Spring 2018

Using Sentiment Analysis and Pattern Matching to Signal User Review Abnormalities

Stefan S. Gloutnikov
San Jose State University

Using Sentiment Analysis and Pattern Matching to Signal User Review Abnormalities

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Stefan S. Gloutnikov

May 2018

The Designated Project Committee Approves the Project Titled

Using Sentiment Analysis and Pattern Matching to Signal User Review Abnormalities

By

Stefan S. Gloutnikov

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2018

Dr. Robert Chun
Department of Computer Science

Dr. Jon Pearce
Department of Computer Science

Mr. Stanislav Georgiev
Principal Information Architect, Salesforce.com

Abstract

User opinions on websites like Amazon, Yelp, and TripAdvisor are a key input for consumers when figuring out what to purchase, or where and what to eat. This means that in order for such websites to provide a better service to their customers, they must guard against fake and targeted reviews. Detecting such users and reviews automatically is a very complex multi-step process, and there is no direct mechanism for solving the problem reliably. Multiple AI and Machine Learning algorithms are coupled together when examining user reviews in determining if a review is fake or not. In this project we propose one such mechanism, which examines past user reviews to detect abnormalities, if any, signaling that they should be looked at more thoroughly from more dimensions. We do so by combining existing sentiment analysis techniques and pattern matching. In order to gain more insight into a review, we break it down into sentences and produce a sentiment value for each one, allowing us to represent a review as a sentiment vector. The sentiment vector then allows us to match various sized tuples against other reviews from the user and compute abnormality scores.

Acknowledgments

I would like to express my deepest gratitude to Dr. Robert Chun for agreeing to be my advisor again after my hiatus, and for his continued guidance. In addition, thank you also to Dr. Jon Pearce for his suggestions and support, and Stanislav Georgiev for working extremely closely with me throughout this project, without whom it would not have been possible. I would also like to thank my friends Alina, Deian, and Yelena for their continuous encouragements. Last but not least, I would like to thank my family for their endless support and love.

Table of Contents

List of Figures

List of Tables

1. Introduction

Since more and more people have been turning to user reviews when deciding on their consumer choices, there has also been a push by marketing agencies or owners to promote certain products and restaurants. A strategic and powerful way to game the system is to insert fake reviews by fake users. To a normal user looking at the popular restaurant, he sees hundreds or thousands of happy customers and is more likely to also give it a try. Websites like Yelp and Amazon need to identify and eliminate these fake reviews, if they want their customers to receive a satisfactory result and return to their platform again in the future.

As fake review bot makers improve, so should the detection methods, and it turns into a "cat and mouse" game between fake review generators and detectors. In a recent article discussing a new Yelp AI review generator from researchers from the University of Chicago, it was determined that AI-generated reviews were "effectively indistinguishable" from the genuine ones and were given a "usefulness" rating of 3.15 by human evaluators, compared to 3.28 for genuine reviews [4]. In response, Yelp stated that they didn't believe such reviews would pose a problem as they have internal methods for spotting fake reviews, and that they use many signals when determining if a review is legit and whether or not to approve it.

Due to the increasing complexity of detecting fake reviews, the detection systems rely on many signals when determining if something looks abnormal. Machine learning and AI algorithms combine together looking at the review text, user location, physical user location (IP address), history, and many more factors in determining if a review belongs to a real person.

In this project, we develop one such signal, which can be used as input into a more complex detection system, when identifying abnormal user activity. To do so, we break down every review into sentences, and perform sentiment analysis on each of these sentences. Modern sentiment analysis techniques (Recursive Neural Tensor Networks) and tools (CoreNLP) were used when scoring each sentence. The sentiment analysis scores of each sentence gives us an insight into the structure of the review, and we can use that to deduce if a user produces a certain pattern in his or her reviews.

From each of the scored sentences, we produce a sentiment vector that is used to generate all possible sentiment tuples for that review. Having all possible tuples, we match for patterns that maybe reoccur often in the user's reviews. If a user constantly exhibits certain patterns in the reviews he or she leaves, we signal that there is something abnormal.

2. Related Work

Detecting fake and suspicious user reviews has been a growing area of research, as online services and web stores have been rapidly growing. This has led to the increased importance of only displaying real and objective opinions online. Advances in natural language processing, machine learning, and AI have produced multiple techniques used to identify such fake, deceptive, or spam reviews. We outline some of these works, and build upon them to propose a unique and novel approach to detecting abnormal user reviews.

Around 10 years ago the paper titled *Review Spam Detection*, by Jindal and Liu [12] was the first to present the problem of detecting spam user reviews. The authors paved the first footsteps for studying review spam and spam detection. They identified that a large number duplicate, and near-duplicate reviews were written by the same or different reviewers for different products. The proposed approach was based on duplicate detection through the use of the shingle method and a 2-class classification (spam/not-spam) machine learning logistic regression predictive model. Later in *Finding Unusual Review Patterns Using Unexpected Rules*, Jindal, Liu, and Lim [10] identify unusual review patterns, which they define as suspicious behaviors, and formulate the problem as finding unexpected rules. To decide what is expected and unexpected, the authors define several types of expectations based on the natural distribution of the

data. The statistical methods used means that the proposed model is domain independent as it only depends on the data and the types of rules, but not the application [10].

In [11], the authors used linguistic features and behavioral features in an attempt to recognize what Yelp is doing in their existing secret spam detection methods. Both methods showed high detection accuracy of their crowdsourced fake reviews, but they concluded that Yelp's algorithms most likely used a behavioral based approach for detecting fake reviews. The following behavioral dimensions were used when examining a user's set of reviews: maximum number of reviews—writing a large amount of reviews in a day is abnormal; percentage of positive reviews—a majority of spammers had more than 80% of their reviews as four or five stars, while normal users showed a more even distribution; review length—a majority of spammers used 135 words or less, while a majority of normal users used more than 200 words; reviewer deviation—since spamming is analogous to incorrect projection, fake reviews are more likely to deviate from the general rating consensus; maximum content similarity—examine whether reviews are similar to other existing reviews [11]. The authors in [15] explore generalized approaches for identifying fake reviews by capturing the differences of language between misleading and truthful reviews. They construct and present a cross-domain gold-standard dataset, which they use to extend the SAGE

Model—a Bayesian generative approach [21]. The findings showed that spammers are more likely to exaggerate their opinions, and are more likely to use strongly opinionated vocabulary.

In [14], the authors detect fake reviews written by the same person using multiple names, by looking at the semantic similarity between words. They argue that the key to catching fake reviews can be found in the review text, because spammers have a limited imagination when it comes down to writing completely new details in every review. Fake reviews are more prone to rephrasing or switching words with their synonyms, and is why they look at synonym relations between words. Simple cosine similarity and variants of cosine similarity are proposed in measuring the results [14]. Another method for identifying the same author among different user names is proposed in [17], by looking at writing styles and other linguistic clues. The authors argued that cosine similarity did not perform well on their dataset and proposed a new method by creating their own binary classifier. The core of the method is based on supervised learning, which learns in a similarity space rather than the document space [17]. Both [16] and [19] also employ their own machine learning methods for identifying and classifying spam reviews. Three different types of ways to look at reviews are proposed in [16]: Untruthful opinions—reviews that mislead readers on purpose by giving undeserving positive or negative reviews. Reviews based on brands only—these are

reviews that do not comment on the products, but only the brands, manufacturers or sellers. Non-reviews—reviews that are not really reviews, but are there to advertise a different product or ask a question.

A different way to spot fake reviews, by identifying groups is presented in [13], where the authors study spam detection in the collaborative setting, by discovering fake reviewer groups. Although identifying and labeling a review or reviewer as fake is hard, this study suggests that labeling reviewer groups is much easier, and propose a supervised machine learning approach to do so. Most of the previously shown publications which use machine learning techniques are highly dependent on domain specific labeled data in order to perform well. *FraudEagle*, presented in [20], is a fast and effective framework for detecting fraudsters and fake reviews, that works in a completely unsupervised fashion, requires no labeled data, and is generalizable. It exploits the network effect among reviewers and products, unlike other methods that focus on review texts or behavioral analysis. Finally, it is able to give a fraud score to each review and user, and is capable of scaling to large datasets due to its nature to grow linearly with the network size. [20]

Sentiment Analysis

Up until recently, the field of sentiment analysis had not been used for detecting fake and spam reviews. The paper *Detecting Spam Reviews through Sentiment Analysis*, by Peng and Zhong, was published in late 2014 and was the first to make this step. The sentiment score and star rating relationship is used as a discriminative rule—a review with a high star rating, but with text that did not reflect the high score (and vice versa), would be flagged as potentially fake. A time series combined with more discriminative rules are used at the end to detect and classify reviews as spam [18].

Although not directly in the domain of detecting fake reviews, sentiment analysis was used very recently with user reviews in [22], for identifying restaurant features in Yelp reviews. The authors used sentiment analysis and the Yelp dataset to answer questions like: What makes a good restaurant? What are the major concerns of customers for a great meal?

Proposed Methods

Building on some of the ideas discussed above, we propose a unique and novel approach for identifying and scoring potentially fake reviews and users. Using a combination of existing sentiment analysis techniques, pattern matching, and statistically derived rules, we score how likely a user is a fraudster. To the best of our

knowledge, sentiment analysis has never been used before for identifying the structure of a review, which gives us the ability to detect reoccurring patterns and abnormalities. As AI fake review bots evolve, they must still obey by some underlying logic and rules. Knowing and comparing the sentiment of each sentences gives us a look into how each review is structured. Like in [20] and [10], our methods are also domain independent, and do not necessarily depend on an existing dataset.

3. Sentiment Analysis

Sentiment analysis is a relatively new field of research that has bloomed rapidly since the 2000's, in most part under the umbrella of the Natural Language Processing (NLP) field. Sometimes referred to as opinion mining, the field of sentiment analysis is the study of analyzing people's opinions, emotions, or attitudes towards a certain subject, topic or event. The terms sentiment analysis and opinion mining can be used interchangeably, with the first most often used in industry, whereas both terms appear in academia to express the same matter.

The desired outcome of sentiment analysis is to be able to produce automated methods for identifying and extracting said sentiment from a written text, plus be able to classify it as accurately as possible. When we say to classify the sentiment extracted from written text, we mean being able to imply whether it holds positive, negative, or neutral

sentiment. In the case of positive and negative sentiment, we can go a little further if our methods allow, and say to what degree it is positive or negative. For example, a friend of ours watched a movie and used the following sentence, "The movie was OK" to describe this movie when we asked how it was. The sentence or language in this case holds a positive sentiment, but does not express a very strong positive opinion. "The movie was one of the best ones that I have seen" is also positive overall, but on the other hand also establishes a much stronger positive opinion. If someone wrote or spoke the latter to us, we would be more inclined to watch this movie ourselves, because of the strong positive opinion we have created in our mind after hearing it. The same can be said about negative sentiment, and the degree of which the text expresses a negative opinion. *Figure 1* displays an example of a statement being put through some sentiment analysis method, and a result being computed. There are various different methods and techniques for producing a result (middle box), which are the main focus of academic research when it comes to sentiment analysis.

Figure 1. Statement and analysis example.

