Happy or Sad? A Twitter Sentiment Analysis

Zoé Baraschi **Louis-Maxence Garret** **Arnaud Boissaye**
`name.surname@epfl.ch`

Abstract

In this report, we present a study of sentiment analysis on Twitter data, where the task is to predict whether the smiley contained in the tweet is happy :) or sad : (. We experimented with today's most common solutions, such as text preprocessing and supervised classification techniques. We mixed-and-matched our algorithms to evaluate how they influenced the accuracy of our predictions. Our predictor obtains an accuracy of: 0.85.

1 Introduction

In this project, we would like to find the best model that determines whether a tweet is positive or negative. Our data is composed of $1'250'000$ negative tweets, $1'250'000$ positive tweets and $10'000$ tweets to predict. We also have a 10% data set to test our implementation with more speed. We first thought about cleaning our data before running different models on it, such as bag of words, TF-IDF and skip gram. We first used BOW and TF-IDF with unigrams and then experimented with bigrams and trigrams.

The whole project is in Python, as it is a very well suited language for data analysis and machine learning.

2 Methodology

Our research consisted of comparing the following methods:

- Bag of Words with unigrams.
- TF-IDF with unigrams.
- BOW and TF-IDF with n-grams.
- Skip gram.

In addition, we decided to see if the different ways of cleaning data have an impact on our prediction efficiency or not. Indeed, we spotted many duplicated tweets, *"user"* tags in (due to the tagging of users in Twitter) and stop words, which we thought would not influence the outputted prediction when removed. Finally, we speculated that in order to gain more consistency, using the root (stem) of a word would be better than using the word itself.

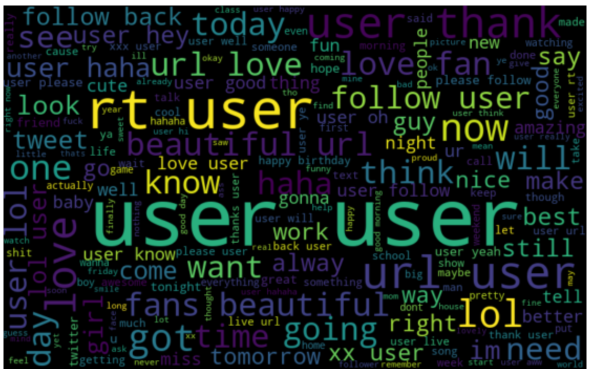


Figure 1: Positive Data Set Word Cloud, Original

2.1 Data Cleaning

Figure 1 and 2 respectively show the word clouds on the original and cleaned positive data set. Those figures clearly show us that some words, for instance *user and rt* which are not usually considered to be related to positive emotions, are redundant in the positive data set and thus appear as positive words. We therefore decided to explore and apply some data cleaning methods to our model

2.1.1 Stemming

This process reduces a word to it's stem *i.e difficulty* $\rightarrow$ *difficult*
Why : This should give more importance to words that are frequently used with some affixes. We should then observe a better comprehension from our model to the data set and thus gain a better prediction.
How : We used a porter stemmer from the [1]*Natural Language Toolkit* library.

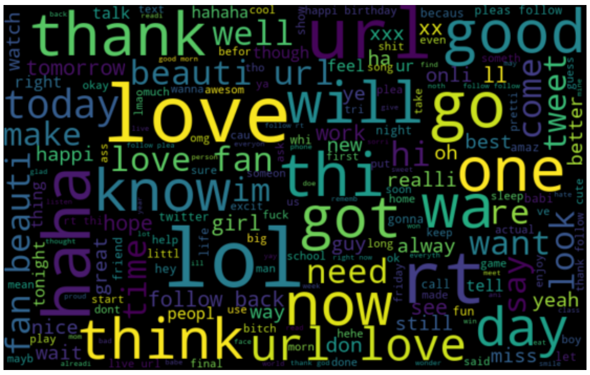


Figure 2: Positive Data Set Word Cloud, Cleaned

2.1.2 Stop Words

Those words are used to connect common words between them. *i.e a, or, the, ...*
Why : They should not give any important information to our model.
How : We used the English stop word dictionary from [1]*Natural Language Toolkit*.

