CS 6604 Digital Libraries
Spring 2017
Final Report

Sentiment and Topic Analysis

Team Members:

Abigail Bartolome <abijbart@vt.edu>
Matthew Bock <mattb93@vt.edu>
Radha Krishnan Vinayagam <vradha@vt.edu>
Rahul Krishnamurthy <rahulk4@vt.edu>

Project Advisor:

Prof. Edward A. Fox

May 3, 2017
Virginia Tech, Blacksburg, VA 24061

Abstract

The IDEAL (Integrated Digital Event Archiving and Library) and Global Event and Trend Archive Research (GETAR) projects have collected over 1.5 billion tweets, and webpages from social media and the World Wide Web and indexed them to be easily retrieved and analyzed. This gives researchers an extensive library of documents that reflect the interests and sentiments of the public in reaction to an event. By applying topic analysis to collections of tweets, researchers can learn the topics of most interest or concern to the general public. Adding a layer of sentiment analysis to those topics will illustrate how the public felt in relation to the topics that were found.

The Sentiment and Topic Analysis team has designed a system that joins topic analysis and sentiment analysis for researchers who are interested in learning more about public reaction to global events. The tool runs topic analysis on a collection of tweets, and the user can select a topic of interest and assess the sentiments with regard to that topic (i.e., positive vs. negative).

This report covers the background, requirements, design and implementation of our contributions to this project. Furthermore, we include a user manual and a developer manual to assist in any future work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last decade, social networking has played a major role in archiving global events. Nowadays, many members of diverse communities contribute to archiving events by microblogging (i.e., Twitter). Collections of Internet archives have grown over the last decade with the increase of stream-oriented communication on Twitter. With such large collections of text, linguists need a tool that will help them analyze language trends.

The Digital Library Research Laboratory has a collection of webpages and over 1.5 billion tweets that were collected from the Integrated Digital Event Archiving and Library (IDEAL) and Global Event and Trend Archive Research (GETAR) projects [7] [6]. This collection began in 2007 and is continuously updated as global events occur and become the subject of tweets. The tweets are grouped into collections based on the real-world events for which they are tweeted. Linguists can assess topics of interest within a community by running topic analysis on a collection of tweets written by that community. Sentiment analysis can be made on the tweets corresponding to each topic to determine if the community has, for example, more positive or more negative sentiments associated with the topic.

The Sentiment Team has built a user-friendly tool that will allow linguists and sociological researchers to find topics of interest within a collection of tweets. The team has produced a workflow that uses Latent Dirichlet Allocation (LDA) to extract topics of interest from a collection of tweets. The user can interact with the topic analysis results in such a way that if LDA yields a topic that the researcher knows is of little interest, the researcher can omit the topic and re-run LDA. The user is also able to select a topic of interest and read through the tweets and filter out specific sentiments that they would like to study. For example, in a collection of tweets regarding the Newtown school shooting, the user can select a topic such as "gun control" and read through the more positive or more negative tweets about gun control.

This semester, we have completed an extensive literature review, developed tools to clean tweets, extracted key features from each tweet, run LDA on a collection of tweets, use self-labeled data to build training sets for sentiment classification, built a binary sentiment classifier, performed lexicon based sentiment analysis, and created a GUI for the topic analysis component of our tool. For our final product, we integrated all of the aforementioned components into a system with a user interface.

Chapter 2

Literature Review

2.1 Textbook

Chapter 17 of *Text Data Management and Analysis* [25] focuses on topic analysis and was a very helpful guide to Latent Dirichlet Allocation. We define a topic as a main idea discussed in a document, which is represented as a distribution of words. We can look at the task of topic analysis as having two distinct parts: discovering topics and seeing which documents fit which topics. Topics can be defined as terms such as science or sports; when defined this way, we can see the occurrence of the terms within the document. We can score the terms we choose as topics by using TF-IDF weighting so that the topics chosen will be terms that are fairly frequent but not too frequent. Another way to represent the topics is as a word distribution that allows the topics to be more expressive and deal with more complicated topics. This is normally done by making a probability distribution over the words. By using LDA we represent the topics that are representative of this set as a probability distribution. Additionally, LDA can be used as a generative model to apply to new unseen documents.

Chapter 18 of *Text Data Management and Analysis* [25] gives an overview of opinion mining and sentiment analysis, as well as two sentiment mining algorithms that can be used in different situations. The first mines for sentiment polarity (on a 1-k scale from negative to positive) and the second looks for latent aspects of opinionated text where the sentiment polarity is already known. The chapter also gives good definitions for what constitutes an opinion and which of its components are most important. Specifically, it says that opinions consist of three components: the holder of the opinion, the subject of the opinion, and the content of the opinion, from which context and sentiment can be mined. The chapter also explains an approach to sentiment analysis that considers it as a classification problem, where you build classifiers for each level of polarity and analyze from most to least positive. The other useful piece of information is the discussion of feature selection in the context of human-written text. It gives a few examples of features like n-grams of varying sizes, syntactic and semantic word classes, and recognized name entities.

2.2 Papers

2.2.1 Topic Analysis

Latent Dirichlet allocation is a popular topic modeling approach in natural language processing, so we decided to read "Latent Dirichlet Allocation" by David M. Blei, Andrew Y. Ng, and Michael I. Jordan [3]. It was useful in giving us an in-depth understanding of how LDA works. We define a topic to be a distribution of words. LDA is a probabilistic model of a corpus and treats the documents as random mixtures over latent topics. LDA uses a three-level model with a topic node that is repeatedly sampled, thus allowing documents

to be associated with multiple topics.

2.2.2 Sentiment Analysis

Aspect extraction is an important component of sentiment analysis. Because we would like to find the sentiments of specific topics within collections of tweets, we looked to "Aspect Extraction with Automated Prior Knowledge Learning" by Zhiyuan Chen, Arjun Mukherjee, and Bing Liu [12]. Many unsupervised topic models extract latent topics in collections of documents, where a topic is a distribution of words. Unfortunately, these unsupervised topic models do not always produce coherent topics that are intuitive to humans. This paper describes a method of automating prior knowledge learning by mining prior knowledge from a large corpus of relevant data. The research hypothesized that this prior knowledge learning could be attained by focusing on the aspects shared across multiple domains.

Since our project focuses on processing documents from Twitter, our documents are no longer than 140 characters. Since authors of tweets have to communicate their ideas in 140 characters or less, it can be difficult for tweet authors to use standard language practices (i.e., slang and abbreviations are more common in tweets than other traditional documents). This adds a challenge to tweet sentiment classification, because it is difficult to to use standard natural language processing techniques. This challenge led us to reading "Enhanced Twitter Sentiment Classification Using Contextual Information" [22]. This paper studied the relationship between tweet metadata (e.g., author, time tweet was written, location) and tweet sentiment. The study hypothesized that "people are generally happier on weekends and certain hours of the day, more depressed at the end of summer holidays, and happier in certain states of the United States" [22]. While many of our tweets are reactions to real-world events, it could be interesting to use our tweet metadata to explore correlations between sentiment and tweet metadata, and compare our results to this hypothesis.

Sentiment Word Identification (SWI) is a sentiment analysis technique that can determine critic opinion, tweeter classification, review summary, etc. Most sentiment analysis strategies employ seed words to determine positive or negative sentiments within a document. However, studies have shown that using seed words can be unreliable across multiple domains, and that missing a key word in a set of seed words can inhibit sentiment analysis performance. "Identifying Sentiment Words Using an Optimization-based Model without Seed Words" proposed a method of SWI, called WEED, that identifies sentiment words without seed words [24]. It exploits a phenomenon, referred to as "sentiment matching", that means that the polarities of a document and its most component sentiment words are the same. That is to say that if a word is found mostly in positive documents, it likely is a positive word; if a word is found mostly in negative documents, it is a negative word. This process does require pre-labeled documents, so it could be difficult to include across tweet collections of different domains.

Since tweets can not be longer than 140 characters, it would be beneficial to have a large-scale sentiment lexicon from Twitter that could be adapted to any collection of tweets in the GETAR project. We read "Building Large-Scale Twitter-Specific Sentiment Lexicon: A Representation Learning Approach" to understand how to handle the niche language used in Twitter documents [20]. A sentiment lexicon is a list of terms, such as "excellent" and "terrible", where each term is "assigned a positive or negative score reflecting its sentiment polarity and strength" [20]. The paper proposes employing a representation learning approach to build a large-scale sentiment lexicon from Twitter. The proposed method involves finding the continuous representation of phrases and using them as features for sentiment classification. The method then adds these phrases to a smaller list of seed words to collect training data for sentiment classification.

Another particularly interesting approach was to use emoticon-annotated tweets as a self-labeled training set. "Positive, Negative, or Neutral: Learning an Expanded Opinion Lexicon from Emoticon-annotated Tweets" suggested a supervised framework for expanding a sentiment lexicon for tweets [4]. This methodology treats tweets with emoticons as self-labeled data, where a tweet with ":-)" is a positive document, and a tweet with ":-(" is a negative document. Each term in the lexicon has a probability distribution to describe

how positive, negative or neutral the term is. For example, "terrible" is more negative than "unsatisfactory". Each entry in the lexicon is associated with a corresponding part-of-speech so as to differentiate homographs with different parts of speech.

2.2.3 Lexicon-Based Methods

Lexicon based techniques take advantage of annotated lexicons. Entries in lexicons are preassigned a polarity score. Lexical entries form the building blocks of tweets, so from the known polarity scores in lexicons we can predict the polarity of tweets or any target entities within the tweet. One of the drawbacks of the lexicon based techniques is that they do not consider the context in which the word is used. The sentiment polarity of a word may be different in the presence of other terms. Contextual polarity of the word should be considered while deciding the sentiment of the overall tweet.