In this project we will use one of the more modern approaches developed at Stanford around 2013 for our sentiment analysis steps— Recursive Neural Tensor Network with

Sentiment Treebank [2]. The reasoning behind this choice is that this model performs very well and comes with well-developed software tools for using it, which we discuss below. To back up our results, we will also use a more classical approach, in the use of the Naive Bayes classifier. Alternative methods include using a few other modern approaches, word2vec and fastText [6][27-29].

3.1. Background

Sentiment analysis has become a very active and popular research topic over the last several years, and has seen tremendous growth since the early 2000's. Not only has it grown in academia, but also multiple large tech companies like Google, Microsoft and Amazon have become more invested in sentiment analysis by dedicating more resources to studying, improving and implementing it in their businesses. Thanks to the various applications and interest from both the industry and academic fields, sentiment analysis has seen steady and continuous growth since the turn of the century to present day. *Figure 2* shows the Google interest over time (Google Trends) for the term 'sentiment analysis'.

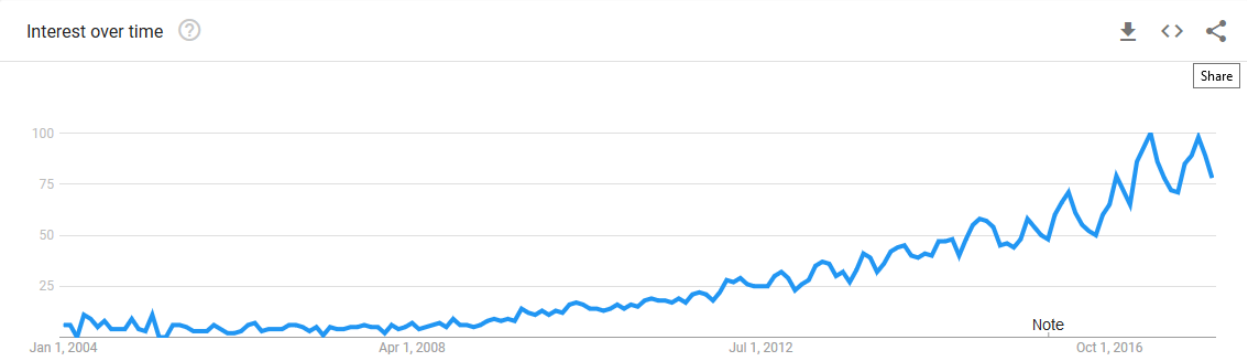

Figure 2. Google Trends for the term 'sentiment analysis'. [7]

The start of the sentiment analysis field may be attributed to multiple factors, but the most influential one was the boom of the internet after the turn of the century, or what we most often refer to as Web 2.0. Although Sentiment Analysis, like Natural Language Processing and Linguistics, involves the study of text, there was no research related to it prior to the year 2000. This is due to the fact that even though a lot of written or spoken opinions existed before that time, close to none of it existed in digital format.

The rapid growth of the web and social media sites have allowed for large and continuous streams of structured and unstructured opinion data to be stored in digital formats. Web pages can easily be mined for text. Text can also be stored and accessed in databases, allowing software developers and scientists to develop different tools and techniques in improving the sentiment analysis field. For example, giant web stores like Amazon house millions of reviews on products sold through their store. Online communities on various forums or reddit also contain large amounts of openly available opinion data related to news, politics, sports, products, etc.

From the opinion data sources out there, the biggest one is social media. Sites like Facebook, YouTube, and Twitter allow billions of users from any country, in any part of the world to instantly express their views on any topic. They also allow for users to easily connect and communicate among each other. Social media sites have become embedded in our everyday lives and society to such a degree, that we are now talking about them being used to influence opinions, and having the capability of swinging political elections. The data generated from these social media websites is one of the biggest contributors to the increasing popularity of the study of sentiment analysis. [7-8].

3.2. Sentiment

In order to perform and understand sentiment analysis, we must first clearly define what sentiment is. Although there are multiple dimensions and characteristics to analyzing, defining, and presenting sentiment, which unlock more interrelated sub-problems, we will concentrate on only the main characteristics, and define an abstraction of the sentiment analysis problem.

Sentiment is the underlying attitude, feeling, or emotion associated with an opinion. We represent it as a tuple,

(o, i),

where ‘o’ represents the orientation of the sentiment, and ‘i’ represents the intensity of the sentiment. The orientation of the sentiment can sometimes also be referred to as polarity, class, or semantic orientation.

- Sentiment orientation: Orientation can be positive, negative, or neutral. In the event that a sentiment orientation is classified as neutral, that usually means an absence of sentiment, or no sentiment. An example of a neutral statement could be something on the lines of, “I don’t know if I liked the movie. I should watch it again in a quiet environment.”
- Sentiment intensity: In addition to orientation, we can also add a value to intensity, when looking at a statement of opinion. Sentiment intensity refers to how strong of an opinion the statement expresses. In the introduction section of this paper we gave examples of two statements describing a movie. Both were positive, but one of them portrayed a much stronger opinion.

In more practical and real-world applications for example, we could see sentiment intensity as a five-star rating system. Where five stars means highly positive, four stars as somewhat positive, three stars as neutral, two stars as somewhat negative, and one star as highly negative.

Using a range from -1 to 1 is also very often seen as a good way to simplify sentiment orientation and intensity. A positive orientation is represented by +1, 0 a neutral, and -1 a negative orientation. The numbers in between can be treated as a scale of how negative or positive the intensity is. In a previous example we saw that the statement "Exercise is great" produced a result (through some algorithm) of positive 0.7—which is very positive. This simplified approach is most often used in academia and sentiment analysis systems, and is what we will prefer in this project [8].

3.3. Problem and Characteristics

We would love for sentiment analysis to be exact and produce accurate results for all of our input, but that is simply impossible in today's day and age. If we go back to the definition of the problem that sentiment analysis is trying to solve, it is to develop automatic tools, such that given an input text they output a classification. Language is simply too complex for us at this stage to develop automatic methods for detecting and classifying sentiment with 100% accuracy. We can only hope for high percentage of accuracy, and in reality, that should be enough for us. Given a large input size and high probability of accurate results, means that we should be able to produce an accurate enough result even though we could have misclassified some inputs.

As mentioned before, sentiment analysis is a deep and complex field of research, in which during the analysis phase, multiple characteristics can be considered. For this

project we will only be concerned with the main such characteristics. Next, we describe and discuss these characteristics and challenges in more detail.

Objective vs. Subjective Sentences

Unlike factual texts, sentiment and opinion have one important characteristic, and that is that they are subjective. The first step in sentiment analysis usually involves distinguishing between subjective and objective text. In the event that a sentence or text is classified as objective, no other steps are necessary. However, if a sentence is classified as subjective, its orientation and intensity (positive, negative, neutral) are to be estimated. *Fig 3* demonstrates this process, and the basic workflow when classifying a sentence.

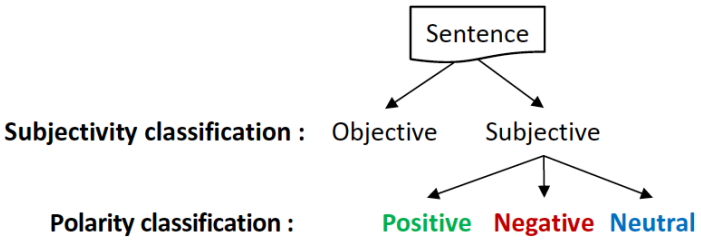

Figure 3. Classification workflow. [7]

We call the task that distinguishes between an objective (factual) sentence and a subjective (express views and opinions) sentence, subjectivity classification. Polarity classification is the step that determines if a sentence is positive, negative, or neutral.

"The MacBook is a laptop" is an example of an objective sentence, and "The MacBook is a great laptop" is an example of a subjective sentence, with positive polarity. It is important to note that a subjective sentence can sometimes not portray positive or negative polarity. The sentence "I think the MacBook shipment has arrived" is neither positive nor negative, and is thus classified as neutral [7].

3.3.2. Levels of Analysis

When performing sentiment analysis on a text, there are multiple levels at which the text can be analyzed. Generally speaking, when analyzing social networks this can be done essentially at three levels. *Fig. 4* shows these levels [7].

Figure 4. Levels of analysis. [7]

- Message level: The goal is to determine the orientation/polarity of the entire opinionated message. An example of this might be a product review on an online store like Amazon. The system determines what the entire review is expressing,

positive, negative or neutral overall opinion about the product. The assumption here is that the entire message expresses only one opinion on a single entity (a product in our example).

- Sentence level: The goal is to classify the orientation/polarity of each sentence contained in the entire text. The assumption is that each sentence in the entire text expresses a single opinion on a single entity.
- Entity and aspect level: The goal is to perform a deeper analysis than message and sentence levels. This level of analysis assumes that an opinion consists of sentiment and a target (of opinion). For example, "The MacBook is a great laptop, but it needs work on security issues and battery life". This sentence evaluates to three aspects: MacBook – positive, security – negative, and battery life – negative.

Regular vs. Comparative Opinion

An opinion can be of different shades, and can be part of one of the following groups:

- **Regular opinion:** Sometimes referred to as a standard opinion in literature. There are two main subtypes:
    - **Direct opinion:** Refers to an opinion expressed directly about an entity. For example, "The retina display on the MacBook is gorgeous."
    - **Indirect opinion:** Refers to an opinion expressed indirectly on an entity, on the basis of its effects on some other entity. For example, "After I

upgraded my MacBook OS, I lost all my settings!" Shows a negative effect on "my settings", which indirectly gives a negative sentiment to the MacBook.

- **Comparative opinion:** Expresses a relation of similarities or differences between two or more entities. For example, "MacOS performs much better than Windows 10" and "MacOS is the best operating system" both express comparative opinions. A comparative opinion is usually expressed with the use of a comparative form of an adjective or adverb.

3.4. Applications

Another reason behind the growing popularity in academia and industry for sentiment analysis, other than that it is a very interesting topic for research, is the real-life applications that can be developed around it. Knowing the opinions of individuals, whether they are customers, consumers, or voters can be a very powerful tool. Sentiment analysis enables multiple, different and interesting applications, in almost every possible domain.

Opinions, and the understanding of those opinions, through sentiment analysis is very important to businesses and organizations, because they want to find out what their customers or the public thinks about their products and services. Answers to important

questions like, “Why are consumers not buying this product?” or “Why and what are customers liking about a product, so that we can make it even better in future revisions?” can be of tremendous help to marketing or design teams at a company. In addition, these opinions can also be beneficial to a customer in helping them to decide whether to buy a product or not. Not only that, but if there are multiple variations or configurations of a product, which variation to buy. Consumers are no longer limited to only asking friends or family for their opinions, but have the opinions of hundreds or thousands of other customers.

Another domain that has used sentiment analysis and has seen beneficial results, is in politics. Simply put, opinions matter a whole lot in politics. One of the first to openly admit to using sentiment analysis and social media in the US was president Barack Obama’s campaign. During the 2008 presidential elections, sentiment analysis was used to gauge the feelings of core voters. Understanding the opinions of voters can help campaign managers and candidates understand what issues most concern the public, and ultimately swing an election.