2.1.3 Pattern Replacement

The data set has a lot of patterns such as *"user"* which represent the anonymization of a user's user name.
Why : Those patterns are theoretically not relevant to our model. We chose to replace these patterns with a simple space.
How : We used the [2]*regular expression library*.

2.1.4 Remove Duplicates

Duplicates are everywhere in the data set. We have the same tweet multiple times.
Why : Having duplicate tweets may bias our model and give more importance to some words that should theoretically not influence it. In order to avoid this issue, we could simply remove those duplicates
How : Dropping duplicates using the [4]*pandas library*.

2.2 Methods

2.2.1 Bag of Words (BOW) with unigrams

BOW is a widely used model in document classification. *Why :* As it is a very common model, we started with this one as starting point.
How : The idea is to use the occurrence of each word as a feature for the classifier. Each row in our data set then becomes a "bag of words", where you have each word and its occurrence. Then, you merge all of the bags together to get a bigger set of words and their occurrences. At the end, you get a single bag with all the words in the data set and their occurrences. To do so, we used the [5]*scikit-learn library*

2.2.2 TF-IDF with unigrams

Term Frequency - Inverse Document Frequency measures the relevance of a word inside the word corpus (i.e. the list of all words in all documents). The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.
Why : This model will devalue a word with a high occurrence. This will give more importance to words which seem to be less important because they are less often in the corpus but should be still relevant.
How : It first gets the frequency of each word in the data set (TF) but some words will appear substantially more than others. Those words get less relevance points that the others (IDF). To do so we used the [5]*scikit learn library*

2.2.3 N-grams

This represents all possible combinations of consecutive words in a given sentence, from 1 to N words in each combination. *Why :* Considering combinations of 2 or more consecutive words could be relevant. This comes from the language semantic, some words are preceded or followed by some other words. This is part of the understanding of the data set. Considering such combinations could increase our accuracy as it will give more meaning to some words.
How : We integrate this into our BOW and TF-IDF implementations and consider each combination as a "word". For our model we choose to use N = {1,2,3}. To do so we used the [5]*scikit learn library*

2.2.4 Skip gram

This is a generalization of n-grams where the words do not have to be consecutive.
Why : As n-grams use consecutive words, we thought that depending on the cleaning methods, we could get the same accuracy as n-grams with no cleaning which could lead us to better run time.
How : We used the Fasttext library. Rather than building a wrapper around fasttext binaries, we used the pip Fasttext package that already provides bindings for the skip gram and CBOW models. Unfortunately, the binding of fasttext for skip-gram and CBOW does not allow for fine tuning. This leads to an impossibility to change the seed used for randomization, hence an impossibility to reproduce results done using the fasttext library. We decided therefore not to use it.

