Approaches to Automatically Constructing Polarity Lexicons for Sentiment Analysis on Social Networks

Thesis

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate School of The Ohio State University

By

Vinh Khuc, B.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2012

Thesis Committee:

Prof. Rajiv Ramnath, Advisor

Prof. Jay Ramanathan

Abstract

Sentiment analysis is a task of mining subjective information expressed in text, and has received a lot of focus from the research community in Natural Language Processing in recent years. With the rapid growth of social networks, sentiment analysis is becoming much more attractive to Natural Language Processing researchers. Identifying words or phrases that carry sentiments is a crucial task in sentiment analysis. The work in this thesis concentrates on automatically constructing polarity lexicons for sentiment analysis on social networks.

One of the challenges in sentiment analysis on social networks is the lack of domain-dependent polarity lexicons and there is a need for automatically constructing sentiment lexicons for any specific domain. Two proposed methods are based on graph propagation and topic modeling. Our experiments confirm the quality of the polarity lexicons constructed using these two algorithms.

Dedication

Acknowledgements

Vita

June 2008 …….. B.S.

Department of Applied Mathematics

and Computer Science

Moscow State University

September 2009 – present …….. Graduate Student

Department of Computer Science

and Engineering

The Ohio State University

Publications

Khuc, V. N., Shivade, C., Ramnath, R., & Ramanathan, J. 2012. Towards building large-scale distributed systems for twitter sentiment analysis. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12). ACM, New York, NY, USA. pp. 459--464.

Field of Study

Major Field: Computer Science

Table of Contents

List of Tables

List of Algorithms

List of Figures

Chapter 1: Introduction

Social networks have become very popular in the past few years. Users on social networking sites usually express opinions about topics that they are interested in. Recent surveys have stated that both customers and product producers are being influenced more by opinions on online reviews and social networks than by traditional media [1]. For that reason, sentiment analysis on social networks has attracted a lot of interest in the field of Natural Language Processing (NLP) in recent years. In sentiment analysis, identifying polarity words is a crucial task. Due to the issue of domain-polarity dependency, there is a need for automatic construction of polarity lexicons for any specific domain. In NLP, there are two popular approaches: graph-based and probabilistic modeling-based. The work in this thesis focuses on graph-based and statistical algorithms for automatically building polarity lexicons for lexicon analysis on social networks.

1.1. Sentiment Analysis

Sentiment analysis is a task of determining polarity in opinions, feelings, and attitudes expressed in text sources. A sentiment analysis tool should predict whether the underlying opinion in a given text is positive, negative, or neutral. Increasing interest in this research area is due to many useful applications associated with it: calculating public opinion polls of presidential elections in blogosphere, measuring customer satisfaction from product reviews, and

customer feedback from websites (Amazon, BestBuy, etc.) and social networks (Twitter, Facebook, etc.), to name a few.

1.2. Social Networks

Over the last few years, social networks have emerged as a new communication channel between people. Social networking services are connecting people who have the same interests and activities regardless of their geographic borders. As of March 2012, Twitter, an online social networking and microblogging service for publishing brief message updates, is estimated to have more than 140 million active users. These users are creating approximately 340 million messages, which are called tweets and are limited to 140 characters, a day [2]. Twitter has drawn a dramatic surge of interest from the research community [3,4,5,6,16]. Unlike other Web-based social networking services that focus on friendship connections, Twitter employs the “following-follower” model, where a Twitterer is able to choose who he/she wants to follow without any granted permission. Conversely, he/she can also be followed by other Twitterers without granting permission. With very low latency of message delivery, Twitter is considered a real-time communication platform that allows users to receive updated posts as soon as they are published by others.

1.3. Twitter Sentiment Analysis

Twitter is also considered a customer relationship platform, where customers are able to easily post reviews about products and services. Providers also can interact with their customers by replying directly to these posts. Many

companies have started collecting Twitter data in order to measure customer satisfaction about their products.

1.4. Research Challenges

In this section, some research challenges in the field of Twitter sentiment analysis are highlighted and serve as the basis for the methods developed in this thesis.

Twitter messages are short and may contain misspelled words. Thus, traditional NLP techniques, which are designed for working with formal languages, perform poorly when applied to tweets [6].

Tweets have the following special characteristics. First, Twitters have the tendency to include emoticons and punctuation characters to express their sentiment, for example, “downloading apps for my iphone! So much fun :-) There literally is an app for just about anything”. Second, tweets may contain abbreviated words due to the limit of 140 characters: “Watchin Espn..Jus seen this new Nike Commerical with a Puppet Lebron..sh*t was hilarious...LMAO!!!” Last, repeated characters are used to express the degree of sentiment, such as “Stopped to have lunch at McDonalds. Chicken Nuggetssss! :) yummmmmy.” Thus, misspellings are introduced in tweets as a result of the described characteristics.

Traditional opinion lexicons such as SentiWordNet [7] are constructed for detecting sentiment in formal language documents such as customer feedback forms, news articles, etc. Those opinion lexicons would not help us extract correct sentiment from tweets. For instance, in the tweet “This is soo goooddd!!!,”

the word “goooddd” is obviously a misspelled word and is not included in traditional sentiment lexicons. Therefore, the tweet would be classified as “neutral.”

Another challenge is that Twitter users can express their opinions about different topics. When talking about wine, one may post a negative message such as “This wine is green!” The word “green” in this tweet carries a negative meaning about the wine product. However, this word would be neutral without knowing the context. Apparently, “green” is not contained in any general-purpose opinion lexicon as a negative word. In addition, adding sentiment words manually for each topic consumes a lot of time and effort.

The identified challenges are the basis for us to develop algorithms for automatically building sentiment lexicons for a given topic.

Chapter 2: Related Work

This section briefly summarizes related work on sentiment analysis/opinion mining. In addition to the lexicon-based approach, we will review machine learning-based methods as well as methods that utilize both sentiment lexicons and machine learning techniques.

2.1. Lexicon-Based Approach

The lexicon-based approach has been developed to perform opinion mining at sentence level and document level by searching for polarity words from a predefined word list [8,9,10,11]. This list is called an opinion lexicon and it contains positive and negative words. If there is no polarity word presented in a sentence/document, the sentence/document is labeled as neutral. We emphasize that existing sentiment lexicons have been constructed from formal language text sources, and therefore do not include slang words, misspelled words, or emoticons.

A graph-based algorithm has been introduced for deriving opinion lexicons from Web data [12]. In this algorithm, a graph $G = (V, E)$ is first constructed. Each node $v_i \in V$ represents a unique word w_i and two nodes $v_i,\ v_j$ are connected by an edge $e_{ij} = (v_i,\ v_j) \in E$ if two words $w_i,\ w_j$ have a similar score above a certain threshold. A list of negative and positive words are identified initially and marked as seed words, which are represented as seed nodes in the graph G. In

order to discover other sentiment words, the authors propagate sentiment scores from seed nodes into their neighbor nodes. In Chapter 3, we will describe an adapted version of this technique to build sentiment lexicons for Twitter using only a small set of emoticons.

2.2. Machine Learning-Based Approach

This approach has been used by Pang et al. [13] to analyze sentiment in movie reviews. Three supervised machine learning algorithms, Naïve Bayes, Maximum Entropy, and Support Vector Machines (SVM) are used for comparison. The experiment results confirm that the SVM algorithm gives better accuracy than Naïve Bayes and Maximum Entropy. Go et al. [3] compared the same algorithms on Twitter data using distant supervision and obtained similar results. The works from both Pang et al. and Go et al. classify text into negative and positive categories only.

2.3. Hybrid Approach

The hybrid approach combines lexicons and machine learning algorithms for sentiment analysis. Zhang et al. [4] introduced an entity-level method in which training data is created automatically by assigning an unlabeled tweet into positive or negative categories based on its included sentiment words. The training data is then fed into an SVM classifier. Their results show that the accuracy of the SVM classifier is improved using the augmented training data.

Another method to utilize both sentiment lexicons and machine learning algorithms is proposed by Agarwal et al. [14]. In their work, opinion words are generated by extending the Dictionary of Affect in Language using WordNet [15].