The possibilities are near endless, and more and more creative ways are being thought out of to apply sentiment analysis in our everyday lives. Spam detection algorithms use sentiment analysis in their pipeline when detecting if an e-mail is spam or not.

Advertisers like Google can gain an insight into what a successful ad looks like for a specific user on their platform like YouTube, where each ad is tailored as much as possible to the viewer, and where sometimes an ad can be skipped after 5 seconds. Last, but not least, the financial and medical field also see benefits in using sentiment analysis. In a study, sentiment analysis was applied to examine how exposure to messages about a drug used to treat nicotine addiction affected the decision of smokers to use it or not [7-8][22][25].

3.5. Previous Research

Because there are a lot of different ways to approach the problem, sentiment analysis research has branched into several academic fields, where many different techniques to tackling the problem have been, and are being developed. One such field is Natural Language Processing (NLP), in which sentiment analysis is commonly seen as a subarea. Some researchers have gone as far as saying that every subproblem of NLP is also a subproblem of sentiment analysis, and vice versa. The argument is that sentiment analysis touches every core area of NLP, such as lexical semantics, coreference resolution, and word sense disambiguation. In general, it can be said that sentiment analysis is a semantic analysis problem, but it's highly focused and restricted because a sentiment analysis method does need to fully "understand" each text—it merely needs to assimilate some forms of it—mainly positive and negative opinions on an entity. It is

not then uncommon to see that in the following sentiment analysis techniques or approaches, NLP is involved in one way or another [24-26].

3.5.1. Naïve Bayes Classifier

Naïve Bayes Classifier is a popular supervised learning method that stems from the Machine Learning field, and has been adopted for sentiment analysis in some of the earliest research. As we mentioned before, there are multiple algorithms and methods, some of which more effective than others, but Naïve Bayes is a very good baseline and is fairly effective itself. It is a probabilistic classifier based on the Bayes Theorem. Because sentiment analysis is a text classification problem, different supervised learning methods can be applied, including support vector machines (SVM) and naïve Bayes classification. Pang, Lee, and Vaithyanathan were one of the first to propose and experiment with using a multinomial naïve Bayes classifier for sentiment analysis in 2002, for their paper *Thumbs up? Sentiment Classification using Machine Learning Techniques* [23]. The authors of the paper were able to achieve near 80% accuracy using the Bayesian classifier in their experiments. Since then, there has been an increase in the amount of research using this method, and numerous other papers have been published.

The Bayesian classifier is called naïve because it makes independent (naïve) assumptions about how features interact and are ordered. For example, when looking at a text, it does not take into effect the order of words appearing, or if phrases can be identified by seeing the same groups of words following or preceding each other. The text is represented as if it were a "bag-of-words" or a set, with only a frequency count of the number times that a word appeared in that text, and their positions are ignored. *Fig. 5* demonstrates how a text document, in this case a movie review, is broken down into words with only the frequency count of each word tracked. The word "seen" appeared 2 times, the word "it" appeared 6, etc. [9].

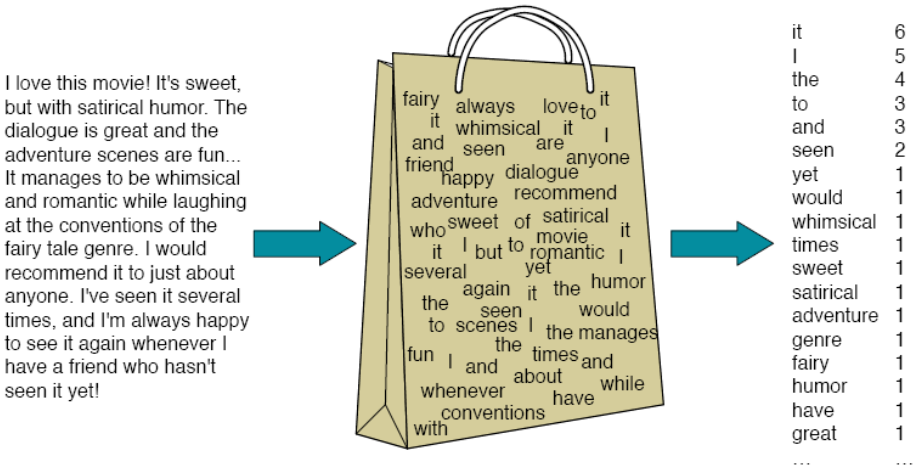

Figure 5. A "bag-of-words" assumption for Naïve Bayes Classifier. Word position is ignored, and a frequency of each word is stored. [9]

The implementation of the algorithm can be best described by the following three phases:

- **Training Phase:** The algorithm is trained on documents already classified as positive, negative, or neutral. This can be manually classified training sets,

dictionary words, phrases, etc. We want some baseline data for which we are very certain of its classification.

- **Testing Phase**: After training, the algorithm is tested to calculate the accuracy. Sometimes adjustments like tweaking the probabilities for how missing training words are handled, or adding more training data is used to improve accuracy during in this step.
- **Classification Phase:** The algorithm is given previously unseen text as input to classify as positive, negative, or neutral.

Outline

Let's look at a step-by-step outline of how the classifier works, after which we will follow with a basic worked out example in the next section.

1. We take some training documents for which we know the classification. For each document, we will break it down into words, count how many times each word appears, only keeping a vector of word and frequency at the end.
2. From the training documents, compute the probability of each class, P(c). That is, if we have 10 training documents, and 4 of them are positive, P(+) = 4/10.
3. For all classes, compute the probability of each word: $P(w_k|c) = \frac{n_k+1}{n+|vocabulary|}$. Let n, be the total number of words in the class c. Let n_k be the number of times that a word k occurs in the class c. Note that if a word does not appear in a class it

will still have some small probability value, and not equal 0. Just because an event has not happened before, does not mean that it will never happen. This is referred to as additive smoothing or Laplace smoothing, and the smoothing factor can be tuned accordingly.

4. Test with a test document unseen by the training phase to verify. Compute all the probabilities of the test case for each class multiplied by the probability of the class, and select the one with the highest probability: $V_{NBC} = \underset{v_j \, \varepsilon \, V}{\operatorname{argmax}} P(v_j) \prod_{w \, \varepsilon \, words} P(w|v_j)$. Let V stand for value or class. If a word in the test case does not appear in the training data, the simplest and most common solution is to remove it.
5. Finally, use the classifier like in the test phase on never before seen documents we want to classify.

Example

We show what running the Naïve Bayes Classifier looks like on a set of very set limited data. For simplicity we will only work with two classes (positive and negative), but adding more (like neutral) is trivial, and will not change anything previously defined. We would still compute the probability of each class in the usual way, and select the class with the highest probability. Let's consider the following very basic documents and classifications as our training and test data [9]:

ID	Case	Class (+ / -)	Document
1	Train	Positive	Very powerful
2	Train	Positive	The most fun film of the summer
3	Train	Negative	Just plain boring
4	Train	Negative	Entirely predictable and lacks energy
5	Train	Negative	No surprises and very few laughs
6	Test	?	Predictable with no fun

Table 1. Training and predict data for Naïve Bayes Classifier.

We first compute the number of occurrences of each word, then compute the probability of each class, and finally the individual probability of each word in each class. We will again use a chart to better visualize this. The probability of each class is:

$$P(+) = \frac{2}{5} \quad P(-) = \frac{3}{5}$$

For the positive class:

ID	Very	Powerful	The	Most	Fun	Film	Of	The	Summer
1	1	1	0	0	0	0	0	0	0
2			1	1	1	1	1	1	1

Table 2. Naïve Bayes word count of positive class.

Each word appears only once, but in the event that it appeared more we would count it. The number of words in the positive class (n) is 9, and the size of the vocabulary is 20. The same table can be drawn for the negative class documents. Next, we compute the individual probabilities of each word for each class. To simplify things again, we will only show the words that we need for the test phase, "predictable", "no", and "fun", but in order to speed things up for future computations the algorithm implementation in reality will compute all the words in the training data. In the real-world we could be dealing with thousands or millions of training documents. The word "with" was

discarded as mentioned before, because it does not appear in any of the training documents for any of the classes.

$$P(\text{predictable}|+) = \frac{0+1}{9+20}\ ; P(predictable|-) = \frac{1+1}{14+20}$$

The word "predictable" does not occur in any documents classified as positive, there are 9 total words in the positive class, and there are 20 total words in the vocabulary. "Predictable" appears once in the negative class, there are 14 total words in the negative class, and there are 20 words in the vocabulary.

$$P(no|+) = \frac{0+1}{9+20}; P(no|-) = \frac{1+1}{14+20}$$

$$P(fun|+) = \frac{1+1}{9+20}; P(fun|-) = \frac{0+1}{14+20}$$

Finally, we compute the total probabilities for each class for the sentence V = "Predictable with no fun" and select the larger value.

$$P(+)P(V|+) = \frac{2}{5} * \frac{1*1*2}{29^3} = 3.2 * 10^{-5}$$

$$P(-)P(V|-) = \frac{3}{5} * \frac{2*2*1}{34^3} = 6.1 * 10^{-5}$$

The value with the higher probability is the negative class, and thus our classifier has concluded that the sentence V is negative. Of course, this is only a simple example to give an overview and in real-life applications these computations become much more complex. Log-space reduction is used in order to increase speed and avoid underflow.

3.5.2. Recursive Neural Models

Recursive Neural Models are characterized by their use of vector representations. Vectors (ordered set of numbers) are used to represent words, as well as all sub-sentences related to an input's syntax tree. Word representations are trained with a model, and the representations of sub-sentences are calculated with a compositionality function. To calculate the sub-sentence's representations, a bottom-up compositionality function is applied according to the input's parse tree. Finally, all vectors are fed to the softmax classifier to determine the sentiment. The differences between the recursive models come in the choice of compositionality function. The most popular models are Recursive Neural Network (RNN), Matrix-Vector RNN, and the newly developed Recursive Neural Tensor Network (RNTN) discussed in the paper below.

Sentiment Treebank

The paper titled *Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank* by Socher R., Perelygin A., Wu J., Chuang J., Manning C.D., Ng A., and Potts C., was published in 2013 and first proposed the use of a Sentiment Treebank and Recursive Neural Tensor Networks (RNTN). The paper stated that in order to further progress for more accurate sentiment analysis techniques, richer supervised training and evaluation resources and more powerful models of composition were needed. The main ideas behind the Sentiment Treebank was for it to aid semantic word spaces

(representations of natural language able to capture meaning) in expressing the meaning of longer phrases in a principled way—a model to train on. RNTNs were originally designed for sentiment analysis and to analyze data that had hierarchical structure. Unlike naïve Bayes classifier, where words are considered individually, RNTNs compute the sentiment of a sentence not only on its words, but on the order in which they are syntactically grouped.

The results were very promising, as the authors were able to increase positive/negative single sentence classification up to 85.4% from 80%. Also, the accuracy of predicting fine-grained sentiment labels for all phrases was at 80.7%, 9.7% above baselines in previous research. The model was also able to accurately capture the effects of negation and its scope for positive and negative phrases [2].

