Natural Language Processing

Rafael Barón González - Ripoll
UCLM
Ciudad Real, Spain
Rafael.Baron@alu.uclm.es

Samuel González Linde
UCLM
Ciudad Real, Spain
Samuel.Gonzalez3@alu.uclm.es

Author Keywords
Machine Learning; Twitter; Classification

INTRODUCTION

Given a dataset made out of thousands of real tweets we need to classify them to say if the tweet is offensive or not. For that, we need to transform and clean the tweets, and also vectorize them for the algorithms to work.

We made all the text preprocessing to get a list of tweets transformed to apply a classification algorithm. For that we use techniques such as stemming, vectorization and a preprocess to remove contractions or unuseful data.

DATASET

The dataset is composed by thousen of tweets with (most) very bad language, it contains tags of people that consider them offensive, not offensive or neither.

Preprocessing Data

Emoji Replacement

In order to replace emoticons with their correspondant strings, we have decided to use the library **emoji**. This library contains a set with emoji codes defined by the unicode consortium. Each emoji has a representation with a string. If we take a look to the *emoji.EMOJI_UNICODE* we can see a set containing a string value with the corresponding emoji.

Exploring the data contained in the tweets we have found that some of the emoticons are coded with numeric character references instead of Unicode characters. So that is an added difficulty when using the emoji library. We found that these numeric references were html code which can be converted to unicode using the library **html**.

Then for each tweet we are using html.unescape in order to convert any possible emoji into unicode and then passing it to emoji.demojize to have a more readable solution.

Tokenization

Once we have replaced the references to the emoticons with text, we are going to proceed with the tokenization.

Tokenization is the process of turning a piece of data, usually text, into atomic units called tokens. Those tokens can be words, characters or subwords and serves as a reference to the original data.

It is even possible to tokenize by sentences but in our case each row contains the data for one tweet which it is already a sentence so we are going to use *world_tokenize* from the library **nltk** to slice each sentence into tokens (or words) in order to work with them.

While splitting our tweets into tokens we are going to check that these words are not contained in a set called *stopwords* where it is stored words that has poor meaning or value and therefore we can remove them from the tweet. In our case as the idiom we are going to select is the language of the tweets, english.

Removing Contractions

In this case we came up with an easy solution for the contraction removement. Taking the most common contractions into a dictionary with their key set with the constraction form and the value with the non-contracted form so we remove almost every contraction from the tweets.

Removing Unuseful Data

Just as we did with the contractions, we have made a selection of characters that are not useful to us and therefore can be eliminated. A simple algorithm is in charge of finding and removing them from the tokens that correspond to the symbols stored in a list

Remove Capital Letters

Running a simple *lower* function for each token from each tweet allows us to lowercase every character.

Correcting Wrong Words

First thing we tried to use for correcting wrong words was the spell correct given us in the project pdf. Once runing the algorithm with the whole dataset of tweets we ended up discovering that after a long time waiting Google Collab would end up running out of memory

So we decided to split the process and every time a tweet is processed store and continue with the next instead of going through the whole dataset all at once.

Also we decided to use **pyspellchecker** as it didn't need any external document (As the other one needs the *big.txt* to run) and it works the same way.

It checks the word given and it makes sure that is not misspelled and if it is, the algorithm corrects it by using a word frequency list.

This process has a great computational cost and it lasts a lot of time because of the big amount of tweets it has to process.

Lemmatisation and Stemming

We algo apply techniques in order to change the words to get only the root of the word or origin word. For that we use the functions aviable in the ntlk library and apply it to every tweet.

To get the root of the word we use the technique of stemming. This give us results such as eliminating the 'ing', 'ed' and 'ly' at the end of some words. The result is useful to get words simplified and generalize the tweets for a better context and connections between tweets.

To generalize even more we can do the lemmatization step. This is even more powerfull because it apply a morphological analysis to the words of the tweet. Lemmatization not only eliminates 'ing' and others, it transform verbs such as 'was' in to 'be', generalizing even more.

Vectorization

In order to *map* the tweets we do the step of vectorization, transforming the words into numbers. This is a trivial step to classify the words beacuse the classification algorithm words with numbers.

We use the vectorizer from *sklearn.feature_extraction.text*. Applying it for every tweet gives us a list of arrays which every array is a list of the vectorization of the tweet.