TwiSent: A Multistage System for Analyzing Sentiment in Twitter

Subhabrata Mukherjee[†], Akshat Malu[†], Balamurali A.R.[†‡], Pushpak Bhattacharyya[†]

[†]Dept. of Computer Science and Engineering, IIT Bombay

[‡]IITB-Monash Research Academy, IIT Bombay

{subhabratam, akshatmalu, balamurali, pb} @cse.iitb.ac.in

ABSTRACT

In this paper, we present *TwiSent*, a sentiment analysis system for Twitter. Based on the topic searched, TwiSent collects tweets pertaining to it and categorizes them into the different polarity classes *positive*, *negative* and *objective*. However, analyzing micro-blog posts have many inherent challenges compared to the other text genres. Through *TwiSent*, we address the problems of 1) *Spams* pertaining to sentiment analysis in Twitter, 2) *Structural anomalies in the text* in the form of incorrect spellings, nonstandard abbreviations, slangs *etc*., 3) *Entity specificity* in the context of the topic searched and 4) *Pragmatics* embedded in text. The system performance is evaluated on manually annotated gold standard data and on an automatically annotated tweet set based on *hashtags*. It is a common practise to show the efficacy of a supervised system on an automatically annotated dataset. However, we show that such a system achieves lesser classification accurcy when tested on generic twitter dataset. We also show that our system performs much better than an existing system.

Keywords: Sentiment Analysis, Twitter, Micro blogs, Spam, Entity Specific Twitter Sentiment

1. INTRODUCTION

Social media sites, like Twitter, generate voluminous amounts of data which can be leveraged to create applications that have a social and an economic value. In this paper, we present a hybrid system, *TwiSent*, to analyze the sentiment of tweets based on the topic searched in Twitter. Even though Twitter generates a large amount of data, a text limit of 140 characters per tweet makes it a noisy medium for text analysis tasks. Compared to other text genres like News, Blogs *etc*., it has a poor syntactic and semantic structure. For example, consider the following tweet "*Had Hella fun today with the team. Y'all are hilarious! &Yes, i do need more black homies..... "*. Apart from the irregular syntax, the following sentence has other problems like *slangs, ellipses, nonstandard vocabulary etc.* A direct analysis of such noisy text using commonly applied Natural Language Processing (NLP) tools would be futile, as it may not give the desired results. Further, the problem is compounded by the increasing number of *spams* in Twitter like *promotional* tweets, *bot-generated* tweets, *random links* to other websites *etc*. In this paper, we tackle the following problems which are exclusive to a micro-blog genre like Twitter for assessing the sentiment content: **Twitter based spam, Spell checker for noisy text, Entity detection** and **Pragmatics.**

2. RELATED WORK

[1] provides one of the first studies on sentiment analysis on micro-blogging websites. [2] and [4] both cite noisy data as one of the biggest hurdles in analyzing text in such media. [1] describes a distant supervision-based approach for sentiment classification. They use hashtags in tweets to create training data and implement a multi-class classifier with topic-dependent clusters. [2] proposes an approach to sentiment analysis in Twitter using POS-tagged n-gram features and some Twitter specific features like hashtags. Our system is inspired from *C-Feel-IT*, a Twitter based sentiment analysis system [3]. However, *TwiSent* is an enhanced version of their rule based system with specialized modules to tackle Twitter spam, text normalization and entity specific sentiment analysis.

There has not been much work in the area of text normalization in the social media, although some work has been done in the related area of sms-es [5]. We follow the approach of [6] and attempt to infuse linguistic rules within the minimum edit distance [7]. We adopt this simpler approach due to lack of publicly available parallel corpora for text normalization in Twitter.

Unlike in Twitter, there has been quite a few works on general entity specific sentiment analysis. Many approaches have tried to leverage dependency parsing in entity-specific SA. [8] exploits dependency parsing for graph based clustering of opinion expressions about various features to extract the opinion expression about a target feature. We use dependency parsing for entity specific SA as it captures long distance relations, syntactic discontinuity and variable word order.

