18th International Conference on Information Fusion
Washington, DC - July 6-9, 2015

A Unified Approach for Domain-Specific Tweet Sentiment Analysis

Patrícia L V Ribeiro and Li Weigang
TransLab, Department of Computer Science
University of Brasilia, Brasilia, DF, Brazil
Email: {patlustosa,weigang}@cic.unb.br

Tiancheng Li
BISITE group, University of Salamanca
37008 Salamanca, Spain
Email: tiancheng.li1985@gmail.com

***Abstract*—Twitter is an online social networking (OSN) service that enables users to send and read short messages called "tweets". As of December 2014, Twitter has more than 500 million users, out of which more than 284 million are active users and about 500 million tweets are posted every day. Tweet sentiment analysis (TSA) identifies a valuable platform for the OSN study which provides insights into the opinion of the public about culture, products and political agendas and thereby is able to predict the trends in specific domains. In order to execute efficient TSA on a particular topic or domain, a TSA approach with unified tool, UnB TSA, is proposed consisting of four steps: tweets collection, refinement (excluding noisy tweets), sentiment lexicon creation and sentiment analysis. As a key part, the lexicon is domain-specific that incorporates expressions whose sentiment varies from one domain to another. Four algorithms including expanding limited hashtags into a larger and more complete set to collect tweets have been implemented. Experiments on the 'iPhone 6' domain which obtains convincing results in all of the four phases, showing the superiority of the domain-specific TSA approach over a generic one.**

I. Introduction

Twitter is an online social networking service where users post status messages called tweets. According to Twitter website[1], there are 271 million monthly active users and 500 million tweets are sent per day. This huge amount of data can be used to provide insights into the opinion of the public in various topics, from political affairs, hot news to commercial products. To be able to extract these information from tweets, applications need to perform sentiment analysis of those messages.

Sentiment analysis is a topic of natural language processing (NLP) in which, the goal is to identify the overall sentiment (or opinion) of a given text or phrase. Sentiment analysis of texts has been acknowledged and investigated intensively within the Tweets context. Most of existing work use a sentiment lexicon, which is a list of sentiment words or phrases. The performance of the algorithms is closely related to the quality of the lexicons. In order to achieve good results on evaluating tweets in a particular domain, it is interesting to create a lexicon that is specific for these messages and related to that domain, so that to incorporate expressions where their individual sentiment varies from one domain to another.

The objective of this paper is to present a unified tool that executes efficient TSA on a particular domain. The user inputs a set of initial terms and/or hashtags related to the intended topic for analysis and the tool outputs a collection of tweets with tagged sentiments. To cover all the work needed, we divided our approach into four steps. The first step deals with collecting tweets related to a specified topic. It is important to allow the collection of a larger set of related tweets with a few inputs. The second step is to select tweets relevant to our application, for instance, English and non-spam tweets. This is necessary due to the amount of spam tweets that exist, especially with regards to trending topics. The third step is to build a domain-specific tweet sentiment lexicon. This is essential to capture specificities of the domain. Finally, the last step is to perform sentiment analysis of these tweets. An overview of the system architecture is shown as Figure 1.

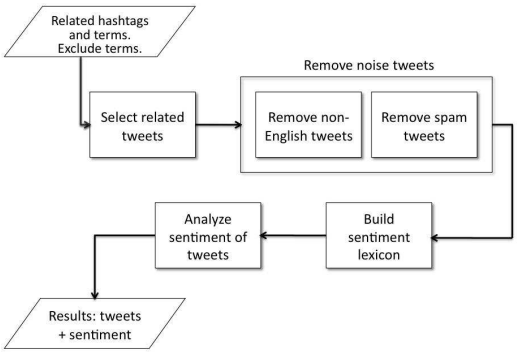

Fig. 1. Overview of the System Architecture

The proposed TSA tool is tested on the 'iPhone 6' domain which has obtained convincing results on all four phases. These results show the superiority of the domain-specific TSA toll over a generic one. The developed tool builds a lexicon in an automated and unsupervised way, which makes it easier to be applied to other domains.

II. Related Work

In this section, we review the related literature and studies concerning the four phases of our TSA tool, respectively.

A. Hasgtag Expansion

Ozidikis *et al.* [1] presented an event detection method in Twitter based on clustering of hashtags by using the semantic

[1]http://www.twitter.com

similarity between them. Carter *et al.* [2] proposed a method for finding related hashtags in a target language, given a hashtag from the source language. Another approach that is similar to our problem is to recommend hashtags. An automatic hashtag suggestion tool that returns a short list of relevant hashtags when given a tweet was proposed by Mazia *et al.* [3]. Kywe *et al.* [4] worked in a similar way, with the benefit of the recommendation being personalized for the user.