SentiCircle [17] is one of the lexicon based techniques, which builds a dynamic representation of words that consider the contextual semantics. This scheme uses external lexical resources such as sentiwordnet or MPQA [23]. However, the sentiment values of each term are not static. The sentiment value changes according to the contextual vector, which comprises of all terms that co-occur with the target term. The paper introduces a degree of correlation metric which influences the sentiment value of terms. The scheme allows measurement of impact of context words on the sentiment orientation and on the sentiment strength of a target-word separately. This paper handles negation by reversing the sentiment value of terms that are near to a negation term. Finally, to compute entity-level sentiment polarity, a median of all the sentiment values of terms that occur together with the target entity in tweets is calculated. In order to measure tweet level sentiment, two approaches are proposed. In the median method, first the sentiment median of all terms that appear in that tweet is computed. Then a median of all those sentiment medians is computed. The final median value is used to predict the sentiment of the tweet. The second technique proposed in the paper to find a tweet's sentiment value is called the pivot method. In the pivot method, the terms tagged as common noun, proper noun and pronoun are considered pivot terms. One of these terms is the target of sentiment expressed in the tweet. For each sentiment label, the sentiment impact that each pivot term receives from another is accumulated. The sentiment label with the highest value is assigned as the sentiment of the tweet.

Natural language processing (NLP) is another field which has proven to be effective in extracting information from tweets. In [15], work has been done to leverage Twitter for providing people necessary information during a natural disaster. In [15], NLP has been successful in creating a system which enabled quick response during natural disasters. Important information such as person names and locations were filtered from a highly unorganized and difficult-to-process information source using NLP techniques. To make effective use of Twitter, a labeled corpus of tweets was created which would help in information extraction of unlabeled tweets. NLP techniques such as word segmentation, morphological analysis, part-of-speech tagging and name entity recognition were very important in building this Twitter based disaster response system. This work shows that NLP has the potential to help understand tweets and it would be interesting to explore its potential in the field of Twitter sentiment analysis.

Performance of supervised sentiment classifiers depends on amount of training data and hence they require a large amount of annotated data to obtain higher performance. Moreover, these classifiers are domain dependent. They need to be re-trained on a different data set in order to do well in a separate domain. This forms the motivation to explore unsupervised classifiers. One such unsupervised classification scheme proposed in [21] employs a metric called semantic orientation of a phrase to identify polarity of movie reviews. A movie review is classified positive if the average semantic orientation of the phrases which contain an adjective or adverb in the review is positive else that review is predicted as negative. The scheme makes use of reference words: one each for positive and negative reference. The semantic orientation of a word is then found to be either closer to positive reference words or negative reference

words. More specifically, semantic orientation was computed by using the equation given below.

$$SO(p) = PMI(p, excellent) - PMI(p, poor) \tag{2.1}$$

Here p is a phrase whose semantic orientation is being computed. A positive seed word is 'excellent' and negative seed word is 'poor'. PMI refers to pointwise mutual information. The technique was applied to 410 reviews and obtained an accuracy of 74%.

2.2.4 Sentiment-LDA Model

Sentiment analysis of text data, especially Twitter content, is a well-researched topic. Since the sentiments of words are also domain specific, in the past decade, many researchers have explored how to combine the approaches of topic analysis and sentiment analysis. Li et al. [11] have developed a joint Sentiment-LDA model, by adding a sentiment layer to the regular LDA model. The regular LDA model has three layers: document layer, topic layer, and word layer. Sentiment-LDA model is a four layer topic model which contains a sentiment layer between the topic layer and the word layer. In this way, they are associating words in the documents with both topics and sentiment labels. The authors claim that this model will also be helpful in determining the sentiment of the sub-topics in a document or a collection.

Chapter 3

Requirements, Design, And Implementation

3.1 Requirements

This section outlines the requirements that we laid out for our project after much planning and discussion with our team members, Dr. Fox, and the rest of the class. We divided our system requirements into three components: A tool that successfully performs topic analysis, a tool that successfully performs sentiment analysis, and a system that efficiently uses the results from topic analysis to yield focused sentiment analysis within topics on the tweet level. We also identified requirements related to user interaction with our system. These requirements included a way to interface with the system without needing programming knowledge, the ability for the user to refine results if they are looking for something specific, and the ability to let the system run automatically and find general results if the researcher does not know what topics to look for or is generally surveying the collection of tweets.

3.1.1 System Requirements

Topic Analysis

The first of the three requirements for our system was to be able to run topic analysis on an arbitrary collection of tweets. We were able to use LDA [3] to extract the most popularly discussed topics in a collection. This tool has successfully been used to extract topics from several collections, but it was refined to handle collections of larger size. The next major step in satisfactorily extracting topic models from a collection of tweets was that the user be able to interfere if they encountered a topic they they believed was not meaningful or reflective of the collection. We successfully delivered this requirement, providing the user the ability to eliminate the topic(s) that are not meaningful and rerun LDA so that the results are more useful to the user.

Sentiment Analysis

The second of the three requirements for our system was to have a tool that could perform sentiment analysis on an arbitrary collection of tweets about a specific event or topic. Our primary goal to achieve from this requirement was to be able to run polarity analysis on all of the tweets and determine whether they expressed positive or negative emotion. We would have liked to move on to full emotion analysis, where we would identify emotions being expressed in the tweets (such as happiness, anger, disappointment, etc.). This would have been an area for users to be able to interact with the system. They could specify emotions that they are

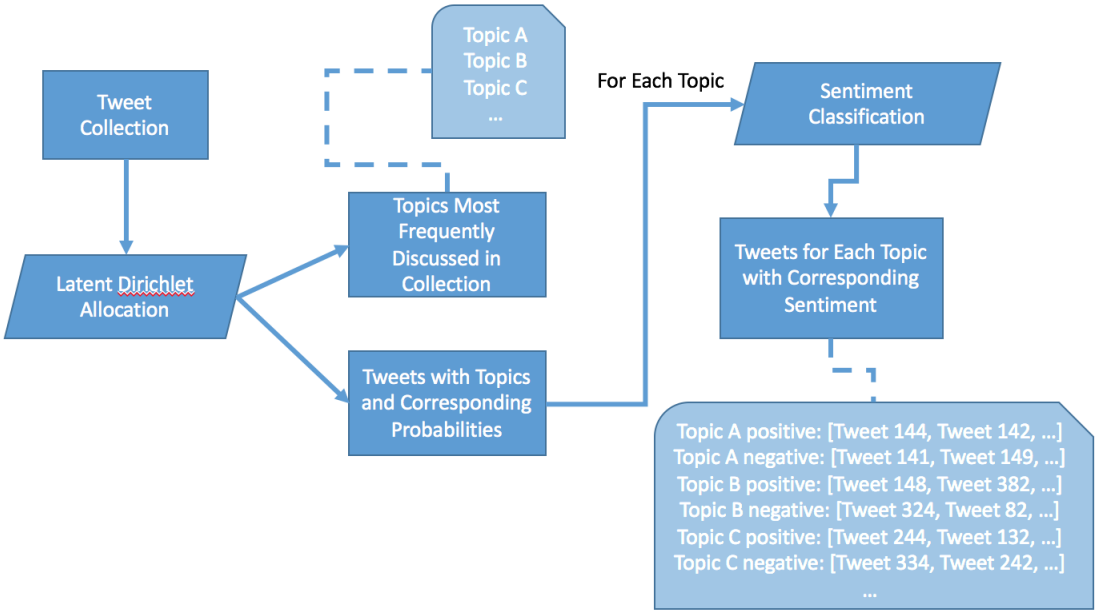

Figure 3.1: The planned flow of our system's tools

interested in analyzing and tailor the results accordingly. However, during our work on this particular effort, we found some interesting results in calculating sentiment scores using dependency trees. Understanding this technique for sentiment analysis became the focal point of our efforts, and we chose to maintain our initial focus on positive and negative sentiments. This is discussed further in Chapter 4.

Combined Sentiment-Topic Analysis

A key component of our system was to have topic and sentiment analysis working together. The goal here was for researchers to be able to search for sentiments within specific topics. We set the input to be a tweet collection where every tweet is tagged as belonging to a certain topic (as determined by the LDA approach explained above). The output sentiment-tagged sub-collections of tweets belonging to each topic. Our data flow model is presented in Figure 3.1.

We provided functionality for users to interact with the system to guide the process along, either by focusing the results towards something specific that they are looking for or applying their domain knowledge to help enrich the process. More specifically, the users are able to intervene throughout the topic analysis process by disregarding topic words that they determined are not meaningful to the domain. The user may also choose not to intervene in the system, and allow LDA to determine topics uninterrupted. This flexibility enables users to be able to search for specific topics and sentiments that they are interested in, or get a broad set of results if they are interested in general behavior. Figure 3.2 shows a modified version of our tool flow which includes optional user interaction.

3.1.2 User Interface Requirements

One of the other goals for this project was to provide a way for users who do not have knowledge of programming to be able to run topic analysis on data sets that interested them. This made the tool very useful to researchers without computational backgrounds, who were still interested in leveraging the GETAR and IDEAL projects to perform research in their own research areas.

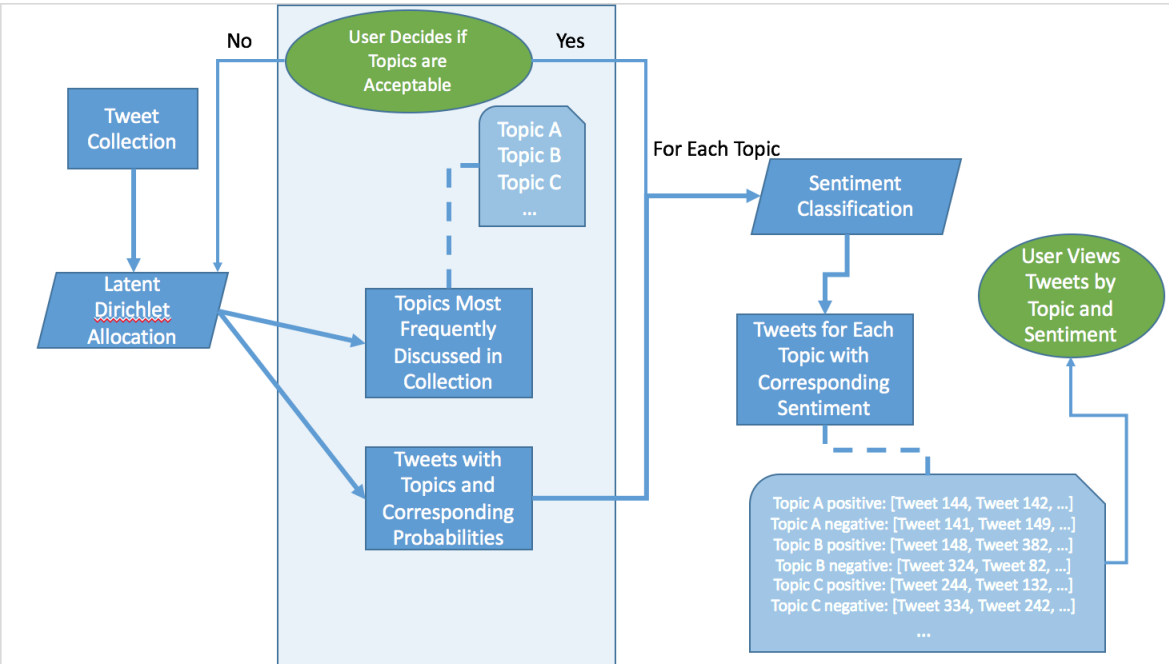

Figure 3.2: The planned flow of user interaction with our system's tools

3.2 Design

Our design aimed to meet all of our requirements effectively and efficiently. Section 3.2.1 explains how our system flow works, and it is illustrated in Figure 3.1. Additionally, we drafted how our users can interact with the system, as described in Section 3.2.2 and illustrated in Figure 3.2.