The Sentiment Treebank contains fine grained sentiment labels over five classes for 215,154 phrases. In addition, it houses a collection of 11,855 sentences with fully labelled parse trees. The paper proposes a Sentiment Treebank used for training, which includes labels for every syntactically plausible phrase in thousands of sentences. In order to improve analysis results, the word order in a sentence should be taken into account, and the sentiment treebank assists in doing so. It allows for better predictions of longer,

more complex phrases as well as more accurately predicting negation of phrases. “Not very good,” is an example of such a phrase [2].

To build the Sentiment Treebank, the authors used movie reviews from the rottentomatoes.com website. Their original dataset included 10,662 sentences, half of which were positive and the other half negative. Different phrases were then extracted from the reviews and classified by humans on Amazon’s Mechanical Turk platform. This is how the resulting 215,154 phrases were labeled over five classes, from very negative to very positive.

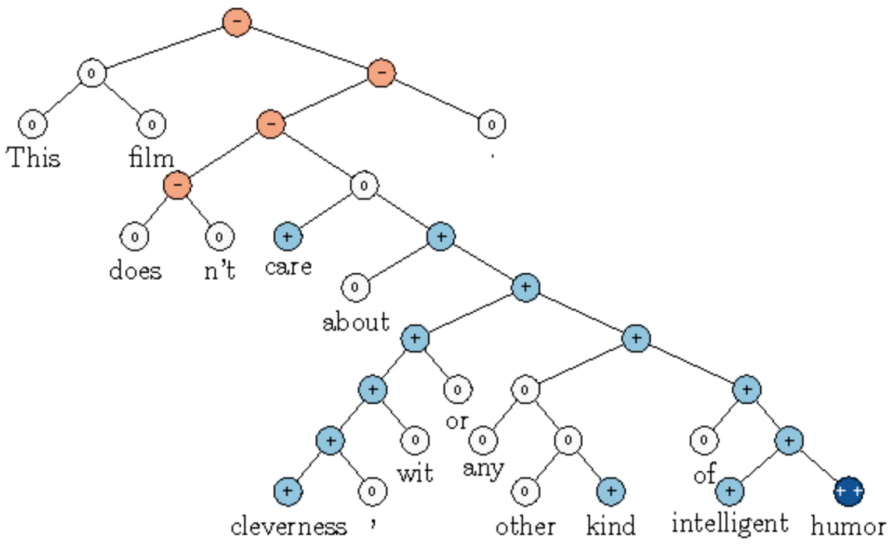

Figure 6. Example of a phrase inside the Sentiment Treebank. [2]

In *Fig. 6* we show an example of a phrase inside the Sentiment Treebank—each data point is the binary syntax tree of a rotten tomatoes review. The tree’s root, as well as its child nodes, are labeled with sentiment values between 1 and 25, with 25 being the best

possible review and 1 being the worst (crowdsourced on Amazon Mechanical Turk). RNTNs will use these parse trees to compute parent vectors in a bottom up fashion.

Recursive Neural Tensor Network

The Neural Tensor Network (RNTN) is a model to used learn these fine-grained sentiment labels, and when trained on the new Sentiment Treebank, the model is shown to outperform previous methods on multiple metrics. A deeper examination into how the classification function and data flows recursively through an RNTN can be found in [2][5]. We will show a brief overview of how the RNTN looks and works.

Breaking it down, an RNTN is a binary tree with a root and two children as shown in *Fig. 7.* Each child and root are a collection of neurons, and the number of the neurons depends on the complexity of the input. The child or leaf nodes receive input words and the root uses a classifier to produce a class and a score.

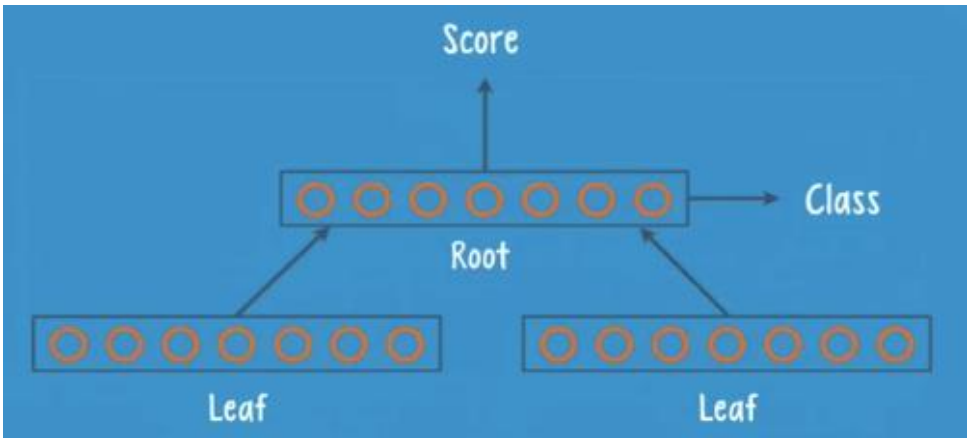

Figure 7. Structure of a RNTN. [5]

As its name implies, data flows recursively through the network. Given an input sentence, the network produces a parse tree, as depicted in *Fig. 8* for the sentence "The car is fast." In the first step, the first two words are used as input to the leaves at the very bottom. Because these networks work best with vector representations of the words, when we say that they are used as input, we mean that their vector representations are used. The two vectors move up to the root where they are processed, and a class and score are computed. The score represents the quality of the current parse, and the class represents an encoding of a structure in the current parse. At this stage is where the recursion begins and the tree is built out upward as shown in *Fig. 8*. In the next step, the current parse is used as one leaf and the next vector representation of the word "is" is used as the other. At this stage the new root would produce the score of a parse that is three words long ("The car is"). The recursion continues until all the input words are exhausted.

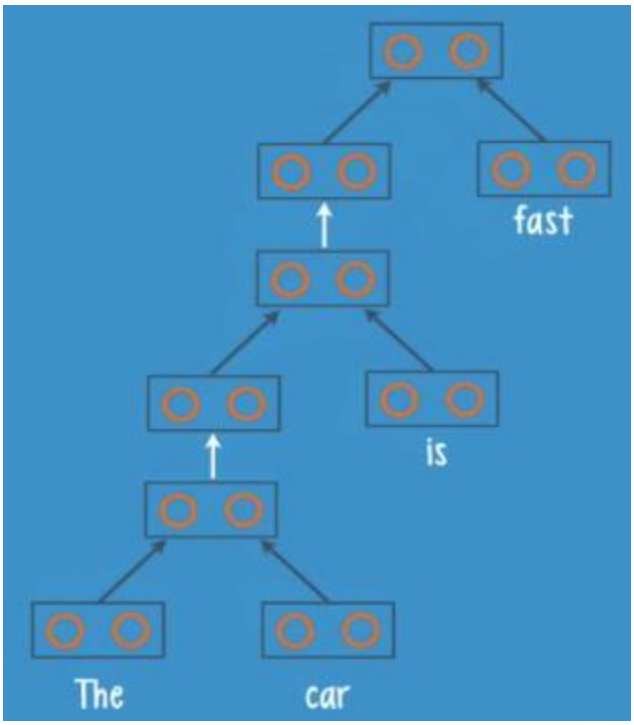

Figure 8. Parse tree of a sentence produced by a RNTN. [5]

We just showed a very basic parse tree being produced, but in a real-life application, more complex recursive processes are encountered. For example, instead of always using the next word in the sentence for the second leaf node, the RNTN would compute new parse trees from all of the remaining words. This way, the RNTN is able to pass through and score every possible parse of the input sentence. In order to pick the best one at the end, the network uses the score produced at the root node in each recursion. *Fig 9* shows the other two possible parse trees for the example sentence. Once the RNTN has found the optimal parse tree, it backtracks through it in order to figure out the correct grammatical label for each part of the sentence. For example, it will go back and label "The car" as a noun phrase, and "is fast" as a verb phrase.

RNTNs are trained with backpropagation by comparing the predicted sentence structure with the proper sentence structure obtained from a set of labeled training data. Once trained, the RNTN will give higher scores to structures that are more similar to the parse trees that it saw in the training phase [5].

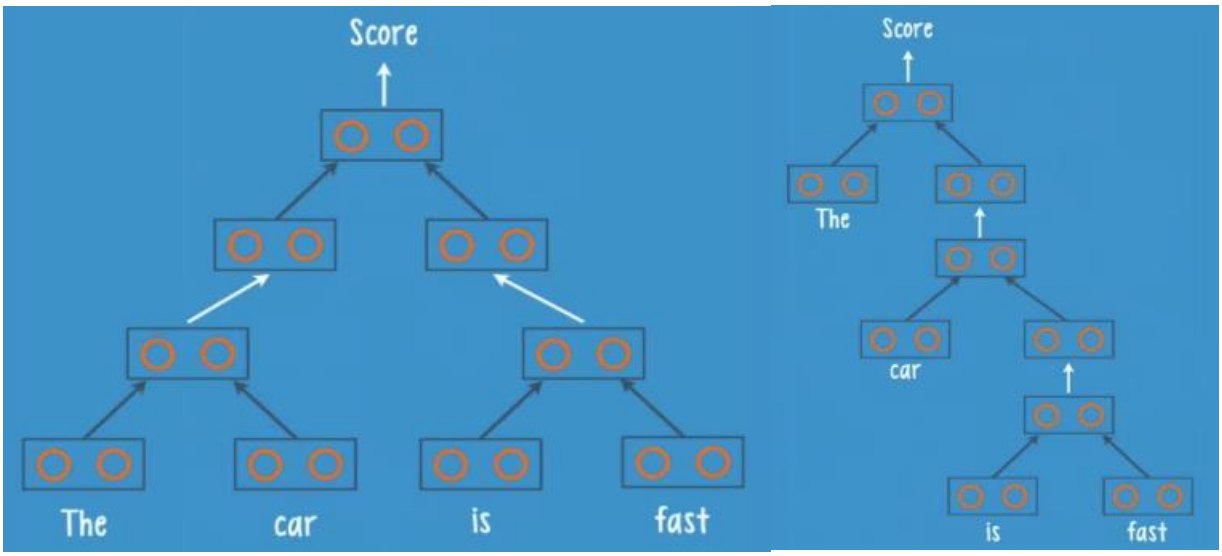

Figure 9. Remaining two possible parse trees for the input sentence. [5]

Stanford's CoreNLP Library

The CoreNLP library is an open-source natural language processing library from Stanford that contains many language processing tools (called annotators), one of which is to perform sentiment analysis [3]. The sentiment analysis annotator is based entirely off of the above research and comes pre-trained, giving us the ability to quickly process the sentiment of an input sentence with certain accuracy. For an input sentence, the library produces an integer score in the range 0-4:

Score	Mapping
0	Very Negative
1	Negative
2	Neutral / No Sentiment
3	Positive
4	Very Positive

Table 3. CoreNLP sentiment scoring.

Another annotator that we will use from this library is the "ssplit" annotator to split our review text into multiple sentences. After tokenizing a text, the ssplit annotator produces a list of sentences that are contained within the original text. Each sentence then will be the input to the sentiment analyzer.