In the research of Tsur *et al.* [5] for clustering tweets, they partioned the clustering task into two distinctive tasks: batch clustering of user annotated data and online clustering of a stream of tweets. Rangrej *et al.* [6] focused on clustering tweets based on its content. They compared various document clustering techniques and showed that graph-based approach using affinity propagation performs the best in clustering short text data in the sense of clustering error.

B. Spam Detection

There are several articles related to identifying spam accounts on Twitter, such as Benevenuto *et al.* [7] which collected 54 million user profiles and labeled the users with manual inspection. Based on this data, they ranked features that identify spammers. Stringhini *et al.* [8] created honeypot profiles and analyzed the friend requests and the messages sent to them. Klassen [9] explored attribute reduction and data preprocessing from the aspect of Twitter spam detection using various machine learning algorithms.

Some studies mention spam detection as a pre-processing step for their algorithms ([10], [11] and [12]). There are also some studies that focus on detecting spam tweets instead of spam accounts; the same is done in our approach. A method of identifying spam tweets from a stream source using information about the user along with features of the tweets was proposed by Miler *et al.* [13]. Martinez *et al.* [14] presented a methodology involving two aspects: the detection of spam tweets in isolation and the application of a statistical language analysis to detect spam in trending topics.

C. Lexicon Building

General lexicons are widely used, since they can be applyed to several domains. The disadvantage is that they are usually hard to capture the specificities of the domains, which leads to low recall percentage. Hu and Liu [15] manually compiled a lexicon since 2004 which has about 6.800 words. A large and general polarity lexicon has been developed semi-automatically from the web using a graph propagation algorithm [16].

Intensive research has been devoted into building domain-specific lexicon that are not tweet specific. Tan and Wu [17] proposed an approach to construct domain-oriented sentiment lexicon by using labeled data from other domain. Their work is based on a random walk model by utilizing all relationships among documents and words from both old and new domains. The same problem was solved by using a deep learning approach [18] and integer linear programming [19].

Mohammad *et al.* [20] built two twitter-specific lexicons: NRC Hashtag Sentiment Lexicon and Sentiment140 Lexicon. The first one was built from 78 seed positive and negative hashtags using a big set of tweets and pointwise mutual information. The second one used the same approach, but using Sentiment140 corpus, which is a collection of tweets that contain both positive and negative emoticons. A Twitter-specific lexicon was created by using an approach to supervised feature reduction using n-grams and statistical analysis [21]. The authors augmented this reduced Twitter-specific lexicon with brand-specific terms for brand-related tweets, such as the brand 'Justin Bieber'. This work is domain and tweet specific, but their approach is highly manual and hard to be applied to other domains. Chen *et al.* [22] presented an optimization-based approach to automatically extract sentiment expressions for a given target from a corpus of unlabeled tweets.

D. Sentiment Analysis

With regards to sentiment analysis, there are some comprehensive surveys of the algorithms and applications, such as [23], [24] and [25]. The research of Gonçalves *et al.* [26] compared several sentiment analysis algorithms and developed a new method that combines existing approaches, providing the best coverage results and competitive agreement. A support vector machine classifier was created to detect the sentiment of tweets, using a lexicon and several other features [20]. A supervised machine learning approach has been developed to build a tweet sentiment classifier [27]. Saif *et al.* [28] proposed using semantic features in Twitter sentiment classification and explored three different approaches for incorporating them into the analysis; with replacement, augmentation, and interpolation.

These studies use supervised or semi-supervised approaches for sentiment analysis. They can obtain better results, but at the cost of needing to input labeled data. In contrast, we will develop an unsupervised algorithm, which can be easily applied to any domain.

III. Collection of Related Tweets

The first step consists of collecting tweets related to a specific topic. Instead of a lot of terms and hashtags, the user is only required to provide a small set of initial terms and/or hashtags about the topic and related tweets will be collected by our approach.

In order to extend the initial set of terms and hashtags into a larger and more complete set, we propose an algorithm namely the Hashtag Expansion Algorithm. Based on this larger set, we can search on the Twitter API the tweets that contain these hashtags and then collect tweets related to that topic.

Particularly, we define that two hashtags are related if they are referred to the same topic (e.g.: WorldCup2014 and Brazil2014 talk about the same event), some attributes of that topic (e.g.: iPhone6 and iPhone6Battery) or a closely related topic (e.g.: ecig and eliquid).

The input of the Hashtag Expansion Algorithm consists of three sets: input hashtags, input terms and ignore terms.

Input hashtags provide the seed hashtags for the algorithm, i.e., the hashtags concerning the topic chosen. Input terms are alternative terms related with the topic. Ignore terms are the terms that we want to exclude from the final set of tweets.

