Towards Building Large-Scale Distributed Systems for Twitter Sentiment Analysis

Vinh Ngoc Khuc, Chaitanya Shivade, Rajiv Ramnath, Jay Ramanathan
Department of Computer Science and Engineering
The Ohio State University
{khuc, shivade, ramnath, jayram}@cse.ohio-state.edu

ABSTRACT

In recent years, social networks have become very popular. Twitter, a micro-blogging service, is estimated to have about 200 million registered users and these users create approximately 65 million tweets a day. Twitter users usually show their opinion about topics of their interest. The challenge is that each tweet is limited in 140 characters, and is hence very short. It may contain slang and misspelled words. Thus, it is difficult to apply traditional NLP techniques which are designed for working with formal languages, into Twitter domain. Another challenge is that the total volume of tweets is extremely high, and it takes a long time to process. In this paper, we describe a large-scale distributed system for real-time Twitter sentiment analysis. Our system consists of two components: a lexicon builder and a sentiment classifier. These two components are capable of running on a large-scale distributed system since they are implemented using a MapReduce framework and a distributed database model. Thus, our lexicon builder and sentiment classifier are scalable with the number of machines and the size of data. The experiments also show that our lexicon has a good quality in opinion extraction, and the accuracy of the sentiment classifier can be improved by combining the lexicon with a machine learning technique.

Categories and Subject Descriptors

I.2.7 [**Natural Language Processing**]: Text analysis – *Opinion mining.*
H.3.4 [**Systems and Software**]: Distributed systems – *MapReduce, Hadoop, Mahout, HBase*

General Terms

Algorithms, Performance, Design, Experimentation.

Keywords

Twitter, Social Networks, Opinion Mining, MapReduce

1. INTRODUCTION

With the explosion of social networks, Twitter has become a popular micro-blogging website. It is estimated that Twitter users create approximately 50 million posts a day [13]. These users post their status messages, which are called "tweets", to express opinions on a variety of topics. These messages can be (their) thoughts about current political issues, their complaints, or satisfaction with a product they have just bought. Many companies have started collecting tweets in order to summarize public sentiment about their product.

However, analyzing the sentiment of Twitter posts has challenges. The first problem is that tweets are short and may contain misspelled words; hence, traditional NLP techniques [10] perform poorly when applied to Twitter data. The second problem is that customers are requiring that sentiment analysis has to be done in a real-time fashion; therefore, there is a need to develop a classifier that scales well. The last challenge is that classification results can be incorrect and the sentiment classifier must have a capability of updating its knowledge so that the accuracy can be improved in the future. Further, and in addition to gauging sentiment, companies also wish to perform location-based and temporal analyses of tweets with respect to their topics of interest. This involves introducing intermediate layers that leverage GIS services, translation services and temporal analyses. The need of the hour is to address all these issues and at the same time maintain real-time delivery of insights. However, standalone systems fall short of the computing power necessary for scale. In this paper, we describe a distributed computing framework aimed at addressing the problem of scalable, real-time analysis of tweets. Although our analysis is limited to that of gauging sentiment, the scalable nature of our framework will make it easy to incorporate intermediate layers of analyses.

In our work, sentiment analysis can be performed using a list of opinion words known as an "opinion lexicon". Opinion words are words that contain positive or negative sentiments such as "great" and "bad". By searching for opinion words in a sentence, we can assign the sentence one of three labels: positive, negative, or neutral. This method is called lexicon-based approach [8], [16]. However, Twitter posts have the following special characteristics. Firstly, people use emoticons and punctuation characters to express their sentiment in tweets, for example, "Back when I worked for Nike we had one fav word : JUST DO IT! :)". Secondly, Twitter users abbreviate words because tweets are limited to 140 characters, for example, "@sketchbug Lebron is a hometown hero to me, lol I love the Lakers but let's go Cavs, lol". Lastly, degrees of sentiment are often expressed by repeating characters, for example, "This is sooo goooddd!!!". Further, these repeated characters result in misspellings. Thus, if traditional opinion lexicon such as SentiWordNet [15] is used for analyzing sentiment in tweets, we would not assign this tweet the correct (positive) sentiment. The tweet would instead be considered as neutral since the words "sooo" and "goooddd" are simply not present in the general opinion lexicon. Such sentiment words are