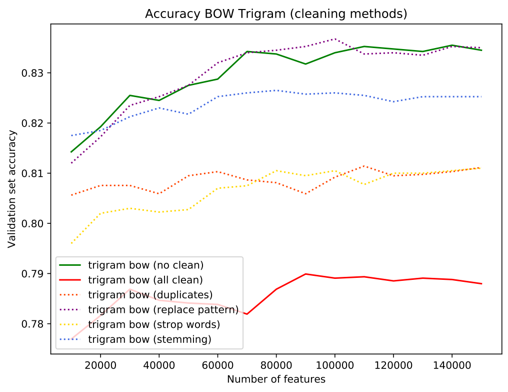


Figure 3: BOW Cleaning Comparison

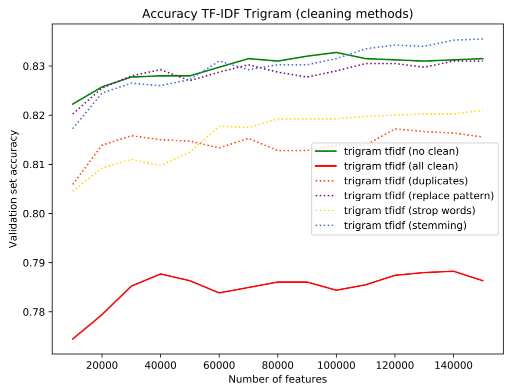


Figure 4: TF-IDF Cleaning Comparison

3 Results

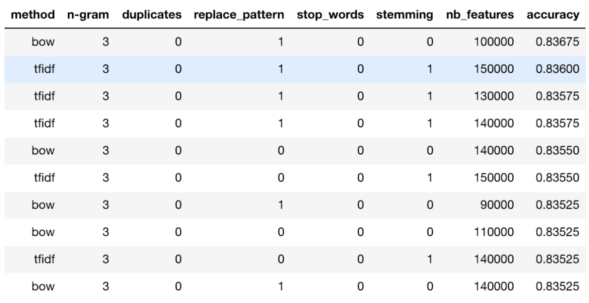
We ran all the possible combinations between our methods and our data cleaning. This lead us to 1440 different combinations to run. The plots in the next sections are based on 10% of the data set.

3.1 Methodology

We first saw that the skip gram method in the Fasttext library was not consistent, since the accuracy changed after each run. We this decided to drop this method. We then wanted to see which combinations of cleaning methods yielded the best results. For this matter, we tested all combinations of cleaning while using BOW, TF-IDF with unigrams, bigrams and trigrams. The following plots aim to visualize the performance of those combinations depending on the number of max features for BOW and TF-IDF. Figure 3 shows the difference of performance of the various data cleaning methods using BOW if we apply them one by one, all of them or none of them. For the sake of clarity of this plot, we did not add combinations of cleaning methods but interesting combinations will be detailed later. We can clearly see that not cleaning the data at all or solely replacing patterns render the best results.

Figure 4 shows the same cleaning methods as for 3 but for TF-IDF. Here we see that replacing patterns, stemming or not cleaning the data render the best accuracy.

Figure 5 displays the 10 best combinations found on the reduced data set. As we can see, the best results use a combination of cleaning methods. We can see that only trigrams yield the best accuracies. We plot n-gram accuracies for n=1,2,3 to verify this.



method	n-gram	duplicates	replace_pattern	stop_words	stemming	nb_features	accuracy
bow	3	0	1	0	0	100000	0.83675
tfidf	3	0	1	0	1	150000	0.83600
tfidf	3	0	1	0	1	130000	0.83575
tfidf	3	0	1	0	1	140000	0.83575
bow	3	0	0	0	0	140000	0.83550
tfidf	3	0	0	0	1	150000	0.83550
bow	3	0	1	0	0	90000	0.83525
bow	3	0	0	0	0	110000	0.83525
tfidf	3	0	0	0	1	140000	0.83525
bow	3	0	1	0	0	140000	0.83525

Figure 5: Top 10 Accuracies using 10% of the Training Set

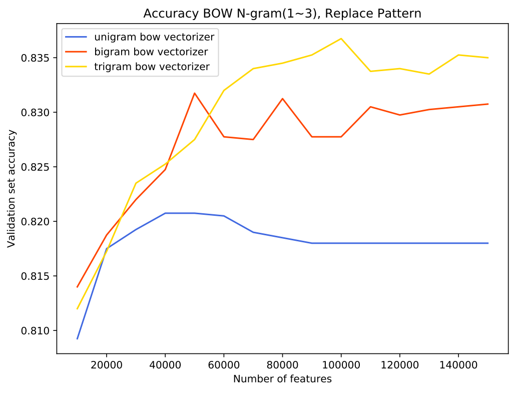


Figure 6: BOW with Replace Patterns

With figure 6, we can clearly see that trigrams outperform bigrams and unigrams, when we increase the number of max features for BOW.

Figure 7 confirms this tendency for TF-IDF, but only once we get to more than 100'000 max features.