For instance, suppose we want to find hashtags about the new Apple product: iPhone 6. The input set could be the hashtag #iphone6, the input terms could be 'iphone 6', 'iphone 6 plus' and the ignore set could include 'iphone2', 'iphone3', 'iphone4', 'iphone5'.

A. Hashtag Expansion Algorithm

The main idea of the algorithm is to find hashtags that contain one or more of the seed hashtags or terms and do not contain any of the ignored terms. The algorithm is developed in order to handle a few misspellings of the hashtag, which occur frequently due to the informal nature of tweets. If a hashtag is similar enough to one of the seed hashtags or terms, it will be considered as related. If a hashtag is more similar to a ignored term than it is to any of the seed hashtags or terms, it will not be considered as a related term.

This similarity is calculated based on an adapted version of the Levenshtein distance and taking into account whether one string is a substring of others. The substring algorithm captures the hashtags that contain one of the initial hashtags or terms as a substring and do not contain any of the ignored terms. This will capture many tweets that talk about some particularity of the original topic.

We use Levenshtein distance with different weights for insertion, deletion and substitution. These weights are customisable in order to achieve the optimal parameters, which give the best results. To implement the idea that one letter changed in a 4-letter word means more than one letter changed in a 12-letter word, a normalized Levenshtein distance is developed, by dividing the distance with the size of the smaller one of the two strings being compared. Mathematically, the normalized distance between two strings a and b is equal to $lev_{a,b}(|a|,|b|)/min(|a|,|b|)$ where $|a|$ is the size of the string a, lev is calculated according the equation 1, α is the insertion cost, θ is the deletion cost and γ is the substitution cost. We establish a threshold on how different the hashtags could be which can still be considered related. This threshold is customizable. This step helps to capture some misspellings that are common on an informal platform as Twitter.

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j) + \alpha \\ lev_{a,b}(i,j-1) + \theta \\ lev_{a,b}(i-1,j-1) + \gamma_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases} \quad (1)$$

In addition to that, we only take into consideration the hashtags that appear in more than a defined percentage of the tweets with our seed hashtags. This threshold is customizable and is proposed with the intention of ignoring the hashtags that appear just a few times in our dataset. We also ignore hashtags with the size smaller than the established threshold, since they mean nothing.

When optimizing an algorithm, the metric is important. If, in the intended application of the hashtag expansion algorithm, we want to minimize false negatives, in order to avoid to leave out any related hashtag, we should optimize the algorithm to maximize the recall percentage. If we want to minimize false positives, in order to avoid to select hashtags that are not related, we should optimize the algorithm to maximize the precision. As we do not want to run a manual verification of the selected hashtags, the best choice is to maximize precision.

In our approach, we use the F-score F_b as the metric where the weight parameter b can be adjusted. As discussed, we want to maximize precision, but we cannot ignore the recall. For that reason, we choose to maximize $F_{0.5}$.

Algorithm 1: Hashtag Expansion Algorithm

Data: SH, ST, IT
Result: $hashtags$
/* SH stands for seed hashtags, ST for seed terms and IT for ignored terms. The algorithm return a collection of related hashtags. */
$hashtags = \{\}$;
$allHashtags =$ getHashtagsThatCooccur$(ST, percentage)$;
/* Return all the hashtags that co-occur in at least percentage of the tweets with one of the seed hashtags. */
for $hashtag \in allHashtags$ **do**
  for $seed \in SH$ **do**
    $distance :=$ NLD$(hashtag, seed)$;
    /* NLD means Normalized Levenshtein Distance */
    $minDist1 :=$ minimum$(distance, minDist1)$;
  end
  for $term \in ST$ **do**
    $distance :=$ NLD$(hashtag, term)$;
    $minDist1 :=$ minimum$(distance, minDist1)$;
  end
  for $term \in IT$ **do**
    $distance :=$ NLD$(hashtag, term)$;
    $minDist2 :=$ minimum$(distance, minDist2)$;
  end
  if $minDist1 < t$ *AND* $minDist1 < minDist2$ **then**
    $hashtags.add(hashtag)$;
  end
end
return $hashtags$;

The code of Hashtag Expansion Algorithm was build in Python 2.7, using a MySQL database with tweets and hashtags. The tweets were captured using the 140dev Streaming Twitter API Framework [2]. This framework is available online, which is written in PHP and stores the tweets in a MySQL database. This database was accessed from our Python code to build the set of related hashtags. All the algorithms presented in this paper use this architecture. Algorithm 1 shows the pseudo-code of the algorithm.