3.2.1 System Flow

1. Perform topic analysis on a cleaned collection of tweets.
2. Optionally refine topic analysis results by choosing topics to discard (re-run with more focus). At this stage the user will be able to interact with the workflow.
3. Perform sentiment analysis within sub-groupings of tweets that are within topics of interest.

3.2.2 User Interaction Flow

1. User selects a collection for using our linguistic analysis tool.
2. Optionally reject resulting topics produced by the tool and rerun. Note that the user can be satisfied with the resulting topics and continue on with the tool's flow.
3. View the tweets by topic and sentiment.

3.2.3 Design Deliverables

We successfully built a tool that processes datasets and performs topic analysis using LDA. The tool also allows users to exclude words that are not of interest. We are able to produce k topics where k is determined by the user, as well as a result file containing each tweet's ID and its corresponding topic probability

distributions.

We have also developed a sentiment analysis tool that can extract tweets that contain the ":)" emoticon, label those tweets as a positive training dataset, and so classify tweets that are positive. Similarly, we have a sentiment analysis tool that can extract tweets that contain the ":(" emoticon, label those tweets as a negative training dataset, and so classify tweets that are negative. We learned about this process of using self-labeled data through emoticon-annotated tweets through Bravo-Marquez, Frank, and Pfahringer's paper [4] on expanding lexicons through emoticon-annotated tweets.

Our efforts built on this idea by using tweets with emojis as self-labeled tweets. We labeled selected emojis as being positive or negative, as demonstrated in Appendix A. We have a script that extracts tweets with the emojis of interest and places them into respective CSV files for positive and negative tweets. We integrated the workflow used in this script with the sentiment analysis tool that we created.

We implemented a basic Scala user interface for topic analysis. Using the GUI, a user can input all the required parameters for LDA. We also designed a basic interface to present the LDA results to the user. Through this interface, the user will be able to tweak the model to exclude irrelevant topics if any. We were able to successfully integrate it with our current LDA wrapper code.

3.3 Implementation

Our project was implemented using the tools provided by the Apache Hadoop project [1]. Specifically, we used Spark for massively parallel data processing, and HDFS for distributed data storage. We tried to restrict ourselves to these tools because they are supported by the cluster maintained by the Digital Library Research Lab (DLRL). We had intended to use the cluster for all of our data processing, and we had used it for testing our topic analysis and one of our sentiment classifiers. However, because we wanted to include a GUI component, we used a Cloudera virtual machine for our GUI development and testing.

Our project is partly based on API code that was developed for Matthew's thesis project. A large part of his thesis is focusing on enabling developers to more easily access data that is stored on the cluster. The tool provides us with data structures that will abstract away the complex Spark code that is involved in reading data off of the cluster, decoding it from binary format into readable text, parsing the fields, and storing them in a collection to be processed. In addition, the tool provides us with a large suite of functions that we can use to clean, filter, and otherwise modify the data set to suit our needs. Finally, it also includes wrappers for, among other things, Spark's LDA implementation that take the tweet collection data structure and run LDA analysis on it. These three components are the parts that are of use to us.

We built tools that could stand on their own, before integrating them with each other. We built a standalone sentiment classification component, a standalone topic analysis component, and a prototype user interface. The sentiment classification component works by extracting tweets which contain a positive or negative emoticon (currently either ":)" or ":(" respectively) and building a word2vec vector space with them using Spark's word2vec implementation. We then use the word2vec vector space as training data for a logistic regression classification model. This model will use the training data to classify new tweets as either positive or negative, and return the tweet collection with each tweet labeled as positive or negative. The topic analysis component uses Matthew's library to run LDA topic analysis on the tweet collection, and returns a list of topics and the most likely terms in each of those topics. The user interface is implemented in Scala using the Swing library. We are currently working on integrating the wireframe GUI with the functional back-end code we have developed.

The tweets extracted from the database had emojis that were encoded in a UTF-8 format that contained non-alphanumeric codes. We wrote a script to convert emojis in that format to the standard UTF-8. Both the raw and formated emojis are shown in Figure 3.1. Figure 3.2 shows a part of a lookup table that was used to identify positive and negative emojis. The intent was that we would incorporate this extraction method

Raw emoji	Processed Emoji (UTF-8)
Ã°Å¸Ëœâ€ž	\xF0\x9F\x98\x93
Ã°Å¸ËœÆ’	\xF0\x9F\x98\x94
Ã°Å¸ËœÆ’	\xF0\x9F\x98\x95
Ã°Å¸Ëœâ,¬	\xF0\x9F\x98\x96
Ã°Å¸Ëœâ€°	\xF0\x9F\x98\x9E
Ã°Å¸ËœËœ	\xF0\x9F\x98\xA0

Table 3.1: Emoji's before and after being processed

UTF-8	Positive emoji
\xF0\x9F\x98\x83	smiling face
\xF0\x9F\x98\x84	smiling face with open mouth and smiling eyes
\xF0\x9F\x98\x85	smiling face with open mouth and cold sweat
\xF0\x9F\x98\x86	smiling face with open mouth and tightly-closed eyes
\xF0\x9F\x98\x89	winking face
\xF0\x9F\x98\x8A	smiling face with smiling eyes

UTF-8	Negative emoji
\xF0\x9F\x98\xA0	Angry
\xF0\x9F\x98\xA2	crying face
\xF0\x9F\x98\x9F	worried face
\xF0\x9F\x98\xA7	Anguished face

Table 3.2: Sample lookup table for positive and negative emoji's

with our sentiment classifier. However, it became evident that the self-labeled data would not suffice for developing "not negative" and "not positive" training sets. Our tests showed an alarming number of false-negatives, so we took another approach in our sentiment analysis, which we discuss in Chapter 4.

Chapter 4

Lexicon Based Sentiment Analysis

4.1 Lexicon Based Sentiment Analysis

In lexicon based sentiment analysis, we employ one or more sentiment dictionaries to perform sentiment analysis. The performance of this technique depends not only on the quality of the sentiment lexicon used, but also on how this lexicon was used in performing sentiment analysis. The task of performing sentiment analysis using a lexicon can be subdivided in two sub-tasks as follows:

1. Analysis of tweet structure
2. Calculation of sentiment using lexicon

4.1.1 Analysis of Tweet structure

In this task, we look for specific features in the tweet that have influence in deciding the sentiment of the overall tweet. After these features are identified, the lexicon is then used to find the sentiment type and raw score of features. The scores are raw and can be further modified depending on the neighboring words in that tweet. The following are the features that we extracted in a tweet structure:

- Negation word (example: couldn't, wouldn't)
- Polarity reversal words (example: prevent, diminish, subside)
- Analysis of syntactic structure of the tweet, by generating its parse tree
  - Identify head words
  - Identify the modifiers of head words
- Emoticons

4.1.2 Calculation of sentiment using lexicon

In this task, we find the scores for the n-grams that were extracted from the tweet. The way we combine the individual scores to compute the overall score determines the accuracy of sentiment analysis. A naïve approach to perform this task is to find the overall sentiment of the tweet by adding the sentiment score of individual unigrams. This approach will have the least accuracy since it does not consider the structure in which different words in the tweet are connected to each other. In a sentence structure, the sentiment score of individual words vary depending on the word's proximity to a special category of words. Examples

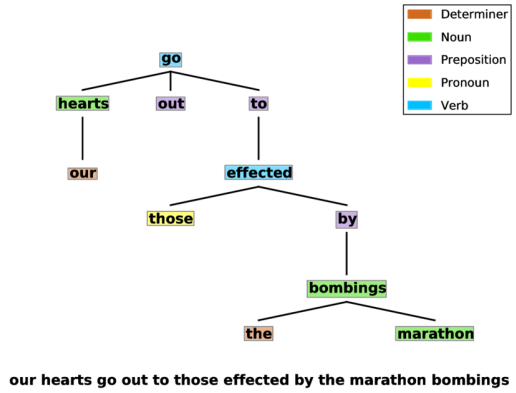

Figure 4.1: Dependency tree of an English sentence

of special categories of words include negation words and intensifiers. Once we determine the technique to compute an overall sentiment, we have to then set thresholds for positive and negative sentiments. A tweet is classified positive if the overall score of that tweet is above the set positive threshold. If the overall sentiment score of the tweet is below a negative threshold then it is classified as a negative polarity tweet. A neutral tweet will have a score between the positive and negative threshold. Following are score computation approaches that we followed:

- Add sentiment scores of each token in a tweet. The result of this addition will give the sentiment score of the overall tweet.
- Computation of score based on the dependency of words in a parse tree.

Sentiment Score Using Dependency Tree

A dependency tree of a sentence represents the connection between words in a sentence. The tree consists of two types of nodes. One node is called the head and the other node is called the modifier of head node. The modifier node is also called the dependent node. Figure 4.1 shows an example of a dependency tree.

