Sentiment Classification of Tweets

Pietro Cagnasso
Politecnico di Torino
pietro.cagnasso99@gmail.com

TABLE I
COUNT OF NULL AND UNIQUE VALUES ON A PER-DATASET BASIS.

	Development		Evaluation	
Field	**# null**	**# unique**	**# null**	**# unique**
sentiment	0	2	-	-
ids	0	224716	0	74977
date	0	189779	0	70508
flag	0	1	0	1
user	0	10647	0	10647
text	0	223106	0	74624

***Abstract*—Report on Data Science Lab's exam project. This project consists of a simple sentiment analysis of a series of tweets extracted from the well-known Sentiment140 dataset. This task is an internal competition between students to achieve the highest f1 score.**

I. PROBLEM OVERVIEW

The proposed task is the sentiment classification of a set of tweets, between positive and negative. We worked on two datasets: a labeled development dataset and an evaluation dataset with unlabeled tweets.

II. DATA EXPLORATION

The datasets are composed of 224994 and 74757 entries, respectively. In both datasets, we did not find any null value nor completely duplicated entries (all features with the same value).

Both datasets describe each entry with the same set of features: id, date, flag, user, and text. The development set also includes the target value sentiment.

A. Sentiment

The prediction of this feature is the target of our task. It represents the sentiment expressed in the text of the tweet. It is a categorical variable whose possible values are: 1 meaning positive, and 0 meaning negative. We have 130157 positives and 94837 negative tweets. This 37.2% unbalance among classes may lead to overfitting if this does not correctly represent the underlying distribution.

B. Ids

This feature represents the tweet's unique identifier. Despite its nature to be unique, as we can also see in Tab. I, we have fewer ids values than entries in the training dataset. With a manual inspection of those non-id-unique entries, we can see that those entries only differ in sentiment value. Thus, we should drop those entries to reduce the possible misclassifications. In Fig. 1 we can easily identify an area, corresponding to the highest values of id, in which only negative tweets occur. This feature should not carry much information, thus we drop it.

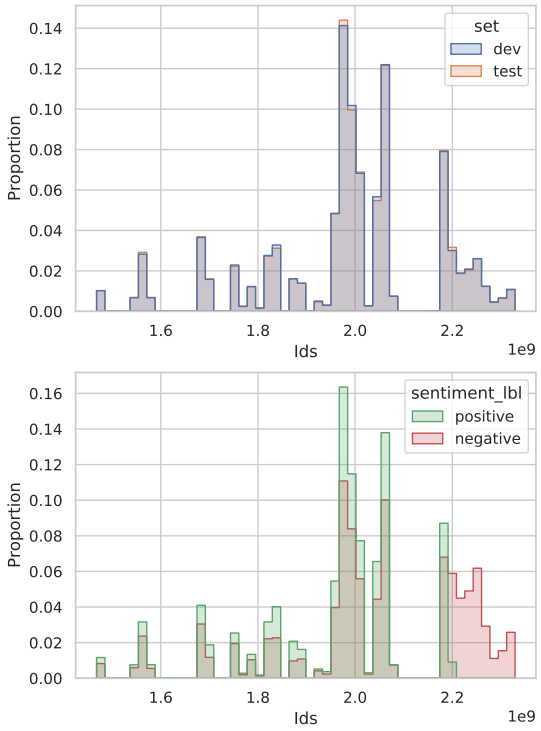

Fig. 1. Distribution of ids both by dataset and sentiment. The representation as proportion allows for direct comparison of the two categories.

C. Date

This feature represents the date on which the tweet was published. This gives us a time location of the tweet, in theory, this could carry some information (e.g., at Christmas we may expect to see more positive tweets than negative ones).

In Fig. 2 trends look very similar to the ones already seen in the case of ids. Analyzing the correlation between the id and the date in UNIX time[1] we can notice a value of 0.994. This means that ids are assigned progressively, and thus their inclusion would have brought information about time.

In Fig. 3 you can find a more detailed analysis of the entries' distribution by weekday and hour of the day.

[1]Seconds from January 1^{st} 1970.