SAC'12, March 25-29, 2012, Riva del Garda, Italy.

commonly used in Twitter and are not included in lexicons since these lexicons are built for analyzing sentiment in formal English documents such as news articles, customer reviews, etc. In fact, to the best of our knowledge, there is no opinion lexicon constructed specifically for the Twitter domain. Also, manually adding sentiment words for Twitter data consumes a lot of time. Thus, we propose a method to automatically construct a large lexicon for Twitter sentiment analysis using Hadoop [2], an open source MapReduce framework [7] and HBase [3], a Bigtable-like database [6]. Since our method for building Twitter opinion lexicon is implemented using the MapReduce framework, it can scale well simply by increasing the number of computers, all of which may be built using commodity hardware. Also, HBase tables are used for backing MapReduce jobs. The whole graph for constructing the lexicon is stored in an HBase table which is a distributed storage for structured data, and hence, the graph is able to scale up well.

Besides lexicon-based approaches, machine-learning techniques are also used to performance sentiment analysis [12], [14]. Machine-learning based classifiers are trained on a collection of documents associated with three labels: positive, negative and neutral. The classifier is then used to predict sentiment of future documents. Although this approach works well with test data in the same domain on which the classifier is trained, it performs poorly in a different domain [5]. Twitter data is very generic since people express their thoughts about a varied set of topics. Therefore, using machine-learning techniques alone for Twitter sentiment analysis does not work well for a wide range of topics. In this paper, we propose a method to combine the lexicon-based and machine-learning based approaches to achieve higher accuracy than the lexicon-based approach. Our machine learning algorithm uses Online Logistic Regression algorithm from the Mahout framework, a machine learning and data mining library built on top of Hadoop [4]. This algorithm is chosen because of its fast training time and its ability to adjust the trained model with new data.

2. RELATED WORK

This section briefly summarizes related work on sentiment analysis/opinion mining.

The lexicon-based approach has been used to perform opinion mining in documents or sentences by searching for polarity words [8], [16]. This approach relies primarily on an opinion lexicon which is a dictionary of positive and negative words. If a sentence or a document does not include any polarity words, that sentence or document will be classified as neutral. Existing lexicons have been constructed from news articles, which do not contain slang, misspelled words, or emoticons. Therefore, we have adapted the algorithm proposed by Velikovich et al [17], which is used for deriving lexicon from Web data, in our lexicon builder to construct an opinion lexicon for Twitter using a small collection of emoticons as seed words.

Machine-learning based approach has been used by Pang et al to perform opinion mining in movie reviews [14]. The authors use supervised learning techniques Naïve Bayes, Maximum Entropy and Support Vector Machines (SVM). They report that SVM gain the best performance compared to the remaining learning algorithms. Go et al applied the same classifiers for Twitter data using distant supervision and reported similar results [12]. However, their classifiers were only tested with positive and negative classes. Our task is more difficult than theirs since we also consider the neutral class.

Some researchers have tried to combine lexicon-based and machine-learning based approaches. Zhang et al propose an entity-level method for Twitter sentiment analysis [19]. They create training data automatically by identifying opinionated tweets which contain words from an opinion lexicon. Their conducted experiments show that augmented training data improves performance of the SVM classifier. Another way to utilize both opinion lexicons and machine learning methods is the work of Agarwal et al [1]. They proposed a way to construct senti-features, which are derived from Part-of-Speech (POS) tags and prior polarity of opinion words.

Our classifier works with three sentiment classes: positive, negative, and neutral. We use Online Logistic Regression algorithm from the Mahout framework, due to its flexibility as compared to SVM. Note that although Logistic Regression cannot achieve the same performance as SVM, its accuracy is still very competitive to SVM [18]. Because misclassification errors cannot be avoided in the real world, there is a need to re-train the classifier using new data and corrections from users. Available SVM classifiers do not support updating the trained model; hence, they have to be re-trained from the beginning with the original training data plus the new data. With the amount of Twitter data being huge, this strategy of training is not appropriate for a system that must operate in the real-world, in real-time.