The works [1][12][13] evaluate their system on a dataset crawled and auto-annotated based on *emoticons* while [14] annotate the crawled data based on *hashtags*. We show, in this work, that a good performance on such a dataset does not ensure a similar performance in a general setting.

3. SYSTEM ARCHITECTURE

In this section, we give an overview of the complete system and define the functionality of each module. *Figure 1* presents the architecture of the system.

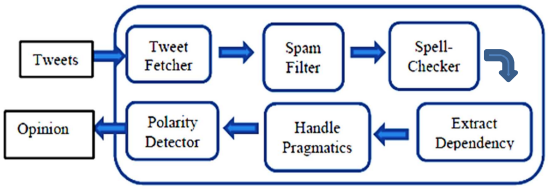

Figure 1. TwiSent Architecture Diagram

3.1 Tweet Fetcher and Polarity Detector

A Twitter API is used to obtain live feeds from Twitter. Based on the search string, we retrieve the latest 200 tweets in English. The

tweets are in XML format which needs to be parsed to extract the tweet bodies. The tweet polarity is determined by a majority voting of four sentiment lexicons, following the approach in [3], namely, *SentiWordNet, Subjectivity, Inquirer* and *Taboada*.

3.2 Spam Filter

The Spam is the use of electronic messaging systems to send unsolicited bulk messages indiscriminately. [9] identifies three types of spam: *Untruthful opinions, reviews on brands* only and non-reviews. However, we provide a more detailed categorization of Twitter spams as: *Re-tweets, Promotional tweets, Tweet containing links, Tweets in foreign language or having incomplete text, Bot-generated tweets, Tweets with excessive off-topic keywords or hashtags* and *Multiple tweets with same template*.

The list is not exhaustive as new categories of spams are generated regularly. Thus, adaptation of the algorithm to these new instances of spam requires human supervision. We adopt a partially supervised approach to alleviate this problem. In this setting, we have labeled training examples of only one category namely, the *non-spam* class and a mixed set of unlabeled examples containing *spams* as well as *non-spams*. A classifier is trained on these sets, which tries to identify the non-spam tweets out of the mixed bag. The approach discussed here (*Algorithm 1*) uses Naive Bayesian text classification to implement a partially supervised learning based on *Expectation Maximization* [10].

Input: Build an initial naive bayes classifier NB- C, using the tweet sets *M* and *P*

1: Loop while classifier parameters change

2: for each tweet $t_i \in M$ do

3: Compute $Pr[c_1 | t_i]$, $Pr[c_2 | t_i]$ using the current NB

//c_1 - non-spam class , c_2 - spam class

4: $Pr[c_2 | t_i] = 1 - Pr[c_1 | t_i]$

5: Update $Pr[f_{i,k} | c_1]$ and $Pr[c_1]$ given the probabilistically assigned class for all t_i ($Pr[c_1 | t_i]$). //f denotes the feature set (a new NB-C is being built in the process)

Algorithm 1. Spam Filter Algorithm

The following set of features is used in the spam filter module:

1. *Number of Words / Tweet*	**8.** *Freq. of First POS Tag*
2. *Average Word Length*	**9.** *Freq. of Foreign Words*
3. *Freq. of "?" and "!"*	**10.** *Validity of First Word*
4. *Numeral Character Freq.*	**11.** *Presence / Absence of links*
5. *Frequency of hashtags*	**12.** *Freq. of POS Tags*
6. *Frequency of @users*	**13.** *Character Elongation*
7. *Extent of Capitalization*	**14.** *Frequency of Slang Words*

Table 1. Spam Filter Features

The algorithm begins with assigning all the samples in the non-spam class *P* as non-spam, and all the samples in the mixed unlabeled set *M* as spam. In the first iteration, all the feature values are calculated using the above set of features. The class probabilities are calculated considering individual feature weights leading to probabilities for each tweet to be in either class. All the tweets in the mixed set *M*, which are more probable to be in the *non-spam class* than in *spam class*, are reassigned to the set *P*. A tweet is reassigned from the spam category to one of the three classes (*positive, negative and objective*) for which the probability is highest, if the difference between the probability for this class and the spam class is greater than a threshold. The algorithm halts when there is no further reassignment to any other category.

