and we achieved similar results.

A.9. Using Different Values of η_0

We repeated Test 7 (weighted reviews) several times with different values of η_0 as well as Test 11 (non-weighted reviews) with different value of η_0. These results are in tables A.3 and A.2. In addition, Delta TFIDF results are in Table A.4. We also performed weighted review tests on an unbalanced dataset (i.e. unequal number of positive and negative reviews). Those results are in Table A.5.

Table A.2. Test 7 (Non-Weighted Reviews) with Different Values of η_0

η_0	**Accuracy**	**Time**	**FPR**	**FNR**	**TPR**	**TNR**	**PPV**	**F1**
-0.25	71.51%	44 min.	38.43%	18.58%	81.42%	61.57%	67.99%	74.10%
0	71.64%	1hr 12 min	38.01%	18.73%	81.27%	61.99%	68.20%	74.16%
0.25	75.60%	53 min	33.37%	15.45%	84.55%	66.63%	71.76%	77.63%
0.5	75.99%	54 min.	33.44%	14.61%	85.39%	66.56%	71.91%	78.07%
0.75	75.40%	1 hr 1 min	34.31%	14.92%	85.08%	65.69%	71.32%	77.59%
1.25	74.56%	59 min	34.83%	16.08%	83.92%	65.17%	70.72%	76.76%
1.5	73.96%	1 hr 1 min	35.67%	16.43%	83.57%	64.33%	70.14%	76.27%

Table A.3. Test 11 (Weighted Reviews) with Different Values of η_0

η_0	**Accuracy**	**Time**	**FPR**	**FNR**	**TPR**	**TNR**	**PPV**	**F1**
-0.25	71.04%	49 min.	42.48%	15.47%	84.53%	57.52%	66.61%	74.51%
0	71.22%	55 min.	42.09%	15.50%	84.50%	57.91%	66.81%	74.62%
0.25	74.80%	1 hr. 3 min.	37.69%	12.74%	87.26%	62.31%	69.89%	77.61%
0.5	75.66%	1 hr 8 min	37.07%	11.65%	88.35%	62.93%	70.50%	78.42%
0.75	74.76%	1 hr. 6 min	38.58%	11.93%	88.07%	61.42%	69.59%	77.75%
1.25	74.15%	1 hr 8 min	38.79%	12.94%	87.06%	61.21%	69.23%	77.13%
1.5	73.31%	1 hr 5 min	39.96%	13.46%	86.54%	60.04%	68.47%	76.45%

Table A.4. Delta TFIDF Results

Accuracy	**FPR**	**FNR**	**TPR**	**TNR**	**PPV**	**F1**
75.43%	34.14%	15.03%	84.97%	65.86%	71.39%	77.59%

Table A.5. Unbalanced Tests

	Accuracy	**Time**	**FPR**	**FNR**	**TPR**	**TNR**	**PPV**	**F1**
$\eta = 0.5$	79.82%	1 hr. 33 min.	36.95%	11.98%	88.02%	63.05%	82.98%	85.43%
$\eta_0 = 1$	78.81%	1 hr. 26 min.	37.79%	13.08%	86.92%	62.21%	82.47%	84.64%
Delta TFIDF	67.23%	24 min	99.61%	0.10%	99.90%	0.39%	67.23%	80.37%
NB (control)	77.53%	55 hr. 33 min.	35.61%	16.05%	83.95%	64.39%	82.83%	83.38%

Table A.6. Accuracy Per Star Review

Test #	**1 Star**	**2 Star**	**3 Star**	**4 Star**	**5 Star**
Test 1	75.64%	67.91%	48.79%	76.40%	90.02%
Test 2	74.94%	67.98%	50.14%	74.95%	88.79%
Test 3	68.87%	59.98%	41.06%	81.55%	92.35%
Test 4	77.41%	70.39%	52.24%	75.60%	89.64%
Test 5	76.78%	69.99%	51.31%	75.85%	89.92%
Test 6	75.05%	66.21%	46.54%	79.67%	91.73%
Test 7	78.86%	71.80%	52.93%	75.94%	90.13%
Test 9	75.77%	68.85%	50.54%	75.23%	89.14%
Test 10	78.00%	71.14%	52.13%	75.89%	90.30%
Test 11	76.29%	67.35%	47.13%	79.99%	92.10%
Test 12	75.81%	67.32%	46.87%	80.16%	92.31%

A.10. Focusing of Particular Star Reviews

We would except 5 and 1 star reviews to be more useful because of the clear sentiment they should express (extreme like/dislike). Thus, we decided to look at the accuracy of each star review. These results are presented in table A.6.