We conclude this section by summarizing our contributions:

1) We adapt the original algorithm [17] (which is designed for deriving lexicon from Web data) into Twitter domain. Also, and unlike the original algorithm, we use a Twitter part-of-speech (POS) tagger [11] to remove non-sentiment words from the training tweets.
2) We create the first automatically built opinion lexicon for Twitter sentiment analysis. Moreover, our lexicon builder is capable of scaling with data size and the number of machines.
3) We suggest a way to combine lexicon-based and machine learning-based approaches to improve the accuracy of the sentiment analyzer.
4) We show that our sentiment classifiers scale well with the volume of data and the number of machines.

3. SYSTEM ARCHITECTURE

3.1 Lexicon Builder

Our lexicon builder is adapted from the original work of Velikovich et al [17]. The lexicon builder constructs Twitter polarity lexicon from a list of positive and negative seed words that are emotional icons (emoticons) such as ":)", ":D", ":(", ":-(", etc. In the original algorithm, the authors first compute the co-occurrences for each pair of words/phrases from a collection of Web documents. A cosine similarity score between two arbitrary words/phrases is derived from their co-occurrences. A word graph is then constructed with each node representing a single word or a phrase. Two nodes are connected by an edge with a weight that equals the cosine similarity score between two words/phrases represented by these nodes. Edges having weights below a certain threshold are discarded. Sentiment scores are propagated from seed words, which are initially positive and negative words, into other nodes.

In our lexicon builder, tweets are first normalized. The reason is that Twitter users frequently express their sentiments by repeating characters in words, such as "soooo", "goooddddd", etc. Assuming that the words "goooddddd" and "goooodd" have the same sentiment, we map them into the same word "goodd" where every "ooo" and "ddd" are replaced by "oo" and "dd" respectively. We argue that this way of mapping is useful since even if in the training data, there does not exist the word "gooddd", its mapping word is still included in the lexicon (this normalization technique is similar to the technique used by Go et al in [12]). We utilize a Twitter POS tagger[1] created by Gimpel et al [11] to extract (only) nouns, adjectives, adverbs, verbs, interjections, abbreviations, hashtags, and emoticons from tweets since only these words can contain sentiment. This preprocessing step is done so as to reduce the total nodes in the graph.

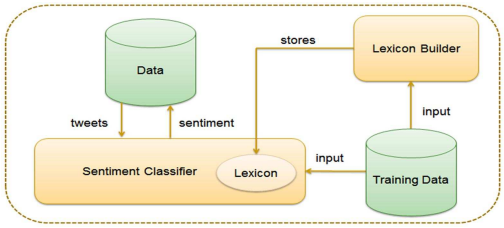

Figure 1. Architecture of the large-scale distributed system for real-time Twitter sentiment analysis.

3.1.1 Generating co-occurrence matrix

A MapReduce job is used for computing co-occurrences between words/phrases in the training tweets. Phrases are formed by connecting continuous words. In our experiment, we only used bi-gram phrases. The co-occurrence matrix is stored in an HBase table described in Table 1, where k_{ij} is the number of occurrences between w_i and w_j.

Table 1. Co-occurrence table

key	column family "cooccurrence"		
w_1	w_1:k_{11}	…	w_n:k_{1n}
…	…	…	...
w_n	w_1:k_{n1}	…	w_n:k_{nn}

3.1.2 Computing cosine similarity and creating the graph of words

The cosine similarity between two words W_i and W_j is computed by dividing the inner product of two vectors w_i and w_j with coordinates $(k_{i1}, k_{i2}, \ldots, k_{in})$, $(k_{j1}, k_{j2}, \ldots, k_{jn})$ respectively, by the product of their lengths. In order to avoid re-computing vector lengths, all vectors w_i are first normalized to unit vectors w'_i, then the cosine similarity values are calculated as the pairwise dot product between the unit vectors $w'_i = (k'_{i1}, k'_{i2}, \ldots, k'_{in})$:

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

A MapReduce algorithm for computing pairwise inner product of a large collection of vectors has been proposed by Elsayed et al [9]. The idea is that the index c in formula (1) contributes to the right hand side sum, if and only if both coordinates k'_{ic} and k'_{jc} are not equal to zero. If we denote as C_i the set of indices c for which the coordinates k'_{ic} of vector w'_i are different from zero, then (1) can be rewritten as:

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

Vector length normalization and the set of indices C_i are calculated in one MapReduce job as described in Algorithm 1a.