The constructed opinion words contain approximately 8,000 English words in which each word has a sentiment score ranging from 1 to 3. These scores are then normalized by diving by 3. Negative words are those that have sentiment scores below 0.5, whereas words with scores above 0.8 are positive. Remaining words with scores ranging from 0.5 to 0.8 are considered neutral. These sentiment scores are then used as prior polarity. Part-of-Speech (POS) tags together with prior polarity are then used to derive senti-features for SVM classifier.

Chapter 3: A Large-Scale Distributed Lexicon Builder for Twitter Sentiment Analysis

In this chapter, a large-scale distributed system is proposed for extracting sentiment words from Twitter posts. This lexicon builder adapts the idea from the work of Velikovich et al, [12], in which a graph connecting words is constructed to identify sentiment words with a given set of seed polarity words. However, the original method was designed for constructing a lexicon from Web data, which has different properties from Twitter data.

3.1. Architecture

Our method is implemented using Hadoop [18], an open source implementation of the MapReduce framework [19], and hence, it scales well by simply adding commodity computers. The graph data is stored in HBase [20], a Bigtable-like database [21] built on top of Hadoop, and therefore, scales well also.

The resulting sentiment lexicons are then consumed by a sentiment classifier that uses a lexicon-based approach to detect sentiment in tweets. The system architecture is described in Figure 1.

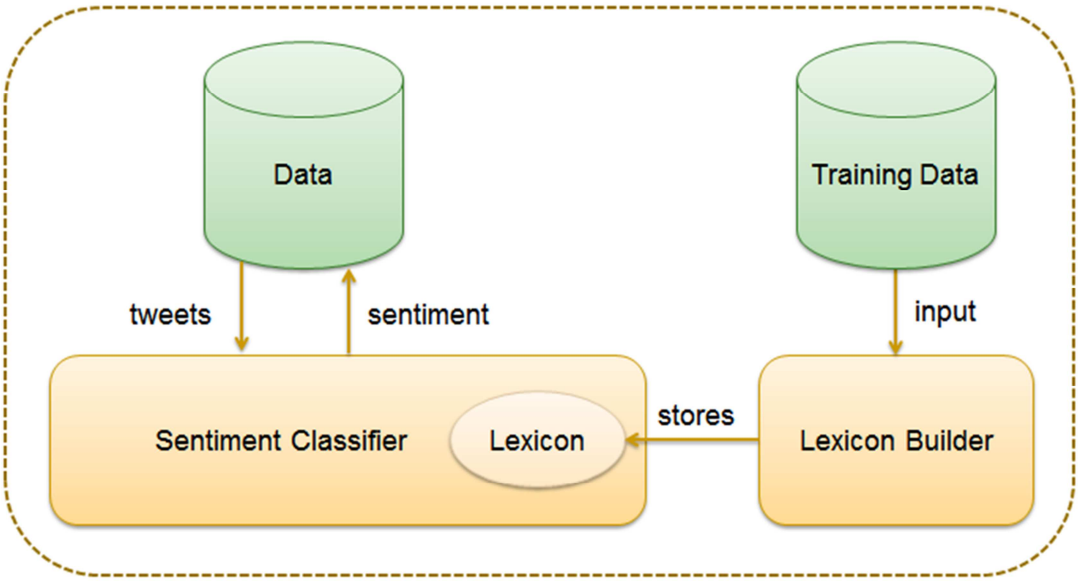

Figure 1. Large-scale distributed lexicon builder and classifier for Twitter sentiment analysis.

3.2. Hadoop

Hadoop [18] is an Apache open-source software that supports data-intensive applications. It is an implementation of Google's MapReduce idea [19]. The MapReduce framework is a programming model designed for scalable and distributed computing on clusters of commodity computers.

A MapReduce job consists of Map and Reduce steps. In the Map step, the large input data is divided into smaller parts by a master node and those parts are fed into worker nodes in the cluster for processing. The partial results produced by worker nodes are then collected back by the master node in the Reduce step. These gathered partial results are combined in an appropriate way to calculate the final output for the original large input data.

3.3. HBase

Apache HBase [20] is a Bigtable-like [21] database software built on top of Hadoop and has been used as a large-scale and distributed data storage. HBase tables can serve as input and output for Hadoop's MapReduce jobs, and therefore, are chosen to store the large graph data for our lexicon builder.

3.4. Algorithm

Our lexicon builder uses the idea from the work of Velikovich et al. [12] to construct sentiment lexicons for Twitter data where the seed polarity words are emotional icons (or emoticons) such as ":D", ":)" or ":-(" as examples. Our lexicon builder is implemented in a scalable and distributed way with the help of Hadoop and HBase.

The co-occurrences for each pair of words are calculated first and then are used to compute the cosine similarity scores to construct a word graph. Each node in the graph represents a word and two nodes are connected by an edge with the weight equal to the cosine similarity scores of words represented by these nodes. Edges with low weights are then discarded. After that, sentiment scores are propagated from the seed nodes into their neighbor nodes to help identify polarity words (Figure 2).

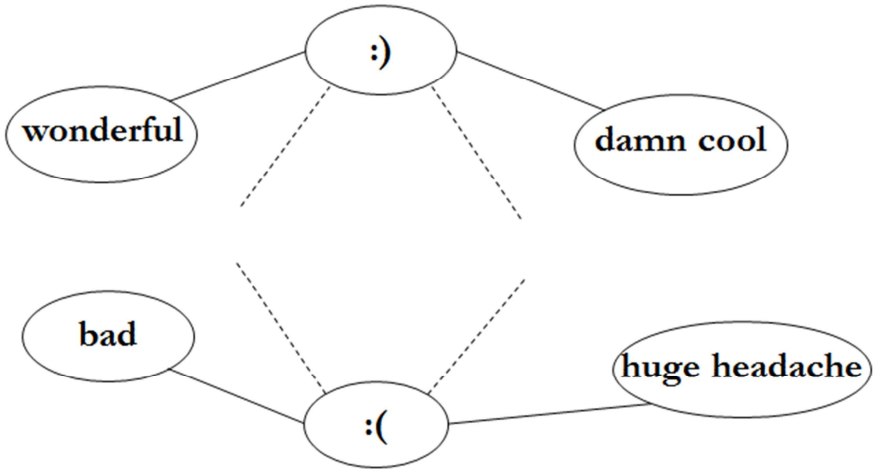

Figure 2. Graph connecting words/phrases.

In our implementation, tweets are normalized first so that repeated characters are reduced. For example, we map both words “goooddddd” and “goooodd” to “goodd” by replacing every “ooo” and “ddd” by “oo” and “dd”, respectively. This approach is similar to the approach used in the work by Go et al. [3]. Next, a POS tagger[1], which is created specifically for Twitter data by Gimpel et al. [16], is used to extract nouns, adjectives, adverbs, verbs, interjections, abbreviations, emoticons, and hashtags from the input tweets with the assumption that only those words contain sentiment. This filtering step is done for reducing the size of the word-graph.

Next, we will describe our algorithm, which is based on the MapReduce framework.

3.4.1. Co-occurrence matrix. Co-occurrence matrix A^{NxN} (N is the number of words) contains information about how many times the word i co-

[1] http://code.google.com/p/ark-tweet-nlp/

occurs with the word j. The MapReduce job for calculating the co-occurrence matrix is described in Algorithm 1.

For each tweet t:

1. Lowercase t.
2. Separate t into a list of tokens: $word\_list_t$ = twokenize(t).
3. Remove tokens which are not nouns, adjectives, adverbs, verbs, interjections, abbreviations, hashtags, or emoticons from $word\_list_t$.

In mapper:

a) For each word w from $word\_list_t$, emit <w, v> where v is a word or a phrase within a window size T from w.
b) Similarly, for each phrase p, emit <p, v>

In reducers:

a) Count the number of entries <u, v> as the number of co-occurrences k between u and v.
b) Insert the value k into table co-occurrence_matrix at row u and column v in the column family “co-occurence”.

Algorithm 1. Computing co-occurrence matrix.