We also have results for Test 7 (weighted reviews) with different values of η_0 as well as Test 11 (non-weighted reviews) with different value of η_0. These star review accuracy for those results are in tables A.8 and A.7. In addition, Delta TFIDF star review accuracy results are in Table A.9. We also calculated star review accuracy for weighted review tests on an unbalanced dataset. Those results are in Table A.10 As expected, the 5 star reviews are easier to classify. The 3 star reviews seem ambiguous, probably due to there neutral nature. Furthermore, weighting the reviews seems to increase the accuracy of the 5 and 1 star reviews, which if the star number was

hidden and only the positive/negative label was provided could prove useful in order to help ignore potentially distracting 3 star reviews. In essence, if we only cared about classifying 1 and 5 star reviews correctly, weighting them by useful reviews could be very helpful.

Table A.7. Accuracy Per Star using WSM

η_0	**1 Star**	**2 Star**	**3 Star**	**4 Star**	**5 Star**
-0.25	69.22%	66.04%	53.09%	72.19%	87.04%
0	70.66%	66.46%	52.78%	71.97%	86.92%
0.25	81.47%	72.24%	52.16%	75.62%	89.97%
0.5	80.28%	72.46%	52.72%	76.55%	90.77%
0.75	79.24%	71.44%	52.08%	76.41%	90.35%
1.25	76.88%	70.61%	53.10%	74.86%	89.43%
1.5	75.53%	69.46%	52.83%	74.24%	89.25%

Table A.8. Accuracy Per Star using wWSM

η_0	**1 Star**	**2 Star**	**3 Star**	**4 Star**	**5 Star**
-0.25	66.69%	62.02%	47.91%	76.41%	89.46%
0	68.08%	62.45%	47.55%	76.42%	89.42%
0.25	79.07%	67.80%	46.49%	79.68%	91.86%
0.5	79.25%	69.16%	46.98%	81.00%	92.82%
0.75	76.94%	67.30%	46.28%	81.01%	92.36%
1.25	74.85%	66.81%	47.61%	79.33%	91.76%
1.5	72.81%	65.30%	47.31%	78.55%	91.40%

Table A.9. Accuracy Per Star Review (Delta TFIDF)

1 Star	**2 Star**	**3 Star**	**4 Star**	**5 Star**
79.02%	71.52%	52.58%	76.19%	90.31%

Table A.10. Accuracy Per Star Review (Unbalanced Reviews)

	1 Star	**2 Star**	**3 Star**	**4 Star**	**5 Star**
η_0=0.5	78.09%	68.30%	47.07%	80.48%	92.76%
η_0=1	75.93%	66.87%	47.70%	78.98%	91.90%
Delta TFIDF	0.61%	0.39%	0.20%	99.87%	99.92%
NB (control)	77.00%	69.17%	50.74%	74.96%	89.58%

APPENDIX B

SELECTED CODE SNIPPETS

B.1. NLTK Code

This code was more straightforward, as we used a python package that did most of the heavy lifting. See for [42] more information.

B.2. Prepossessing the Data

```
library(tm,qdap)
library(stringr)
library(wordcloud)
set.seed(1234)
totalLength = length(yelp_train$label)
posLength = length(yelp_filtered_positive$label)
negLength = length(yelp_filtered_negative$label)
#pos words
#pos <- Corpus(VectorSource(yelp_filtered_positive$text))
pos <- strsplit(yelp_filtered_positive$text, " ")
#neg words
#neg <- Corpus(VectorSource(yelp_filtered_negative$text))
neg <- strsplit(yelp_filtered_negative$text, " ")
#get unique words in each review
pos<- lapply(pos,unique)
neg<- lapply(neg,unique)
#get freq
# most frequent words
mfw = sort(table(unlist(pos)), decreasing=TRUE)
```

B.3. Calculate List of Positive and Negative Words

```
mfwNeg = sort(table(unlist(neg)), decreasing=TRUE)
#get list of negative words frequencies
posTest <- as.data.frame(mfw)
colnames(posTest)<- c("word","posFreq")
#get list of negative words frequencies
negTest <- as.data.frame(mfwNeg)
colnames(negTest)<- c("word","negFreq")
#merge two columns with outer join, replce NA with zero
word_freq <- merge(posTest,negTest,by="word",all=TRUE)
word_freq[is.na(word_freq)] <- 0
word_freq$totalFreq <- word_freq$posFreq+word_freq$negFreq
word_freq$notInNeg <- posLength - word_freq$negFreq
word_freq$notInPos <- negLength - word_freq$posFreq
word_freq$totalNotFreq <- word_freq$notInNeg+word_freq$notInPos
word_freq_top50 <-word_freq[word_freq$totalFreq>=50,]
#convert text to vector
yelp_train$text <- sapply(yelp_train$text,strsplit,split=" ")
yelp_train$text <- sapply(yelp_train$text,unlist)
#only keep words with freq >= 50
yelp_train$text <- sapply(yelp_train$text,intersect,y=word_freq_top50$word)
#combine vector back to text
yelp_train$text <- sapply(yelp_train$text,paste,collapse=" ")
```