The parameters of Algorithm 1 were optimized to achieve the highest $F_{0.5}$. We used two domains (World Cup 2014 and Ebola) to train the algorithm to get the best parameters. The ground-truth was built manually from a large database of tweets. Then we applied the parameters to the domains iPhone 6 and E-cigarettes to compare the results. Table I shows the

[2]http://140dev.com

results for the domains used to train the parameters and table II shows the results for the domains used to test Algorithm 1. We can see that the results are consistent, since the train set produces better results than the test set and both of them produce a good $F_{0.5}$.

TABLE I
RESULTS WITH TRAIN SET - HASHTAG EXPANSION

Lexicon	Precision	Recall	$F_{0.5}$
#worldCup2014	1.00000	0.65217	0.90361
#ebola	1.00000	0.94737	0.98901
Average	**1.00000**	**0.79977**	**0.94631**

TABLE II
RESULTS WITH TEST SET - HASHTAG EXPANSION

Lexicon	Precision	Recall	$F_{0.5}$
#iphone6	0.76470	0.97500	0.79918
#ecig	0.97142	0.97842	0.97282
Average	**0.86806**	**0.97671**	**0.88600**

IV. REMOVE NOISY TWEETS

In order to select only relevant tweets from the ones collected in the previous step, we need to exclude the noisy tweets. Firstly, we need to exclude non-English tweets, since this tool is made for English texts. Secondly, we need to exclude spam tweets, such as advertisement, requests for followers and automatically generated tweets (e.g. tweets generated by games). For identifying non-English tweets, we used a Python code and the language information reported by Twitter.

In our approach, we choose to focus on detecting spam tweets, as we are dealing with a stream of tweets.

Algorithm 2: Spam Detection Algorithm

```
Data: tweet, parameters
Result: sentiment
/* Return whether the tweet is a spam                    */
if tweet.user.accountAge <= parameters[0] then
    return true;
end
if tweet.user.followersCount <= parameters[1] then
    return true;
end
if tweet.hashtagsCount <= parameters[2] then
    return true;
end
if tweet.spamWordsCount <= parameters[3] then
    return true;
end
if tweet.urlCount <= parameters[4] then
    return true;
end
if tweet.containsUrlBlacklisted() then
    return true;
end
return false;
```

A. Spam Detection Algorithm

Algorithms that focus on identifying spam accounts usually take into consideration features regarding the user account and the history tweets published by that user. Examples of such features are the age of the account, the number of followers and followees and the amount of tweets published by day. Algorithms that focus on identifying spam tweets usually analyse a tweet separate of other tweets from the same user. These algorithms commonly use tweet-specific features, such as the number of hashtags and URLs in that tweet. Sometimes they also use some features of the user, such as the age of the account.

With that in mind, we compare the features mentioned in previous studies that could be applied to our algorithm, identifying the optimal combination that gives the best results. These features were then consolidated in a single algorithm and they are account age in days, number of followers, number of URLs, number of hashtags, number of spam words and whether it contains a blacklisted URL. The number of spam words is a feature very useful to detect spam emails that is also applied for spam tweets. We consolidate a list of spam words that contain words that are usually spam in many applications as well as words that are spam in Twitter, such as 'follow' and 'please'. These words usually indicates a user asking for more followers.

Another feature chosen is the account age in days, because spammer accounts tend to have a short period of life, as they are usually identified as spam after a few days and excluded from Twitter. The number of followers is also a significant feature, as spammer accounts usually have less followers than a regular account. The number of URLs was chosen because spammers usually post tweets with URLs, since tweets have no more than 140 characters, which is usually not enough to disseminate the information. Related to this, another feature is whether the tweet contains a blacklisted URL. We used PhishTank blacklist [3] to make this verification. Finally, the last feature is the number of hashtags, which is important since spammers tends to use a lot of hashtags to appear in searches and trending topics.

A threshold for each of these features was established by a training process intended to maximize F_2. We choose this metric because we want to minimize false negatives. This is due to the fact that the impact of ignoring some no-spam tweet is not critical, as we have a big set of tweets to use, but the impact of selecting a spam tweet and use it to build the lexicon is important.

We use 3.000 tweets collected in the previous phase that are related to iPhone 6 to train Algorithm 2 and the other 500 to test. The result can be seen in table III. The train and test results are very similar, which means that our training is consistent. From the training we observe that the number of hashtags is the most significant feature to identity spam tweets in this domain.

