Sentiment analysis with TF-IDF and Word2Vec embedding techniques

Giovanni Mantegna
Politecnico di Torino
Student Id : s296555
s296555@studenti.polito.it

Domenico Mereu
Politecnico di Torino
Student Id : s302151
s302151@studenti.polito.it

***Abstract*—In this paper we use two different word embedding approaches for sentiment analysis classification. After the preliminary steps of data cleaning and preprocessing we perform two different words embedding techniques: Word2Vec and TF-IDF. We evaluate different models for the classification task in both cases. Our best result in sentiment classification task uses a TF-IDF embedding and logistic regression classification model. It achieves a 0.804 f1-score. Finally, we propose a solution that makes use also of date and user features and achieves a f1-score of 0.861**

I. Problem overview

This paper shows two possible solutions for sentiment analysis classification applied on a data set that contains 223106 tweets. This data set was collected during a period from 2009-04-06 to 2009-06-25. Our data set has 5 different features.

- *ids* : The ids is a unique identifier for a tweet
- *date* : The date of posting
- *flag* : A flag that shows how tweets were collected
- *user* : The name of the user who posts the tweet
- *text* : The text of a tweet

Our target label is *sentiment*, it can assume value 1 if the tweet shows a positive sentiment or 0 if it shows a negative sentiment. The final classification results are computed on an evaluation set with 74999 tweets.

TABLE I
Variables cardinality

Variable	Cardinality
sentiment	2
ids	224716
date	189779
flag	1
user	10647
text	223106

In a preliminary step of data exploration, we notice that *ids* values do not match with corresponding unique values of *text*. We found about 200 duplicates and we decide to drop these rows. The sentiment value count shows that our data set has unbalanced classes. We found 130157 positive tweets and 94837 negative ones. This data set has no missing values but the quality of the text corpus is very raw and required a lot of preprocessing. The main feature for sentiment analysis is *text*. As shown in the word clouds of the following figures, the feature *text* uses different words in case of positive sentiment Fig 1 or negative sentiment Fig 2

Fig. 1. WordCloud extracted by our data set filtered by positive *sentiment* excluding common words form both sentiment

Fig. 2. WordCloud extracted by our data set filtered by negative *sentiment* excluding common words form both sentiment

II. Proposed approach

A. Preprocessing

Preprocessing is a necessary data preparation step for sentiment classification. Preprocessing of tweets includes following three main tasks: *Noise Removal*, *Normalization* and *Negation handling*.

Noise Removal: In sentiment analysis noise removal is referred as the process of cleaning or removing irrelevant data from a huge collection of extracted data. The text extracted from social media sites is full of noise. This text contains URLs, symbols, undesired punctuations and some special communication symbols e.g. @, # ,etc. These symbols and tags have no role in sentiment classification tasks and therefore all such kinds of punctuations must be eliminated before mining and analysis. In this phase of preprocessing we removed these undesired symbols and tags using regular expressions.

Normalization: Text normalization is the process of transforming a text into a canonical (standard) form. For example, "gooood" and "gud" can be transformed to "good", its canonical form. Tweets often contain non-standard words, ill-formed words, negation words and out of vocabulary words. In the following, different processes are done for normalization:

- *Emoticon replacement*: We use the emot python library [1] for replacing the emoticons with their meaning. (e.g. :) $\rightarrow$ smiley).
- *Expand acronyms*: Acronyms are informal expressions and are replaced by their canonical forms using table consisting of 290 such expressions and their replacements. For e.g. LOL is replaced by laughing_out_loud.
- *Replace elongated word*: A word is elongated when it contains a character that is repeating more than two times, like the word 'greeeeat'. It is important to replace words like this with their source words, so that they can be merged. Otherwise, the classifier will treat them as different words, and probably the elongated ones will be ignored because of their low frequency of occurrence. For this purpose we use regular expressions.
- *replace contractions*: One technique that can be used in preprocessing is the replacement of contractions, i.e. words like 'won't' and 'don't', that will be replaced with 'will not' and 'do not', respectively. This helps in the next phase of the handling of the negations. Also for this task we use regular expressions.