In Figure 4.1, the word "bombings" is a head node of words "the" and "marathon".

After forming the dependency tree, we compute the sentiment score of the sentence based on the connection of words in the tree. In the literature, we found two approaches which have been used to compute sentiment scores from a dependency tree. These rules are as follows:

1. Voting with Polarity reversal [14]
   In this rule we compute the polarity of each word from a lexicon. We then reverse the polarity score of a word if there are an odd number of polarity reversal words in the ancestor list of that word. For example, in Figure 4.1, if we want to compute the polarity score of a word "bombings", then we will first look up the word "bombing" in a lexicon. Then we will compute a list of ancestors for the word "bombing", which in this case consist of words: {"by","effected","to","go"}. If any of the ancestor words are a polarity reversal word, then we will reverse the polarity score of word "bombings". Mathematically this rule can be written as :

$$polarity = sgn(\sum_{i=1}^{n} score_i \prod_{j \in H_i} (-1)^{r_j}) \tag{4.1}$$

   Here, polarity refers to the polarity of the whole tweet. $score_i$ refers to the polarity score of i^{th} word

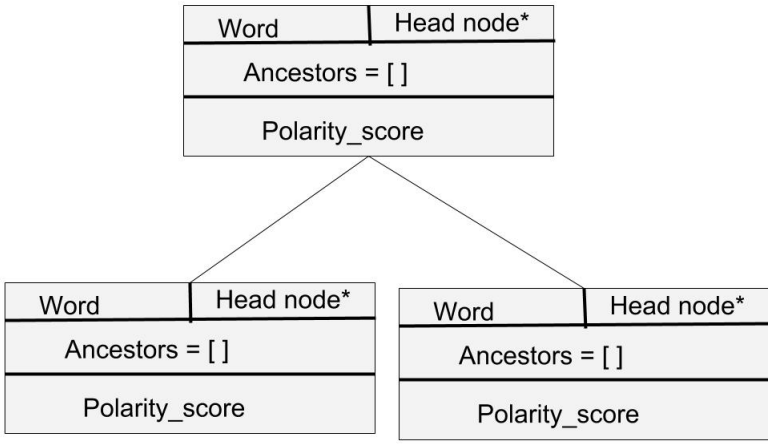

Figure 4.2: Tree data structure of dependency tree

node in a tree. H_i means the ancestor list of word node i. r_j is 1 if the j^{th} word in the ancestor list is a polarity reversal word. In all other cases the value of r_j is 0.

2. Deterministic Rule [14]: The polarity of a subjective sentence is deterministically decided based on rules, by considering the sentimental polarities of dependency subtrees.
   The polarity of the dependency subtree whose root is the i^{th} word node is decided by voting the prior polarity of the i^{th} word node and polarities of dependency subtrees whose root nodes are the modifiers of the i^{th} word node. Also, the polarities of the modifiers are reversed if their head node has a reversal word.
   Based on the above rule we compute the polarity of subtrees recursively starting from the leaf node till we reach a root node. The polarity of the tweet is then determined by the polarity of the root node. This rule is mathematically represented as follows:

$$polarity\_score\_i = sgn(root\_i \sum_{j:h_j=i} polarity\_score\_j(-1)^{r_i}) \tag{4.2}$$

   Here we are computing polarity of the i^{th} subtree which is denoted by polarity_score_i. The term root_i represents the polarity score from the lexicon for node i which is the root of this subtree. Here, h_j represents the head of the j^{th} node.

Figure 4.2 represents the data structure that we are using to store the dependency tree. Each node in the tree stores the word, a pointer to head node, a list of ancestors and polarity score for that node. This polarity score is the score of the dependency subtree which is rooted at this node.

4.1.3 Lexicon Used

In order to create a sentiment analysis tool for Twitter, we needed a social media specific lexicon. Content observed on Twitter and Facebook poses serious challenges to practical applications of sentiment analysis. Contextual sparseness resulting from shortness of the text and a tendency to use abbreviated language conventions to express sentiments makes sentiment analysis using standard English lexicons difficult. In our

initial experiments, we employed Sentiwordnet [2] and found that the sentiment of most of the tweets were classified as neutral. We got such a result because the words that are used in Twitter were not found in Sentiwordnet and hence the polarity score of such words was considered to be neutral (a score of zero). This observation prompted us to search for lexicons which have Twitter specific words. We found VADER [10] which uses a high quality lexicon specifically made for sentiment analysis of text in social media.

4.1.4 Experiments

Our objective was to determine whether or not we could use dependency parse trees to determine the sentiment of tweets. We started our work by going through rule based sentiment analysis that was given in [14]. This paper [14] focuses on polarity reversal words and hence we started our experiment by analyzing such words.We used polarity reversal words that were given in General Inquirer [8]. We integrated VADER's lexicon, polarity reversal words and the sentiment analysis rules with dependency parse tree to determine the sentiment polarity of tweets. We focused on very limited words because we were testing the polarity of tweets manually. We focused on words such as 'depression', 'anxiety', and 'stress'. These words have negative sentiment scores. Then we started looking for tweets which had these negative words along with polarity reversal words like 'abate', 'diminish', 'reduce' and 'decrease'. The following are some of the tweets and the corresponding output of VADER:

1. "study shows a significant decrease in depression after taking psilocybin"
    - Vader score = -.4404
2. "CBD help reduce depression"
    - Vader score = -.25
3. "Learning math games to decrease math anxiety."
    - Vader score = -.1779
4. "When people are doing what they love, depression and anxiety will decrease."
    - Vader score = -.0516
5. "Singing helps reduce feelings of depression and anxiety, increases oxygen to your lungs."
    - Vader score = -.4215
6. Escape to nature, even if just for a 30 minute walk.. it will greatly lower your stress levels and reduce risk of depression
    - Vader score = -.309
7. Listening to music for an hour every day can reduce chronic pain by up to 21% and depression by up to 25%
    - Vader score = -.7906

The observation that we found in the above tweets was that when we have only polarity reversal words in tweets with a negative sentiment word, then the output of VADER is different from the expected value. This list of tweets is not sufficient to draw any concrete conclusion, and hence we cannot make any claims

about the accuracy of VADER. Our objective of such a test was to find a category of tweets for which it is difficult to predict sentiment. The following are tweets that we tested further and found that when we have a negation word in close proximity or higher positive polarity word close to a polarity reversal word, then these words dominated in deciding the polarity of tweet.

1. "I know but that doesn't help abate the anxiety spiral that has me nauseous and upset."
2. My brain hurts. Anxiety probably won't abate until after Monday.
3. At School for the Dogs, Halloween cone craft workshop helps abate doggy costume anxiety "
4. My claustrophobic and lacerating loops of extreme anxiety, fear, and worry typically do not ever abate.
5. How my gratitude journal helped abate my ANXIETY!
6. "Thanks to the @united woman who helped me get a window seat on an almost full flight to help abate my flight anxiety."
7. Bless Frank Ocean's music for its ability to abate my anxiety.
8. Can't wait to get to the gym. Hoping it will abate this anxiety.
9. Even free cookies can not abate the anxiety of startups in post-Brexit London http://wapo.st/29lga1E @ylanmui
10. Making Weight: Exercise, medicine abate anxiety
11. U.S. Gas Pipeline Capacity Remains Short, But Anxiety Levels abate
12. I hate anxiety
13. You will fall asleep faster, if you sleep next to someone you love. This way you can also reduce depression.

We found that when we applied the dependency tree generated by Syntaxnet [9], the rules failed to give accurate sentiments in certain tweets. This may be due to the parse tree structure that is generated by Syntaxnet. The parse tree that was used in research paper [14] to test these rules was different. For tweets we can have more than one parse tree possible and hence we have probabilities associated with each parse tree that is generated. In this work we focused on using parse trees generated by Syntaxnet and hence we modified these rules so that they can be applied with our parse tree. The following section gives details of our approach to identify sentiment polarity of tweet using Syntaxnet parse trees.

4.1.5 Our Approach

We found that the sentiment of tweets which have negation in leaf nodes of a parse tree could not be predicted correctly by these rules because the rules only search for polarity reversal word and negation words in the parent node.

Secondly, we found that when we have two independent phrases in a tweet, the parse tree generated by Syntaxnet could not be integrated with these rules. For example, the tweet : "Conversations reduce stigma and increase understanding" gave a negative sentiment polarity, when we used rules.

On further analysis of that tweet we found that the word 'reduce' is incorrectly connected as the head node of the word 'increase' in the parse tree. Since word reduce is a reverse polarity word, it changes the

polarity of word ‘increase’ from positive to negative and hence we get a negative score as the sentiment of this tweet. We came to the conclusion that we can not apply these rules in a straight forward manner with Syntaxnet. Our approach was as follows: We wanted to detect two independent clauses from a parse tree. We can do this by checking the nodes connected by a conjunction. If these nodes are leaf nodes, then we are sure that the conjunction is connecting two words and not two clauses. If the conjunction was connecting two clauses then we decided to do further analysis. The objective of our analysis was to find whether or not to reverse the polarity of a word1 that is connected by a word2. The word1 and word2 belonged to two different clauses in a sentence. We came up with the following modified rule :

Rule: Do not reverse the polarity of a word due to a polarity reversal word in head node, if

1. The child is a verb and accompanied by a subject in its neighbor
2. The child has an object as its dependent

Also if there is a ‘neg’ part of speech word in a sub-tree then we would reverse the polarity of all the words in that sub-tree.

Integration of rule with Syntaxnet

In the topic analysis stage we get tweets for various topics in text files. We provide these text files as input to Syntaxnet. Syntaxnet creates a parse tree for each tweet and stores it in a data structure that is shown in Figure 4.2. We parse this data structure by Depth-first scheme and simultaneously apply the rules to find the value of ‘Polarity_score’ attribute for each node in the tree data structure. Finally, the polarity of the tweet is given by the ‘Polarity_score’ attribute value of the root node.

Chapter 5

User Manual

5.1 Scala User Interface

Scala's user interface for topic analysis provided us with a way to give users the ability to run LDA without using the command line. Additionally, it also helped users to interact with the underlying topic model.