3.3 Spell Checker and Text Normalization

Multiple spell-checkers are available today, but they are not effective in handling noisy text present in the social media. We give an overview of some of the most prevalent abbreviations and noisy text in Twitter. The list is compiled from the tagged tweets for this work and from [11]: **1.** Dropping of Vowels - Example: *btfl (beautiful), lvng (loving).* **2.** Vowel Exchange - Exchange between pairwise vowels due to phonetic similarity. Example: *good vs. gud (o,u).* **3.** Mis-spelt words - Example: *redicule (ridicule), magnificant (magnificent).* **4.** Text Compression - Example: *shok (shock), terorism (terrorism).* **5.** Phonetic Transformation - Example: *be8r (better), gud (good), fy9 (fine), gr8 (great).* **6.** Normalization and Pragmatics - Example: *hapyyyyyy (happy), guuuuud (good).* **7.** Segmentation with Punctuation - Example: *beautiful, (beautiful).* **8.** Segmentation with Compound Words - Example: *breathtaking (breath-taking), eyecatching (eye-catching), good-looking (good looking).* **9.** Hashtags - Example: *#notevenkidding, #worthawatch.* **10.** Combination of all - Example: *#awsummm (awesome), gr88888 (great), amzng,btfl (amazing, beautiful)*

We implement a *minimum edit distance based spell checker* to resolve all the *identified errors*.

Input: For string s, let *S* be the set of words in the lexicon starting with the initial letter of *s*.

```
/* Module Spell Checker */
for each word w ∈ S do
  w'=vowel_dropped(w)
  s'=normalize(s)
/*diff(s,w) gives difference of length between s and w*/
  if diff(s' , w') < offset then
   score[w]=min(edit_distance(s,w),edit_distance(s,w'),
   edit_distance(s' , w))
  else
    score[w]=max_centinel
  end if
end for
Sort score of each w in the Lexicon and retain the top m
entries in suggestions(s) for the original string s
for each t in suggestions(s) do
  edit1=edit_distance(s' , s)
  /*t.replace(char1,char2) replaces all occurrences of char1
 in the string t with char2*/
  edit2=edit_distance(t.replace( a , e), s')
  edit3=edit_distance(t.replace(e , a), s')
  edit4=edit_distance(t.replace(o , u), s')
  edit5=edit_distance(t.replace(u , o), s')
  edit6=edit_distance(t.replace(i , e), s')
  edit7=edit_distance(t.replace(e, i), s')
  count=overlapping_characters(t, s')
  min_edit= min(edit1,edit2,edit3,edit4,edit5,edit6,edit7)
  if (min_edit ==0 or score[s'] == 0) then
    adv=-2 /* for exact match assign advantage score */
  else
    adv=0
  end if
  final_score[t]=min_edit+adv+score[w]-count;
end for
return t with minimum final_score;
```

Algorithm 2. Spell Checker Algorithm

3.4 Handling Pragmatics

Pragmatics is a subfield of linguistics which studies how the transmission of meaning depends not only on the linguistic knowledge (*e.g.* grammar, lexicon *etc.*) of the speaker and listener, but also on the context of the utterance, knowledge about the status of those involved, the inferred intent of the speaker *etc.* We identified the different forms of pragmatics in Twitter as: **1.** *Happiness, joy or excitement is often expressed by elongating a word, repeating alphabets multiple times* - Example: *happppyyyyyy, goooooood*. **2.** *Use of Hashtags* - Example: *#overrated, #worthawatch*. **3.** *Use of Emoticons* is common in social media and micro-blogging sites where the users express their sentiment in the form of accepted symbols. Example: ☺ *(happy),* ☹ *(sad).* **4.** *Happiness, joy, sorrow, hatred, enthusiasm, excitement, bewilderment etc. are also commonly expressed by capitalization* where words are written in capital letters to express intensity of user sentiments. *Full Caps* - Example: *I HATED that movie. Partial Caps*- Example: *She is a Loving mom.* All these forms are given more weightage than other commonly occurring words by repeating them twice.