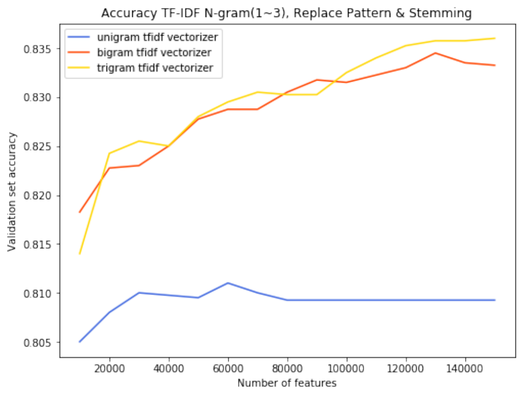


Figure 7: TF-IDF with Replace Patterns and Stemming

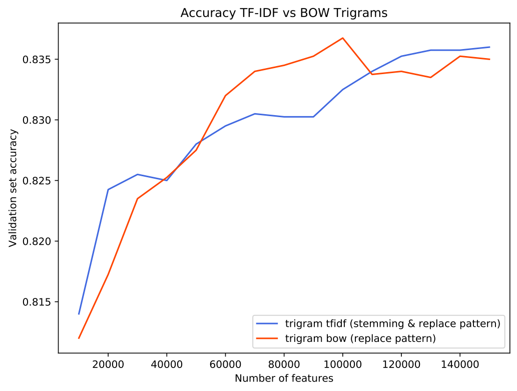


Figure 8: BOW vs. TF-IDF (using best parameters)

Finally, we can compare TF-IDF trigrams with its best cleaning parameters (stemming & replace pattern) versus BOW trigrams with its best cleaning parameters (replace pattern). While CBOW gives a better accuracy for a max number of features ranging between 40'000 and 110'000, we have better results than with TF-IDF for a max number of features ranging between 110'000 and 150'000.

To ensure that our model did not overfit for the reduced set compared to the full set, we ran our models on the full dataset using the combinations of parameters of the 10 best accuracies results displayed in figure 5. For those, we generated the submission csv and uploaded them on the [1]*CrowdAI* platform. This allowed us to find that on the full dataset, the best combination of parameters was the 3rd in 5, i.e. using TF-IDF trigram with max features of 130'000 and cleaning with replace pattern and stemming methods. This gave us an accuracy of 0.85 on the CrowdAI leaderboard.

3.2 Cross Validation

We ran a five-fold cross validation. Here is a summary of our results on our best model:

	precision	recall	f1-score	support
0	0.86	0.83	0.85	1225015
1	0.84	0.87	0.85	1224985
avg	0.85	0.85	0.85	2450000

Table 1: Cross Validation

4 Discussion

4.1 Methods

We had trouble to run our model over the whole data set due to the lack of computational power. We tried the suggested approach with gLove and embeddings but it was confusing. We thus decided to search the internet for a more intuitive approach, which you can find in our references section. We also tried to implement a neural network but as we got at most the same results as our best method we did not explore this further. We also could have tried to experiment with a convolutional neural network.

4.2 Data

The data set was composed of some tweets that are not relevant at all in terms of sentiment analysis:
i.e.13x35 Custom Picture Frame / Poster Frame 2" Wide Complete Honey Pecan Wood Frame (8936): This frame is manufa... http://amzn.to/tUqHS3 . As the tweet contains the following pattern: *(...):* which is identified as a negative tweet even if does not convey negative sentiments. Such tweets are found in both the training and testing sets. As they are not relevant for sentiment analysis removing such tweets, from both sets, could be interesting.

5 Conclusion

To conclude, the best result was obtained using TF-IDF trigram, set with a maximum of 130'000 features and using the replacement of certain patterns and stemming as cleaning methods for the data set. This yielded an accuracy of 0.85 on CrowdAI.

References

[1] https://www.nltk.org/ *Natural Language Toolkit*. last update 3.4 - 17 Nov 2018.

[2] https://docs.python.org/2/library/re.html *regular expression library*. last update - 08 Nov 2018.

[3] https://fasttext.cc *Fasttext*.

[4] https://pandas.pydata.org *Pandas*. last update 0.23.4 - August 2018

[5] https://scikit-learn.org/stable/documentation.html *Scikit library*. last update 0.20.1 - 23 Nov 2018

[6] https://github.com/epfml/ML_course/tree/master/projects/project2/project_text_classification *EPFL - Machine Learning Course Repository on Github*.

Inspiration

[7] https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-4-count-vectorizer-b3f4944e51b5

[8] https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/

[9] https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/