To run the LDA program, the user needs to select the input data file, specify various parameters (i.e., number of topics, number of iterations, etc.). All of these parameters can be specified through the UI window given in Figure 5.1. Results from the topic analysis are presented to the user through a separate window, as given in Figure 5.2. In the results window, the user can also select words which are irrelevant and re-run the LDA to modify the underlying topic model.

To do this, the user should highlight the words that are not meaningful, press the "»" button, so moving the word out of the topic words box (bottom left box). Once the user is satisfied with the words in the topic words box, the user can re-run the LDA by clicking the "Re-Run" button. Once the user is satisfied with the LDA results, the user can click the "Finish" button. Thus, using our topic analysis interface, a user can interactively create a topic model according to his/her preferences. Figures 5.3, 5.4, and 5.5 illustrate this process of topic refinement.

Since demoing the tool for the screenshots shown in the aforementioned figures, we have made a design enhancement to label the bottom two boxes in the results window. The bottom left box shows the topic words that were part included in the word distributions for each topic. The bottom right box is for the words that should not be included in the topics' word distributions. Figure 5.6 shows this enhancement.

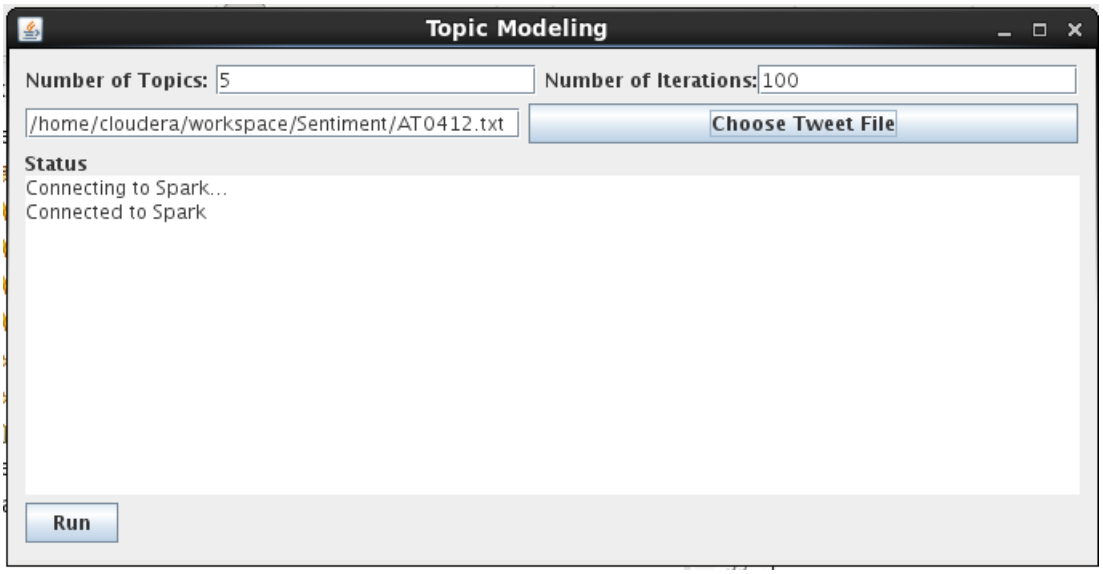

Figure 5.1: Topic analysis UI- start screen

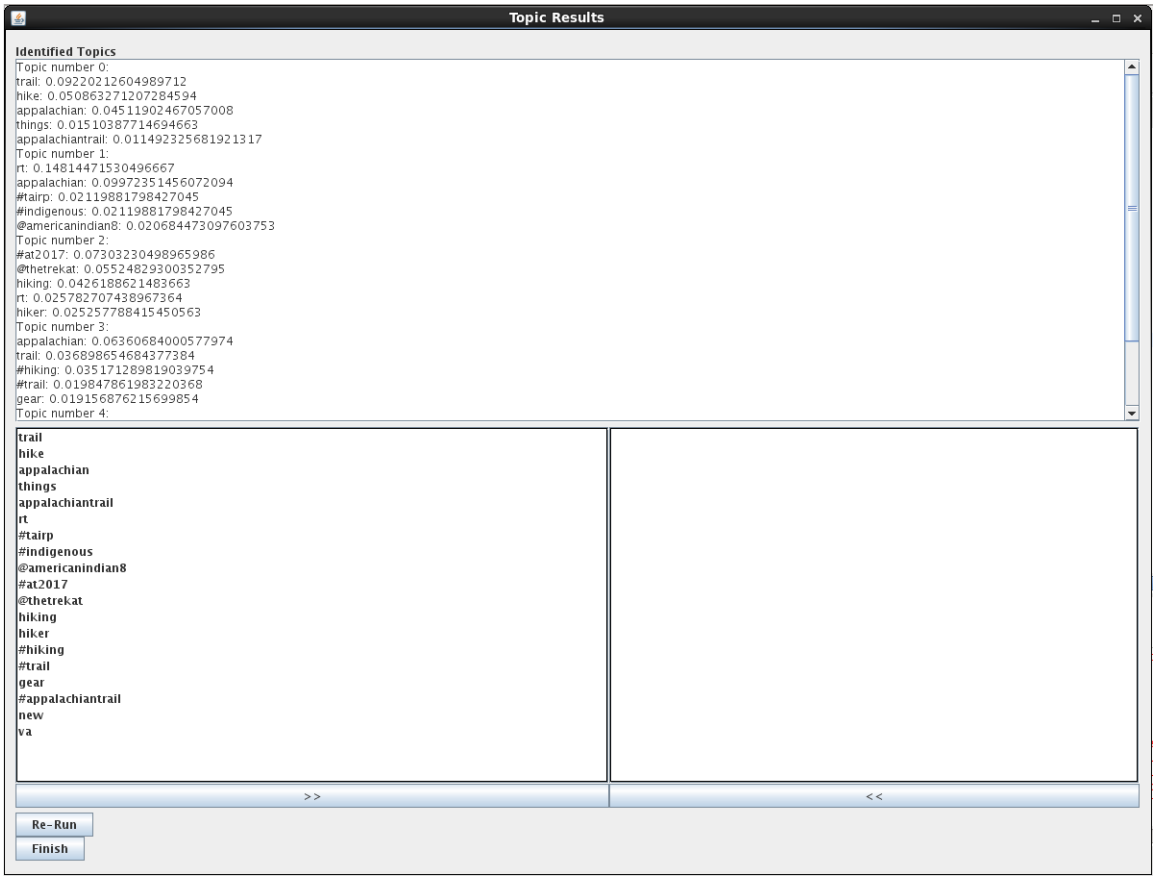

Figure 5.2: Topic analysis UI- first round results

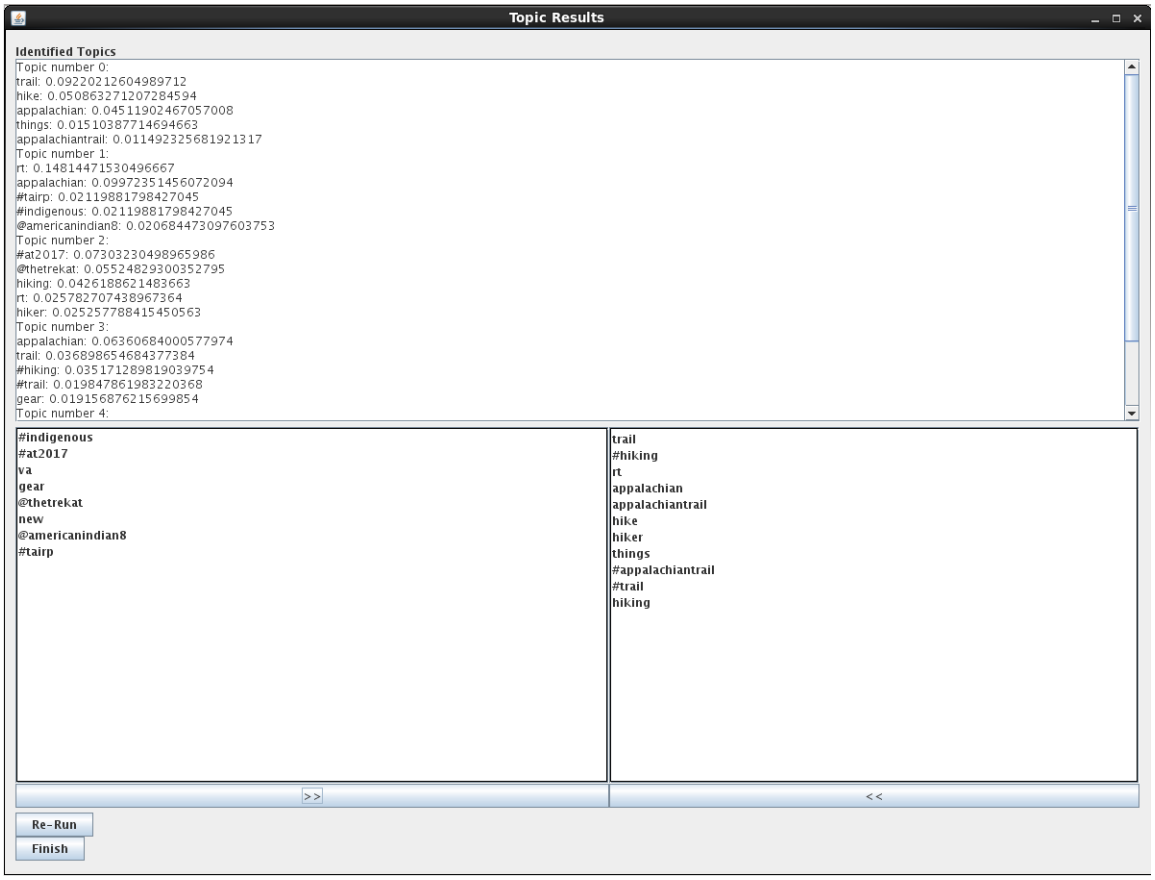

Figure 5.3: Topic analysis UI- second round results