B.4. Calculating Information Gain

Notice how use use the intersect function to remove unwanted words. Thus, assuming $m > n$, instead of searching for m words to remove, we search for n words to keep.

```
word_freq$IG <- ((word_freq$totalFreq/totalLength) * ((word_freq$posFreq /
    word_freq$totalFreq) * log2( 1+word_freq$posFreq / word_freq$totalFreq )+
                                            (word_freq$negFreq /
                                               word_freq$totalFreq) * log2(
                                               1+word_freq$negFreq /
                                               word_freq$totalFreq))
            + (1-(word_freq$totalFreq / totalLength)) *
                ((word_freq$notInPos / word_freq$totalNotFreq) * log2(
                1+word_freq$notInPos / word_freq$totalNotFreq)+
                                             (word_freq$notInNeg /
                                                word_freq$totalNotFreq) *
                                                log2( 1+word_freq$notInNeg
                                                / word_freq$totalNotFreq)))


#sort by IG column
word_freq_sorted <- word_freq[order(-word_freq$IG),]
#only keep first 1000 rows
word_freq_sorted <- word_freq_sorted[1:1000,]
yelp_train_ig$text <-
    sapply(yelp_train_ig$text,intersect,y=word_freq_sorted$word)
```

B.5. Calculating η_0

We refer to "η_0" as "mew" in the code. In this example, $\eta_0 = 0.75$

```
mew <- 0.75
word_freq_top50$diff_v2 <- word_freq_top50$posFreq- mew *
    word_freq_top50$negFreq
#sort by descending (pos)
word_freq_sorted_mew <- word_freq_top50[order(-word_freq_top50$diff_v2),]
#only keep first 500 rows
word_freq_sorted_mew <- word_freq_sorted_mew[1:500,]
#sort by ascending column (neg)
temp <- word_freq_top50[order(word_freq_top50$diff_v2),]
#only keep first 500 rows, combine with pos rows
word_freq_sorted_mew <- rbind(word_freq_sorted_mew,temp[1:500,])
#convert text to vector of words
yelp_train_diff_v2$text <- sapply(yelp_train_diff_v2$text,strsplit,split="
    ")
yelp_train_diff_v2$text <- sapply(yelp_train_diff_v2$text,unlist)
#
#only keep top 1,000 (500 pos, 500 neg) diff words
yelp_train_diff_v2$text <-
    sapply(yelp_train_diff_v2$text,intersect,y=word_freq_sorted_mew$word)
#
#combine vector back to text
yelp_train_diff_v2$text <- sapply(yelp_train_diff_v2$text,paste,collapse="
    ")
```

B.6. Calculate Accuracy and Accuracy Per Star

```
accurcy_per_star<-function(test_name){
  print(test_name)
  for(star in 1:5){
    yelp_test_temp <- as.data.frame(yelp_test)
    yelp_test_ig_weighted<-yelp_test_temp[yelp_test_temp$stars==star,]
    yelp_test_ig_weighted <- yelp_test_ig_weighted[ !is.na(
        yelp_test_ig_weighted[[test_name]]), ]
    accurcy_results_ig <- sum( yelp_test_ig_weighted$label ==
        yelp_test_ig_weighted[[test_name]]) /
        length(yelp_test_ig_weighted$label)
    print(paste("accurcy for ",star," review=",accurcy_results_ig,sep=" "))
  }
}
#
accurcyFun <- function(test_column){
  #
  yelp_test_diff <- yelp_test
  #
  #remove empty/na rows
  yelp_test_diff <- yelp_test_diff[!is.na( yelp_test_diff[[test_column]]),
      ] #note the reference the column by a string
  #
  print(length(yelp_test_diff$review_id))
  #[1] 18530
```

B.7. Calculate Values for Confusion Matrix

```
  TP <- sum(yelp_test_diff$label==1 & yelp_test_diff[[test_column]]==1)
  TN <- sum(yelp_test_diff$label==0 & yelp_test_diff[[test_column]]==0)
  FP<- sum(yelp_test_diff$label==0 & yelp_test_diff[[test_column]]==1)
  FN <-sum(yelp_test_diff$label==1 & yelp_test_diff[[test_column]]==0)
  #
  TPR <- TP/(TP+FN)
  FPR <- FP/(FP+TN)
  TNR <- TN/(TN+FP)
  FNR <- FN/(FN+TP)
  Accuracy <- (TP+TN)/(TP+TN+FP+FN)
  #
  PPV <- TP/(TP+FP)
  NPV <- TN/(TN+FN)
  FDR <- 1-PPV
  FOR <- 1-NPV
  #
  print(paste("Accuracy for ",test_column,"=",Accuracy,sep=" "))
  print(paste("FPR for ",test_column,"=",FPR,sep=" "))
  print(paste("FNR for ",test_column,"=",FNR,sep=" "))
  print(paste("TPR for ",test_column,"=",TPR,sep=" "))
  print(paste("TNR for ",test_column,"=",TNR,sep=" "))
  print(paste("PPV for ",test_column,"=",PPV,sep=" "))
}
```