Machine Learning Project 2
Text Classification

Franck Dessimoz, Paul Jeha, Pierre Latécoère
CS-433 Machine Learning, EPFL, Switzerland

***Abstract*—Text classification is a core topic of Machine Learning. It has a broad range of applications such as sentiment analysis of social media posts, email spam detection or articles classification. Using Natural Language Processing, ...**

I. Introduction

In this paper we are going to investigate which tools can can be used in order to allow a computer to understand the english language. More precisely, given a set of tweets, we we will train a model to enable a computer to decide wheter a given tweet reflects a positive or negative sentiment.

The data we are given are tweets which originally ended with either a :) smiley for positive tweets or a :(smiley for negative tweets and predict their sentiment from which the smileys where removed.

To train our model, we were given a set with only positive sentiment tweets and a set with only negative sentiment tweets.

We used the following specialized libraries in order to get the best accuracy possible :

- Pandas, to manipulate and analyze the data.
- Scikit-learn, providing us :
  - Classification functions
  - Validation functions, such as grid search with k-fold cross-validation, to optimize the model's hyperparameters.
  - Preprocessing functions, such as TfidfVectorizer, used to tranform tweets into vectors.
- NLTK, to work with human language data.

II. Models

A. First Model

A critical point of text classification is the transformation of words or sequence of words into vectors. Assume we can map k sequences of words to k vectors of dimension N, with N being the number of features. Then we can naturally construct a matrix of dimension KxN. Furthermore, assume that we know the sentiment of each sequence of words. Denote by 1 a positive sentiment and by -1 the negative sentiment. We can then feed the KxN matrix along with the list of sentiments to a classifier, which will output a model.

We started with the following simple model to get acquainted with the data. As a first step, we mapped every word into a vector using the GloVe algorithm. The GloVe algorithm is a variant of Word2Vec that allows us to obtain vector representations for words. Training is performed using a co-occurence matrix, with entries corresponding to the number of times a specific word occurs together with a context word.

Now that we have a vector representation for each word, we need to obtain the vector representation for each tweet. Our first approach to obtain the vector representation of a tweet was to take the average of the vector representation of the words constituing this tweet.

The vector representation of each tweet corresponds to our training features while the actual sentiment of the tweet corresponds to the train prediction. Having both the features and the predictions we ran many different classifiers to obtain the best model possible.

—COMPARAISON OF STANDARD CLASSIFIERS SGDCLASSFIER, LOGISTICREGRESSION, LINEARSVC—

By observing the above graphs, we can see that the accuracy of our model is bounded by XXX. This means that we have to improve our preprocessing step in order to get a bettwer accuracy.

B. Second Model - Taking into account the weight of each word into the tweet

The main aspect of our first model we can improve is our strategy to construct the vector representation of the tweet based on the vector representation of its individual words.

In our first model, we simply took the unweighted average of the vector representation of each word constituing the tweet. This is not an optimal way to form the vector since we do not take into account the importance of the words. For example stopwords such as 'a', 'the' or 'this' don't bring the same amount of information as 'happy' or 'sad', so they shouldn't be treated as having the same importance. It is therefore natural that attributing weights to words depending on their importance would result in a greater accuracy. We implement this improvement using the TfidfVectorizer function of the Scikit learn library.

The TfidfVectorizer function converts a collection of raw documents to a matrix of TF-IDF features. Now let us explain simply what is a TF-IFD matrix.

Let's now see what the TF-IDF matrix does. Our aim is to give weights to the words according to their importance. A natural way to get the importance of a word is to take its frequency in a given document (an article, a book, a tweet, ...). But if a word also appears often in other documents, the importance of this word is reduced. Take stopwords for

example, they appear in all documents and therefore should have a very small weight as they don't bring any relevant information.

The TfidfVectorizer maps list of documents to their vector representation, taking into account the importance of each word, the TF-IDF 'score'. In our case, the list of documents is the list of tweets. Indeed each tweet can be seen as a document. TfidfVectorizer directly computes the vector representation of a tweet instead of first computing the vector representation of individual words.

