# Tweet sentiment classification problem

Nicolò Vergaro
*Politecnico di Torino*
Student id: s295633
s295633@studenti.polito.it

***Abstract*—In this report, I propose a possible approach to sentiment analysis (positive or negative) of tweets using classification techniques. My solution consists of extracting information from the tweet corpus and using data about the user and the date.**

## I. Problem overview

The proposed task is a sentiment analysis problem on a dataset of tweets. The training set contains 224994 tweets which have the following features:

- *ids*: the unique identification of the tweet;
- *date*: the date on which the tweet was written. It is expressed in PDT timezone;
- *flag*: the query used to collect the tweet;
- *user*: the user who wrote the tweet;
- *text*: the corpus of the tweet;

The feature sentiment is what we will have to predict. It can be either 0 or 1 if the sentiment is negative or positive.

The evaluation set contains 74999 tweets and does not contain the label.

The dataset contains 35k more positive tweets than negative ones. I tried balancing the dataset by removing them naively (removing the last 35k positive tweets) and undersampling. However, it has always led to worse scores, so the population tends to write more positive tweets.

The dataset contains almost 2000 duplicates. I will deal with this aspect later on in the preprocessing section.

After further analysis, I found that the evaluation and development sets contain 10647 unique users. This can be a strong point since the model can also learn what these users tweet and better predict their tweets' sentiment. I will give a further interpretation on this aspect in the Discussion section.

The feature *date* is expressed in the Pacific Daylight Time timezone, used in western North America. I observed that there are only three unique months in both sets: April, May, and June. Given that there are limited users, I think that tweets were extracted from a community of people in a limited period. Date and user could be excellent indicators of how a particular user behaves and what kind of tweets he writes.

Fortunately, the dataset does not contain missing values. This simplifies the problem since we do not have to deal with inferring values or removing whole records (or columns) that could be critical.

By analyzing the length of the tweets, I noticed that there are three tweets that exceed the limit of 280. After further inspection, I noticed that they contain encoded characters. I tried to discard them, but I obtained a lower result, so I kept them also because they are just three. The information about the length of the tweet does not improve my score, probably because the lengths are distributed in almost the same way for both sentiments, as we can see from Fig. 1. The main difference is in height, and that is because there are more positive than negative tweets.

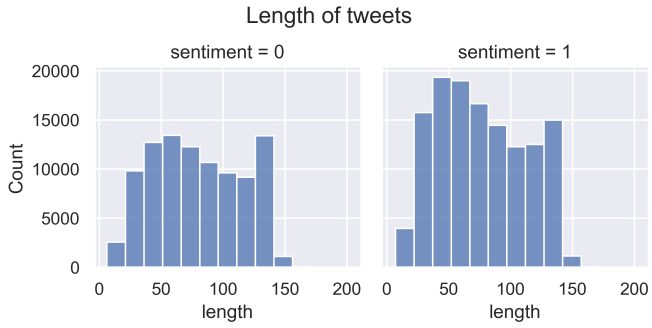


Fig. 1. Length of tweets aggregated by sentiment

## II. Proposed approach

### A. Preprocessing

Concerning duplicate tweets, I found out that:

- 1500 can be found by checking for duplicates using the tuple (user, text). I found out that they are mostly spam tweets. Removing these worsened my score, so I kept them.
- 500 can be found by checking the id of the tweet. The sentiment is different in many of these tweets, even if they are the same tweet. Removing these improved my score.

Unfortunately, in the evaluation set, 44 duplicate ids can not be removed to correctly upload on the platform. After this step, the feature *ids* is removed because it will not be used for classification.

The feature *flag* has the same value for all the records in both sets, that is "NO_QUERY", so I decided to drop it in both sets since it provides no value.

For the feature *date*, I split it into:

- day of the week (e.g Monday, Tuesday,..);
- hour of the day (24-hour format);
- day+month, as a single feature.