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 $norm_{ic} = k_{ic}/len_i$.

Reducers: Do nothing.

Algorithm 1a. Vector normalization and non-zero coordinate indexing

Another MapReduce job is executed for calculating cosine similarity scores and creating the graph table as described in Algorithm 1b. The output graph table has the schema with a column family "weight" (similar to the co-occurrence table).

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 $cosine\_sim_c = norm_{uc} * norm_{vc}$

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

Algorithm 1b. Cosine similarity calculation.

3.1.3 Removing edges with low weights

The purpose of this step is to keep only edges with top weights so that we can avoid non-sentiment words and improve the speed of sentiment score propagation. An edge (w_i, w_j) is kept if it is one of the TOP_N highest weighted edges adjacent to nodes w_i and w_j. The MapReduce algorithm for removing low weighted edges is described in Algorithm 2 below.

Mappers: For each key w_i in the graph table,

Calculate a list L_1 of 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, ∅>

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

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

Algorithm 2. Removing edges with low weights

3.1.4 Sentiment score propagation

This is the main algorithm which propagates sentiment scores from seed nodes into other nodes which are located within a distance D from them. Score propagation is implemented with the help of an HBase table named "propagate". In this table, nodes reachable from a seed node are stored as qualifiers in a column family named "visited" in the row with the key equal to the seed node's id. The MapReduce implementation of sentiment score propagation is described in Algorithm 3.

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

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:

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} which 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}>>.

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 propagate table entry <w_{seed_i}, <visited: w_{iv}, ‘’>> to mark w_{iv} as visited from w_{seed_i}.

Algorithm 3. Sentiment score propagation

3.1.5 Sentiment score calculation

After the sentiment score propagation step is completed, each node w_i in the graph will be assigned two types of sentiment scores: $score_i^+$ - sum of scores propagated from positive seed nodes, $score_i^-$ - sum of scores propagated from negative seed nodes. The final sentiment score of node w_i is computed based on $score_i^+$ and $score_i^-$ with the following formula [17]:

$$score_i = score_i^+ - \beta * score_i^-, \text{ where } \beta = \frac{\sum_i score_i^+}{\sum_i score_i^-}$$

This step is executed with three MapReduce jobs. The first job is launched for calculating $score_i^+$ and $score_i^-$ of each node. The second one calculates β. The last one computes $score_i$.

3.2 Sentiment Classifier

3.2.1 Lexicon-based approach

In this approach, every tweet is first chunked into sentences. The total scores of a sentence are computed as the sum of scores associated with sentiment words/phrases in it. The tweet sentiment is positive, negative, or neutral if the total sentiment score of all sentences is positive, negative, and zero respectively. For each sentence, we extract sentiments based on sentence types. Although it is similar to the work of Zhang et al [19], we only consider the following sentence types:

a) **WH question**: a sentence is a WH-question if it begins with “when”, “who”, etc. then followed by an auxiliary verb such as “do”, “did”, etc. and ends with *one* question mark ‘?’.
b) **Exclamatory sentence:** a sentence finishes with a character sequence containing exclamation marks ‘!’.
c) **Other:** a sentence that is not in either type a or type b.

A sentence of type a is always classified as neutral. For a sentence of type b, the sentiment score is computed as $\gamma * \sqrt{l} * \sigma$, where σ is sum of sentiment scores of all sentiment words/phrases in the sentence, l is the number of characters in the last token which contains the exclamation mark, and γ is a constant larger than 1. The intuition is that if a sentence in a tweet ends with exclamation marks, it should have a higher weight than others. The sentiment score of a sentence of type c is calculated as the sum of scores of its sentiment words/phrases. At the word level, we use a POS tagger for Twitter [11] to tag words in sentences. Only nouns, adjectives, adverbs, verbs, interjections, emoticons, abbreviations and hashtags are considered. In addition, if a word/phrase is preceded by a negation word such as “not”, the sign of its sentiment score is changed.