Negation handling : Negations are very important in linguistics because they affect the polarities of other words. Negations include words such as no, not, shouldn't etc. When a negation appears in a sentence it is important to determine the sequence of words which are affected by this term. The scope of negation may be limited only to the next word after a negation or may be extended up to other words following negation. We have combined two different approaches:

- *Replace negations with antonyms*: We search in each sentence for the word 'not' and then we check if the next word has an antonym. If yes, we replace both words with the antonym. For example, the phrase 'not good' will be replaced with the word 'bad', using WordNet [2].
- *Next word negation technique*: When text analysis is performed at a word level, it is very challenging to handle negation. One method that is widely used by researchers is the detection of words that imply negation and the addition of the prefix 'NOT_' in every word after them until the first punctuation mark. We have modified this strategy and instead of negating all the words till punctuation, we have negated the very next word following the negation word. So, if the sentence "Dog does not eat Chinese food" is received then instead of converting it to "Dog does NOT_eat NOT_Chinese NOT_food", we have converted it to "Dog does NOT_eat Chinese food".

B. Word Embedding

After the preprocessing step, that is useful for every text embedding models, we decide to try two different approaches and compare the results as shown in some papers.

- *Word2Vec* : This model [3] creates an n-dimensional vector representation for every word in our data set. An implementation of Word2Vec algorithm is provided by Gensim [4]. The model is based on a two-layer neural network that processes text by vectorizing words. The relative distance of one word from another depends on the semantic area. Moreover, words with opposite means in n-dimension representation will have different orientation. Fig 3 shows a t-SNE visualization of Word2Vec space for our model. Due to the large amount of words we selected a subset of relevant words to plot. Words with similar means are near each other in this two-dimensional representation.

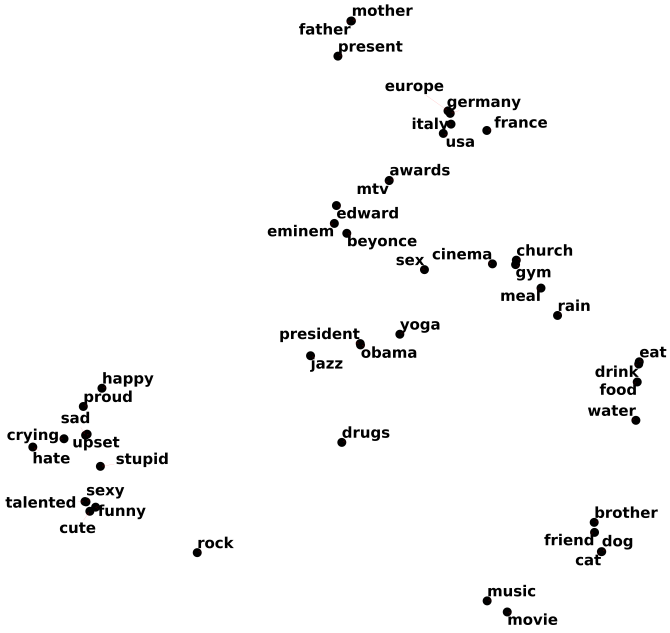

Fig. 3. An example of t-SNE visualization o Word2Vec model space

We can also see that some of the most similar words form our model are semantically strongly correlated. Gensim model provides `most_similar` method. Using this method we provide a word as an input and receive as output the most similar word. For example: sad → upset, stupid → lame, cute → adorable. For this particular classification task we create the model and train it with all tokenized words then we use a standard method characterizing a tweet. We consider the mean of word vectors. The dimension size of the mean vector is 400 and we filter out words with a count less then 30.

- *TF-IDF* : The most successful unsupervised term weighting scheme is TF-IDF (term frequency-inverse document frequency). The TF-IDF algorithm is the product of two terms. The first frequency term (Luhn, 1957) represents frequency of a word in the document. Normally we want to downweight the raw frequency a bit so we generally use the log_{10} of the frequency. The second factor is inverse document frequency or idf.It is used to give a higher weight to words that occur only in a few documents.