In order to reduce to the size of the matrix, we include only unigram and bigram words into $word\_list_t$, where a bigram word is formed by two consecutive

unigram words. An HBase table is used to store the co-occurrence matrix and has the schema described in Table 1.

Key	Column family “co-occurrence”			
w_1	w_1:k_{11}	w_2:k_{12}	…	w_n:k_{1n}
…	…	…	…	…
w_n	w_1:k_{n1}	w_2:k_{n2}	…	w_n:k_{nn}

Table 1. Co-occurrence table.

3.4.2. Computing similarity scores and constructing the word-graph.

The similarity score between two words W_i and W_j is defined as the cosine similarity $\text{cosine\_sim}(w_i, w_j)$ between their presenting vectors $w_i = (k_{i1}, k_{i2}, \dots, k_{in})$ and $w_j = (k_{j1}, k_{j2}, \dots, k_{jn})$, respectively. In order to save computing time by not re-computing vector lengths, all vectors $w_i = (k_{i1}, k_{i2}, \dots, k_{in})$ are normalized into unit vectors $w'_i = (k'_{i1}, k'_{i2}, \dots, k'_{in})$. Hence, $\text{cosine\_sim}(w_i, w_j)$ is calculated as the pairwise dot-product between the unit vectors w'_i as follows:

$$\text{cosine\_sim}(w_i, w_j) = \sum_{c=1}^{n} k'_{ic} * k'_{jc} \quad (1)$$

In the work of Elsayed et al. [17], a MapReduce algorithm was introduced for computing pairwise cosine similarity for a very large collection of vectors. The idea behind their work is based on the following observation: the index c in

Formula (1) contributes to the right-hand side sum if both coordinates k'_{ic} and k'_{jc} are non-zero. Therefore, if we denote $C_i = \{ c \mid k'_{ic} \neq 0 \}$, then Formula (1) is equivalent to the following formula:

$$\text{cosine\_sim}(w_i, w_j) = \sum_{c \in C_i \cap C_j} k'_{ic} * k'_{jc} \quad (2)$$

The pseudo-code for the MapReduce job for vector length normalization and calculating the set of indices C_i is described in Algorithm 2.

In mappers:

For each vector w_i:

a) Compute its length len_i.

b) For every non-zero coordinate w_c, emit <w_c ,<w_i, $norm_{ic}$>> where $norm_{ic} = k_{ic}/len_i$.

In reducers:

Do nothing.

Algorithm 2. Vector normalization and non-zero coordinate indexing.

The cosine similarity scores are calculated in a separate MapReduce job as follows:

In mappers:

For each key w_c:

For every pair of its values <w_u, $norm_{uc}$> and <w_v, $norm_{vc}$>, emit <<w_u, w_v>, $cosine\_sim_c$>, where $cosine\_sim_c = norm_{uc} * norm_{vc.}$

In reducers:

For each key <w_u, w_v>:

a) Take cosine similarity score between w_u and w_v as the sum of all associated values $cosine\_sim_c$: $\text{cosine\_sim}(w_u, w_v) = \sum_c \text{cosine\_sim}_c$.

b) If $\text{cosine\_sim}(w_u, w_v) \geq \alpha$, insert into the graph table two edges (w_u, w_v) and (w_v, w_u) with the same weight $cosine\_sim(w_u, w_v)$.

Algorithm 3. Cosine similarity calculation and graph table creation.

Similar to the co-occurrence table, the graph table schema has the format described in Table 2.

Key	Column family “weight”			
w_1	w_1:p_{11}	w_2:p_{12}	…	w_n:p_{1n}
…	…	…	…	...
w_n	w_1:p_{n1}	w_2:p_{n2}	…	w_n:p_{nn}

Table 2. Graph table.

3.4.3. Discarding edges with low weights. The purpose of discarding edges with low weights is to avoid propagating sentiment scores from seed words to non-sentiment words, and also to improve the speed of sentiment score propagation. As described in the pseudo-code in Algorithm 4, an edge $e_{ij} = (w_i, w_j)$ is retained if it is in the list of $\mathrm{TOP\_N}$ highest weighted edges adjacent to nodes w_i and w_j.

In mappers:

For each key w_i in the graph table:

Calculate a list L_1 of $\mathrm{TOP\_N}$ highest weighted edges adjacent to w_i and a list $L_2 = L \backslash L_1$ where L – list of all edges adjacent to w_i.

a) For each edge $e \in L_1$, emit <e, L_1>.

b) For each edge $e' \in L_2$, emit <e, ∅>.

In reducers:

For each key e, there are two associated values L', L'',

a) If one of the associated values is $\emptyset$, remove edge e from the graph table.
b) If e is not in one of the $\mathrm{TOP\_N}$ highest weighted edges in the combination list $\mathrm{L} = \mathrm{L}' \cup \mathrm{L}''$, edge e is also removed.

Algorithm 4. Discarding edges with low weights.

3.4.4. Propagating sentiment scores. Sentiment scores are propagated from seed nodes into their neighbor nodes located within the distance D from the seed nodes. The HBase table "propagate" has the following structure:

Key	Column family "visited"		
w_{seed_1}	w_{i1}:''	…	w_{j1}:''
…	…	…	...
w_{seed_k}	w_{ik}:''	…	w_{jk}:''

Table 3. Propagate table.

In Table 3, we put nodes that are reachable from seed nodes as qualifiers under the column family "visited". For instance, in the row corresponding to the

seed node w_{seed_1}, the qualifiers $w_{i1}, \ldots, w_{j1}$ are the nodes reachable from the seed node w_{seed_1} within the distance D. Algorithm 5 illustrates how score propagation is implemented in MapReduce jobs.

Initial:

a) Put into the propagate table, rows <w_{seed_i}, <visited: w_{seed_i}, ‘’>>, where i = 1, 2, …, k and k is the number of seed nodes.

b) Create a column family named “alpha” in the graph table and insert values < w_i, <alpha: w_i, 1>>.

Repeat the following MapReduce job D times:

In mappers:

For each key w_{seed_i} in the propagate table:

For each qualifier w_{iu} of column family “visited” in row w_{seed_i}:

For each node w_{iv} that is not in column family “visited” in row w_{seed_i} and is adjacent to w_{iu} in the word graph, emit < w_{seed_i}, < $w_{iu,}$ w_{iv}>>.

In reducers:

For each key w_{seed_i}:

a) Retrieve α_{iu} from entry < w_{seed_i}, <alpha: w_{iu}, α_{iu}>> in the graph table. If that entry does not exist, assign $\alpha_{iu} := 0$.

b) Similarly for α_{iv}.

c) Get the weight ω_{uv} of edge ($w_{iu,}$ w_{iv}) from the graph table.

d) If $\alpha_{iu} < \alpha_{iv} * \omega_{uv}$, update $\alpha_{iu} := \alpha_{iv} * \omega_{uv}$ in the graph table.

e) Insert into the propagate table entry <w_{seed_i}, <visited: w_{iv}, ''>> to mark w_{iv} as visited from w_{seed_i}

Algorithm 5. Sentiment score propagation.

3.4.5. Computing final sentiment scores. This is the final step where the sentiment scores are computed and sentiment words are discovered. As can be seen, each node w_i in the graph has two types of scores: $score_i^+$ is the sum of scores accumulated from positive seed words and $score_i^-$ - from negative seed words. These two scores are then combined to get the final sentiment score $score_i$ as shown in the following formula:

$$score_i = score_i^+ - \beta * score_i^- \quad (3)$$

where β is defined as

$$\beta = \frac{\sum_i score_i^+}{\sum_i score_i^-}$$

The final sentiment scores of all nodes in the graph are computed in three MapReduce jobs. The first one calculates $score_i^+$ and $score_i^-$. The next job is

executed to get the value β. Finally, the scores $\mathrm{score_i}$ are generated according to Formula (3).

3.5. Experiment

In this section, we will conduct an experiment to build sentiment lexicons for Twitter using the described lexicon builder. Its performance and scalability also will be evaluated.

3.5.1. Setup. The input Twitter corpus[2] for the lexicon builder consists of only tweets that have either smileys ":)" or frownies ":(".