3. Dataset and Overview

In order to run our experiment, we use a rich dataset of real life Yelp reviews provided by Yelp as our baseline data input. [1] The entire dataset consists of 5.2 Million reviews for 174,000 different businesses. Although the dataset also contains many more interesting features, the main ones we care about are the review texts and users.

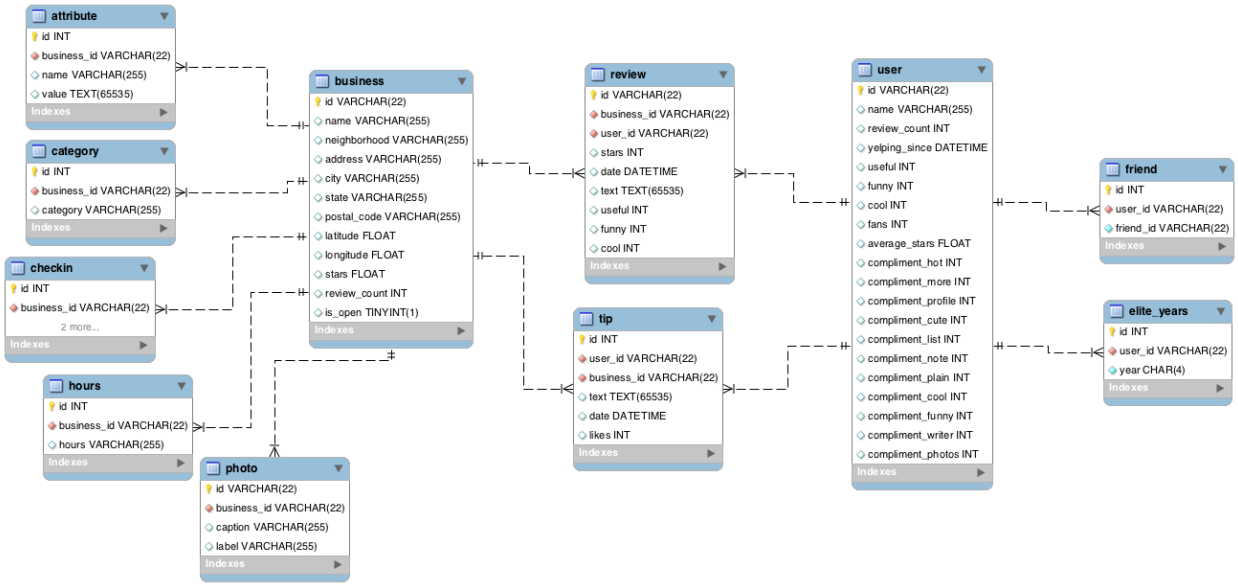

Figure 10. Schema representation of the Yelp Dataset.

3.1. Preprocessing

From the total dataset, only users with 50 or more reviews are extracted and the rest are discarded. We do this because there should be a large enough set of reviews written by the same user in order to detect patterns in his or her behavior. What is left is the following dataset which we use as our starting point:

Total Users:	9,701
Total Reviews:	1,079,812

Table 4. Working dataset size.

3.2. System Overview

Given the nature of our operations and the size of our starting dataset, we will be producing large amounts of data with multi-million records. The CoreNLP library used for our sentiment analysis and sentence tokenizer is native to Java and is run in a parallel across multiple machines and threads to speed up our computations.

Intermediate and final results are persisted in a Mongo database. Finally, to piece everything together and compute the final results, Apache Spark is used as the compute engine. Spark is extremely efficient at parallelizing computations by performing them in different stages and can handle inputs in the hundreds of millions without any trouble.

4. Sentiment Vectors and Tuples

We first get a feeling for the structure of what each review looks like so that we can match if there are similar patterns found in other reviews left by the same user. As mentioned before, we perform sentiment analysis on each review, but on a sentence level. This gives us insights into the opinions expressed by the user throughout the entire review. The end result of the sentiment analysis phase is the production of a "sentiment vector". The sentiment vector contains the sentiment scoring for each sentence in the review. For example, if the user left a five-sentence review, with the first three showing positive sentiment, the fourth sentence showing neutral or no sentiment and the last sentence showing negative sentiment, then the sentiment vector for that review will look like this: 33321.

From each sentiment vector, we generate all possible sentiment tuples (orderings, analogous to n-grams in computational linguistics) of the vector by looking at smaller and smaller pieces of it. This allows us to break down longer reviews and it is how we

will identify repeating patterns if they exist. For example, for the sentiment vector mentioned above (33321), we generate all possible sentiment tuples of length 3 or longer (3 is used as the minimum because 2 is too ambiguous and does not provide much insight into an existing pattern): 3332, 3321, 333, 332, 321.

For a sentiment vector of length n, there will be $\frac{(n-2)(n-1)}{2} - 1$ total tuples of length ≥ 3.

4.1. Defining Abnormality

Given what we now know about each review, we are able to identify reoccurring patterns via matchings in our tuples and answer a few questions. Does a user leave the same patterns or partial patterns within his or her reviews? Does this pattern appear often in all of the reviews left by the user? Is the pattern being discovered throughout the reviews long? Answering yes to any of these questions is considered abnormal, and we should account for any of these scenarios. We attribute a weight to each of these cases and quantify how abnormal each of these occurrences is.

Repetition

We care about a pattern reappearing throughout a user's reviews because it is abnormal and should not be happening in a normal scenario. We account for this by measuring how often an exact pattern has appeared in the user's review vs. the normal (expected) occurrence. Thus, a large tuple appearing only once will cancel itself out and have zero

score, while the opposite is true for a long one that reappears. A tuple appearing only a few times will be treated as normal and not contribute much to the repetition tuple score as expected. For each of the user's tuples we compute:

Times observed tuple T = $\frac{\#\ of\ \text{occurrences of T}}{total\ occurances\ of\ tuples\ of\ length\ of\ T}$

Expected to observe tuple T = $\frac{1}{total\ unique\ tuples\ of\ length\ of\ T}$

Repetition tuple score = $|Times\ Observed\ T - Expected\ to\ Observe\ T|$

For example, if we are looking at a user's tuples of length 5 and they contain: 33321, 33321, 33321, and 7 others that do not repeat and are unique, we have:

Times observed = $\frac{3}{10}$

Expected to observe = $\frac{1}{8}$

Repetition tuple score = $\left|\frac{3}{10} - \frac{1}{8}\right| = 0.175$.

Frequency

The next thing we care about is the frequency of the tuple—if a certain tuple appears multiple times, across multiple reviews, this is abnormal. For each tuple we count how many times this tuple appears in a review and divide by the total number of reviews.

Tuple frequency = $\frac{\#\ of\ reviews\ this\ tuple\ occurs\ in}{total\ number\ of\ reviews\ the\ user\ has}$

If a specific tuple occurs in 10 of the 50 total reviews a user has, it has a tuple frequency of 0.2.

Length

Last but not least, we also care about the tuple length. If a tuple is reoccurring multiple times or with higher frequency, and is also with a large length, this is clearly abnormal. The chances of that happening for a normal user are extremely low, and we should take it into account. For the tuple length score, we simply use the length of the tuple.

4.2. Abnormality Score

For each tuple we are now able to produce a score and quantify how abnormal what we have observed is. The higher the score, the more abnormal this tuple is. We square and multiply all of the values together giving appropriate boosts to more significant values, like the length for example.

Tuple abnormality score = $Repetition\ Tuple\ Score^2 * Tuple\ Frequency^2 * Length^2$

We now have for every user, and every tuple for that user, a tuple abnormality score. Summing them all up will give us the total abnormality score for the user itself.

5. Implementation

We use two different methods for sentiment analysis, and later compare the results. We do this in order to gain a higher confidence in our results, and the users we flag as abnormal.

CoreNLP

After preprocessing, we run the sentiment analysis on each review, tokenizing it into sentences, and generating a sentiment vector for each review. We use the source code found in *Appendix A1* for this step. The entire process has to be parallelized as this is the slowest computation phase of the entire project, which took around ten days to complete. The 1.1M initial reviews generated 12.6M sentences, each of which is passed through sentiment analysis. A local 6-core CPUs was used as well as a 64-core Google Cloud Compute instance to process all of the data in this step. Since the CoreNLP library comes pre-trained we can simply feed it our input, and it will return a sentiment value for that input.

After completion and merging of all results, we are left with two intermediate tables. A reference table holding the user id, review id, sentence, sentence position in the review, and the sentiment score for each sentence. And a second table holding the sentiment vector for each review, which we use next to generate the sentiment tuples from. We now are able to associate a specific review to a sentiment vector.

The next step is to dive deeper and allow ourselves to find patterns for the reviews by generating the sentiment tuples for each vector. For each sentiment vector longer than 3 we generate tuples of length n-1 to 3, and if the vector is of length ≤ 3 we simply use it

as a tuple. We use the source code found in *AppendixA2* for this step. We now are able to associate a sentiment vector with all possible tuples of length 3 or more for that vector. This allows us to perform pattern matching and see if we are able to find any reoccurring patterns among a user's reviews. This intermediate table is the final table needed in order for us to compute the associated tuple abnormality scores and user abnormality scores.

The final step is to compute the results and save them in a new table for later review. From the generated sentiment tuples intermediate table, we compute the results table that contains every unique tuple associated with each user and all statistics and scores for that tuple. We use Apache Spark for this last step and the source code in *Appendix A3*. Tuples scoring high are considered abnormal. Summing up the abnormalities per user gives us the total user abnormality score, and sorting by that score gives us the top users.

Naïve Bayes

For the naïve Bayes sentiment analysis method, we need to mirror the exact steps above in the CoreNLP method, with only one difference: we will need to train our model. Unlike CoreNLP, which comes pre-trained, the naïve Bayes model needs to be trained in order to predict a result. We use the remaining data which we discarded during our

preprocess phase. Namely, the reviews from users with less than 50 total reviews amounting to roughly 4.1M in total.

In order to improve sentiment results, we also remove stop words from our dataset. Words like “a”, “and”, “but”, “how”, “or”, “what”, etc. do not add value to sentiment and are removed in order to not pollute our word counts. We train and save our model with the source code in *Appendix B1*. Next, with *Appendix B2* we run our model to predict the sentiment for the 12.6M sentences previously identified. At this stage we are ready to produce our review sentiment vectors and tuples—done so with the source code provided in *Appendix B3*. Apache Spark and *Appendix A3* is used again for the final computation of our results. After completing our results table, we sum up the scores for all the users, sort them in descending order by their abnormality score and have the top users identified for abnormalities in their reviews.

6. Results

Validation

We first look at our computed user abnormality scores and validate that each of the sentiment analysis methods produced viable results, by looking at the user score distributions. Flagged users, who had abnormal reviews that triggered our abnormality filters can be identified by looking at users with scores one ($> 1\sigma$) and two ($> 2\sigma$) standard deviations away from the mean. From the 9,701 users studied, we see only a

small number of users that stand out, have bubbled up, and should be examined more closely. *Figure 11* describes the user abnormality score distribution for the different CoreNLP and Naïve Bayes sentiment analysis methods. We are concerned with roughly looking at less than 1% of the total users studied, which validates our initial thoughts and definition of abnormality. That is, that only a small number of users, if any, should be bubbling up.