3.2.2 Combining lexicon-based and machine learning-based approaches

The lexicon-based approach relies primarily on searching for opinion words in the lexicon. However, if none of the words of the tweet are present in the lexicon, the tweet is classified as neutral. It can be a misclassification error if the lexicon does not cover the sentiment word included in the tweet. In order to address this issue, we propose a second classifier, which uses the sentiment lexicon together with a machine learning technique. For the machine learning algorithm, we choose the Adaptive Logistics Regression algorithm from Mahout library [4]. For a tweet in the training data, we first identify all the sentiment words in it. These identified sentiment words are then used to compute the total score for the tweet using the lexicon-based approach. We limit our implementation to the use of binary features since it is shown to achieve good accuracy in opinion mining [14]. The total sentiment score is used as another feature. In other words, even if no sentiment word found in a tweet (in that case, the total sentiment score is 0, and hence the feature represented by it is considered as absent), the classifier can still classify it by utilizing the remaining features.

In our system, the sentiment classifier component is executed as a MapReduce job, where each mapper runs a sentiment classifier instance. Note that we assume tweets are independent with each other, so it is not necessary to run reducers for aggregation.

4. EXPERIMENTS

4.1 Lexicon Builder

The Twitter dataset[2] for constructing the lexicon includes only tweets having smileys “:)”, etc. and frownies “:(“, etc. Since tweets are usually written in spontaneous ways, simply separating words using white spaces does not work well. For example, in the following tweet “I lovee iPad:)”, the emoticon :) is not separated from the word iPad by white space characters. In our experiment, a Twitter tokenizer tool called Twokenizer[3] is applied for each tweet before POS tagging. The positive and negative seed words are only smileys and frownies respectively. Only words/phrases with absolute sentiment score above 1.0 are saved into the final lexicon.

We choose the following parameter values: window size T = 6, threshold α = 0.01, TOP_N = 100, and distance D = 4. We use Hadoop 0.20.2 and HBase 0.90.3. Our cluster consists of 5 machines running on Amazon EC2. Each machine is a medium instance with 1.7 GB of memory and 2 virtual cores with 2.5 EC2 Compute Units each. Two mappers and two reducers are executed on each machine. We measure the running time of our lexicon builder with different datasets containing 100000, 200000, and

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

300000 tweets respectively. The number of words/phrases extracted from these datasets is reported in Figure 2.

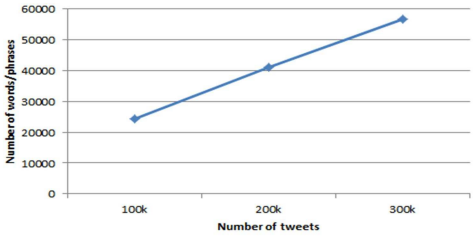

Figure 2. Number of words/phrases by number of tweets.

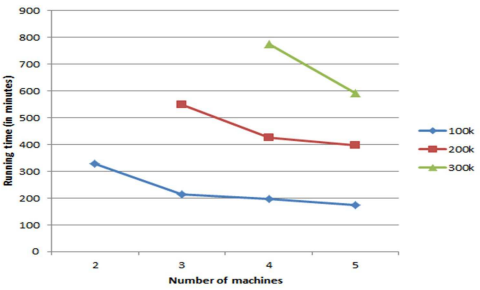

Figure 3. Running time of the lexicon builder with different number of machines and different amount of training tweets.

The overall performance of our lexicon builder is shown in Figure 3. Due to many OutOfMemory errors when training large data with small number of machines, we do not report the running time of the lexicon builder for 200000 tweets in a cluster of 2 machines, and 300000 tweets in a cluster of less than 4 machines. The results show that the time for training the lexicon decreases when we increase the number of machines. In particular, for 100000 tweets, the training time is decreased by 35% when increasing from 2 to 3 machines, and 40% for 2 to 4 machines, 47% for 2 to 5 machines. For 300000 tweets, we are able to decrease the running time by 23% when moving from a cluster with 4 machines to a larger one with 5 machines. However, the training speed is not proportional to the number of machines due to the latency of IO performance of HBase tables with default configurations (i.e. without Bloom filter, compression, etc.).