Because Twitter posts do not follow any specific grammar structure, simply separating words by white spaces does not work. For example, in the tweet "I lovee iPhone:)", the emoticon ":)" is not separated from the word "iPhone" by white spaces. Therefore, a tokenizer created specifically for tweets, called Twokenizer[3], is used for extracting unigram and bigram words.

In addition to the emoticons, nouns, adjectives, verbs, interjections, abbreviations, and hashtags are used for constructing the word graph. The emotions serve as the seed nodes. When computing sentiment scores in the last step, we keep only nodes with absolute sentiment scores at least 1.0 and save them as polarity words.

In our experiment, the parameters are set as follows: distance $\mathrm{D} = 4$, $\mathrm{TOP\_N} = 100$, window size $\mathrm{T} = 6$, threshold $\alpha = 0.01$. We run the lexicon builder on a Hadoop cluster with 5 nodes on Amazon EC2. Each node is a medium

[2] https://sites.google.com/site/twittersentimenthelp/for-researchers
[3] https://github.com/vinhkhuc/Twitter-Tokenizer

instance with 2 virtual cores (2.5 EC2 Compute Units each) and 1.7 GB of memory. Two mappers and two reducers are executed on each node. We measure lexicon builder's running time with datasets consisting of 100,000, 200,000, and 300,000 tweets chosen from the input Twitter corpus.

3.5.2. Results. Figure 4 shows the performance of the lexicon builder with different datasets and different number of nodes. The number of unique words (unigrams and bigrams) for each dataset is shown in Figure 3. The number of unique words increases linearly with the number of tweets due to our naive way of forming bigram phrases and the small amount of training tweets being used.

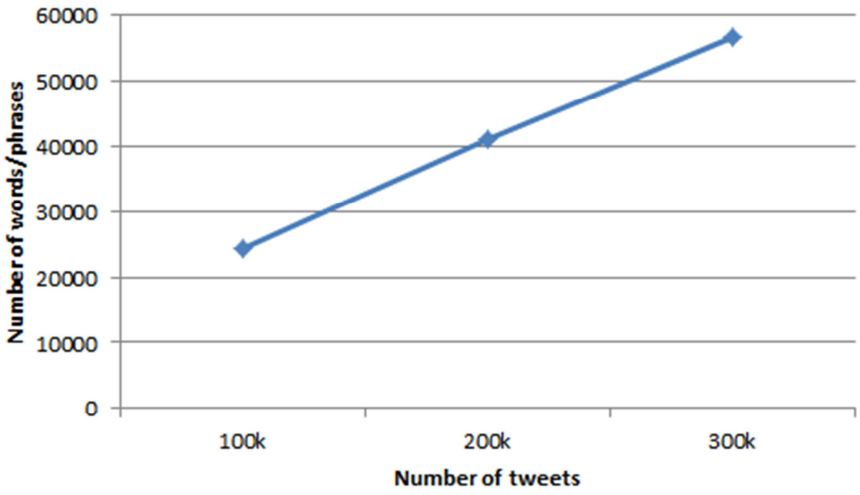

Figure 3. Number of unique words by the number of tweets.

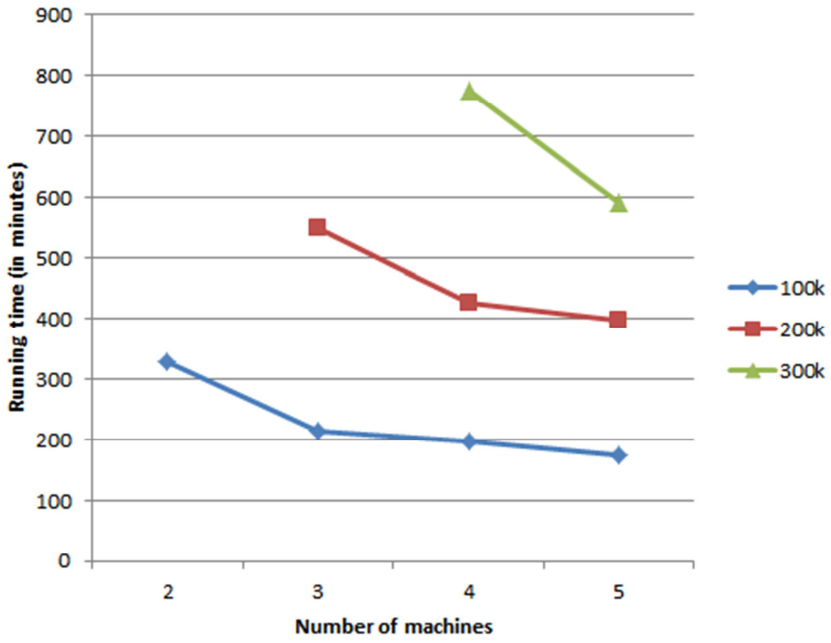

Figure 4. Running time of the lexicon builder with different numbers of nodes and different amounts of tweets.

In Figure 4, the execution time of our lexicon builder with the dataset of 200,000 tweets for a cluster of 2 machines, and the dataset of 300,000 tweets for a cluster of less than 4 machines is not reported due to many OutOfMemory errors when running Hadoop and HBase in a cluster of a small number of nodes. The results have confirmed that lexicon builder's running time decreases when more machines are added. In particular, for the dataset of 100,000 tweets, the execution time is decreased by 35% when moving from 2 machines to 3 machines, by 40% when moving from 2 to 4 machines, and by 47% when moving from 2 to 5 machines. Similarly, with the dataset of 300,000 tweets, the running time is decreased by 23% when increasing from 4 to 5 machines. However, the

running time does not decrease linearly due to the latency of Hadoop and the default settings of HBase tables.

Since the settings of HBase tables are kept as default (i.e., without block caching, Bloom filter, compression, etc.), the execution time may be reduced further when these settings are turned on.

Table 4 shows sample sentiment words in the lexicon constructed using a dataset of 384,397 tweets in which 232,442 tweets contained smileys and 151,955 tweets contained frownies. The final lexicon has 2,411 positive and 1,018 negative words.

Positive	Negative
awesome	damn
lovee	stupid
long weekend	completely exhausted
honey moon	proper weird
pain stop	totally helpless
yess	eww
wohoo	fml
hurray	boored
finish schoolwork	home sick

Table 4. Sentiment words/phrases from the resulting lexicon.

3.6. Discussion

The experiments have shown the scalability of our lexicon builder in automatically generating sentiment lexicons according to a given training dataset, in this case, a collection of tweets containing either smileys or frownies. The execution time can be reduced by adding machines into the cluster thanks to Hadoop and HBase.

Although the experiment has not been conducted for constructing sentiment lexicons for any specific topic, such as food, movie, etc., we argue that our lexicon builder is capable of generating sentiment lexicons for any topic if the input training dataset is relevant to that topic.

However, there are some issues that need to be solved in future work:

- First, some positive words may be assigned incorrectly with negative sentiment scores, and similarly for negative words. This issue occurs due to the fact that Twitterers may write tweets in a negation manner. For example, the bigram word “stop coughing” frequently follows a negation word, such as “cannot stop coughing :(so much for sleeeping.” In this example, a sentiment relationship between “stop coughing” and the seed word “:(“ is likely to be established. Consequently, “stop coughing” may be extracted eventually as a negative word.
- Second, in our implementation, the word graph is difficult to update using added training data without re-constructing from scratch. Moreover, it is hard to maintain different word graphs for different topics.

- Third, in order to reduce the size of the word graph, a POS tagger is used to filter out words that may not contain sentiments. Therefore, we have to use different POS taggers for different languages. As far as we know, the Twitter POS tagger [16] is the first POS tagger created specifically for Twitter data and it only works for tweets in English.

Chapter 4: Topic Modeling-Based Approach to Construct Sentiment Lexicons for Twitter

4.1. Topic Modeling

Topic models are probabilistic models for extracting potential topics that occur in an archive of documents. Probabilistic topic models [22,23] have drawn interest from researchers from different backgrounds, including NLP, machine learning, information retrieval, etc. Topic modeling has been used in topical analysis of scientific journals [24], Wikipedia articles [25], social networks [26], etc.