As already said, there are only three months out of twelve, but it is not a problem since the evaluation set contains only

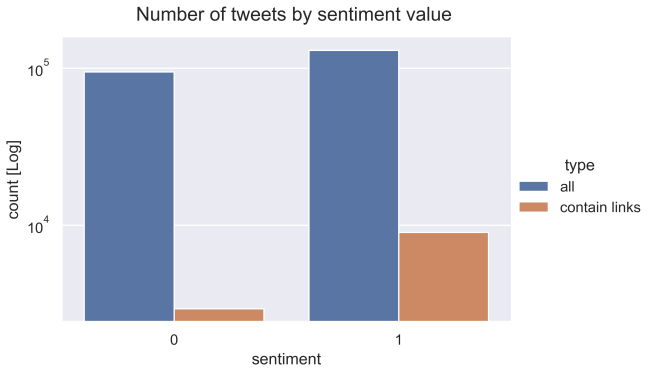


Fig. 2. Number of tweets grouped by sentiment and divided by all tweets vs. only tweets that contain links

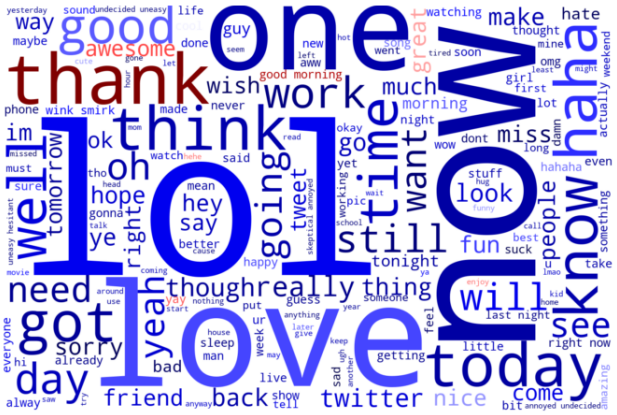


Fig. 3. Wordcloud of the most frequent words in the collection of tweets

these three months. The day of the week and the day+month are textual features and will be one-hot encoded. The hour is a numerical feature, so it does not need encoding. After this process, I obtained 56 additional features (1 from hours, 7 from days, and 48 from day+month).

The most intensive preprocessing is done on the corpus of the tweet. The first things I do are:

- I substitute all HTML characters (", &, <, >) with their real representation (", &, <, >). This is beneficial because now emoticons appear correctly.
- I remove links
- I remove hashtags
- I substitute emoticons with their description, which I found on GitHub.It is good enough.

From the *nltk* library, I use the *TweetTokenizer* module to split the tweet into tokens. The main difference with the classic *word_tokenize* is that TweetTokenizer keeps hashtags as a token (I had already removed them) and, most importantly, does not split words that contain an apostrophe, such as contracted forms (e.g. I'm, won't, don't). So, I remove words that are not alphanumeric (that includes @usernames), but I keep contracted words. I also remove words that are shorter than two characters. I do not remove stop words from the nltk library since I found out that it led to worse scores.

I also extracted a new feature called *contains_links*, which is a feature that indicates whether or not there is a link in the tweet. I found out that it is indeed a good indicator of sentiment since there are three times more positive tweets that contain a link than negative ones. This is not explained by the fact that the dataset is unbalanced since that difference is 1.3 times more positive than negative tweets. We can observe this difference in Fig. 2 (represented in logarithmic scale so the same distance will represent the same percentage changes).

The same operation was done with the feature *contains_mentions*, which works with @usernames.

The relevant information is extracted from the text using a text mining technique called *tf-idf*[1]. This technique computes a score for each word representing its importance in the single document, also considering the collection of documents. It computes the frequency of each word in the single document and the inverse frequency in the whole collection of documents. Words that appear too often in the collection will have a lower value of tf-idf, while words that do not occur often in the collection characterize the document and have a higher value. The result is a sparse matrix with columns the different words found in the collection of documents with their respective values. To exclude terms that appear too often and too rarely, I will tune the parameters *max_df* and *min_df*. Furthermore, to consider context, I will also consider the parameter *ngram_range*, which is used to take into consideration not just single words but groups of more (n) words. These are the hyperparameters for the preprocessing that will be discussed in II-C.