3.5 Entity Specificity

A *tweet* may have multiple entities and the user may express a different opinion expression regarding each entity there. Thus, it is of utmost importance to extract the specific opinion expression relating to a particular entity. Consider the tweet, "*The film bombed at the box office although the actors put up a reasonable performance*". Here the sentiment of the tweet with respect to *film* is negative whereas that with respect to the *actors* is positive. [8] proposes a *Dependency Parsing* based method to capture the association between any specific feature and the expressions of opinion that come together to describe that feature. The underlying hypothesis is that: *More closely related words come together to express an opinion about a feature.*.

Consider a sentence *S* and 2 consecutive words $w_i, w_{i+1} \in S$. If $w_i, w_{i+1} \notin StopWords\_List$, then they are directly related. This helps to capture *short range dependencies*. Let *Dependency_Relation* be the list of significant dependency parsing relations (like *nsubj, dobj, advmod, amod etc.*). Any 2 words w_i and w_j in *S* are directly related, *if* $\exists D_l \; s.t \; D_l(w_i, w_j) \in Dependency\_Relation$. Through this *long range dependencies* are captured. The direct neighbor and dependency relations are combined to form the master *relation set* $\boldsymbol{R}$. Given a sentence *S*, let $\boldsymbol{W}$ be the set of all words in the sentence. A Graph $G(W, E)$ is constructed such that any $w_i, w_j \in W$ are directly connected by $e_k \in E$, *if* $\exists R_l \; s.t \; R_l(w_i, w_j) \in R$. All the *Nouns* in the given tweet are extracted by a POS-Tagger which form the feature set *F*. Let $f_t \in F$ be the target feature *i.e.* the feature with respect to which we want to evaluate the sentiment of the sentence.

Let there be '*n*' features where *n* is the dimension of *F*. We initialize '*n*' clusters C_i, corresponding to each feature $f_i \in F$ *s.t.* f_i is the clusterhead of C_i. We assign each word $w_i \in S$ to the cluster whose clusterhead is closest to it. The distance is measured in terms of the number of edges in the shortest path, connecting any word and a clusterhead. Any 2 clusters are merged if the distance between their clusterheads is less than some threshold. Finally, the set of words in the cluster C_t, corresponding to the target feature f_t gives the opinion about f_t.

4. EXPERIMENTAL EVALUATION

Twitter was crawled using Tweet Fetcher module and 8507 tweets (*Dataset 1*) were collected based on a total of around 2000 different entities from over 20 different domains. These were manually annotated by 4 annotators into four classes: *positive, negative, objective-not-spam and objective-spam*. The Twitter API was used to collect another set of 15,214 tweets (*Dataset 2*) based on *hashtags*. Hashtags *#positive, #joy, #excited, #happy etc* were used to collect tweets bearing positive sentiment, whereas hashtags like *#negative, #sad, #depressed, #gloomy, #disappointed etc.* were used to collect negative sentiment tweets.

The crawled tweets were pre-processed before the spam filtering phase. All the links (urls) in the tweets were replaced by "#link". All the user id's in the tweets were replaced by "#user". A dictionary [15] was used to map the standard abbreviations and slangs to their proper words in the lexical resources. An emoticon dictionary was used to map each emoticon to *positive* or *negative* class. The following negation operators like *no, never, not, neither* and *nor* were used and the polarity of all words in the forward context window of *five* from the occurrence of any of these operators were reversed.

We compare our system performance on both the datasets to *C-Feel-It* [3], which is a rule-based system, using a weighted polarity scoring based on four sentiment lexicons, like ours. C-Feel-It has the same *Tweet Fetcher* and *Polarity Detector* module as TwiSent, but lacks the remaining modules.