4.1.1. Latent Dirichlet Allocation. In recent years, Latent Dirichlet Allocation (LDA) has gained popularity among NLP researchers. By modeling documents as a mixture of distributions over words, LDA [23] is capable of discovering hidden “topics”, i.e. distributions over words, in a large collection of documents. In LDA, a topic, which is not assigned a name, is a cluster of words that tend to co-occur frequently in the same document. In general, words under one topic are connected to each other through similar semantic relations. Therefore, one is able to understand the contents of a document by simply reading words under the discovered topics with highest probabilities.

Topic 1 (Airline)	Topic 2 (Food+Movie)	Topic 3 (Bank)
united	king	bank
pilot	moon	america
new	eating	just
flight	eat	credit
drunk	like	get
con	team	new
travel	jacob	card
american	cheese	like
arrested	ship	chase
charged	twilight	time

Table 5. Latent topics discovered from a collection of tweets using LDA.

Table 5 shows an example of three latent topics discovered from a Twitter corpus using LDA. By reading through the words corresponding to each topic, it is obvious that Topic 1 is Airline, Topic 3 is Bank, and Topic 2 is a mix of Food and Movie topics. In LDA, because the number of topics is a parameter, when we provide smaller numbers of topics than the actual number of latent topics existing in the corpus, some of them may combine into one, as seen in Topic 2.

LDA relies on the assumption that the topic distribution has a Dirichlet prior and each document is a mixture of different topics. Moreover, word order in a document is ignored, i.e., words are independent from each other. Because of the bag-of-words assumption, some topics are difficult to interpret by just reading through their associated words.

As a fully unsupervised algorithm, LDA does not offer a way of incorporating supervised labels into its learning process. In other words, LDA has no mechanism to tune extracted topics to suit user's needs even if labeled resources are available.

4.1.2. Labeled Latent Dirichlet Allocation. In some problems, such as document browsing, the user may want to see all documents that are relevant to a chosen label. One approach to these problems is finding associations between words in documents and their most suitable labels. For example, if the word “yummy” is known to be related to the label “food” and the word “melody” is associated with the label “music”, we can easily extract documents for those labels. However, these tasks cannot be done with LDA because topic names are not known beforehand and LDA often extracts topics in an unsupervised manner.

Labeled Latent Dirichlet Allocation (Labeled LDA) [27] is an LDA’s supervised learning version in which a constraint is made by establishing a one-to-one correspondence between latent topics and user-defined labels. Labeled LDA, therefore, allows word--label relationships to be learned directly by constraining the topic model to extract only topics that are in a predefined list.

4.1.3. Topical N-grams. The topic models such as LDA and its extension Labeled LDA rely on the bag-of-words assumption where word order in a document is totally ignored. However, in the real world, this is not true because word order is very important for lexical meaning. In other words, the aggregation of individual words in a phrase cannot provide the same meaning as the whole phrase. For instance, the phrase “sex and the city” is about movie, but it will have a totally different meaning if we sum up the meanings of the individual words “sex”, “and”, “the”, “city”. Similarly, the phrase “kill cancer” does not carry the same meaning as its constituent words “kill” and “cancer.”

A topic model called Topical N-grams (TNG) [28] was proposed for discovering both topical unigram words and n-gram phrases that are associated with latent topics. TNG is an extension of LDA, where observed bigram statuses are introduced to determine whether two consecutive unigram words can be connected to form a phrase. In particular, TNG is able to distill n-gram phrases depending on the nearby information.

4.2. Topic Modeling Approach for Generating Sentiment Lexicons

In this section, we propose our topic model for automatically generating sentiment lexicons. In LDA, if we consider positive and negative as two “topics”, then the words associated with these two topics become sentiment lexicons that we wish to obtain.

Our topic model is called Labeled Topical N-grams (Labeled TNG), which is inspired by two topic models, Labeled LDA and TNG. Sentiment label supervision is incorporated so that the topic model is constrained to use only labels from the document’s label set. The extracted sentiment phrases are very useful for sentiment analysis because we can capture the whole meaning of phrases and, therefore, we are able to avoid the problem of incorrectly summing up the meanings of individual words.

4.2.1. Graphical model representation. The graphical model representation of Labeled TNG is described in Figure 5.

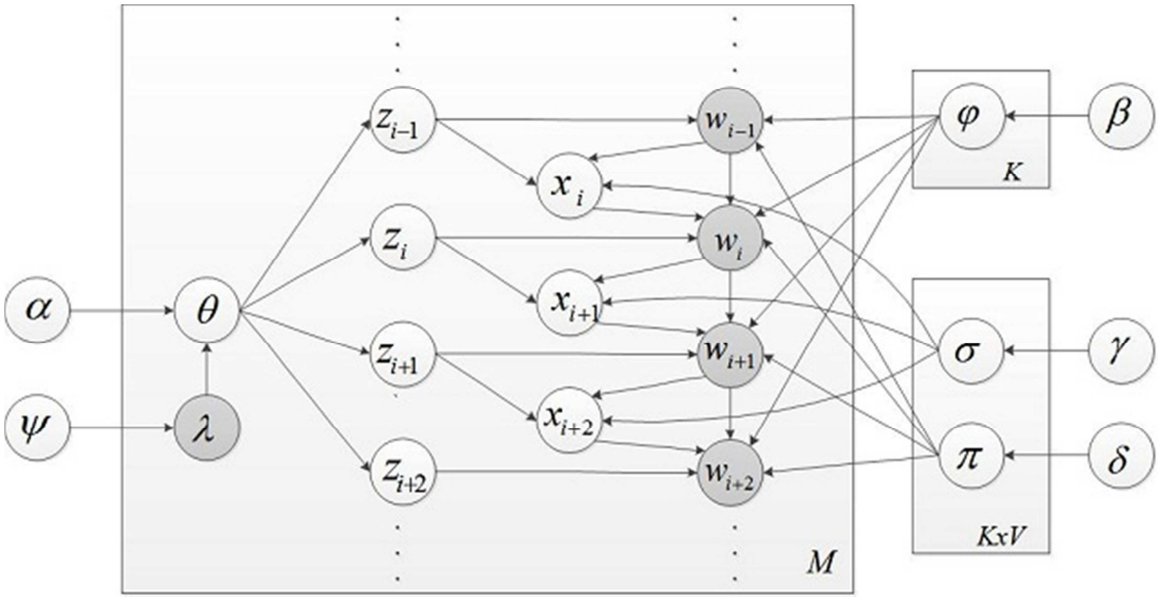

Figure 5. Graphical model representation of Labeled TNG.

Symbol	Description
K	Number of topics
M	Number of documents
V	Number of unique words
N_m	Number of word tokens in document m
w_i^m	The i^{th} word in document m
z_i^m	The topic associated with the word w_i^m in the document m

Table 6. Notation for Labeled TNG (continued)

Table 6 (continued)

x_i^m	The bigram status for the words w_{i-1}^m and w_i^m in the document m
θ^m	The Discrete distribution of topics w.r.t. document m
ψ_z	The Bernoulli distribution of presence/absence of observed topic z in document m
φ_z	The Discrete distribution of words w.r.t. topic z
σ_{zw}	The Bernoulli distribution of bigram status variables w.r.t. topic z and word w
π_{zw}	The Discrete distribution of words w.r.t. topic z and word w
λ^m	The topic presence/absence indicators for document m
α	The Dirichlet prior of θ
α^m	The Dirichlet prior of θ^m w.r.t the topics presented in document m
β	The Dirichlet prior of φ
γ	The Beta prior of σ
δ	The Dirichlet prior of π
ψ	The Beta prior of λ
L^m	The document-specific label projection matrix w.r.t document m

The generative process of Labeled TNG can be described as follows:

1. For each topic z, draw Discrete distribution φ_z from Dirichlet prior β.
2. For each topic z and each word w, draw Bernoulli distribution σ_{zw} from Beta prior γ.
3. For each topic z and each word w, draw Discrete distribution π_{zw} from Dirichlet prior δ.
4. For each document m:
    a. For each topic z, draw λ_z^m from Bernoulli distribution ψ_z.
    b. Draw $\alpha^m = L^m * \alpha$.
    c. Draw a Discrete distribution θ^m from Dirichlet prior α^m.
    d. For each word at position i in document m:
        i. Draw a bigram status x_i^m from Bernoulli distribution $\sigma_{z_{i-1}^m w_{i-1}^m}$.
        ii. Draw a topic z_i^m from Discrete distribution θ^m.
        iii. If $x_i^m = 0$, draw a word w_i^m from Discrete distribution $\varphi_{z_i^m}$; otherwise, draw w_i^m from Discrete distribution $\pi_{z_i^m w_{i-1}^m}$.