For what concerns the encoding of the user, I tried encoding this information using one-hot encoding, but it resulted in a very vast sparse matrix. I could not concatenate it with the also very vast sparse feature matrix obtained from the text, not even with NumPy optimized methods. Instead, I decided to include the user at the end of the text I obtained after the preprocessing steps. In this way, I can consider it a word and, through tf-idf, encode it and append it in the feature matrix. Using tf-idf, I can also obtain information about how much a user tweets, that combined with the information about the sentiment of his tweets can enhance the classifier's ability to predict (for example, just by knowing if a user has tweeted a few times with negative sentiment, we can think that his following tweet will be negative).

Fig. 3 shows the most relevant words extracted using this technique. Words are colored according to the positive sentiment they carry. We can observe the words used primarily on positive tweets such as thank, good morning, or awesome and those primarily used in negative tweets such as now or one.

### B. Model selection

The models I chose to work with are:

[1]tf-idf: term frequency - inverse document frequency

- Random forest: it is an ensemble[2] classification technique that leverages the power of many decision trees (trained on different portions of both the dataset and the features) to obtain a **local** optimal result. I immediately discarded this algorithm since it takes a very long time to run (one hour and a half) without feature elimination.
- Support Vector Machines: this algorithm creates a hyperplane (decision boundary) by maximizing the *margin*[3]. I chose it because it is among the best classifiers. In particular, I used the LinearSVC classifier since it does not take too long to run and provides very good results.
- Naive Bayes: this algorithm is based on Bayes' rule with the *naive* assumption that the features are independent. For each class, it computes the probability that each record belongs to the class. These probabilities represent the model. I used in particular the ComplementNB classifier that outperforms other kinds of naive Bayes classifiers such as MultinomialNB. It is especially used for text classification, as explained in [1].

I also tried the decision tree algorithm, but it provided inferior results ($F1 \approx 0.789$), so I moved to the random forest technique (which I will eventually discard). I preferred to discard the KNN algorithm since the feature matrix quickly became very large in the number of attributes, so it becomes unfeasible to use[4]. For all models except for the random forest, I searched for the best hyperparameters to further improve my starting score, as discussed in the next section.

### C. Hyperparameters tuning

Hyperparameters can be divided into two groups based on each phase:

- Preprocessing: *ngram_range max_df min_df*;
- Prediction: hyperparameters for LinearSVC and ComplementNB

First, I have tuned the hyperparameters regarding the first phase, and then I moved on to the second phase. For the first phase, I ran a grid search using an 80/20 hold-out split on the default LinearSVC model and evaluated its F1 score. After finding the best configuration, I tuned the hyperparameters for the classifiers. For this tuning I ran another grid search, but this time with cross-validation using $K = 5$ folds; the development set was split in 20% test and 80% train-validation, which was further split during the cross-validation phase. A summary of the hyperparameters can be found in Table I

## III. Results

The tuning of the preprocessing hyperparameters has been done on both LinearSVC and ComplementNB and found that the best configuration is:

- LinearSVC: {*max_df*= 0.4, *min_df*= 2, *ngram_range* = (1, 3)} ($F1 \approx 0.845$);

[2]generate many models and combine them to improve results

[3]distance between the two parallel hyperplanes that touch the closest instance of both classes respectively

[4]because of the curse of dimensionality

| Model | Parameter | Values |
|---|---|---|
| Preprocessing | *ngram_range* | {(1,2), (1,3)} |
| | *max_df* | {0.1, 0.2, 0.4, 0.8} |
| | *min_df* | {2, 5, 30} |
| LinearSVC | *C* | 0.1 → 1, step 0.1 |
| | *random_state* | 42 |
| | *max_iter* | 10000 |
| | *penalty* | {'l1', 'l2'} |
| ComplementNB | alpha | 0 → 2, step 0.1 |
| | norm | {True, False} |