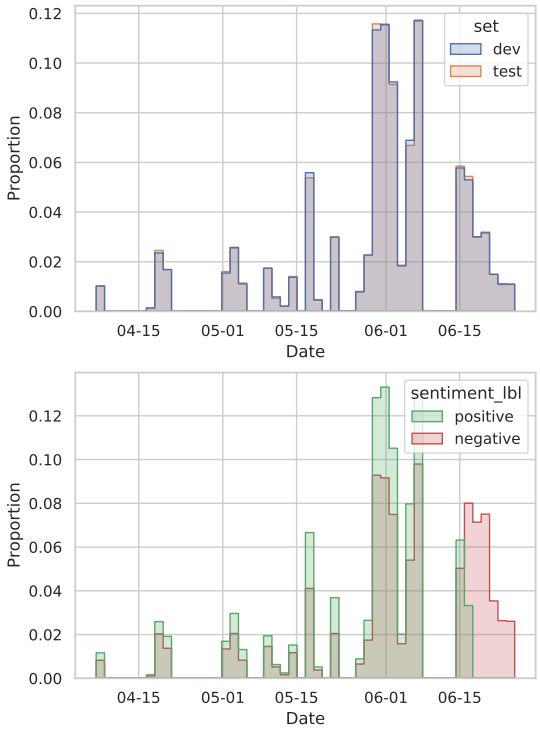

Fig. 2. Distribution of dates both by dataset and sentiment. The representation as proportion allows for direct comparison of the two categories.

D. Flag

This feature contains the same value: the string "NO_QUERY", which means that this feature carries no information.

E. User

This feature represents the user that wrote the tweet. In a broader context, we should avoid using this kind of information as it is (but a user profiling and a distance metric between the users based on this could be useful) since in general, this would not generalize at all. But in this case, we can notice that the two sets of users are the same. Thus, aiming at maximizing our chances of increasing the f1 score, we should include somehow the information about the user who wrote the tweet.

We have on average 21.132 ± 16.058 and 7.021 ± 5.416 tweets per user in the development and the evaluation set respectively. Such a high variability means that the information value carried by the user varies across the dataset.

F. Text

This feature is the main element of the analysis. The maximal lengths recorded are 359 and 296 characters for training and testing datasets. Since the maximal length allowed by Twitter was 140 characters per tweet until 2017 [1] this is an unexpected result. To partially justify this behavior we inspected the dataset using regular expressions founding HTML entities, URLs, user tags, and happy and sad emoticons (Tab. II).

TABLE II
COUNT OF CLASSIFIED STANDARD SEQUENCE OF CHARACTERS.

Field	Development	Evaluation
HTML entities	20003	6636
user tags	165807	55196
URLs	15167	5068
happy emoticons	3409	1068
sad emoticon	11806	3979

Using fastlangid we detected 127 languages, but more than 95% of tweets are in English. With a manual inspection of some tweets classified as Italian or French (languages we know), we found out that these are wrongly classified, so we can expect that most of those classified as non-English tweets are written in English.

III. Proposed approach

For reproducibility reasons, we carried out all tests using 20 as a seed for all the classes that make use of random number generation.

A. Preprocessing

We dropped the entries with duplicated ids and from the remaining dataset, we dropped the columns ids and flag.

1) Date: We have included this feature in our preprocessing in several ways: we have converted the date up to seconds in the corresponding UNIX time, extracted the weekday and the hour of the day, and finally added the date up to the day to the textual feature.

2) User: As done with the date we included this information in two ways. First, we added the name of the user to the textual feature. Secondly, we computed a feature called *user orientation* that should represent how much the user u is inclined to write positive or negative comments. We defined the orientation as:

$$o(u) = \frac{n_{u,+} - n_{u,-}}{\max_{u \in U} n_u}$$

3) Text: We have extracted from text information at two levels:

- *character* level: number of upper case characters, number of punctuation characters (in `string.punctuation`), number of exclamation and question marks and finally the number of words
- *tweets'* level: number of user tags, number of hashtags, number of URLs, number of happy and sad emoticons

Then we substituted happy emoticons with the string "_HAPPY_EMOT_" and sad emoticons with "_SAD_EMOT_". Finally, we added to the text the date up to the day and the user.

The resulting text was deprived of stopwords, POS-tagged and lemmatized using `nltk` [2]. The stopwords we used are a subset of nltk's stopwords for english that we selected to avoid losing elements of the text like negations.

To make the text usable for a model of choice we used tf-idf to move it into the numerical space and then SVD to reduce its dimensionality.

