Text Sentiment Classification

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

***Abstract*—**

I. Introduction

II. 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).

The methods that we implemented were inspired from the respective methods of *GloVe* [1] 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., #lovemyjob). Splitting a hashtag into token is a difficult problem [2] since we may have multiple, ambiguous splits (e.g., #homestore can be split into either *home store* or *homes tore*). 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 [3].

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.

III. 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 [4] 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 II. 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 [5] 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 $TF - IDFs$ 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

Word Embeddings [6] is a recently proposed representation that maps words to high dimensional vectors of real numbers. This representation tries to capture the multiple contexts in which a word can appear in a text. Hence, it guarantees that words that appear in similar context, will have vectors that are very close each other. The advantage of Word Embeddings comparing to the previous representations is that they have fixed size for each word which is independent of the size of the used vocabulary.

Thus, in our problem, we can create for all the words in our vocabulary a $|V| \times D$ matrix representation, where D is the dimension of our vectors. In order to construct this matrix we implemented the methods presented bellow.

1) GloVe Training Methods: GloVe [1] has an open-source implementation of its training methods. We experimented with both the given implementation and the *glove-python*[1] that uses *SGD* for training the vectors. As we confirmed with the experimental evaluation, the vectors trained with *glove-python* were more accurate than the ones trained with the given implementation.

2) Pretrained GloVe Embeddings: We also employed a pretrained set of Word Embeddings published by GloVe[2]. These vectors were trained with 2 billions tweets containing 1.2 millions words. The train set of these vectors was an order of magnitude bigger than the one we were given (both the positive and the negative train sets contain 2.5 millions tweets). Thus, these vectors are very well trained, capturing accurately the meaning on each word. Also, since the corpus that was used for the training was Twitter, they cover a high percentage of the words used in our given train set.

3) Hybrid Method: Since not 100% of the words of our train set were found in the pretrained Word Embeddings, we decided to train our own vectors for these words and append them to the pretrained ones. However, the union of the vectors gave us actually worse accuracy than just the pretrained ones. One explanation is that the two sets of vectors were unequally trained. The GloVe vectors were trained for more than a day in a very powerful server, while ours in a common PC for a couple of hours.

E. From Word Embeddings to Tweet Embeddings

IV. Model Selection

V. Conclusions

[1] http://github.com/maciejkula/glove-python

[2] http://nlp.stanford.edu/projects/glove/

REFERENCES

[1] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[2] S. Khaitan, A. Das, S. Gain, and A. Sampath, "Data-driven compound splitting method for english compounds in domain names," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, ser. CIKM '09. New York, NY, USA: ACM, 2009, pp. 207–214. [Online]. Available: http://doi.acm.org/10.1145/1645953.1645982

[3] 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.

[4] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[5] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[6] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *CoRR*, vol. abs/1310.4546, 2013. [Online]. Available: http://arxiv.org/abs/1310.4546