Table 7. The generative process of Labeled TNG.

For each document m, the document vector label is defined as $\tau_m = \{ z \mid \lambda_z^m = 1 \}$ [27]. For instance, if $\lambda_z^m = \{ 1, 0, 1, 0 \}$, then $\tau_m = \{ 1, 3 \}$. The label project matrix L^m of size $M_m * K$, where $M_m = |\tau_m|$, is defined as follows:

$$L_{ij}^m = \begin{cases} 1, & \text{if } \tau_i^m = j \\ 0, & \text{otherwise} \end{cases}$$

Hence, the Dirichlet prior α is projected to α^m with the project matrix L^m as follows:

$$\alpha^m = L^m * \alpha = \{ \alpha_{\tau_1^m}, \alpha_{\tau_2^m}, \dots, \alpha_{\tau_{M_m}^m} \}$$

For instance, with $\tau_m = \{ 1, 3 \}$, i.e., document m is assigned with two topics 1 and 3, we have $\alpha^m = \{ \alpha_1, \alpha_3 \}$.

4.2.2. Learning and inference. Similar to the original topic model TNG [28], we use collapsed Gibbs sampling [29] for inferring Labeled TNG as follows:

$$P(x_i = l \mid w, z, x_{-i}, \alpha, \beta, \gamma, \delta) \propto$$

$$\frac{n'_{eul,-i} + \gamma_l}{\sum_{l'=0}^{1}(n'_{eul',-i} + \gamma_{l'})}$$

$$* \begin{cases} \dfrac{n_{kt,-i} + \beta_t}{\sum_{t'=1}^{V}(n_{kt',-i} + \beta_{t'})}, & \text{if } x_i = l = 0 \\ \dfrac{n''_{kut,-i} + \delta_t}{\sum_{u'=1}^{V}(n''_{kuu',-i} + \delta_{u'})}, & \text{if } x_i = l = 1 \end{cases}$$

and

$$P(z_i = k \mid w, z_{-i}, x, \alpha, \beta, \gamma, \delta) \propto$$

$$\frac{n_{mk,-i} + \alpha_k}{\sum_{k'=1}^{K}(n_{mk',-i} + \alpha_{k'})}$$

$$* \begin{cases} \frac{n_{kt,-i} + \beta_t}{\sum_{t'=1}^{V}(n_{kt',-i} + \beta_{t'})}, \text{if } x_i = l = 0 \\ \frac{n''_{kut,-i} + \delta_t}{\sum_{u'=1}^{V}(n''_{kuu',-i} + \delta_{u'})}, \text{if } x_i = l = 1 \end{cases}$$

The hyper-parameters are estimated as follows:

$$\theta_{mk} = \frac{n_{mk} + \alpha_k}{\sum_{k'=1}^{K}(n_{mk'} + \alpha_{k'})}$$

$$\varphi_{kt} = \frac{n_{kt} + \beta_t}{\sum_{t'=1}^{V}(n_{kt'} + \beta_{t'})}$$

$$\sigma_{eul} = \frac{n'_{eul} + \gamma_l}{\sum_{l'=0}^{1}(n'_{eul'} + \gamma_{l'})}$$

$$\pi_{kut} = \frac{n''_{kut} + \delta_t}{\sum_{u'=1}^{V}(n''_{kuu'} + \delta_{u'})}$$

where x_{-i} is the bigram status for all words except word w_i,

z_{-i} is the topics of all words except word w_i,

$n_{kt,-i}$ is the number of times the word t except the word w_i is assigned the topic k,

$n_{mk,-i}$ is the number of times a word in document m is assigned to topic k except word w_i,

$n'_{eul,-i}$ is the number of times the bi-gram status $x_i = l$ w.r.t the previous word $u = w_{i-1}$ and its topic $e = z_{i-1}$,

and

$n''_{kut,-i}$ is the number of times the word t except the word w_i is assigned to the topic k followed by the previous word u in a bi-gram phrase.

4.3. Experiment

For the experiment, we use Labeled TNG to build sentiment lexicons using the Stanford Twitter data set[4]. This dataset consists of 1,600,000 tweets in which 800,000 tweets are positive and 800,000 tweets are negative. These tweets are assigned labels automatically by searching for emoticons.

In our observation, tweets may be assigned both positive and negative labels. For instance, "@wolkenmalerin someone showed it to me ;D i think its awesome. so sad :(but it's a great documentation. everyone should watch it. <33". Therefore, in addition to the sentiment labels provided in the original dataset, we added the negative labels into positive tweets if they contains frownie icons and similarly we added positive labels into negative tweets if they contain smiley icons.

Unlike the experiment conducted in Chapter 4 for the graph-based approach, for Labeled TNG, we use the whole dataset of 1,600,000 tweets. Also, we do not remove stop words except those words with a single character.

We set the following values for $K, \alpha, \beta, \gamma, \delta$:

$K = 2$

$\alpha = (\alpha_1, \alpha_2, \dots, \alpha_k)$, where $\alpha_i = \frac{50}{K}$

$\beta = (\beta_1, \beta_2, \dots, \beta_V)$, where $\beta_i = 0.01$

[4] https://sites.google.com/site/twittersentimenthelp/for-researchers

$\gamma = (\gamma_1, \gamma_2)$, where $\gamma_1 = 0.2$ and $\gamma_2 = 1000$

$\delta = (\delta_1, \delta_2, \ldots, \delta_V)$, where $\delta_i = 0.01$

We ran Gibbs sampling in 10,000 iterations for inferring Labeled TNG on Ubuntu 12.04 server 64-bit with 2.2 GHz Dual Core 2 and 4GB of memory. The resulting lexicon is post-processed so that stop words and phrases in which more than half of constituent words are stop words are removed. Table 8 shows polarity words and phrases in the final lexicon.

N-grams	Positive	Negative
1-grams	good, love, wonderful	sad, stupid, headache
2-grams	thank you, good sleep, happy birthday, cant wait, first kiss	not fun, tummy ache, still sick, too long
3-grams	an amazing boyfriend, was fucking hilarious, my american idol	stuck in traffic, could n't find, had swine flu
4-grams	let 's party tomorrow, let out early today, yuu da fukin best	miss my long hair, outlook not so good, still not feeling well

Table 8. Examples of resulting sentiment lexicon for unigrams and different n-grams.

Each word/phrase has positive and negative scores. To combine them into a single sentiment score, we use the following formula:

$$\text{senti\_score}(w) = \text{pos\_score}(w) - \text{neg\_score}(w)$$

In the final sentiment lexicon, we obtained 648,715 positive and 617,877 negative words/phrases. The distribution of words/phrases in the final lexicon is shown in Figure 6 and Figure 7 for positive and negative categories, respectively.

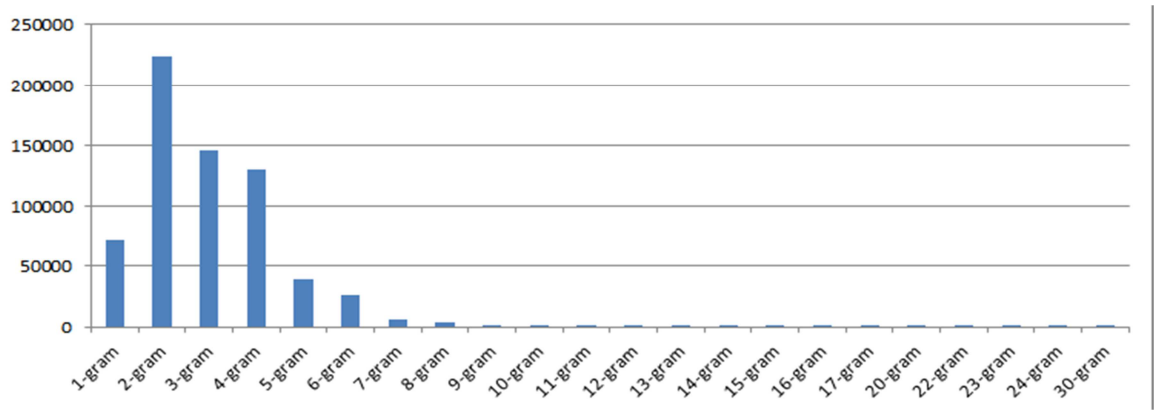

Figure 6. Distribution of positive words/phrases.