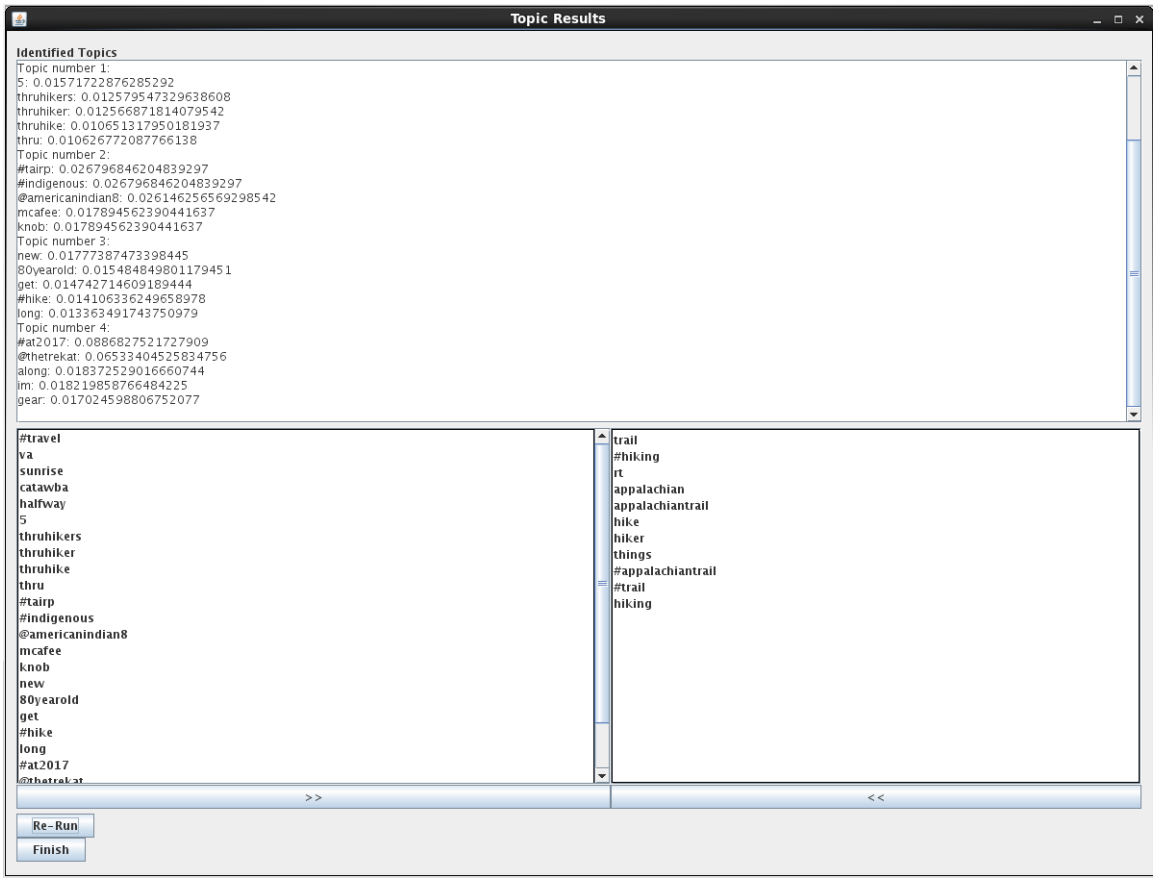

Figure 5.4: Topic analysis UI- third round results

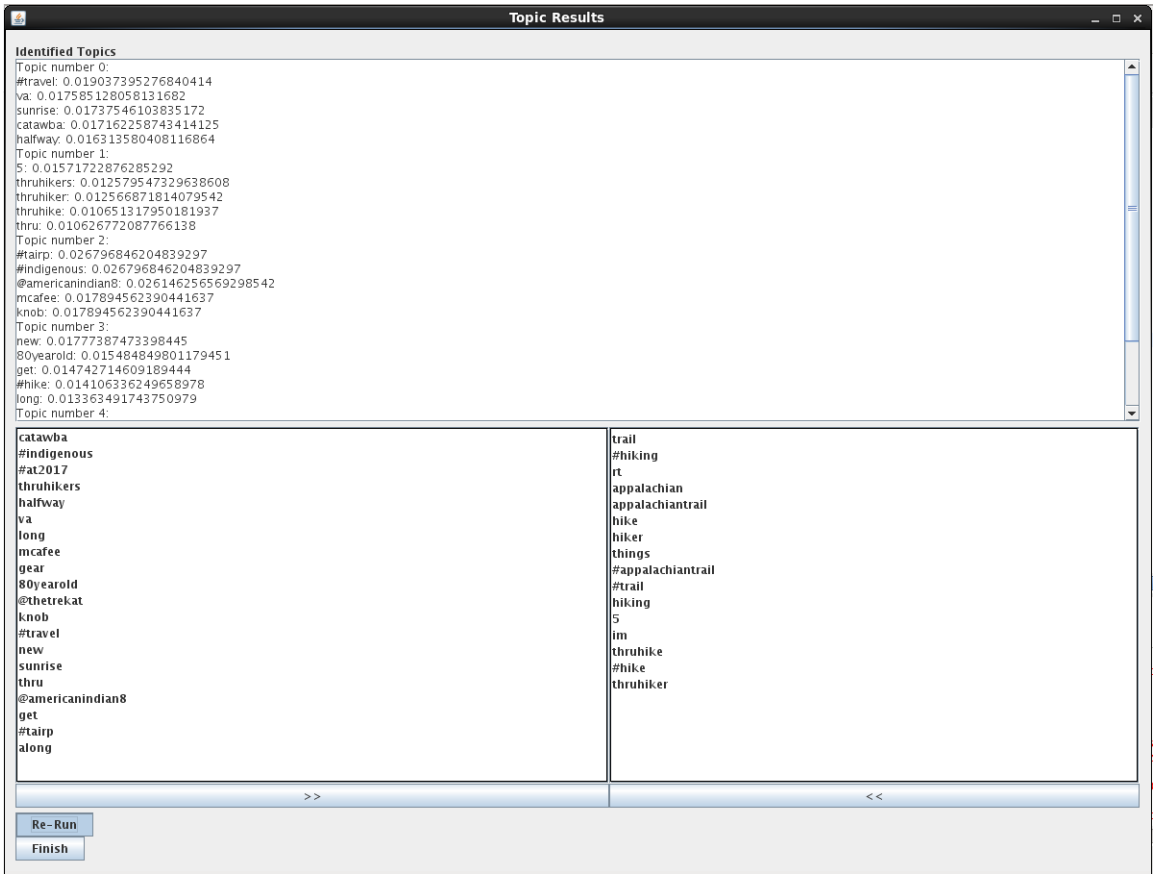

Figure 5.5: Topic analysis UI- fourth round results

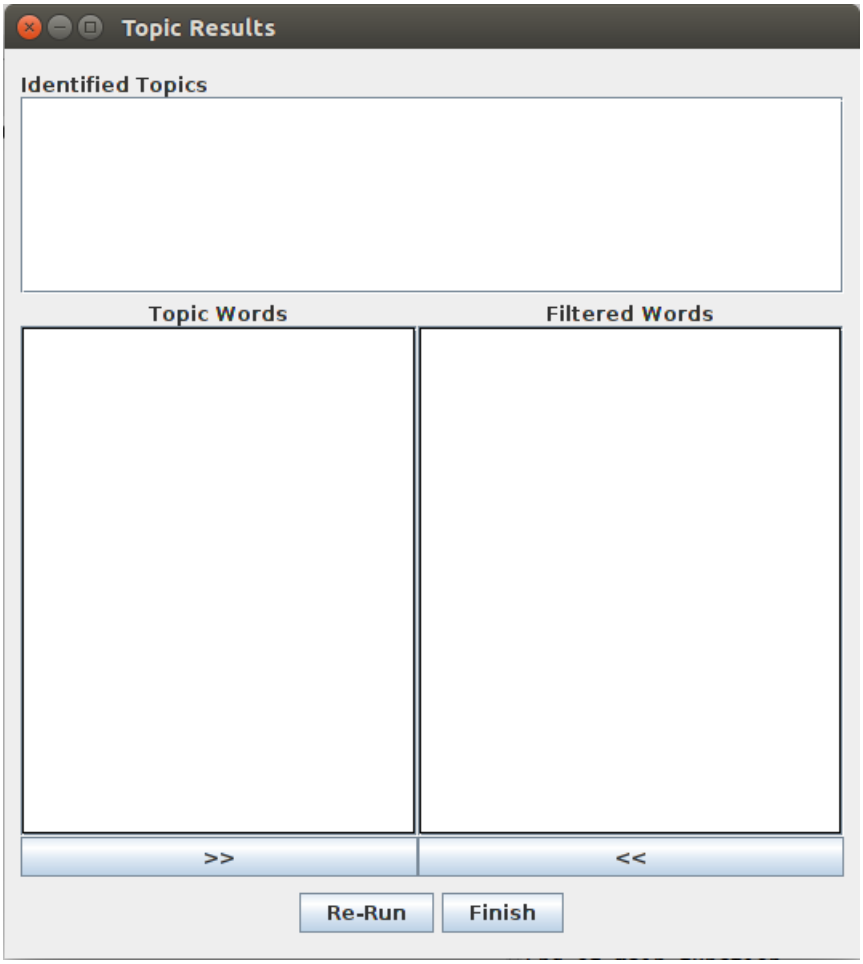

Figure 5.6: Updated topic results window design

Chapter 6

Developer Manual

6.1 Scala Interface

To implement a Scala interface, we could either have used the ScalaFX library [19] or the Scala Swing package. We chose to use the Scala Swing framework to implement the Topic Analysis UI. Scala Swing is a GUI toolkit written in Scala and based on the Java Swing library. It provides wrappers around the Java Swing classes. Thus, to someone who has basic knowledge in Scala and exposure to the Java Swing library, the Scala Swing framework is ideal choice to develop GUIs in Scala. For the GUI programming in Scala, it is beneficial to use an IDE, such as the Scala Eclipse plugin. It enables the developer to run the interfaces interactively. Here is a list of resources that provide a good introduction to Scala GUI programming:

- Introduction to scala.swing [13]
- Programming in Scala - Chapter 32 [16]
- Scala Swing library functions - [18]

6.1.1 Scala UI

Before running the Scala UI project given in our code repository, follow these preparation steps:

- Ensure that JDK8 is installed in your system. If not, you can download it from: `http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`
- Install Scala Eclipse IDE from `http://scala-ide.org/download/sdk.html`

The Scala UI code was developed and tested in Scala 2.11 and with a JDK 1.8 compiler. To run the code, from our repository, import the Scala interface project into your Eclipse projects. Then, open this project in your Scala Eclipse IDE (Figure 6.1).