TABLE I
HYPERPARAMETERS CONSIDERED

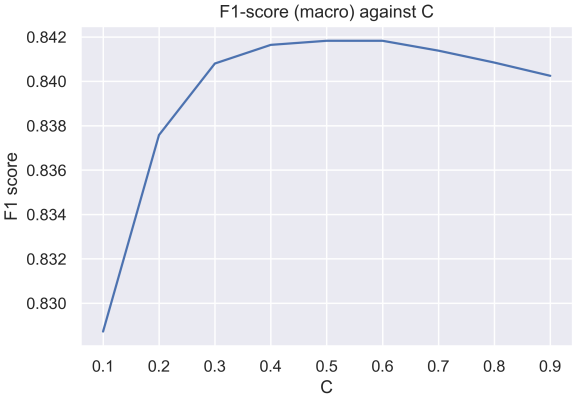


Fig. 4. LinearSVC performances as C varies

- ComplementNB: {*max_df*= 0.1, *min_df*: 5, *ngram_range* = (1, 2)} ($F1 \approx 0.830$);

I chose to use the first one since it provides a sufficiently significant improvement on the F1 score. After finding the best configuration for the preprocessing, I used it to find the best configurations for the classifiers, that are:

- LinearSVC: {*C*=0.5, *penalty*=l2} ($F1 \approx 0.849$)
- ComplementNB: {*alpha*=0.3, *norm*=False} ($F1 \approx 0.83$)

With respect to the LinearSVC classifier, I used *random_state*:42 for consistency and *max_iter*:10000 otherwise I would receive a convergence warning. Furthermore, all parameters seemed to not have a big impact on the performances except for parameter C, so I ran a more in-depth analysis around the maximum C=0.5 (as it can also be seen in Fig. 4) using cross-validation with $K = 5$. The new range of values I looked into are $[0.45 \rightarrow 0.65, step = 0.01]$. I found that the best performing value is still C=0.5, so I had found the best value.

I trained both models on the whole development set using their best configurations. The obtained fitted models have been used to label the evaluation set. After submitting, the public scores I obtained are 0.849 for LinearSVC and 0.832 for ComplementNB.

The scores obtained using the evaluation data reflect the scores obtained on the development data. Given the results, I can safely assume that the private score will reflect the public since there should be no overfitting.

For a final reference, I will propose two naive solutions that are:

- One-hot encoding the raw corpus of the tweet and classify with LinearSVC;
- Random guessing based on a Bernoulli random variable[5] with $p =$ probability of having a positive tweet among all tweets[6];

The public score obtained on the leaderboard are respectively 0.786 and 0.498.

## IV. Discussion

The proposed approach outperforms by far the random guessing solution, and it is a notable improvement compared to the approach of using the raw corpus of the tweet. The second naive solution shows how challenging this task was since, after all my preprocessing and tuning steps, I improved by 0.062, which is not a very big difference. Still, at the time of writing, my solution lacks from the first place of the leaderboard just 0.031 points.

One of the critical factors that increased my score is the prior knowledge of the user who wrote the tweet. This can be seen as an unfair advantage for students in the competition since the classifier (probably) will not perform so well on an unknown user. Instead, this can be legitimized if, for example, this kind of classifier is used by administrators of social media communities (in which the participants are known) to automatically flag tweets by having prior knowledge on who made the tweet.

My two selected classifiers perform similarly on this problem and can achieve satisfactory **f1** scores. Certain aspects that could be taken into consideration to further improve the score, for example:

- Try to adopt more advanced text mining techniques, such as **word embeddings**, which can exploit the potential of neural networks;
- Try to perform more feature engineering to extract more meaningful features (such as the feature I extracted *"contains_links"*);
- Try to perform a more in-depth grid search and explore a wider space of hyperparameters.
- Try to reduce dimensionality and apply the random forest algorithm again.

I think that there is some room for improvement, especially by leveraging techniques such as BERT [2], which was created and open-sourced by Google. It is widely used today to obtain state-of-the-art results on NLP[7]. It is a pre-trained model that we could fine-tune on our dataset and exploit for sentiment analysis.

## References


[1] J. D. Rennie, L. Shih, J. Teevan, and D. R. Karger, "Tackling the poor assumptions of naive bayes text classifiers," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, pp. 616–623, 2003.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.


[5] A Bernoulli random variable can assume value either 0 or 1 based on a given parameter p

[6] Computed as the number of positive tweets divided by the total number of tweets

[7] natural language processing