[3] http://www.phishtank.com/

TABLE III
RESULTS WITH TRAIN AND TEST SET - SPAM DETECTION

Lexicon	Precision	Recall	F_2
iPhone 6 - train	0.644249	0.934724	0.857471
iPhone 6 - test	0.631285	0.911290	0.837037

The amount of spam in iPhone 6 tweets was quite surprising. In our manually labeled results for tweets collected in the period 2014/09/08 to 2014/09/18, there are 49.4% of spam tweets. In other domains, such as World Cup 2014 and Ebola, the amount of spam tweets is around 5.5%, showing that the training should be domain-specific. We believe that this difference is because iPhone 6 launch in 2014/09/09 and iPhone 6 is a trending topic during the period collected, which attract spammers.

V. CONSTRUCTION OF SENTIMENT LEXICON

The goal of this step is to build a domain-specific tweet sentiment lexicon. This is necessary to improve the performance of our TSA tool. Since tweets are short and informal texts and contain a lot of abbreviations and slangs, a traditional sentiment lexicon built for analysing formal texts is not suitable. Because of this, it is important to construct a tweet-specific lexicon, so that the peculiarities of tweets can be handled properly.

According to Liu [29], sentiment words vary a lot across different domains, so building a general sentiment-lexicon will not capture the specificities of each domain. For instance, the word 'loud' is positive when referring to a stereo system but it is negative when referring to a refrigerator. To improve the performance, we need to build a lexicon that is specific to the domain.

There are two ways to build a lexicon, supervised and unsupervised. We choose an unsupervised method to build the lexicon, because of the facility to apply the algorithm to other domains. The input required for our algorithm is an unlabeled set of tweets about the source domain, a set of domain-independent positive hashtags and emoticons and a set of domain-independent negative hashtags and emoticons. The domain-independent sets can be reused to different domains.

Our algorithm is based on the graph propagation algorithm proposed by Velikovich [16] which applied a graph propagation algorithm to a phrase graph built from the web. We proposed a different approach to build our graph by using co-occurence and cosine similarity among tweets.

A. Tokenizing

Tokenizing is the process to split a string into desired constituent parts which is a fundamental technique in Natural Language Processing tasks. Having a tokenizer specific for tweets is important, as tweets have many particularities on its text, such as mentions, hashtags, URLs and emoticons.

All mentions are substituted by '@mention' and all URLs are substituted by 'www.url.com'. The emoticons and hashtags do not need to be changed, as they constitute an important factor in the evaluation of tweet's sentiment. Besides the tweet-specific features, our tokenizer also deals with negation and elongation. All words between a negation word and a clause-level punctuation mark have a '_NEG' appended, which means that this token has a meaning opposed to the meaning of the original word. We also deal with double negation, where the second negation neutralizes the first one. Elongation concentrates on repeating a character to indicate a heightened emotion, for instance 'cooooooooool' gives an emphasis to the word 'cool'. It is very common on tweets and other informal texts. Our algorithm deals with elongation by substituting it by the original word appended with '_LONG'.

To illustrate the behaviour of our lexicon, examples of tweets and its tokenized version are shown as follows.

```
Tweet: RT @patlustosa: this is a
typical Twitter tweet :-) #ICWSM
Tokenization: rt, @mention, this, is, a,
typical, twitter, tweet, :-), #icswm

Tweet: I don't love him anymore. Sooooo
#sad :'( o.O
Tokenization: i, don't, love_NEG,
him_NEG, anymore_NEG, so_LONG, #sad,
:'(, o.O

Tweet: I didn't said I wouldn't buy it.
Tokenization: i, didn't, said_NEG, i_neg,
wouldn't, buy, it

Tweet: Let's just compare them #iPhone6
#iPhone6Plus http://t.co/Zj3b3wfp30
Tokenization:let's, just, compare, them,
#iphone6, #iphone6plus, http://url.com
```

B. Building the Graph from Tweets

Graph propagation algorithms rely on graphs that encode meaningful relationships between nodes. They input an undirected weighted graph $G = (V, E)$, where $w_{ij} \in [0, 1]$ is the weight of edge $(v_i, v_j) \in E$. The node set V is the set of candidate tokens for inclusion in the lexicon. The weight w_{ij} of an edge between two nodes should encode the semantic similarity between these nodes. For instance, $v_i = amazing$, $v_j = good$ and $v_k = disappointing$. The semantic similarity between amazing and good is bigger than between amazing and disappointing, so we would expect that $w_{ij} > w_{ik}$. The algorithm that builds the graph from tweets is given in Algorithm 3.

Algorithm 3: Graph Builder Algorithm

Data: $corpus$
Result: $csMatrix$
/* $corpus$ is the set of tweet tokens */
/* Returns a matrix representing the graph */
1 d =buildCooccurenceMatrix($corpus$);
2 $vocab$ =getSortedVocab(d);
3 $csMatrix$ =buildCosineSimilarityMatrix($vocab, d$);
4 return $csMatrix$;

To build a graph that attends these restrictions, we first run the tokenizer in all tweets and consider all tokens found in the tweets as candidate tokens. Denote T as the set of all input tweets and $tok(t)$ as the set of tokens returned by the tokenizer given t as an input. Then, $E = \cup tok(t), \forall t \in T$.