Manually Annotated Dataset				
#Positive	#Negative	#Objective Not Spam	#Objective Spam	Total
2548	1209	2757	1993	8507
Automatically Annotated Dataset				
#Positive		#Negative		Total
7348		7866		15214

Table 2. Dataset Statistics

Spam Filter module is evaluated in *Dataset 1* as an independent module. It achieved an accuracy of **71.50%** for a four-class classification (*pos, neg, obj-not-spam and obj-spam*) as opposed to **54.45%** for two-class (*obj- spam* vs. *rest*) classification.

For the overall system, we perform a 2-class and a 3-class classification using TwiSent. In the 2-class classification, we consider only *positive and negative* tweets. In the 3-class setting, we consider *positive, negative* and *all objective* tweets as one separate class. *Tables 3* and *4* show the accuracy comparison between *TwiSent* and *C-Feel-It* in *Datasets 1* and *2*, under a 2-class and a 3-class classification setting. Ablation tests (refer to *Table 5*) are performed by removing one module at a time and noting the resulting accuracy of the remaining system. This is done to find the sensitivity of each module. These tests are performed under the *2-class classification setting* using lexicon based classification. A/B significance test [16] was done and the confidence with which the accuracy changes were accepted to be statistically significant is shown in *Table 5*.

Classification	**C-Feel-It Accuracy**	**TwiSent Accuracy**
2-class	52.58%	**66.69%**
3-class	47.23%	**56.17%**

Table 3. C-Feel-It and TwiSent Comparison using Dataset 1

System (2-Class)	**Positive Precision**	**Negative Precision**	**Overall Accuracy**
C-Feel-It	69.06	48.2	58.24
TwiSent	**88.06**	**88.97**	**88.53**

Table 4: C-Feel-It and TwiSent Comparison using Dataset 2

Module Removed	Accuracy	Statistical Sig. Conf.
Entity-Specificity	65.14	95%
Spell-Checker	64.2	99%
Pragmatics Handler	63.51	99%
Complete System	**66.69**	-

Table 5. Ablation Test Results Removing One Module at a Time

5. DISCUSSIONS

5.1 Overall Accuracy

Given a mixed bag of spam and non-spam tweets, the *Spam Filter's* performance improved in a 4-class setting with an overall precision of *71.50%* as opposed to *54.45%* in case of a 2-class classification. This is because merging positive, negative and objective classes into a single class is undesirable as the 3 classes are unique and have different properties altogether. TwiSent achieved a much better accuracy over the baseline system under all the settings. In the 2-class setting the accuracy improvement is over *14%* whereas in the 3-class setting, it is *8.94%*. TwiSent achieves a higher negative precision improvement than positive precision improvement (refer to *Table 4*) over C-Feel-It, which indicates it can capture negative sentiment strongly. Supervised system accuracy suffers due to sparse feature space due to inherent text limit of tweets.

5.2 Ablation Test

The accuracy changes after removing the Entity Specific module, Spell-Checker and Pragmatics Handler are *statistically significant at 95%, 99% and 99% confidence* respectively. The Ablation test shows that removing the Pragmatics Handler decreases the system accuracy most. This indicates that Pragmatism is a very strong feature in the Social Media, but not much work has been done on it. The Spell-Checker also proved to be an important module owing to the tendency of people to mix and match shortenings and abbreviations which cannot be captured in standard lexicons. Hence, without this module, any lexicon-based system would miss out on many important cue words. The entity-specific module, *though important conceptually*, do not contribute greatly because of lack of context due to very short length of tweets, where people express opinions directly to the point unlike in reviews or blogs. The accuracy also gets affected due to the incorrect dependency relations given by the parser due to noisy text (mis-spelt words).