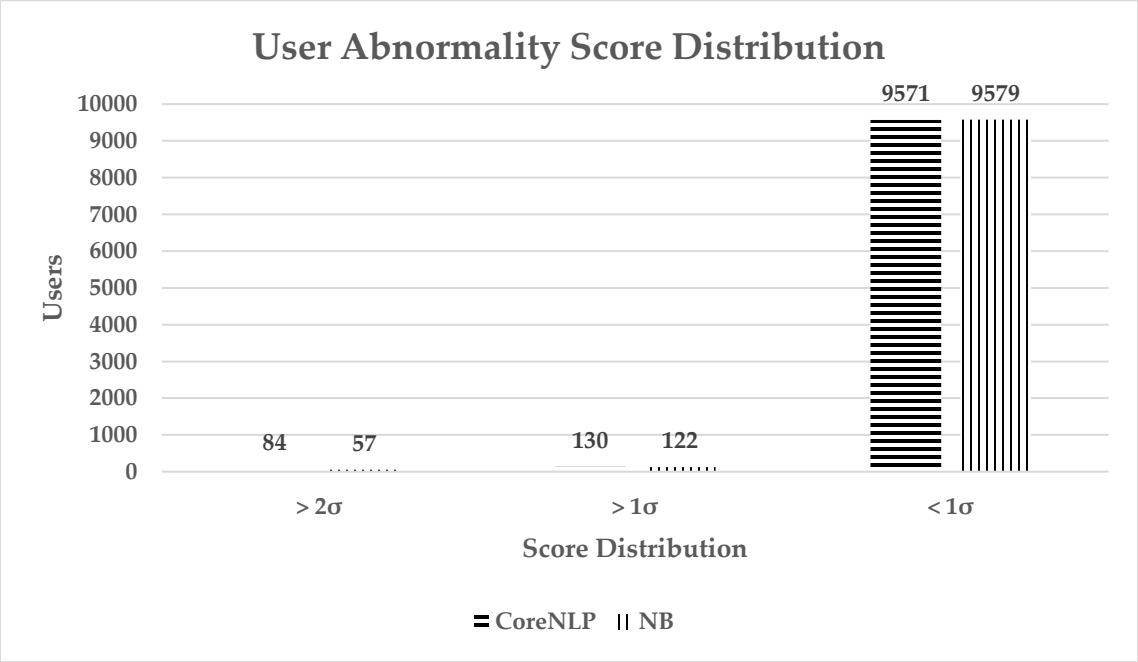

Figure 11. Results for the user abnormality score distribution.

Next, we look to see if our different sentiment analysis methods produced overlapping results, further strengthening our evidence against the top abnormality scored users. If we take the top abnormality scored users from both methods, and see that a user appears in both, we can say with even high confidence that this is not due to a coincidence. This means that a user's reviews triggered the abnormality definitions in

both sentiment analysis methods, and that we have scored these abnormalities accordingly. Looking at our distribution, we should be concerned with looking at roughly the top 150 users from the 9,701 initially studied. *Figure 12* displays the match rates for the top 150 abnormality scored users in both, the CoreNLP and Naïve Bayes methods.

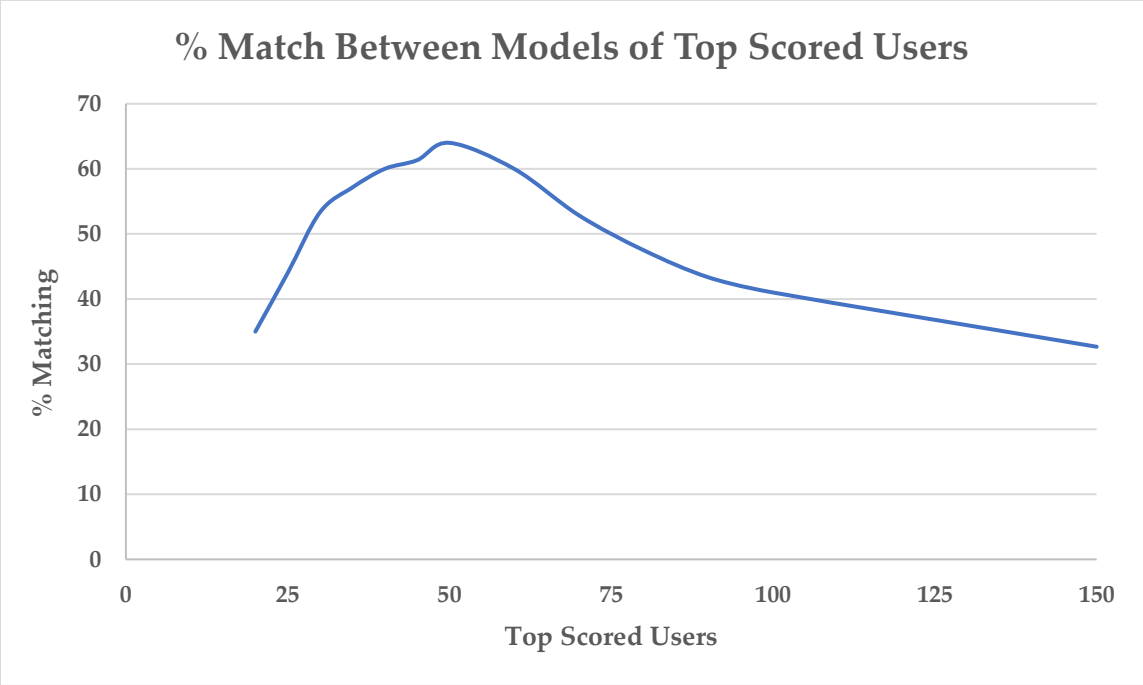

Figure 12. Results for the top abnormality scored users match between the CoreNLP and Naïve Bayes methods.

Examples

Let's look at some concrete examples of users and reviews to see why they have bubbled up in our results. We use users that appear in both the CoreNLP and Naïve Bayes methods, and as *Figure 12* demonstrates that for the top 50 users, we have a 65% match. Here are reviews from two different top scoring users:

"Je ne suis pas d'accord avec la critique précédente de Joshua L. au sujet de ce resto vietnamien. Contrairement à ce qu'il prétend, le service ici est correct, courtois et rapide. Je suis venu ici avec un collègue et le serveur est venu nous voir dès notre arrivé. Nos assiettes ont été servies en 5 minutes, ce qui est un délai très raisonnable"…

"Backwaren, die noch schmecken wie beim Bäcker. Ja ich weiss es gibt tausende Bäcker, und auch der Treiber hat mittlerweile ein Filialangebot das sich auf 18 Geschäfte erstreckt, aber irgendwie ist er eben anders. Zunächst gab es die leckeren Filderwäckle nur auf den namensgebenden Fildern, doch schon bald stellte das clevere Unternehmen fest, dass man auch in der näheren Umgebung und in einigen Stadtteilen von Stuttgart die aromatischen Brötchen, Brezeln und Brote, Kuchen, süße Stückle lieben wird"…

The first thing we notice is unexpected, yet a logical result based on the analysis and pattern matching we performed. We notice that some of the top abnormally scored users are German or French. The sentiment analysis models we used were trained on the English language, and there was no way for them to distinguish users based on their nationality or language. Since the sentiment analyzer didn't understand the text it classified the foreign language sentences very similarly. This is what made the patterns look similar and gave them more weight leading them to bubble to the top. A high percentage of sentences were scored as 1's, producing many similar looking patterns. This on its own can be considered an abnormality in our case and can be viewed as a successful identification.

Next, we look into a user that had an abnormally long tuple of length 55 (1221131131211111331111311111111113412311122121111211121) appear in 4 of his 55 reviews. It turns out the user had copy/pasted the following long text into these reviews, all four for different businesses, and our methods picked up on them:

"This is going to be a long one . . I used Grayline twice during my Vegas Vacation. First, as transportation to and from the airport. . .tip. . DO NOT BOOK ONLINE AHEAD OF TIME. Just get your tickets at the airport---I was charged $2 extra because I pre-booked (price online $16 price at airport $14). The shuttle to the hotel was very hot, sweat stains were visible on every seat--very appealing"…

We also notice a different user using extremely long repetition. A tuple of length 78 of all 2's (neutral sentiment) appeared. The user had used the word "waited" 78 times one after another in one of the reviews, and we classified it as abnormal:

Great place. Beautifully designed space. Food seems ok but since I've never eaten here I can't comment. Service is atrocious. I ordered. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited.Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited. Waited.Waited. Waited. Waited. Then half of my food showed up. Great. Wolfgang Puck is literally across the way. Why did I come here? Bad recommendation from a tasteless friend.

In another case, we notice that a significantly sized tuple (31112) has occurred 51 times in 60 of the user's total reviews. It turns out that this user has prefixed all 51 of these reviews with the same sentences, and later follows up with different text that are actually her review. Table 5 displays the repeating tuple in question, including the sentence and sentiment score breakdown.

Sentence	Sentiment Classification
OK, so check this out.	3
I know I have been long dormant with the Yelp reviews (bad girl, BAD GIRL!)	1
So to make it up to ya'll, I am now embarking on a casino-by-casino tour of my new hometown of LV, NV, and will include reviews of each as I tour them.	1
Now, these will not include room stays, but just schlepping around the premises - eating, drinking, and checking out anything that moves or doesn't move.	1
Got it?	2

Table 5. Example of a reappearing highly scored sentiment tuple's sentence and sentiment classification.

7. Conclusion and Future Work

We proposed a way to look at user reviews and signal for possible abnormalities by detecting patterns in the reviews which they leave. This alone is not enough to say with a high degree of certainty that a user is leaving fake reviews, but it can be used as an input to a more complex detection system to do so.

Our results showed that from the users we looked at, we detected possible abnormalities in less than 1% of them, classifying the rest as behaving normally. Based on the distribution of our results, it is recommended that users with scores placing above the one standard deviation mark should be more thoroughly examined.