We calculate a matrix with the co-occurence count between all $e \in E$. This matrix is then converted into a cosine similarity matrix called CM. The value of the weight between v_i and v_j is the cosine similarity between these two nodes, i.e. $w_{ij} = CM[i][j]$.

C. Graph Propagation

The graph propagation algorithm is given in Algorithm 4. The algorithm works by computing a polarity value for each node in the graph, which corresponds to tweet tokens. These values are equal to the sum over the max weighted path from every seed word to the nodes. Tokens that are connected to multiple positive seeds by highly weighted paths will be considered as positive tokens. The variable b is used to mantain balance to the values, when there is a difference between the number of positive and negative flows in the graph. T is the max path lenght considered by the algorithm and l is a treshold that defines the minimum polarity a token needs to have to be added to the lexicon.

Algorithm 4: Graph Propagation Algorithm

Data: $G = (V, E), w_{ij} \in [0, 1], P, N, l, T$
Result: $map(V, pol)$
/* Returns a polarity value for each token */
$polPos, polNeg, lexicon = map(V, pol)$;
set $a_{ii} = 1$ for i;
set $a_{ij} = 0$ for $i \neq j$;
for $v_i \in P$ **do**
  $F = \{v_i\}$;
  for t *from* 1 *to* T **do**
    for $(v_k, v_j) \in E$ *such that* $v_k \in F$ **do**
      $a_{ij} = max\{a_{ij}, a_{ik} * w_{kj}\}$;
      $F = F \cup \{v_j\}$;
    end
  end
end
for $v_j \in V$ **do**
  $polPos[j] = sum(a_{oj}), \forall i \in P$;
end
/* Repet steps 1 - 14 using N to compute polNeg */
$b = sum(polPos)/sum(polNeg)$;
for $v_i \in V$ **do**
  $lexicon[i] = polPos[i] - b * posPos[j]$;
  if $lexicon[i] < l$ **then**
    $lexicon[i] = 0$
  end
end
return $lexicon$;

D. Consolidation of the Lexicon

As the output of the graph propagation algorithm, we have sentiment values associated with each node v. This set can contain words that are appended with 'NEG' or 'LONG' as they are possible output tokens from the tokenizer. For giving a final polarity to a word, we should consider the polarities of word, word_NEG and word_LONG, as shown in equation 2.

The final lexicon building algorithm incorporates all the steps presented in this section and is shown in Algorithm 5.

$$final\_pol[v] = pol[v] + 1.5 * pol[v\_LONG] - pol[v\_NEG] \quad (2)$$

Algorithm 5: Lexicon Building Algorithm

Data: $tweets, positiveSeeds, negativeSeeds, iteractions$
Result: $lexicon$
/* Returns the lexicon represented as a map of words and values */
$tokenizedTweets =$ tokenize$(tweets)$;
$graph =$ buildGraph$(tokenizedTweets)$;
$tempLexicon =$ graphPropagation$(graph, positiveSeeds, negativeSeeds, iteractions)$;
$lexicon = new\ Map()$;
for $key, value \in tempLexicon$ **do**
  $lexicon[key] = tempLexicon[key] + 1.5 * tempLexicon[key\_LONG] - tempLexicon[key\_NEG]$;
end
return $lexicon$;

E. Test Case

We run the Lexicon Builder Algorithm with 6.000 tweets related to iPhone 6 and obtained a lexicons with 4.576 entries, in which 2.280 are positive and 2.296 are negative. Table IV shows some examples of positive and negative words in the lexicon.

TABLE IV
EXAMPLES OF POSITIVE AND NEGATIVE WORDS

Positive		Negative	
Word	**Value**	**Word**	**Value**
yay	3.39	#pissedoff	-3.12
lol	2.58	#horrorstory	-3.11
:D	2.55	kidney	-2.50
#soexcited	1.36	#outdatedalready	-1.91
ordering	0.93	#camerafail	-0.75

One interesting discovery is that one of the most negative words related for iPhone 6 is 'kidney'. This might be surprising at first, but it makes sense when analysing a few iPhone 6 tweets. By the time of the launch of iPhone 6, many users were complaining about the price of the cellphone and claiming that they would need to sell a kidney in order to buy one. This is an example of domain-dependent words that are not considered by a general lexicon. A domain-specific positive word is 'ordering', which usually means that the user is ordering one iPhone 6 product.

