

# Text Sentiment Classification

Beril Besbinar, Dimitrios Sarigiannis, Panayiotis Smeros  
*Department of Computer Science, EPFL, Switzerland*

**Abstract**—We examine sentiment Analysis on Twitter data. Through the following paper we develop, analyze and compare the most powerfull proposed solutions on the topic. Our Analysis focuses on text preprocessing, feature extraction and machine learning application techniques, the combilation of which have surprisingly achieved a very high accuracy on our final results. Finally, the trade-off between execution time and accuracy is being discussed.

## I. INTRODUCTION

## II. DATA DESCRIPTION

In this section we describe the given Twitter dataset. We were given 2.5 million equally seperated on tweets that used to contain a positive :) and or negative :( smiley and another 10,000 test unlabled tweets. Given, the smiley-filtered test tweets and considering only the remaining text, we were asked to classify them in positive and negative sentiments. The fact that the tweets are equally seperated and the absense of a neutral sentiment made our problem easier. Also, some preprocessing has been done on the given data i.e. all the users and urls have been changed to <user> and <url> tags respectively and all the tweets has been transformed to lowercase characters.

However some other observations on the data had to be considered. Twitter text is written by regular users, in informal language which most of the time contains non existing words which nonetheless makes sense either in the morphological or in the pragmatic layer of a language. Moreover, there are plenty of characters that indicate something meaningfull like punctuation or other types of emoticons. Finally, the existence of compound words, syntactically incorrect sentences, different conjugations of the same word and many other such problems had to be taken into account. In the following section we will present some preprocessing techniques which will make the feature extraction easier in each approach and boost our accuracy.

## III. PREPROCESSING TECHNIQUES

In this section we describe the techniques that we use in order to process the raw tweets. After applying these techniques we were able to emphasize writing styles and words that denote sentiment (e.g., emojis, repeating the last character of a word, etc.) as well as handling most of the spelling errors and slang words usage (e.g., with lemmatization and stemming).

Most of the implemented methods were inspired from the respective methods of *GloVe* [?] and are summarized as follows:

### A. Contractions Expansion

In informal speech, which is widely used in social media, it is common to use contractions of words (e.g., *don't* instead of *do not*). This may result in misinterpreting the meaning of a phrase especially in the case of negations. For this reason we implemented a method that expands these contractions.

### B. Emojis Transformation

Emojis are widely used in social media and they can alone give a good estimate of a post's sentiment. In order to handle the different writing styles of emojis (e.g., :) , :- ) and (: are all smile faces) we transform all of them into the word that describes better the sentiment that they denote. Thus we have smile, lol, neutral and sad faces as well as hearts.

### C. Emphasize Repeated Punctuation

Some punctuation marks like exclamation point (!) denote sentiment especially when they are repeated within a tweet. We implemented a method in which we emphasize that repetition.

### D. Emphasize Repeated Last Characters

Similarly to the punctuation marks, the repetition of the last character of a word reveals sentiment (e.g., *I am happyyyy*). We introduce a method that corrects the spelling of such words (e.g., converts *happyyyy* to *happy*) and adds a special tag to indicate the repetition.

### E. Filter Numerical Expressions

Since numbers in general do not hold any special sentiment semantics, we replace all the numerical expressions with tags that stand for the existence of such expressions.

### F. Split Hashtags

Hashtags are one or more words concatenated with the symbol #. Some of the tokens of the hashtags may hide useful information for the sentiment of a post (e.g., #love-myjob). Splitting a hashtag into tokens is a difficult problem [?] since we may have multiple, ambiguous splits (e.g., #homestore can be split into either *home store* or *hometore*). Having typos and using slang words makes it even more difficult. In order to solve this problem we used a dictionary with the most frequently used English words in

descending order (according to Zipf’s law) and tried to guess the correct split.

#### G. Emphasize Sentiment Words

Similarly to emojis, some English words denote clearly an emotion (e.g., *anxiety* or *happiness*). In order to emphasize the existence of such words we were advised by a lexicon with positive and negative words [?].

#### H. Part-Of-Speech Tagging

Part-Of-Speech tagging helps us understanding the structure of an input text in order to discover its most important words. Since posts in social media are informal, without following many grammatical rules, in most of the cases this method did not help.

#### I. Lemmatization and Stemming

In order to homogenize verbs being in different tenses we implemented a method that applies lemmatization and stemming techniques to the words of the input tweets.

#### J. Stop-words Filtering

Stop-words are very common English words that can be found in almost every tweet and thus do not imply any sentiment (e.g., the articles *The*, *A* etc.). We implemented a method that removes them from tweets. We have also considered <user> and <url> tags as stopwords, since they appear in a very high frequency in the corpus but this did not increase our accuracy. The single-character elimination did also not increased the accuracy.

### IV. REPRESENTATIONS OF TWEETS

In this section we present all the numerical representations that we chose for our input tweets. These representations try to capture the semantic meaning of the tweets while having a constant size, which is independent of the words used in each tweet. Thus, letting  $N$  be the number of input tweets, we create an  $N \times D$  matrix with each line representing a different tweet with  $D$  features.

Below we present the most usable word and text representations that we introduced and used in our classification problem.

#### A. Bag of Words Representation

Bag of Words [?] is a very natural numerical representation of a text. Given that we have a vocabulary  $V$  with finite size, we create an array of size  $|V|$  with 1s in the position of the words belonging in this text and 0s everywhere else. Thus, in our case, we would have an  $N \times |V|$  matrix representing our input tweets.

The problem with this representations is that  $|V|$  can be arbitrary big especially if we do not handle appropriately slang and words with spelling mistakes. One way to solve it, is using the preprocessing methods that we proposed in Section III. Another problem of this representation is that

it does not comprise any semantic information about the significance of each word in a tweet.

#### B. TF-IDF Representation

The significance of the words in tweets is captured by the more sophisticated representation which uses the Term Frequency - Inverse Document Frequency (TF-IDF). TF-IDF [?] works by determining the relative frequency of a word in a specific document compared to the inverse proportion of that word over the entire document corpus. Words that are common in a single or a small group of documents tend to have higher TF-IDF numbers than common words such as articles and prepositions.

This metric can be used as a weight to the respective Bag of Words representation that we have discussed above. Thus, we have again an  $N \times |V|$  matrix representing our input tweets, that contains 0s and  $TF - IDF$ s instead of 0s and 1s.

#### C. N-grams Extension

In both of the above representations the  $N \times |V|$  matrix can become even bigger if we add sequences of contiguous words in our vocabulary. These sequences are called *n-grams* with  $n$  being the maximum number of words in a sequence. For example, for bigrams, in the worst case the size of the vocabulary becomes  $|V| + \binom{|V|}{2}$ . As we will see in the experimental evaluation we tested representations with up to 8-grams.

#### D. Word Embeddings Representation

- 1) *Baseline Build Method:*
- 2) *Glove Python Method:*
- 3) *Pretrained Embeddings:*
- 4) *Hybrid Method:*

#### E. From Word Embeddings to Tweet Embeddings

### V. MODEL SELECTION

### VI. CONCLUSIONS