4.2 Sentiment classifier

4.2.1 Accuracy

The dataset for constructing the final lexicon used in our classifiers included 384397 tweets, in which 232442 tweets containing smileys and 151955 tweets containing frownies. The resulting lexicon contained 2411 positive words/phrases, and 1018 negative words/phrases. Some example sentiment words/phrases are shown in Table 3. For training the Adaptive Logistics Regression classifier, we used a training dataset containing 70000 tweets, in which 60000 sentiment tweets are randomly chosen from the dataset used for training the lexicon and the remaining 10000 tweets are neutral and captured by ourselves. In 60000 sentiment tweets, there are 36282 positive tweets and 23718 negative tweets. We manually annotated 1000 tweets for testing. In the test dataset, there were 584 positive tweets, 270 negative tweets, and 146 neutral tweets. The topic distribution is shown in Table 4.

Table 3. Sentiment words/phrases from the resulting lexicon

Positive	Negative
awesome	damn
long weekend	completely exhausted
pain stop	totally helpless
wohoo	fml

We compared the accuracy of our classifiers against a baseline classifier which only searches for sentiment words in a tweet and classifies it using the total sentiment score. The lexicon used by the baseline classifier is manually constructed and consists of 33 sentiment words associated with scores. The results in Table 5 show that our lexicon provided better quality than the baseline lexicon. The lexicon-and-machine-learning-based classifier, as expected, had higher accuracy than the lexicon-based classifier alone.

Table 4. Topic distribution in the test dataset

Topics	Restaurant	Airline+Hotel	Shopping	Misc.
# Tweets	339	96	230	335

Table 5. Accuracy of the baseline, lexicon-based and lexicon-and-learning-based classifiers

Baseline	Lexicon-based	Lexicon-and-learning-based
55.4%	72.1%	73.7%

4.2.2 Scalability

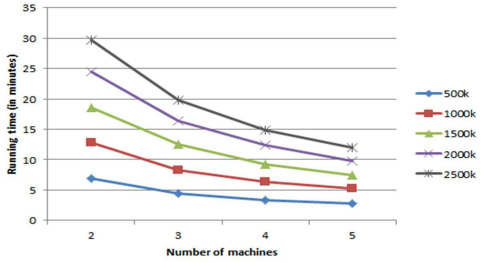

Figure 4. Running time of lexicon-based classifier.

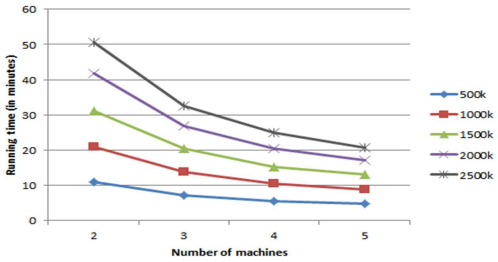

Figure 5. Running time of lexicon-and-learning-based classifier.

We measured the performance of the two sentiment classifiers with different numbers of tweets and with different number of machines. Each run launched a MapReduce job in which each mapper executes an instance of one of the sentiment classifiers. The results in Figures 4 and 5 show the scalability of our sentiment classifier component. In particular, the amount of time for sentiment analysis with different volumes of data decreases

when adding more machines into the cluster. In average, the running time of the lexicon-and-learning-based classifier is 67% higher than the lexicon-based classifier since the former classifier has to do the same task as the latter one in addition to performing its machine learning task.

5. DISCUSSION

Although the resulting lexicon was seen to have a good quality, there are some issues to be addressed in the future work. First, some positive (negative) words/phrases in the lexicon are incorrectly assigned negative (positive) sentiment scores. This is due to the fact that people usually write those words/phrases in a negation manner. For example, the phrase "stop coughing" is frequently used with negation words, such as "cannot stop coughing :(so much for sleeeping". That establishes a sentiment relationship between "stop coughing" and the seed word ":(", and hence, the phrase is assigned a negative score although it is actually positive. Secondly, sentiment words/phrases could have different scores in different topics [8]. For example, in the food domain, "yummy" should have a high positive score while "great song" should be neutral. Third, the word graph cannot be updated with new training data. Since reconstructing the whole graph from scratch is very costly, we will be working on a solution for incrementally updating the lexicon with new training tweets. In addition, the run-time performance of the lexicon builder is not proportional to the size of cluster. In the experiments, we used the default configuration for HBase tables, and that can cause IO performance issues. In the future work, we will be evaluating the performance of our lexicon builder with different HBase configurations. We also would like to focus on improving the accuracy of the lexicon-and-learning-based classifier.