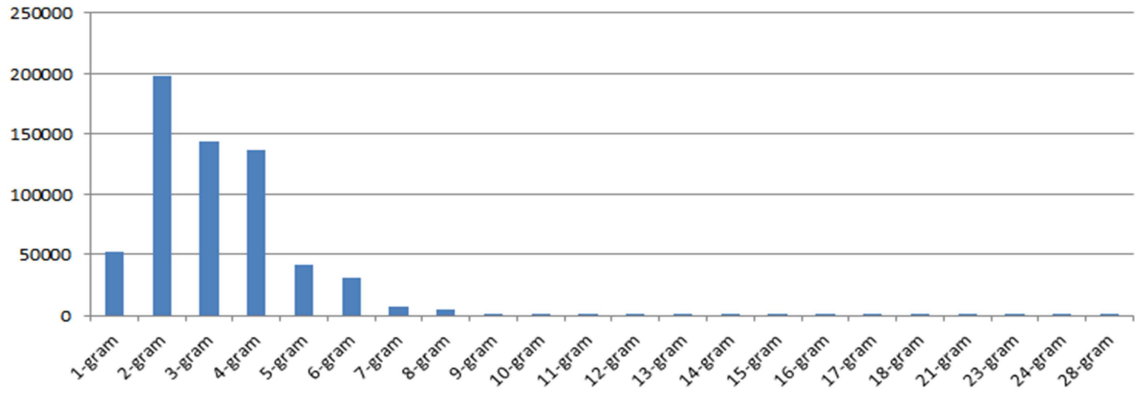

Figure 7. Distribution of negative words/phrases.

4.4. Discussion

As can seen in both Figure 6 and Figure 7, 2-gram, 3-gram, and 4-gram phrases are the major type of phrases in the resulting lexicon. This suggests that Twitter users usually form phrases from 2, 3, or 4 individual words to express their opinions.

In the next chapter, we will run an experiment to evaluate the quality of the lexicons constructed by the graph-based and Labeled TNG-based approaches.

Chapter 5: Evaluation of Constructed Sentiment Lexicons

In this chapter, we will conduct an experiment to evaluate the quality of the sentiment lexicons constructed in Chapter 3 and Chapter 4.

5.1. Lexicon-Based Sentiment Classifier

In order to evaluate the quality of sentiment lexicons built by the graph-based and Labeled TNG-based methods, we use a lexicon-based sentiment classifier which searches for sentiment words/phrases in a tweet and sum up their sentiment scores. If the total score is larger than zero or less than zero, the tweet is assigned the positive or negative label, respectively. The tweet is classified as neutral if the total score is zero, i.e., no sentiment words/phrases are found.

In addition, the resulting lexicons built by both the graph-based and Labeled TNG-based approaches are augmented with emoticons with appropriate scores. A prefix trie T is created for each sentiment lexicon for fast lookup. Algorithm 6 shows how the prefix trie is used in the sentiment classifier.

Input: tweet t, prefix tree p_trie

Output: sentiment label

1. Assign total_score := 0

2. Tokenize the input tweet word_list$_t$:= twokenize(t)
3. For each word $w_i \in$ word_list$_t$:
    a. Assign it to the current phrase p := w_i.
    b. Retrieve a word/phrase u from p_trie that approximately matches p.
    c. If u is p, then augment the phrase p with the next word w_{i+1}, i.e. p := <p, w_{i+1}>. Otherwise, accumulate the sentiment score of phrase p into total_score and reset p with the next word.
4. Output the sentiment label according to the sign of total_score.

Algorithm 6. Lexicon-based sentiment classifier.

5.2. Experiment

We ran the sentiment classifier with the testing dataset from the Stanford Twitter corpus. This testing dataset contains 359 tweets in which 177 tweets are negative and 182 tweets are positive. The testing tweets are labeled manually and are about various topics: company, location, product, etc. [3]. The accuracies obtained by applying Algorithm 6 with sentiment lexicons constructed by both lexicon builders are reported in Figure 8.

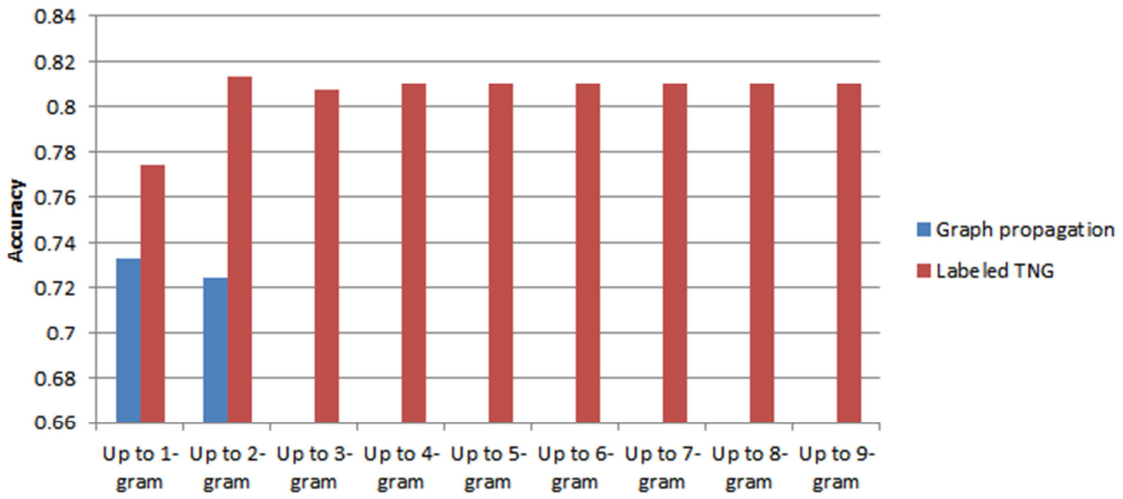

Figure 8. Accuracy of the lexicon-based sentiment classifier for lexicons constructed by graph propagation method and Labeled TNG.

Lexicon	**Graph-based**	**Labeled TNG-based**
Number of training tweets	384,397	1,600,000
Number of positive words/phrases	2,411	648,715
Number of negative words/phrases	1,018	617,877
Total words/phrases	3,429	1,266,592

Table 9. Summary information for each sentiment lexicon.

Table 9 summarizes the number of unique positive and negative words/phrases in lexicons built by two methods, graph propagation and Labeled TNG. We also compared the quality of lexicons constructed by these two

methods with a baseline lexicon obtained from Twitrratr[5] in 2009. This lexicon contains 174 positive and 185 negative unigram words. A unigram search method uses Twitrratr lexicon to count the number of positive and negative single words appearing in a tweet to assign appropriate sentiment label. This method works similarly to our baseline sentiment classifier when the lexicon contains only polarity unigrams and the score of each unigram is either -1 or +1. Table 10 shows the accuracies of sentiment classifiers using three lexicons (where all words and phrases are included).

Lexicon	**Graph-based**	**Labeled TNG-based**	**Unigram Search**
Accuracy	72.42%	81.06%	65.20%

Table 10. Accuracies of the sentiment classifier for each lexicon.

5.3. Results and Discussion

The reported accuracies have confirmed the high quality of our constructed sentiment lexicons. Both sentiment lexicons constructed by graph propagation and Labeled TNG methods provide better quality than the baseline lexicon. The sentiment lexicon built with Labeled TNG provides higher accuracy than the one constructed using graph-based approach. This can be explained by the fact that Labeled TNG is trained with the whole training dataset whereas the graph-based method only uses a part of it.