Before running the project, create a run configuration that points to the .scala file/class name (Figure 6.2). Once everything is set up, you can run the code by pressing the run button in Eclipse.

6.2 Interfacing with Twitter Data

Matthew's thesis framework provides us with a simple and straightforward way to interface with Twitter data stored on the cluster, clean the data, filter the data, and run several different analyses on the data.

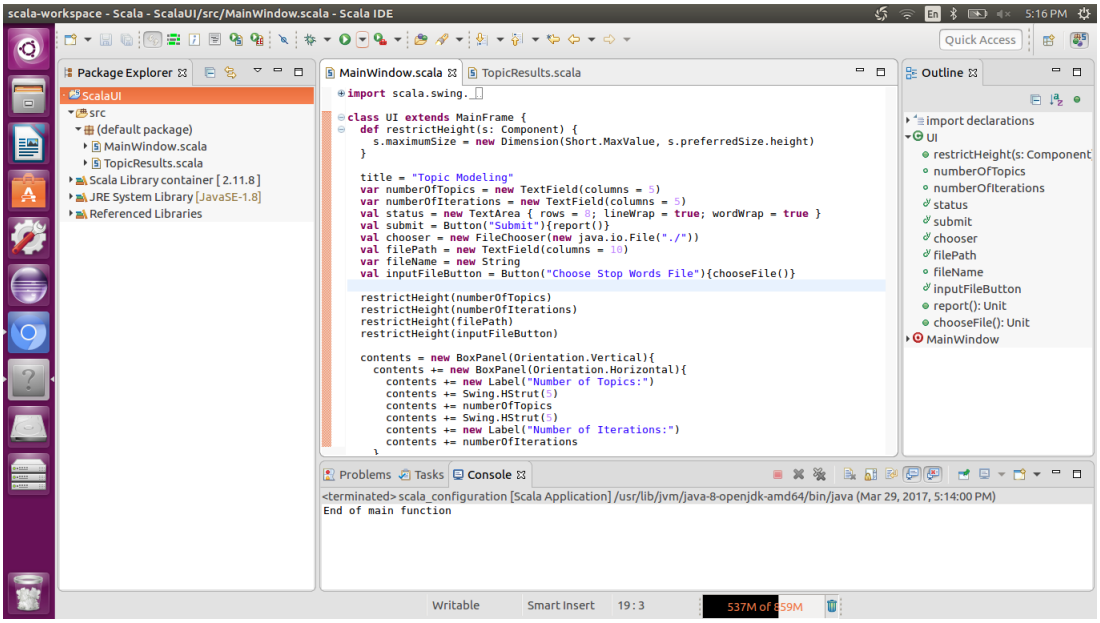

Figure 6.1: Project in Scala Eclipse IDE

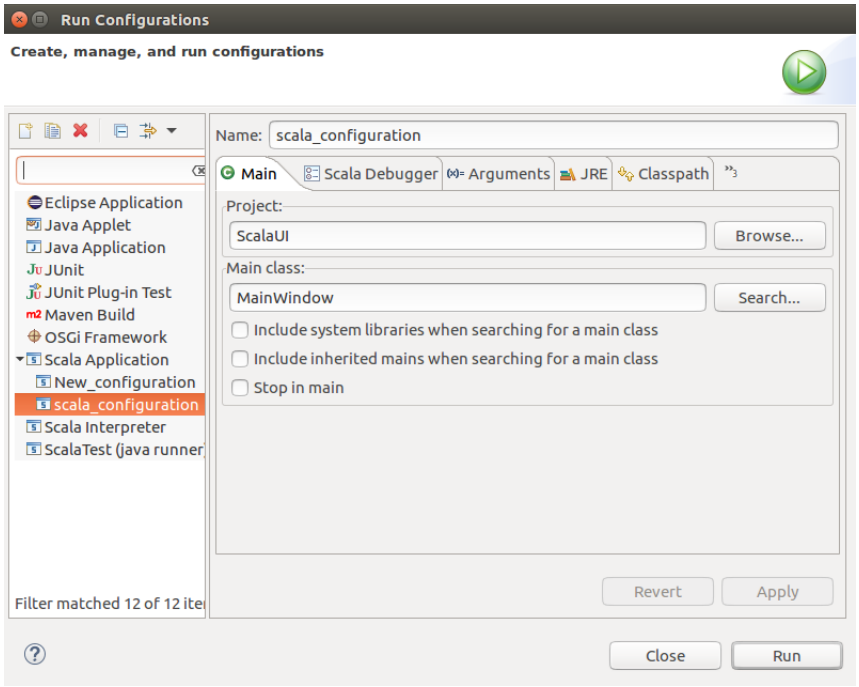

Figure 6.2: Creating run configuration in Eclipse

Name	Type	Description
text	String	Unfiltered text of the tweet
id	String	Unique ID of this tweet
isRetweet	Boolean	True if the tweet is a retweet
tokens	Array[String]	Tweet text broken up into tokens
hashtags	Array[String]	Array of hashtags found in tweet text
mentions	Array[String]	Array of mentions found in tweet text
urls	Array[String]	Array of URLs found in tweet text
payload	Map[String, String]	Payload for holding extra data read in from files or derived by tools

Table 6.1: The fields represented in the Tweet data structure.

Function	Description
cleanRTMarker	Remove the RT marker
cleanMentions	Remove mentions from tokens
cleanHashtags	Remove hashtags from tokens
cleanURLs	Remove URLs from tokens
cleanPunctuation	Remove non-alphanumeric characters except @ and #
cleanRegexMatches(regex*)	Remove tokens matching specified regular expression
cleanRegexNonmatches(regex*)	Remove tokens that don't match specified regular expression
cleanTokens(Array[String])	Remove all instances of specified tokens
toLowerCase	Turn all text lowercase
addToPayload(key: String, value: Any)	Add a key and value to the payload
toStringVerbose	String representation of all data about tweet
toString	String representation of tweet in form "id \t tokens"
equals	Returns true if comparing two tweets with the same ID

**scala.util.matching.Regex*

Table 6.2: The functions provided by the Tweet data structure.

Excerpts from his technical documentation that were relevant to our project are replicated below, with some discussion about how we used the framework to our advantage.

Data I/O

We made use of the framework to handle reading data out of the Avro files stored on the cluster and our own local test files. The framework handles pulling the data, decoding it from the binary format, and turning it into collections of `Tweet` data structures. The `Tweet` data structure is a data structure designed to represent a tweet pulled from Twitter by the tools used in the Digital Library Research Lab. It contains fields to represent each of the types of data stored on the cluster in the Avro files, as well as an extra payload field to hold arbitrary extra data (e.g., topic labels, sentiment tags, etc.). It also breaks the tweet text into an array of tokens for future convenience. The fields are detailed in Table 6.1, and the functions are detailed in Table 6.2. Future developers may wish to leverage the other fields and functions provided here when designing a new approach to the system.

The framework handles creating these `Tweet` data structures and storing them in `TweetCollection` data structures. There is no need for the developer to interface with Spark at all here. Instead, he or she can simply specify a data source to the appropriate `TweetCollectionFactory` function and it will handle the construction of the data structure.

LDA only	Both	Sentiment only
Stopwords	URLs	Mentions
	lowercase	Hashtags
	punctuation	
	RT marker	

Table 6.3: The cleaning done in each step of the system.

Data Pre-Processing

The framework provides a suite of ways to clean, filter, and otherwise modify the tweet collection in ways that will be beneficial to the kinds of digital libraries research that will need this kind of data structure. Cleaning is a very broad topic, and it is likely that future developers may need to clean the data differently to suit their needs. Currently, we have separate cleaning functionalities for the topic analysis and sentiment analysis portions due to the needs of the implementations. These functions are defined in small container classes at the top of the GUI classes in the `MainWindow.scala` file. All of the cleaning functionalities utilized by these cleaning functions are provided to us by the framework. Our cleaning approaches are summarized in Table 6.3.

Topic Analysis

For the topic analysis portion of our project, we are using LDA. We make use of the framework's `LDAWrapper` to run Spark's built-in LDA tool on the tweet collection we have created and cleaned. `LDAWrapper` runs LDA topic analysis on the collection using a set of default parameters. These parameters include the number of iterations to run, the number of topics to search for, which optimizer to use, and a set of terms to ignore. The parameters can be customized before LDA is run, but we are currently using all of the default parameters. We make use of this flexibility in our user interface by changing the parameters based on user input. If future developers chose to use a different algorithm for topic analysis, it would require a lot of re-implementation, including radically changing the user interface to accommodate interfacing with the new approach.

6.2.1 Submitted Files

See Table 6.4 for a description of each file submitted with the project in 6604Files.tar.

Table 6.4: Breakdown of Submitted Files

File	Description
`Sentiment Analysis/ Final_sentiment_analysis.py`	Reads tweet collection and calls syntaxnet using "first3.sh".
`Sentiment Analysis/ first3.sh`	Takes tweet as input and then passes this tweet to syntaxnet.
`Sentiment Analysis/ parse_tree.py`	Data structure to represent the file returned by syntaxnet
`Sentiment Analysis/ reverse_polarity_file`	Polarity reversal and negation words from General Inquirer.
`Topic Analysis/ MainWindow.scala`	Contains the code for our Scala-based User Interface
`Topic Analysis/pom.xml`	The Maven build file to launch our project in Eclipse
`Word2VecSentimentAnalysis.scala`	Our initial attempt at Sentiment Analysis with emojis
`AT0412.txt`	An example test file of Tweet data
`topics1`	Example output file from the Topic Analysis tool
`topics2`	Example output file from the Topic Analysis tool
`topics3`	Example output file from the Topic Analysis tool
`topics4`	Example output file from the Topic Analysis tool

Chapter 7

Plan

7.1 Team member specializations and responsibilities

Table 7.1 explains all of the overall roles that the team members will play in the project, as well as their areas of interest/specialization that they can lend to the project.