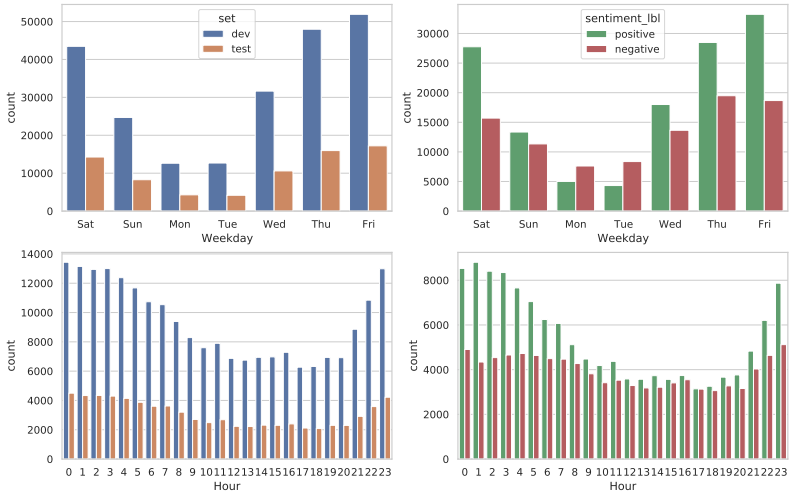

Fig. 3. Detailed analysis of weekdays and hour of the day distribution both by sentiment and dataset.

B. Tools

All the tools used are from the well-known framework Scikit-Learn [3].

`TfidfVectorizer`: (From now on also referred to as TFIDF) This class allows the transformation of a collection of documents into their tf-idf representation. Not just the pure tf-idf representation, but by tweaking the hyperparameters it is possible to produce pure tf or one-hot encoding representations. It produces a sparse matrix to have optimized management of memory.

`TruncatedSVD`: (From now on also referred to as TSVD) This class is used to reduce the dimensionality of the data by exploiting the SVD decomposition. We used SVD instead of PCA because it does not center the data and so does not require to allocate the memory necessary to fully store the matrix.

Applying SVD to a ft-idf matrix is part of the topic modeling area in natural language processing with the name of latent semantic indexing.

`StandardScaler`: We used this class to scale the dataset as needed by SVMs.

`RandomForestClassifier`: (From now on also referred to as RFC) This is a classification model based on decision trees. It implements several trees, each fitting a different portion of the features. Its nature allows the model to reduce the impact of overfitting. This benefit comes with remaining degrees of interpretability through global feature importance, resistance to noise and outliers, and high-speed classification.

`LinearSVC`: (From now on also referred to as LSVC) This is an SVM used for classification purposes. We used this version instead of the common SVC with the linear kernel because its implementation scales almost linearly to millions of samples and features [4].

C. Hyperparameter tuning

To tune the hyperparameter we considered these spaces as orthogonal to one another to reduce the number of possible combinations and so the amount of time requested to tune the models.

TFIDF: We have tuned this class on the following hyperparameter:

- `binary`: this hyperparameter allows to use either the term frequency or the one-hot encoding of terms.
- `used_idf`: used to decide whether or not to use the inverse document frequency weighting.

We used the hyperperameter `ngram_range` to compute the vectorization on both mono-grams and bi-grams together while keeping only the top 5000 occurring tokens as split by nltk's `TweetTokenizer`.

To choose the best TFIDF configuration we reduced the dimensionality to 500 features and used the two models in their default configurations.

With model-optimal hyperparametes for TFIDF we reduced the dimensionality of the resulting matrix to 500 using TSVD. Then we scaled the numerical features with the highest importance from the RFC point of view Fig. 4, added them to the vectorized text, and finally tuned both models on the resulting dataset.

RFC: For this model we tuned as hyperparameters:

- `n_estimators`: this is the number of trees composing the forest. In general for this features the higher, the better: we tested 200, 300 and 400.
- `criterion`: this parameter is used to define the function to use when computing the quality of the split. We tested both gini and entropy.
- `max_features`: this is the number of features to look at while deciding the optimal split. We tested the square root and the $\log_2$ of the number of features.

LSVC: To tune this other model we used these hyperparameters:

- `C`: this is the regularization term used to define how much "hard" the margin should be. We tested values from 0.1 to 1.0 with a step of 0.02.

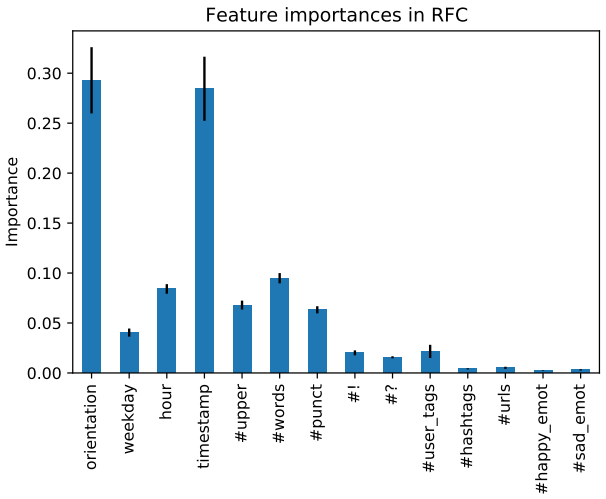

Fig. 4. The numerical features we computed with the corresponding importance given by an RFC. Among those, we picked the ones with an importance over 0.05: orientation, hour, timestamp, #upper, #words, #punct.