  Term frequency: $w_{d_i,t_j} = tf$

  Logarithmically scaled tf: $w_{d_i,t_j} = log(tf + 1)$

  TF-IDF: $w_{d_i,t_j} = tf * log\left(\frac{N}{df}\right)$

  To define similarity between two target words v and w, we need a measure for taking two such vectors and giving a measure of vector similarity. By far the most common similarity metric is the cosine of the angle between the vectors. Below we show three sentences of the dataset and the most similar ones calculated with the cosine similarity matrix:

TABLE II
MOST SIMILAR SENTENCES FOR TF-IDF MODEL

Negative sentence	Most similar sentence
urgh its raining bad	bored and its raining
@scoty_mc i miss you	@AnnikaRaymundo i miss you.
i hate sleeping alone	alone in the house

For this classification task, we used the TfidfVectorizer from the Scikit-learn library with a max_features of 70000 and an ngram_range equal to (1,2). Increasing the number of features and using trigrams weighed down the model and did not improve the classifier.

C. Model Selection

Once we extracted word vectors from our preprocessed text we proceed with a classification task. For both embedding models, we test several machine learning techniques. The following models have been tested for the project:

- Random Forest Classifier: The algorithm trains on multiple decision trees driven on different subsets of data. The random forest algorithm is one of the best among classification algorithms but it is expensive in training time
- Logistic Regression Classifier: Logistic regression is one of the most important analytic tools in natural language processing, logistic regression is the baseline supervised machine learning algorithm for classifying an observation into one of two classes.
- Linear Support Vector classifier: Linear support vector machines have become popular for solving classification tasks due to their fast application to large scale datasets.

TABLE III
MODEL COMPARISON

Model for TF-IDF approach	**f1-score**
`Logistic Regression Classifier`	0.80
`Linear Support Vector Classifier`	0.78
`Random Forest Classifier`	0.77
Model for Word2Vec approach	**f1-score**
`Logistic Regression Classifier`	0.75
`Linear Support Vector Classifier`	0.75
`Random Forest Classifier`	0.74

In table III we can see the f1-score obtained with the three models for the two approaches, TF-IDF and Word2Vec. These results are obtained with a hold-out split of 80% training set and 20% test set. The logistic regression has the highest f1-score score with the TF-IDF approach.

D. Hyperparameter tuning

The model selection phase leads us to use TF-IDF and logistic regression as the most performing model. We perform a grid search in order to increase our results. We use a grid with the following hyperparameters related to TF-IDF vectorizer (specified with "`vect_`") and logistic regression (specified with "`clf_`") :

TABLE IV
GRID SEARCH

Parameter	**Values**
`vect__ngram_range`	[(1, 2), (1, 3)]
`vect__stop_words`	[stopwords, None]
`clf__penalty`	['l1', 'l2']
`clf__C`	[1.0, 10.0, 100.0]
BEST RESULT	
`vect__ngram_range`	(1, 2)
`vect__stop_words`	None
`clf__penalty`	'l2'
`clf__C`	1.0

The model performs better without removing stopwords and without lemmatization and steaming. Our best result with this tuned model is 0.804 f1-score in public leaderboard

III. IMPROVEMENTS USING DATE AND USER

During the exploration phase, we also search for *date* and *user* patterns related to our target, *sentiment*. Sorting our data

set by *date* it is possible to find out a regularity. The Fig 5 shows the time series of cumulative *sentiment* in a brief period of time. The pattern is very clear, we have periods of time with sentiment 1 followed by periods of time with sentiment 0. In the cumulative graph it is easy to distinguish periods of time with positive sentiment (vertical line regions) and periods of time characterized by negative sentiment (horizontal line regions).