5.3 Effect of Artificial Training Data

There has been a lot of work in Twitter that collect data based on specific features like *hashtags* [1], [12], [13], *emoticons* [14] *etc.* and auto-annotate the tweets based on these features. Although these systems achieve a very high accuracy, they remain biased towards these special features. In this work, we showed that although a system may work very well on a dataset based on a specialized feature set with hashtags (*Dataset 2*), it does not necessarily work well in a general setting (*Dataset 1*). This is evident in the performance of TwiSent in *Dataset 2* (created based on *hashtags*) where it attains a high accuracy of *88.53%* compared to the overall accuracy of *66.69%* in *Dataset 1* (manually annotated general purpose data). This shows that the specialized set of features used to crawl the data actually give away the sentiment *explicitly*, unlike in the general dataset which may have latent sentiment based out of *sarcasm*, *jokes*, *teasers* and other *implicit* sentiment, which is quite difficult to detect.

6. Conclusions and Future Work

In this paper, we introduced a Twitter based sentiment analysis system, *TwiSent*. It is a multistage system with specialized modules to tackle the nuances of micro-blogging genres. Our results suggest that we outperform a similar Twitter based sentiment application by *14%*. One of the major contributions of our work is in introducing Twitter based spams in the context of sentiment analysis. Our *Spam Filter* performs well not only as a part of the system but also as a stand-alone application. The *Spell-Checker* module helps in handling the *noisy text*, whereas the *Pragmatics Handler* can loosely capture the pragmatics in text which assists in improving the classification performance. The *Entity-Specific* module helps in capturing sentiment pertaining to the search entity. A more sophisticated approach to Spell-Checker, in presence of a parallel corpora, and Pragmatics Handler may add to the system performance. The system cannot capture *sarcasm* or *implicit sentiment* due to the usage of a generic lexicon in the final stage for classification. Overall, the paper not only highlights the issues associated with the micro-blogs but also presents an effective system to handle many of them. We also show that a superlative system performance on an auto-annotated dataset does not guarantee a similar or comparable performance on real-life micro-blog data.

7. REFERENCES

1. Alec, G.; Lei, H.; and Richa, B. 2009. Twitter sentiment classification using distant supervision. Technical report, Standford University.
2. Barbosa, L., and Feng, J. 2010. Robust sentiment detection on twitter from biased and noisy data. In Proceedings of the Computational Linguistics Posters, 36–44.
3. Joshi, A.; Balamurali, A. R.; Bhattacharyya, P.; and Mohanty, R. 2011. C-feel-it: a sentiment analyzer for microblogs. In Proceedings of ACL Demo Papers, HLT '11, 127–132.
4. Bermingham, A., and Smeaton, A. 2010. Classifying sentiment in microblogs: Is Brevity an Advantage, ACM 1833–1836.
5. Raghunathan, K., and Krawczyk, S. 2009. Investigating sms text normalization using statistical machine translation. CS224NProject Report, Stanford University.
6. Church, K. W., and Gale, W. 1991. Probability scoring for spelling correction. Statistics and Computing 1(2):91–103.
7. Levenshtein, V. I. 1966. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8.
8. Mukherjee, S., and Bhattacharyya, P. 2012. Feature specific sentiment analysis for product reviews. Part 1, Lecture Notes in Computer Science, Springer 7181:475–487.
9. Jindal, N. and Liu, B. 2008. Opinion spam and analysis. In Proceedings of the 2008 WSDM. pp. 219-229.
10. Liu, B., Lee, W., Yu S., and Li X. 2002. Partially supervised classification of text documents. In Proceedings of ICML.
11. Bieswanger, M. 2007. 2 abbrvi8 or not 2 abbrevi8: A contrastive analysis of different shortening strategies in English and german text messages. SALSA XIV.
12. Jonathon Read. 2005. Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In Proceedings of the ACL Student Research Workshop.
13. Pak, Alexander and Paroubek, Patrick. 2010. Twitter as a Corpus for Sentiment Analysis and Opinion Mining. In Proceedings of the LREC.
14. Gonzalez-Ibanez, Roberto and Muresan, Smaranda and Wacholder, Nina. 2011. Identifying sarcasm in Twitter: a closer look, In Proceedings ofACL Short Papers.
15. Website. http://chat.reichards.net/. Retrieved Aug. 11, 2012
16. In Wikipedia. Retrieved on August 11, 2012, from Website http://en.wikipedia.org/wiki/A/B_testing