It is hard to evaluate the quality of a lexicon, because it requires a great manual work and it is difficult to compare with other lexicons. Because of this, sentiment lexicons are usually compared by using a sentiment analysis tool with different lexicons. This is the approach we choose to follow in our comparison given in the next section.

VI. Sentiment Analysis

This step concentrates on analysing the sentiment of the selected tweets. This is important to evaluate the quality of the lexicon built in the third step and to complete the objective of the tool.

The Sentiment Analysis Algorithm is built based on unsupervised learning, which enjoys the facility to apply it to other domains and lexicons, without the need of training and labeled data. The only required inputs are a lexicon and a set of unlabeled tweets.

The idea of the algorithm is to sum the polarity of each tweet token, considering the negation and elongation of words for changing its polarity. If a word is negated, we inverse its polarity and, if a word is elongated, we multiply its polarity by 1.5. The sentiment is derived by the sum s in the following equation 3.

$$sentiment(t) = \begin{cases} positive & \text{if } s(t) > 0 \\ negative & \text{if } s(t) \leq 0 \end{cases} \quad (3)$$

Algorithm 6: Sentiment Analysis Algorithm

```
Data: tweet, lexicon
Result: sentiment
/* Returns the sentiment of a tweet given the
   lexicon                                      */
tokens = tokenizer(tweet);
polarity = 0;
for token ∈ tokens do
    if token contains _NEG then
        polarity := polarity − lexicon[token];
    end
    else
        if token contains _LONG then
            polarity := polarity + 1.5 ∗ lexicon[token];
        end
        else
            polarity := polarity + lexicon[token];
        end
    end
end
if polarity > 0 then
    return Positive;
end
else
    return Negative;
end
```

To evaluate the quality of our lexicon, namely UnB Sentiment Lexicon, we compare the sentiment of iPhone 6 tweets using four lexicons: Bing Liu Opinion Lexicon, NRC Hashtags Sentiment Lexicon, NRC Sentiment140 Sentiment Lexicon and our iPhone6 lexicon. Bing Liu Lexicon is a general sentiment lexicon manually compiled by Bing Liu [15]. The NRC lexicons [20] are tweet specific but not domain specific lexicon.

We run Sentiment Analysis Algorithm with 508 tweets related to iPhone 6. These tweets were manually labeled and used as ground truth, resulting in 413 positive and 95 negative tweets. The results are given in table V.

From these results, we can see that UnB Sentiment Lexicon obtains better recall indexes. This is because this lexicon captures domain-specific words that other lexicons don't. But, the Bing Liu Lexicon has the highest precision. This happens because it is a general lexicon, compiled manually over years, and all its entries are verified by humans. Despite of that, it is not a pratical lexicon for this purpose, because they capture less than half of the positive tweets, as verified by its low recall. The NRC lexicons provide a very similar precision like UnB Sentiment Lexicon, but a smaller recall percentage. Overall UnB Sentiment lexicon outperforms the other lexicons and obtains the best results in all.

TABLE V
Results of Sentiment Analysis

Lexicon	Precision	Recall	F-measure
BingLiu Lexicon	0.87317	0.43341	0.57928
NRC Hashtag	0.82844	0.88862	0.85748
NRC Sent. 140	0.82989	0.87409	0.85142
UnB Sentiment	0.82452	0.94431	0.88036

VII. Conclusions and Future Work

In order to execute efficient TSA on a particular topic or domain, a unified tool is proposed with several critical contributions made. An algorithm for expanding limited hashtags into a larger and more complete set of hashtags is proposed to collect tweets, which is demonstrated to be able to obtain good results with average precision over 90%. To further speed up the algorithm and to decrease the amount of inputs needed, the future work will design a new clustering algorithm to generate the expanded set of hashtags. The spam detection algorithm uses several tweet and user features, excluding spam tweets with recall over 90%. A domain-specific sentiment lexicon is built to incorporate expressions whose sentiment varies from one domain to another, without the use of labeled data. The future work will consider to utilize multiple lexicons, combing the strong points of them. The proposed TSA tool is tested on the 'iPhone 6' domain which obtains convincing results. This tool is readily to be adapted to work with other domains, retaining the lowest input requirement of labeled tweets for spam only.

Acknowledgment

This work has been partially funded by authors individual grant from the Brazilian National Science and Technology Council (CNPq).

References

[1] O. Ozdikis, P. Senkul, and H. Oguztuzun, "Semantic expansion of hashtags for enhanced event detection in twitter," in *Proceedings of the 1st International Workshop on Online Social Systems*, 2012.

[2] S. Carter, M. Tsagkias, and W. Weerkamp, "Twitter hashtags: Joint translation and clustering," *Proceedings of the ACM WebSci11*, pp. 1–3, 2011.