TABLE III
PREPROCESSING OPTIMAL HYPERPARAMETERS CONFIGURATIONS BY MODEL.

Name	RFC	LSVC
TFIDF.`binary`	True	False
TFIDF.`lowercase`	True	
TFIDF.`max_features`	5000	
TFIDF.`ngram_range`	(1, 2)	
TFIDF.`use_idf`	False	
TSVD.`n_components`	500	

- `class_weight`: this hyperparameter is used to adjust the value of C to the weight of each class. We tested both the same weight to each class and a balanced version weighting each class with the inverse proportion of its frequency in the dataset.

As suggested by scikit-learn's documentation we just used the dual problem since the number of features was less than the number of entries.

IV. Results

From local tests, using 75% of the development dataset as training set and the rest as testing set, we found out as optimal hyperparameters for the preprocessing pipeline the one reported in Tab. III.

Using these values of hyperparameters we obtained as optimal configuration of each model: `RFC(criterion="entropy", max_features="sqrt", n_estimators=400)` and `LSVC(C=0.32, class_weight=None, dual=False, max_iter=5000)`.

Exploiting these configurations we obtained as f1-score:

RFC: local tests 0.85213, public leaderboard 0.85208

LSVC: local tests 0.85218, public leaderboard 0.85202

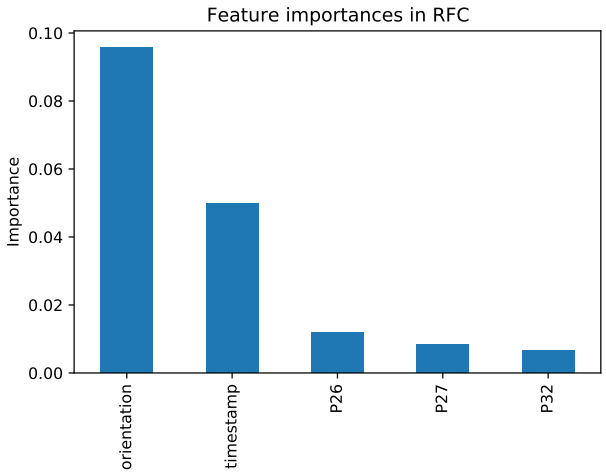

Fig. 5. The most relevant features according to their average importance defined by the ensemble of trees in the RFC.

V. Discussion

Based on the small difference between the results obtained in the local tests and the public leaderboard we can assume that our models did not overfit and the data preparation we did was successful in reducing this eventuality.

As we can see from Fig. 5 the orientation feature we engineered and the timestamp played a key role in the classification. Since we had no overfitting issue we can state that these are in fact good quality features.

To enhance the score of this classification we could have done (out of the scope of the course):

- use of word (e.g. Words2Vec [5]) or sentence (e.g. Doc2Vec [6]) embeddings, to synthesize even better the content of the text some transformer-based tools could have been even more effective (e.g. BERT [7] and its specialization for Tweets BERTweet [8])
- use of more complex and less interpretable, but more capable models (e.g. neural networks)

References

[1] A. Rosen and I. Ihara "Giving you more characters to express yourself", https://blog.twitter.com/en_us/topics/product/2017/Giving-you-more-characters-to-express-yourself
[2] S. Bird, et al. "Natural language processing with Python: analyzing text with the natural language toolkit", O'Reilly Media
[3] F. Pedregosa, et al. "Scikit-learn: Machine Learning in Python", Journal of Machine Learning Research 2011, vol. 12, pp. 2825–2830
[4] "Support Vector Machines", https://scikit-learn.org/stable/modules/svm.html, 1.4.4 Complexity
[5] T. Mikolov, et al. "Efficient Estimation of Word Representations in Vector Space"
[6] Q. Le, T. Mikolov "Distributed Representations of Sentences and Documents"
[7] J. Devlin, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding"
[8] D.Nguyen, et al. "BERTweet: A pre-trained language model for English Tweets"