[5]http://twitrratr.com

The high accuracy of the sentiment lexicon constructed using Labeled TNG is obtained thanks to the use of n-gram words whereas using bigram words from the lexicon built by the graph-based approach seems to hurt the accuracy. This indicates that Labeled TNG is capable of extracting more meaningful phrases than the graph-based method, which natively forms bigram phrases from two consecutive words.

As we mentioned earlier, word order plays a very important role in sentiment analysis. By capturing n-gram words, we are able to capture the semantics of a sentence better and, therefore, gain better accuracy in sentiment extraction.

5.4. Future Work

The results obtained in the experiment are very promising. However, there are still some issues to be solved in future work:

- Due to the limited computing resources, we were not able to create sentiment lexicons using the graph-based approach with the whole training dataset. This issue can be solved in the future by using a larger Hadoop and HBase cluster.
- The current implementation of Labeled TNG only runs in a single machine, and hence, does not scale with the training dataset and the number of topics (for general topic extraction). There have been some efforts to scale LDA algorithm using the MapReduce framework [30, 32]. In the future work, we will incorporate these ideas into our lexicon builder because LDA and Labeled TNG share similar mathematical models.

- Although Labeled TNG is able to handle training tweets with multiple labels, the obtained sentiment scores do not reflect the sentiment strength because the scores are computed based on word frequencies. For instance, the phrase “extremely happy” has a lower score than the word “happy” due to the fact that “happy” is used more frequently in positive tweets than “extremely happy”. This issue can be solved by modifying Labeled TNG so that it can be trained with tweets marked with continuous sentiment scores rather than just discrete labels “positive” and “negative”. This solution is similar to supervised LDA proposed by Blei et al. [31]. However, similar to the original LDA algorithm, supervised LDA relies on bag-of-words assumption where word order is ignored.

References

[1] Pang, B., & Lee, L. 2008. Opinion mining and sentiment analysis. Foundations and Trends in Information Retrieval: Vol. 2: No 1–2, pp 1-135.

[2] Twitter blog. Twitter turns six. http://blog.twitter.com/2012/03/twitter-turns-six.html.

[3] Go, A., Bhayani, & R., Huang, L. 2009. Twitter sentiment classification using distant supervision. Technical report, Stanford.

[4] Zhang, L., Ghosh, R., Dekhil, M., Hsu, M., & Liu, B. 2011. Combining lexicon-based and learning-based methods for Twitter sentiment analysis. Technical report, HP Laboratories.

[5] Weng, J., Lim, E., Jiang, J., & He, Q. 2010. TwitterRank: Finding topic-sensitive influential twitterers. In: Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM '10). ACM, New York, NY, USA. pp. 261--270.

[6] Finin,T., Murnane, W., Karandikar, A., Keller, N., Martineau, J., & Dredze, M.. 2010. Annotating named entities in Twitter data with crowdsourcing. In: Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and

Language Data with Amazon's Mechanical Turk (CSLDAMT '10). Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 80-88.

[7] SentiWordNet – A lexical resource for opinion mining. http://sentiwordnet.isti.cnr.it/.

[8] Taboada, M., Brooke, J., Tofiloski, M., Voll, K., & Stede, M. 2011. Lexicon-based methods for sentiment analysis. Comput. Linguist. 37, (2): 267--307.

[9] Hu, M., & Liu, B. 2004. Mining and summarizing customer reviews. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '04). ACM, New York, NY, USA. pp. 168--177.

[10] Kim, S., & Hovy, E. 2004. Determining the sentiment of opinions. In: Proceedings of the 20th International Conference on Computational Linguistics (COLING '04). Association for Computational Linguistics, Stroudsburg, PA, USA.

[11] Ding, X., Liu, B., & Yu, P.S. 2008. A holistic lexicon-based approach to opinion mining. In: Proceedings of the International Conference on Web Search and Web Data Mining (WSDM '08). ACM, New York, NY, USA. pp. 231-240.

[12] Velikovich, L., Blair-Goldensohn, S., Hannan, K., & McDonald, R. 2010. The viability of web-derived polarity lexicons. In: Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (HLT '10). Association for Computational Linguistics, Stroudsburg, PA, USA. pp. 777--785.

[13] Pang, B., Lee, L., & Vaithyanathan, S. 2002. Thumbs up? Sentiment classification using machine learning techniques. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 79--86.

[14] Agarwal, A., Xie, B., Vovsha, I., Rambow, O., & Passonneau, R. 2011. Sentiment analysis of Twitter data. ACL Workshop on Language in Social Media (LSM '11). pp. 30--38.

[15] Fellbaum, C. 1998. Wordnet, an electronic lexical database. Cambridge, MA, MIT Press.

[16] Gimpel, K., Schneider, N., O'Connor, B., Das, D., Mills, D., Eisenstein, J., Heilman, M., Yogatama, D., Flanigan, J., & Smith, N.A. Part-of-speech tagging for Twitter: Annotation, features, and experiments. 2011. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2 (HLT '11), Vol. 2. Association for Computational Linguistics, Stroudsburg, PA, USA. pp. 42-47.

[17] Elsayed, T., Lin, J., & Oard, D.W. 2008. Pairwise document similarity in large collections with MapReduce. In: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers (HLT-Short '08). Association for Computational Linguistics, Stroudsburg, PA, USA. pp. 265--268.

[18] Apache Hadoop – An open-source software for reliable, scalable, distributed computing. http://hadoop.apache.org/.

[19] Dean, J., & Ghemawat, S. 2008. MapReduce: Simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), pp. 107--113.

[20] Apache HBase – a Hadoop database, http://hbase.apache.org/.

[21] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., & Gruber R.E. 2006. Bigtable: A distributed storage system for structured data. In: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7 (OSDI '06), Vol. 7. USENIX Association, Berkeley, CA, USA. pp.15--15.

[22] Hofmann, T. 1999. Probabilistic latent semantic indexing. In: Proceedings of the International. Conference on Research and Development in Information Retrieval (SIGIR). pp. 50—57.

[23] Blei, D.M., Ng, A.Y., & Jordan, M.I. 2003. Latent Dirichlet Allocation. J. Mach. Learn. Res., 3:993--1022, ISSN 1533-7928.

[24] Blei, D.M., & Lafferty, J.D. 2006. Dynamic topic models. In: Proceedings of the 23rd International Conference on Machine Learning (ICML), pp. 113—120.

[25] Hoffman, M.D., Blei, D.M., & Bach, F. 2010. Online learning for latent Dirichlet allocation. Advances in Neural Information Processing Systems (NIPS), 23:856—864.

[26] Ritter, A., Cherry, C., & Dolan, B. 2010. Unsupervised modeling of Twitter conversations. In: Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics

(HLT '10). Association for Computational Linguistics, Stroudsburg, PA, USA. pp.172--180.

[27] Ramage, D., Hall, D., Nallapati, R., & Manning, C.D. 2009. Labeled LDA: A supervised topic model for credit attribution in multi-labeled corpora. In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1. Association for Computational Linguistics, Stroudsburg, PA, USA. pp. 248--256.

[28] Wang, X., McCallum, A., & Wei, X. 2007. Topical N-grams: Phrase and topic discovery, with an application to information Retrieval. In: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining (ICDM '07). IEEE Computer Society, Washington, DC, USA. pp. 697--702.

[29] Griffiths, T.L., & Steyvers, M. 2004. Finding scientific topics. PNAS, 1: 5228--5235.

[30] Smola, A. & Narayanamurthy, S. 2010. An architecture for parallel topic models. Proc. VLDB Endow. 3, 1-2 (September 2010): 703--710.

[31] Blei, D., & McAuliffe. J. 2008. Supervised topic models. Neural Information Processing Systems 21: 121--128.

[32] Zhai, K., Boyd-Graber, J., Asadi, N., & Alkhouja, M. L. 2012. Mr. LDA: a flexible large scale topic modeling package using variational inference in MapReduce. In Proceedings of the 21st international conference on World Wide Web (WWW '12). ACM, New York, NY, USA, 879-888.