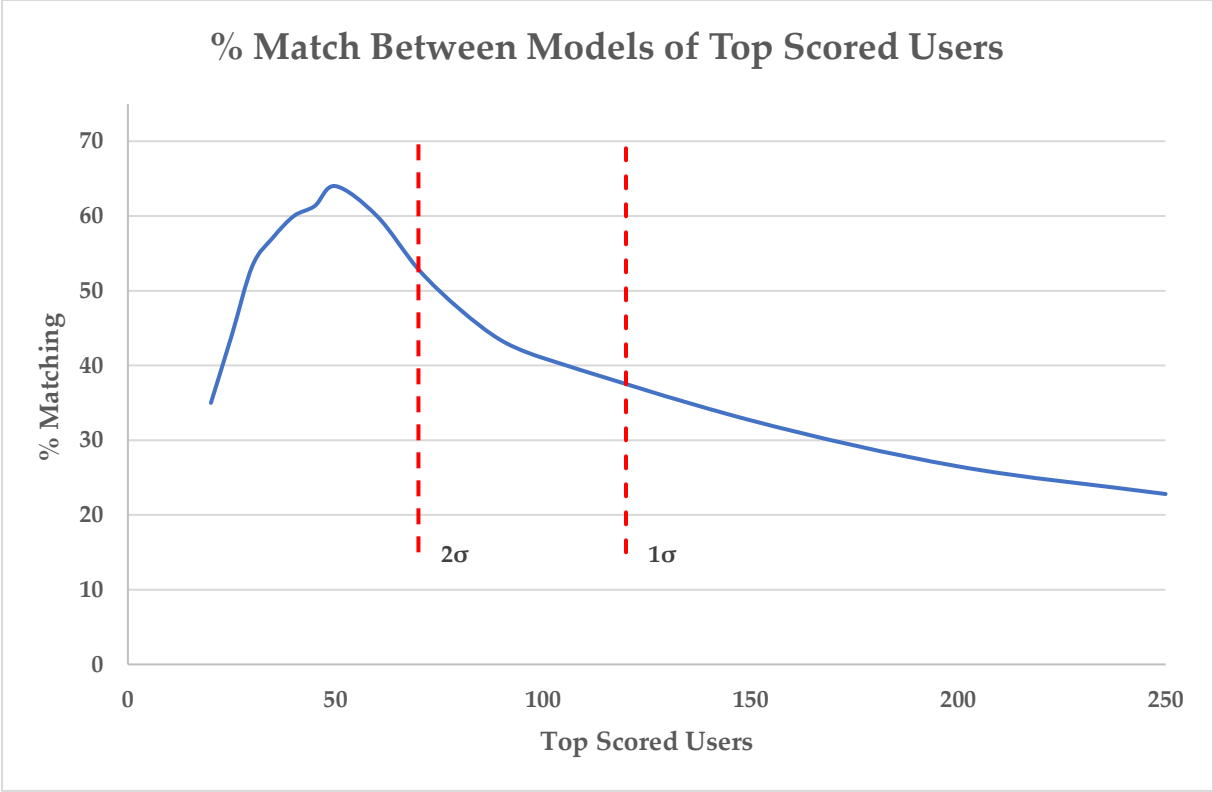

Figure 13. Statistical distribution of the top matching users between CoreNLP and Naïve Bayes.

Figure 13 demonstrates where the standard deviations marks fall under in our results. Scores lower than the one deviation mark become less significant, and are safe to discard from more rigorous tests.

We were able to detect users using repetition and copy/pasting inside of their reviews by properly associating a higher abnormality score to these sentiment tuples. Users using a foreign language also bubbled up in our analysis, but this is to be expected as the sentiment analyzer we ran in order to gain more insight into the review structure was trained on the English language—this on its own can be seen as properly

identifying an abnormality. There was also no way to discard non-English speaking users from the initial dataset.

A possible scenario for work in the future could include running more different types of sentiment analyzers and techniques to see how the results may vary. Some examples could be training and running classifiers based on word2vec and faxtText [27-28]. Another possibility for future work is to tweak the tuple abnormality scoring formula based on different types of input data.

It would be interesting to see how this model holds up against different types of datasets. Combining the different sentiment analysis models and our proposed pattern matching with a dataset from Amazon for example, could open up different features on which abnormalities can be identified and scored against. Instead of looking at the dataset only on a per user basis, expanding it across the entire dataset might also help identify networks of fraudulent users. It could also be that having many fraudulent users, with few reviews also adds to the challenge of identifying them.

References

[1] Yelp dataset challenge, 2018. https://www.yelp.com/dataset/challenge.
[2] Socher, R & Perelygin, A & Wu, J.Y. & Chuang, J & Manning, C.D. & Ng, A.Y. & Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. EMNLP. 1631. 1631-1642.
[3] Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60.
[4] Vincent, James. "AI Trained on Yelp Data Writes Fake Restaurant Reviews 'Indistinguishable' from Real Deal." *The Verge*, The Verge, 31 Aug. 2017, www.theverge.com/2017/8/31/16232180/ai-fake-reviews-yelp-amazon.
[5] *Deep Learning Fundamentals*. Nov. 2017, http://cognitiveclass.ai/courses/introduction-deep-learning.
[6] Joulin, Armand, et al. "Bag of tricks for efficient text classification." *arXiv preprint arXiv:1607.01759* (2016).
[7] Pozzi, Federico Alberto, et al. *Sentiment analysis in social networks*. Morgan Kaufmann, 2016.
[8] Liu, Bing. *Sentiment analysis: Mining opinions, sentiments, and emotions*. Cambridge University Press, 2015.
[9] Jurafsky, Dan, and James H. Martin. *Speech and language processing*. Vol. 3. London: Pearson, 2014.
[10] Jindal, Nitin, Bing Liu, and Ee-Peng Lim. "Finding unusual review patterns using unexpected rules." *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM, 2010.
[11] Mukherjee, Arjun, et al. "What yelp fake review filter might be doing?." *ICWSM*. 2013.
[12] Jindal, Nitin, and Bing Liu. "Review spam detection." *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007.
[13] Mukherjee, A., Liu, B., & Glance, N. (2012, April). Spotting fake reviewer groups in consumer reviews. In *Proceedings of the 21st international conference on World Wide Web* (pp. 191-200). ACM.
[14] Sandulescu, V., & Ester, M. (2015, May). Detecting singleton review spammers using semantic similarity. In *Proceedings of the 24th international conference on World Wide Web* (pp. 971-976). ACM.
[15] Li, Jiwei, et al. "Towards a general rule for identifying deceptive opinion spam." *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2014.

[16] Jindal, Nitin, and Bing Liu. "Opinion spam and analysis." *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM, 2008.
[17] Qian, Tieyun, and Bing Liu. "Identifying multiple userids of the same author." *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 2013.
[18] Peng, Qingxi, and Ming Zhong. "Detecting Spam Review through Sentiment Analysis." *JSW* 9.8 (2014): 2065-2072.
[19] Li, Fangtao, et al. "Learning to identify review spam." *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*. Vol. 22. No. 3. 2011.
[20] Akoglu, Leman, Rishi Chandy, and Christos Faloutsos. "Opinion Fraud Detection in Online Reviews by Network Effects." *ICWSM* 13 (2013): 2-11.
[21] Eisenstein, Jacob, Amr Ahmed, and Eric P. Xing. "Sparse additive generative models of text." (2011).
[22] Yu, Boya, et al. "Identifying Restaurant Features via Sentiment Analysis on Yelp Reviews." *arXiv preprint arXiv:1709.08698* (2017).
[23] Pang, Bo, Lillian Lee, and Shivakumar Vaithyanathan. "Thumbs up?: sentiment classification using machine learning techniques." *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics, 2002.
[24] Dey, Lopamudra, et al. "Sentiment Analysis of Review Datasets Using Naive Bayes and K-NN Classifier." *arXiv preprint arXiv:1610.09982* (2016).
[25] Go, Alec, Richa Bhayani, and Lei Huang. "Twitter sentiment classification using distant supervision." *CS224N Project Report, Stanford* 1.12 (2009).
[26] Pang, Bo, and Lillian Lee. "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts." *Proceedings of the 42nd annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2004.
[27] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).
[28] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.
[29] Goldberg, Yoav, and Omer Levy. "word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method." *arXiv preprint arXiv:1402.3722* (2014).
[30] Babu, Prashanth. *Twitter Sentiment Analysis with Spark MLlib*. Sept. 2016, github.com/P7h/Spark-MLlib-Twitter-Sentiment-Analysis/wiki.
[31] de Toit, Jurgens. *The Bayes Classifier: Building a Tweet Sentiment Analysis Tool*. Aug. 2015, cloudacademy.com/blog/naive-bayes-classifier.
[32] Czerny, Michael. *Modern Methods for Sentiment Analysis*. Mar. 2015, districtdatalabs.silvrback.com/modern-methods-for-sentiment-analysis.

Appendix: Source Code

The entire source code for this project is available at the following Git repository: https://github.com/sgloutnikov/masters-writing-project

A1. CoreNLP Sentiment Analysis and Vectors

```
public class SentenceSentimentWorker implements Runnable {

    String host = "";
    int port = 27017;
    int limit;
    int skip;

    MongoClient mongoClient = new MongoClient( host, port);
    MongoDatabase database = mongoClient.getDatabase("yelp_reviews");
    MongoCollection<Document> sentimentResults =
database.getCollection("sentiment_results");
    MongoCollection<Document> sentimentVectors =
database.getCollection("sentiment_vectors");
    MongoCollection<Document> dataCollection = database.getCollection("review_50");

    public SentenceSentimentWorker(int limit, int skip) {
        this.limit = limit;
        this.skip = skip;
    }

    public void run() {
        // Set annotators
        Properties props = new Properties();
        props.setProperty("annotators", "tokenize, ssplit, parse, sentiment");
        StanfordCoreNLP pipeline = new StanfordCoreNLP(props);

        System.out.println(Thread.currentThread().getName() + " starting...");
        try {
            Gson gson = new GsonBuilder().create();
            JsonParser jsonParser = new JsonParser();

            //Get input from MongoDB
            List<Document> inputList = dataCollection.find().sort(ascending("_id"))
                    .limit(limit).skip(skip).into(new ArrayList<Document>());

            List<Document> sentencesDocList = new ArrayList<Document>();

            for (Document review : inputList) {
                JsonElement reviewJson = jsonParser.parse(review.toJson());
                JsonObject reviewObject = reviewJson.getAsJsonObject();

                String reviewId = reviewObject.get("review_id").getAsString();
                String userId = reviewObject.get("user_id").getAsString();
                String sentimentVector = "";

                Annotation annotation =
pipeline.process(reviewObject.get("text").getAsString());
                List<CoreMap> sentences =
annotation.get(CoreAnnotations.SentencesAnnotation.class);
                for (int i = 0; i < sentences.size(); i++) {
```

```
                    CoreMap sentence = sentences.get(i);
                    Tree tree =
sentence.get(SentimentCoreAnnotations.SentimentAnnotatedTree.class);
                    int sentiment = RNNCoreAnnotations.getPredictedClass(tree);
                    sentimentVector += sentiment;

                    ReviewSentenceSentiment rss = new
ReviewSentenceSentiment(reviewId, userId,
                            sentence.toString(), i, sentiment);

                    Document sentenceDoc = Document.parse(gson.toJson(rss));
                    sentencesDocList.add(sentenceDoc);

                }

                SentimentVector sv = new SentimentVector(reviewId, userId,
sentimentVector);
                Document sentimentVectorDoc = Document.parse(gson.toJson(sv));

                // Save to Mongo
                sentimentVectors.insertOne(sentimentVectorDoc);
                sentimentResults.insertMany(sentencesDocList);
                sentencesDocList.clear();

            }

            System.out.println(Thread.currentThread().getName() + " DONE!");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        finally {
            mongoClient.close();
        }
    }
}

public class SentenceSentimentApp {
    public static void main(String[] args) {

        List<Thread> threadList = new ArrayList<Thread>();
        int NUM_THREADS = 8;
        int limit = 1250;
        int skip;

        for (int i = 0; i < NUM_THREADS; i++) {
            // Num skipped initially of already computed
            skip = 170000 + (i * limit);
            System.out.println("Thread-"+ i + " range: " + skip + "-" + (skip+limit));
            SentenceSentimentWorker worker = new SentenceSentimentWorker(limit, skip);
            Thread thread = new Thread(worker, "Thread-" + i);
            threadList.add(thread);
        }

        for (Thread t : threadList) {
            t.start();
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    }
```

```
}
```