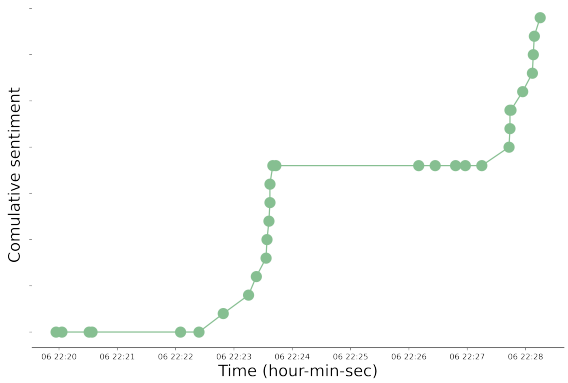

Fig. 4. Time series of cumulative *sentiment*

A similar consideration can be done about *user*. Grouping our data set by user, we observe that some users systematically post mostly negative or positive tweets. In Fig 5 is shown a histogram of variable defined as:

$$Ul = \frac{\#Positive - \#Negative}{\#Positive + \#Negative} \tag{1}$$

where #Positive and #Negative are the number of tweets with positive and negative sentiment of a user. This histogram shows a spike for value 1 i.e users that post only positive tweets.

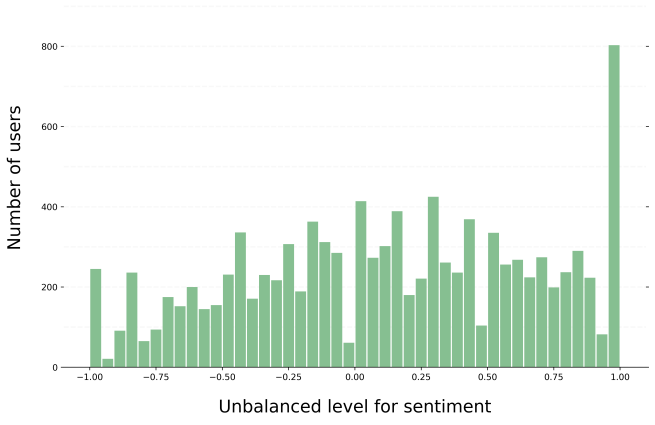

Fig. 5. Histogram of unbalanced level of *sentiment* for users

IV. Results

Due to the patterns shown in section III, we decide to use *date* and *user* as classification features, combining it with our TF-IDF vectors. Our sentiment classification model is based on cascade ensemble classifier [5]. The cascading method we implemented consisted in using the results of a first stage classification with Logistic Regression Classifier as additional columns for a second stage classification using a Random Forest Classifier. First of all, we make a one-hot encode of *user* feature and concatenate it with our TF-IDF vectors. We use our best model shown in section II-C to make a logistic regression. Then we use the previous outcome as an input for a random forest together with *date* converted as an integer using the method `datetime.toordinal()`. This choice increases a lot the classification score (see Table V).

TABLE V
Final Results in test data

Accuracy	
0.86	
Precision	**Recall**
Positive sentiment 0.88	Positive sentiment 0.88
Negative sentiment 0.83	Negative sentiment 0.84

Finally, in public leaderboard we achieve a f1-score of 0.861. Without a cascade model we obtain lower results. The f1-score using a random forest classifier with *date*, *user*, and *text* features is 0.810. However, it is important to say that this approach can be used only with this particular data set. These patterns in *date* and *user* probably have no meaning for a general sentiment analysis classification.

V. Discussion

In this paper we explore TF-IDF and Word2Vec embedding models to perform a classification task. We find out that in this particular sentiment analysis task, TF-IDF perform better than Word2Vec. TF-IDF is more susceptible to preprocessing phase. However, Word2Vec is more useful in order to distinguish semantic areas in our data set. The choice of embedding technique results to be decisive in the performance of the model. TF-IDF seems to be reasonably efficient but does not adopt the known class information of training text while weighting a term. In a forthcoming project we could test supervised term weighting (STW) proposed in many researches [6] or increase Word2Vec efficacy by trying to characterize better the semantic areas related to positive and negative sentiment.

References

[1] https://github.com/NeelShah18/emot
[2] https://www.nltk.org/howto/wordnet.html Supervised term weighting for automated text categorization
[3] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
[4] https://radimrehurek.com/gensim/models/Word2Vec.html
[5] Ensemble Methods Foundations and Algorithms By Zhi-Hua Zhou
[6] Debole and Sebastiani, Supervised term weighting for automated text categorization