7.2 Timeline

Table 7.2 explains a rough weekly breakdown of our work done so far and our plan for the rest of the semester.

Name	Interests	Project Focus
Abigail	NLP	Sentiment Analysis, LDA
Matthew	Spark, Scala	Implementation, Sentiment Analysis
Rahul	NLP	Sentiment Analysis
Radha	Spark, Scala	LDA, UI

Table 7.1: Breakdown of roles and interests of each team member

Weeks	End Date	Tasks
Week 1	21 Jan.	Initial class meetings, discussing potential projects.
Week 2	28 Jan.	Finalization of projects and assignment into teams.
Week 3	4 Feb.	Discussions within team and with class about scope of project.
Week 4	11 Feb.	Literature review and identification of available tools.
Week 5	18 Feb.	Literature review and refinement of scope and approach to project.
Week 6	25 Feb.	Literature review and refinement of approach to project.
Week 7	4 March	Summarization of literature review into Interim Report 1. Planning for how to divide up project responsibilities moving into implementation/prototyping phase.
Week 8	11 March	Initial prototyping and implementation. Work towards segregating positive and negative tweets.
Week 9	18 March	Spring break. Initial prototyping and implementation.
Week 10	25 March	First working prototypes/wireframes ready for evaluation and refinement.
Week 11	1 April	Work towards combining individual functional components (sentiment tool, topic tool, GUI).
Week 12	8 April	Continues work towards combining individual functional components. First end-to-end prototype should be functional by end of week.
Week 13	15 April	Prototype refinement.
Week 14	22 April	Evaluation and preparation for final report and presentation.
Week 15	29 April	Evaluation and preparation for final report and presentation.

Table 7.2: Weekly breakdown of work.

Chapter 8

Future Work

Our topic analysis cleans the tweets by filtering out stopwords, URLs, punctuation, and RT markers. Adding stemming and lemmatization to our data cleaning for our system's topic analysis would probably produce more meaningful topics for the researchers using our system.

In regard to our sentiment analysis component, adding more labeled data to the emoji-based sentiment classifier could improve its results. This could be done by incorporating event specific hand-labeled data to the system (i.e., hand-labeled data for hurricanes could be used for collections about hurricanes). Additionally, supplementing the emoji lookup table with a pre-defined dictionary that has sentiments associated with specific words could also improve the emoji-based sentiment classifier. For our dependency tree based sentiment classifier, combining name entity recognition with the parse trees to get the polarity of entities could improve its performance. Both classifiers would also benefit from using part of speech tagging.

Adding more granular controls, and incorporating sentiment analysis with the user interface would be the next step in assisting researchers without a computing background with their analyses of tweets collected for DLRL. In the future, a more robust user interface could also be created with modern frameworks like Play. Play is a web framework written in Scala to build web friendly applications.

Acknowledgements

Our team would like to thank and acknowledge Dr. Edward Fox for giving us the opportunity to work on a project that aligned with our research interests, as well as for offering advice and guidance on this project. We would like to acknowledge the Digital Library Research Laboratory, especially those who were involved with the Integrated Digital Event Archiving and Library (IDEAL) and Global Event and Trend Archive Research (GETAR) projects. Thus, we would also like to express our appreciation to the National Science Foundation for funding the IDEAL and GETAR projects (grants IIS-1319578 and IIS-1619028, respectively). Furthermore, we would also like to thank the Topic Analysis teams in CS 5604 in the Spring 2016 and Fall 2016 semesters, as their LDA tool was extremely useful to the success of the project.

Bibliography

[1] *Apache Hadoop project*. http://hadoop.apache.org/. Accessed Jan 2017. Apache Software Foundation.

[2] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. "SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining." In: *LREC*. Vol. 10. 2010, pp. 2200–2204.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent Dirichlet Allocation". In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.

[4] Felipe Bravo-Marquez, Eibe Frank, and Bernhard Pfahringer. "Positive, negative, or neutral: Learning an expanded opinion lexicon from emoticon-annotated tweets". In: *IJCAI 2015*. Vol. 2015. AAAI Press. 2015, pp. 1229–1235.

[5] *Emojipedia*. http://emojipedia.org/. Accessed Mar 2017. Emojipedia.

[6] Edward Fox et al. "Global Event Trend and Archive Research (GETAR)". In: (Nov. 2015). NSF grant IIS-1619028 and 1619371. URL: http://www.eventsarchive.org/sites/default/files/GETARsummaryWeb.pdf.

[7] Edward Fox et al. "Integrated Digital Event Archiving and Library (IDEAL)". In: (Sept. 2013). NSF grant IIS-1319578. URL: https://www.nsf.gov/awardsearch/showAward?AWD_ID=1319578.

[8] Roger Hurwitz. *General Inquirer*. 2002. URL: http://www.wjh.harvard.edu/~inquirer/ (visited on 05/03/2017).

[9] Roger Hurwitz. *General Inquirer*. 2002. URL: https://github.com/tensorflow/models/tree/master/syntaxnet (visited on 05/03/2017).

[10] C.J. Hutto and Eric Gilbert. "VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text". In: *Eighth International Conference on Weblogs and Social Media (ICWSM-14). Available at (20/04/16)*. 2014. URL: http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf.

[11] Fangtao Li, Minlie Huang, and Xiaoyan Zhu. "Sentiment Analysis with Global Topics and Local Dependency." In: *AAAI*. Vol. 10. 2010, pp. 1371–1376.

[12] Zhiyuan Liu, Arjun Chen, and Mukherjee Bing. "Aspect extraction with automated prior knowledge learning". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. 2014, 347–358.

[13] Ingo Maier. "The Scala.Swing package". In: *Scala Improvement Process (SID)* 8 (2009). URL: https://www.scala-lang.org/old/sites/default/files/sids/imaier/Mon,%202009-11-02,%2008:55/scala-swing-design.pdf.

[14] Tetsuji Nakagawa, Kentaro Inui, and Sadao Kurohashi. "Dependency tree-based sentiment classification using CRFs with hidden variables". In: *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics. 2010, pp. 786–794.

[15] Graham Neubig et al. "Safety Information Mining-What can NLP do in a Disaster." In: *IJCNLP*. Vol. 11. 2011, pp. 965–973.

[16] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala: A Comprehensive Step-by-step Guide*. USA: Artima Incorporation, 2008. ISBN: 0981531601, 9780981531601.

[17] Hassan Saif et al. "Contextual semantics for sentiment analysis of Twitter". In: *Information Processing & Management* 52.1 (2016), pp. 5–19.

[18] *Scala Swing Library*. `http://www.scala-lang.org/api/2.11.2/scala-swing/`. Accessed Mar 2017.

[19] *ScalaFX Library*. `https://github.com/scalafx/scalafx`. Accessed Mar 2017. BSD Open Source.

[20] Duyu Tang et al. "Building Large-Scale Twitter-Specific Sentiment Lexicon: A Representation Learning Approach." In: *COLING*. 2014, pp. 172–182.

[21] Peter D Turney. "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. Association for Computational Linguistics. 2002, pp. 417–424.

[22] Soroush Vosoughi, Helen Zhou, and Deb Roy. "Enhanced Twitter sentiment classification using contextual information". In: *arXiv preprint arXiv:1605.05195* (2016).

[23] Theresa Wilson, Janyce Wiebe, and Paul 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.

[24] Hongliang Yu, Zhi-Hong Deng, and Shiyingxue Li. "Identifying Sentiment Words Using an Optimization-based Model without Seed Words." In: *ACL (2)*. 2013, pp. 855–859.

[25] Massung S. Zhai C. *Text Data Management and Analysis*. Association for Computing Machinery, Morgan, and Claypool Publishers, 2016.

Appendix A: Labeled Emojis

Tables A.1 and A.2 are screenshots of the table that we used to label emojis. The Unicode and CLDR Short Names were found on the Unicode, Inc. website [5].

Number	Code	CLDR Short Name	Rating
1	U+1F600	grinning face	pos
2	U+1F601	grinning face with smiling eyes	pos
3	U+1F602	face with tears of joy	pos
4	U+1F923	rolling on the floor laughing	pos
5	U+1F603	smiling face with open mouth	pos
6	U+1F604	smiling face with open mouth and smiling eyes	pos
7	U+1F605	smiling face with open mouth and cold sweat	?
8	U+1F606	smiling face with open mouth and closed eyes	pos
9	U+1F609	winking face	?
10	U+1F60A	smiling face with smiling eyes	pos
11	U+1F60B	face savouring delicious food	pos
12	U+1F60E	smiling face with sunglasses	pos
13	U+1F60D	smiling face with heart-eyes	pos
14	U+1F618	face blowing a kiss	pos
18	U+263A	smiling face	pos
19	U+1F642	slightly smiling face	pos
20	U+1F917	hugging face	pos
35	U+1F60C	relieved face	pos
39	U+1F61D	face with stuck-out tongue and closed eyes	pos
40			
41	U+1F612	unamused face	neg
42	U+F613	face with cold sweat	neg
43	U+1F614	pensive face	neg
44	U+1F715	confused face	neg

Table A.1: Labeled emojis

48	U+2639	frowning face	neg
49	U+1F641	slightly frowning face	neg
50	U+1F616	confounded face	neg
51	U+1F61E	disappointed face	neg
52	U+1F61F	worried face	neg
53	U+1F624	face with steam from nose	neg
54	U+1F622	crying face	neg
55	U+1F62D	loudly crying face	neg
56	U+1F626	frowning face with open mouth	neg
57	U+1F627	anguished face	neg
58	U+1F628	fearful face	neg
59	U+1F629	weary face	neg
60	U+1F62C	grimacing face	neg
61	U+1F630	face with open mouth and cold sweat	neg
62	U+1F631	face screaming in fear	neg
63	U+1F633	flushed face	?
64	U+1F635	dizzy face	neg
65	U+1F621	pouting face	neg
66	U+1F620	angry face	neg

Table A.2: Labeled emojis, continued