6. CONCLUSION

We have described a distributed system designed for Twitter sentiment analysis. In this system, we propose a method to automatically construct a large sentiment lexicon for Twitter sentiment analysis. Our lexicon builder is implemented based on Hadoop and HBase, and we have demonstrated that it scales with increasing number of machines and amount of training data. We also implemented two classifiers which utilized the Twitter sentiment lexicon built by the lexicon builder and compared them against a baseline classifier. The high accuracy of the two classifiers has proven that our lexicon has a good indicator of sentiment. Our two classifiers have been shown to scale well with the data size and the number of machines. In addition, we introduce a method to combine the sentiment lexicon with a machine learning algorithm. The experiment has shown that our lexicon-and-learning-based classifier can obtain higher accuracy than the lexicon-based classifier which only relies on searching for sentiment words/phrases.

7. ACKNOWLEDGMENTS

This work was supported by the National Science Foundation's Industry & University Cooperative Research at the Centre for Experimental Research in Computer Systems at Georgia Tech Institute and The Ohio State University. Astute Solutions Inc. (a member of the IUCRC) sponsored the project on which the paper is based and provided us with great help and resources in preparing this paper.

8. REFERENCES

[1] Agarwal, A., Xie, B., Vovsha, I., Rambow, O. and Passonneau, R. 2011. Sentiment Analysis of Twitter Data. LSM 2011.

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

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

[4] Apache Mahout – a scalable machine learning library. http://mahout.apache.org/.

[5] Aue, A., and Gamon, M. 2005. Customizing Sentiment Classifiers to New Domains: a Case Study. RANLP 2005.

[6] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C. , Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R. E. 2006. Bigtable: a distributed storage system for structured data. OSDI 2006.

[7] Dean, J. and Ghemawat, S. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008), 107-113.

[8] Ding, X., Liu, B. and Yu, P. S. 2008. A holistic lexicon-based approach to opinion mining. WSDM 2008.

[9] Elsayed, T., Lin, J. and Oard, D. W. 2008. Pairwise document similarity in large collections with MapReduce. HLT-Short 2008.

[10] Finin, T., Murnane, W., Karandikar, A., Keller, N., Martineau, J. and Dredze, M. 2010. Annotating named entities in Twitter data with crowdsourcing. CSLDAMT 2010.

[11] Gimpel, K., Schneider, N., O'Connor, B., Das, D., Mills, D., Eisenstein, J., Heilman, M., Yogatama, D., Flanigan, J. and Smith, N. A. 2011. Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments. ACL 2011.

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

[13] Measuring tweets. http://blog.twitter.com/2010/02/measuring-tweets.html.

[14] Pang, B., Lee, L. and Vaithyanathan, S. 2002. Thumbs up? Sentiment Classification using Machine Learning Techniques. EMNLP 2002.

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

[16] Taboada, M., Brooke, J., Tofiloski, M., Voll, K. and Stede, M. 2011. Lexicon-based methods for sentiment analysis. Comput. Linguist, 2011, 37, 2 267-307.

[17] Velikovich, L., Blair-Goldensohn, S., Hannan, K. and McDonald, R. 2010. The viability of web-derived polarity lexicons. HLT 2010.

[18] Zhang, J., Jin, R., Yang, Y. and Hauptmann, A. 2003. Modified Logistic Regression: An Approximation to SVM and its Applications in Large-Scale Text Categorization. ICML 2003.

[19] Zhang, L., Ghosh, R., Dekhil, M., Hsu, M. and Liu, B. 2011. *Combining Lexicon-based and Learning-based Methods for Twitter Sentiment Analysis*. Technical report, HP Laboratories, 2011.