This gives us a second set of vectorized tweets. We apply to it the same reasoning as above and obtain the evaluate the difference in accuracy between many classifiers.

—RUN TF-IDF WITH NO PARAMETERS AND PRINT ACCURACIES FOR DIFFERENT CLASSIFIERS WITH NO PARAMETERS SGDCLASSIFIER, LOGISTICREGRESSION, LINEARSVC—

Note that the TfidfVectorizer returns a matrix, whose rows correspond to the vector representation of the tweets. This matrix is sparse, which prevents us to make many transformations on it but allows the computation to be much faster.

C. Third Model

The second model described is a big improvement over the first one. But there still is some things to be improved. Indeed the parameters of the TfidfVectorizer function can be tuned to get a better model. First we can tune the stop words list of TfidfVectorizer instead of using the default one of NLTK which consists of the stop words of the english language. The issue with this default list of stop words is that in general tweets do not follow the usual principles of english grammar and vocabulary. Therefore we decided to search for a list of stopwords more suitable to the context of social media. We found three list of words specifically designed for twitter text analysis which can be found HERE, that we combined in order to obtain our final list of stop words : twitter-stopwords-final.txt. It is obvious that doing so would improve our accuracy since we now use a list of stopwords specifically designed for text classification of tweets.

Secondly, we can also tune the tokenizer parameter of the TfidfVectorizer. The tokenizer is the function that segments a tweet into a list of its component words. TfidfVectorizer has a default tokenizer but once again this default tokenizer doesn't take into account the fact that we are in a social media context. To handle this issue we defined our own tokenizer, specifically designed for our needs. We make use of TweetTokenizer(), a tweet-aware tokenizer, i.e. a function that splits tweets into lists of their component words taking into consideration that we are working with tweets. For example, smileys as ':-)' or words containing a hashtag '#astonmartin' won't be split, while the default tokenizer would return [':', '-', ')'] and ['#', 'astonmartin']. We then apply a lemmatizer function to the tokens. The lemmatizing step is the lexical analysis, which groups words of the same family. This means that each word will be taken back to its root. It groups the different forms a word can take: a name, a plural a verb or an infinitive. For example, 'taking' will be taken back to 'take', 'dogs' will be taken back to 'dog'.

After the tokenizing step, we pass each token into a lemmatizer function. The lemmatizing step is the lexical analysis, which group words of the same family. This means that each word will be taken back to its root. It groups the different forms a word can take: a name, a plural a verb an infinitive and so on. For example, 'taking' becomes 'take', 'dogs' becomes 'dog'.

Having an updated version of the vector representation of the tweets, we ran the same classifiers as above to check for improvements.

—RUN TF-IDF WITH NO PARAMETERS AND PRINT ACCURACIES FOR DIFFERENT CLASSIFIERS WITH NO PARAMETERS SGDCLASSIFIER, LOGISTICREGRESSION, LINEARSVC—

We clearly see an improvement compared to the previous versions of the vector representation of the tweets.

D. Fourth Model

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed at sollicitudin justo, in vestibulum sapien. Duis quis risus iaculis, condimentum risus eget, vulputate magna. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque laoreet pellentesque mauris eget venenatis. In et urna viverra, elementum ligula molestie, maximus odio. Maecenas urna tortor, pellentesque sed ante ut, vehicula egestas mi.

III. Results

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed at sollicitudin justo, in vestibulum sapien. Duis quis risus iaculis, condimentum risus eget, vulputate magna. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque laoreet pellentesque mauris eget venenatis. In et urna viverra, elementum ligula molestie, maximus odio. Maecenas urna tortor, pellentesque sed ante ut, vehicula egestas mi.

IV. Discussion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed at sollicitudin justo, in vestibulum sapien. Duis quis risus iaculis, condimentum risus eget, vulputate magna. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque laoreet pellentesque mauris eget venenatis. In et urna viverra, elementum ligula molestie, maximus odio. Maecenas urna tortor, pellentesque sed ante ut, vehicula egestas mi.