[3] A. Mazzia and J. Juett, "Suggesting hashtags on twitter," in *EECS 545 Project, Winter Term, 2011. URL http://www-personal. umich. edu/~ amazzia/pubs/545-final. pdf*, 2009.

[4] S. M. Kywe, T.-A. Hoang, E.-P. Lim, and F. Zhu, "On recommending hashtags in twitter networks," in *Social Informatics*. Springer, 2012, pp. 337–350.

[5] O. Tsur, A. Littman, and A. Rappoport, "Efficient clustering of short messages into general domains." in *ICWSM*, 2013.

[6] A. Rangrej, S. Kulkarni, and A. V. Tendulkar, "Comparative study of clustering techniques for short text documents," in *Proceedings of the 20th international conference companion on World wide web*. ACM, 2011, pp. 111–112.

[7] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on twitter," in *Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)*, vol. 6, 2010, p. 12.

[8] G. Stringhini, C. Kruegel, and G. Vigna, "Detecting spammers on social networks," in *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM, 2010, pp. 1–9.

[9] M. Klassen, "Twitter data preprocessing for spam detection," in *FUTURE COMPUTING 2013, The Fifth International Conference on Future Computational Technologies and Applications*, 2013, pp. 56–61.

[10] J. Benhardus and J. Kalita, "Streaming trend detection in twitter," *International Journal of Web Based Communities*, vol. 9, no. 1, pp. 122–139, 2013.

[11] J. Nichols, J. Mahmud, and C. Drews, "Summarizing sporting events using twitter," in *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*. ACM, 2012, pp. 189–198.

[12] H. Almuhimedi, S. Wilson, B. Liu, N. Sadeh, and A. Acquisti, "Tweets are forever: a large-scale quantitative analysis of deleted tweets," in *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 2013, pp. 897–908.

[13] Z. Miller, B. Dickinson, W. Deitrick, W. Hu, and A. H. Wang, "Twitter spammer detection using data stream clustering," *Information Sciences*, vol. 260, pp. 64–73, 2014.

[14] J. Martinez-Romo and L. Araujo, "Detecting malicious tweets in trending topics using a statistical analysis of language," *Expert Systems with Applications*, vol. 40, no. 8, pp. 2992–3000, 2013.

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

[16] L. Velikovich, S. Blair-Goldensohn, K. Hannan, and R. McDonald, "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*. Association for Computational Linguistics, 2010, pp. 777–785.

[17] S. Tan and Q. Wu, "A random walk algorithm for automatic construction of domain-oriented sentiment lexicon," *Expert Systems with Applications*, vol. 38, no. 10, pp. 12 094–12 100, 2011.

[18] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 513–520.

[19] Y. Choi and C. Cardie, "Adapting a polarity lexicon using integer linear programming for domain-specific sentiment classification," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*. Association for Computational Linguistics, 2009, pp. 590–598.

[20] S. M. Mohammad, S. Kiritchenko, and X. Zhu, "Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets," *arXiv preprint arXiv:1308.6242*, 2013.

[21] M. Ghiassi, J. Skinner, and D. Zimbra, "Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network," *Expert Systems with Applications: An International Journal*, vol. 40, no. 16, pp. 6266–6282, 2013.

[22] L. Chen, W. Wang, M. Nagarajan, S. Wang, and A. P. Sheth, "Extracting diverse sentiment expressions with target-dependent polarity from twitter." in *ICWSM*, 2012.

[23] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams Engineering Journal*, 2014.

[24] S. Mukherjee and P. Bhattacharyya, "Sentiment analysis: A literature survey," *arXiv preprint arXiv:1304.4520*, 2013.

[25] A. Montoyo, P. Martínez-Barco, and A. Balahur, "Subjectivity and sentiment analysis: An overview of the current state of the area and envisaged developments," *Decision Support Systems*, vol. 53, no. 4, pp. 675–679, 2012.

[26] P. Gonçalves, M. Araújo, F. Benevenuto, and M. Cha, "Comparing and combining sentiment analysis methods," in *Proceedings of the first ACM conference on Online social networks*. ACM, 2013, pp. 27–38.

[27] A. Bakliwal, P. Arora, S. Madhappan, N. Kapre, M. Singh, and V. Varma, "Mining sentiments from tweets," *Proceedings of the WASSA*, vol. 12, 2012.

[28] H. Saif, Y. He, and H. Alani, "Semantic sentiment analysis of twitter," in *The Semantic Web–ISWC 2012*. Springer, 2012, pp. 508–524.

[29] B. Liu, "Sentiment analysis and opinion mining," *Synthesis Lectures on Human Language Technologies*, vol. 5, no. 1, pp. 1–167, 2012.