A2. Tuple Generation

```
public class TupleGenerator {
    public static void main(String[] args) {
        String host = "";
        int port = 27017;
        int skip = 0;
        TupleGenerator tg = new TupleGenerator();

        MongoClient mongoClient = new MongoClient(host, port);
        MongoDatabase database = mongoClient.getDatabase("yelp_reviews");
        MongoCollection<Document> sentimentVectors =
database.getCollection("sentimentVectors");
        MongoCollection<Document> sentimentTuples =
database.getCollection("sentimentTuples");

        MongoCursor<Document> cursor = sentimentVectors.find().sort(ascending("_id"))
                .skip(skip).iterator();
        try {
            while (cursor.hasNext()) {
                Document sentimentVectorDoc = cursor.next();
                String userId = sentimentVectorDoc.getString("user_id");
                String reviewId = sentimentVectorDoc.getString("review_id");
                String sentimentVector =
sentimentVectorDoc.getString("sentimentVector");

                // check if length < 4 before generating, if so just insert sentiment
vector as tuple
                if (sentimentVector.length() < 4) {
                    Document tupleDoc = new Document().append("user_id", userId)
                            .append("review_id", reviewId)
                            .append("sentimentTuple", sentimentVector);
                    sentimentTuples.insertOne(tupleDoc);
                } else {
                    List<Document> tupleDocList = new ArrayList<Document>();
                    for (String tuple : tg.generateTuple(sentimentVector)) {
                        Document doc = new Document().append("user_id", userId)
                                .append("review_id", reviewId)
                                .append("sentimentTuple", tuple);
                        tupleDocList.add(doc);
                    }
                    sentimentTuples.insertMany(tupleDocList);
                }
            }
        } finally {
            cursor.close();
            mongoClient.close();
        }
    }

    /*
    Generate all tuples of length 3 or larger.
     */
    public List<String> generateTuple(String sVector) {
        List<String> sentimentTuples = new ArrayList<String>();
        int length = sVector.length();
        int k = length - 1;

        while (k > 2) {
            for (int i = 0; i + k <= length; i++) {
                String tuple = sVector.substring(i, i+k);
```

```
                sentimentTuples.add(tuple);
            }
            k--;
        }
        return sentimentTuples;
    }
}
```

A3. Computing Results

```
object Results {
  def main(args: Array[String]): Unit = {
    Logger.getLogger("org").setLevel(Level.ERROR)
    val sparkSession = SparkSession.builder()
      .master("local[*]")
      .config("spark.mongodb.input.uri",
"mongodb://localhost:27017/yelp_reviews.sentimentTuples")
      .config("spark.mongodb.output.uri",
"mongodb://localhost:27017/yelp_reviews.userAbnormalityScore")
      .getOrCreate()
    import sparkSession.sqlContext.implicits._
    import org.apache.spark.sql.functions._

    val allTuples = MongoSpark.load(sparkSession)
    val userTupleStats = allTuples.groupBy('user_id, 'sentimentTuple)
      .agg(countDistinct('review_id).as("foundInNumReviews"),
count('sentimentTuple).as("sentimentTupleCount"),
        length('sentimentTuple).as("tupleLength"))

    val usersReadConfig = ReadConfig(Map("collection" -> "users"),
      Some(ReadConfig(sparkSession)))
    val users = MongoSpark.load(sparkSession, usersReadConfig)
    val userTupleExtendedStats = userTupleStats.join(users, Seq("user_id"))
      .drop('_id).withColumnRenamed("count", "totalReviews")

    val userTupleExtendedStatsWithFreq =
userTupleExtendedStats.withColumn("tupleFrequency",
      'foundInNumReviews.divide('totalReviews))

    val userTupleLengthStats = userTupleExtendedStatsWithFreq.groupBy('user_id,
'tupleLength)
      .agg(sum('sentimentTupleCount), countDistinct('sentimentTuple))
      .withColumnRenamed("sum(sentimentTupleCount)", "totalTuplesOfLength")
      .withColumnRenamed("count(DISTINCT sentimentTuple)", "uniqueTuplesOfLength")

    val userTupleFullStats = userTupleExtendedStatsWithFreq.join(userTupleLengthStats,
      Seq("user_id", "tupleLength"))
        .withColumn("observedTupleLengthFreq",
'sentimentTupleCount.divide('totalTuplesOfLength))
        .withColumn("expectedTupleLenthFreq", lit(1).divide('uniqueTuplesOfLength))
        .withColumn("absDiffObsExpected",
abs('observedTupleLengthFreq.minus('expectedTupleLenthFreq)))
        .withColumn("abnormalityScore", pow('tupleFrequency,
2).multiply(pow('tupleLength, 2))
          .multiply(pow('absDiffObsExpected, 2)))

    val userAbnormalityScore =
userTupleFullStats.groupBy('user_id).agg(sum('abnormalityScore)
      .as("userAbnormalityScore")).sort(desc("userAbnormalityScore"))

  }
}
```

B1. Naïve Bayes Train

```
object NaiveBayesTrain {
  def main(args: Array[String]): Unit = {
    Logger.getLogger("org").setLevel(Level.ERROR)
    val spark = SparkSession.builder()
      .master("local[*]")
      .config("spark.mongodb.input.uri",
"mongodb://localhost:27017/yelp_reviews.review_lte50")
      .getOrCreate()
    import spark.sqlContext.implicits._
    val trainReviews = MongoSpark.load(spark)

    // Tokenize
    val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")
    val regexTokenizer = new RegexTokenizer()
      .setInputCol("text")
      .setOutputCol("words")
      .setPattern("\\W")
    val reviewsWithWords = regexTokenizer.transform(trainReviews)

    // Remove Stop words
    // Custom or Default Stop Words
    //val stopWordsList  = spark.sparkContext.textFile("stopwords.txt").collect()
    val stopWordRemover = new StopWordsRemover()
      .setInputCol("words")
      .setOutputCol("wordsFiltered")
      //.setStopWords(stopWordsList)
    val reviewsFiltered = stopWordRemover.transform(reviewsWithWords)

    // Train NB Model
    val hashingTF = new HashingTF()
    val labeledReviews = reviewsFiltered.select('stars, 'wordsFiltered).rdd.map {
      case Row(stars: Long, filteredWords: Seq[String]) =>
        // Start sentiment from 0
        LabeledPoint(stars - 1, hashingTF.transform(filteredWords))
    }
    labeledReviews.cache()
    val naiveBayesModel: NaiveBayesModel = NaiveBayes.train(labeledReviews, lambda =
1.0, modelType = "multinomial")
    naiveBayesModel.save(spark.sparkContext, "./NBModelLambda1")
  }
}
```

B2. Naïve Bayes Sentiment Analysis

```
object NaiveBayesPredict {
  def main(args: Array[String]): Unit = {
    Logger.getLogger("org").setLevel(Level.ERROR)
    val spark = SparkSession.builder()
      .master("local[*]")
      .config("spark.mongodb.input.uri",
"mongodb://localhost:27017/yelp_reviews.sentiment_results")
      .config("spark.mongodb.output.uri",
"mongodb://localhost:27017/yelp_reviews.nb_sentiment_results")
      .getOrCreate()
    import spark.sqlContext.implicits._

    val reviewSentences = MongoSpark.load(spark)
```

```
    val naiveBayesModel: NaiveBayesModel = NaiveBayesModel.load(spark.sparkContext,
"./NBModelLambda1")

    val regexTokenizer = new RegexTokenizer()
      .setInputCol("sentence")
      .setOutputCol("sentenceWords")
      .setPattern("\\W")
    val sentencesWords = regexTokenizer.transform(reviewSentences)

    // Remove Stop words
    val stopWordRemover = new StopWordsRemover()
      .setInputCol("sentenceWords")
      .setOutputCol("wordsFiltered")
    val reviewsFiltered = stopWordRemover.transform(sentencesWords)

    val hashingTF = new HashingTF()

    val NBSentimentResults = reviewsFiltered.select('review_id, 'user_id, 'sentence,
'wordsFiltered, 'position).map {
      case Row(reviewId: String, userId: String, sentence: String, filteredWords:
Seq[String], position: Int) =>
        val sentiment = naiveBayesModel.predict(hashingTF.transform(filteredWords))
        (reviewId, userId, sentence, filteredWords, position, sentiment.toInt)
    }.withColumnRenamed("_1", "review_id").withColumnRenamed("_2",
"user_id").withColumnRenamed("_3", "sentence")
      .withColumnRenamed("_4", "wordsFiltered").withColumnRenamed("_5",
"position").withColumnRenamed("_6", "sentiment")

    MongoSpark.save(NBSentimentResults)
  }
}
```

B3. Vectors and Tuples

```
public class NBVectorMaker {
    public static void main(String[] args) {
        String host = "localhost";
        int port = 27017;
        TupleGenerator tg = new TupleGenerator();

        MongoClient mongoClient = new MongoClient(host, port);
        MongoDatabase database = mongoClient.getDatabase("yelp_reviews");
        MongoCollection<Document> sentimentResults =
database.getCollection("nb_sentiment_results");
        MongoCollection<Document> sentimentVectors =
database.getCollection("nb_sentimentVectors");
        MongoCollection<Document> sentimentTuples =
database.getCollection("nb_sentimentTuples");

        MongoCursor<Document> cursor = sentimentResults.find()
                .sort(ascending("review_id", "position")).iterator();
        String sentimentVector = "Init";
        String currReviewId = "Init";
        String currUserId = "Init";
        try {
            while (cursor.hasNext()) {
                Document review = cursor.next();
                String userId = review.getString("user_id");
                String reviewId = review.getString("review_id");
                String sentiment = String.valueOf(review.getInteger("sentiment"));
                // Still the same review
```

```
                if (reviewId.equals(currReviewId)) {
                    sentimentVector += sentiment;
                } else {
                    // New Review - Save sentiment vector
                    Document vectorDoc = new Document().append("review_id",
currReviewId)
                            .append("user_id", currUserId)
                            .append("sentimentVector", sentimentVector);
                    sentimentVectors.insertOne(vectorDoc);
                    // check if length < 4 before generating, if so just insert
sentiment vector as tuple
                    if (sentimentVector.length() < 4) {
                        Document tupleDoc = new Document().append("user_id",
currUserId)
                                .append("review_id", currReviewId)
                                .append("sentimentTuple", sentimentVector);
                        sentimentTuples.insertOne(tupleDoc);
                    } else {
                        List<Document> tupleDocList = new ArrayList<Document>();
                        for (String tuple : tg.generateTuple(sentimentVector)) {
                            Document doc = new Document().append("user_id",
currUserId)
                                    .append("review_id", currReviewId)
                                    .append("sentimentTuple", tuple);
                            tupleDocList.add(doc);
                        }
                        sentimentTuples.insertMany(tupleDocList);
                    }
                    // Reset
                    currReviewId = reviewId;
                    currUserId = userId;
                    sentimentVector = "";
                    sentimentVector += sentiment;
                }
            }
        } finally {
            cursor.close();
            mongoClient.close();
        }
    }
}
```