Sentiment Analysis of Twitter data

Francesco Lacriola
Politecnico di Torino
Student id: s292129
francesco.lacriola@studenti.polito.it

***Abstract*—This report focuses on sentiment analysis, a natural language processing technique that identifies and extracts subjective information through contextual mining of text. It is often useful when the analysis is conducted on tweets where the point of view and contentions are remarkably unstructured and heterogeneous. In this paper, we show our proposed feature selection and feature engineering steps as well as a suited text cleaning stage, and we conduct a comparative analysis of machine learning algorithms for sentiment classification, like Logistic Regression, Decision Trees and Support Vector Machine, together with evaluation metrics.**

I. Problem overview

The addressed task is a binary classification problem on a dataset containing tweets written by random users during a 3 months period in 2009. The objective of the analysis involves classifying opinions in the text of tweets as "positive" or "negative". The whole assigned dataset is split into two chunks:

1) a development set, with 224,994 records containing tweets for which the true sentiment is provided;
2) an evaluation set, containing 74,999 records without the target sentiment, but with the same other attributes.

Let's analyze the kind of attributes we are dealing with:

- *sentiment*: this is the target variable; 0 if the global opinion in the tweet is negative, 1 if it is positive
- *ids*: each tweet has a unique identification number
- *date*: the date on which the tweet has been posted
- *flag*: an attribute containing the same value for every record
- *user*: categorical attribute containing the username of the person who wrote the tweet
- *text*: the raw text of the tweet

According to preliminary checking, 556 records, with the same id and content but different label have been found. Duplicated rows are often present in this kind of dataset taken from rich sources and this presence could bias the models. Missing values are not present; they could have been impacting the results of the machine learning models and/or reducing their scores.

An exploratory data analysis to derive a first overall picture of the data and to discover important insights, which we might be exploiting in further phases, has been conducted. Overall, there is some imbalance in the sentiment target distribution. In particular, the positive records are 130,157 whilst the negative ones are 94,837.

Fig. 1. Wordcloud of the merged datasets before preprocessing

Including also the development set in the analysis, when possible, will make us understand how, in general, the data is distributed. That is what we did to generate the word-cloud in the Figure1, specifically we used a dataset given from the union of the development and evaluation sets. One important thing one can notice from this image is that HTML character entity references, like *&quot* and *&amp*, are two of the biggest represented words, which means they are among the most frequent ones; therefore, this suggested to us to clean up the text from all HTML references.

For which regards the *user* column, we find out that unique usernames were present in the same number in both the evaluation and development set, namely 10,647; this fact could not have been only a coincidence and led us to consider the opportunity to use this attribute as a feature for our models.

Considering the *date* column, since the dataset considers only three months, we supposed that sentiments could be highly correlated with time periods. This assumption was confirmed from the frequency plots we used to perform our analysis: since raw dates, in the format they are provided in the dataset, are too dense to be considered as single features, we extracted 4 different features from each record converting to the *datetime* format using pandas: the date (in the format yyyy-mm-dd), the day of the week, the month and the hour. Therefore, as we can see in Figure2, plotting the date-sentiment frequency plot, we find out that there are some dates in which only records of negative tweets are present. This might be an important discriminating factor between negative and positive tweets and can help to mitigate the imbalance issue. From the frequency plot for the day of the week, for example, we

can see that this can be really informative because negative tweets are fairly more than positive ones on Wednesday and on Thursday. From the same data visualization applied to the month column, we advise that records are equally numerous in July, whilst in June positive tweets doubled negative ones. Visualizing the hour-sentiment distribution, instead, we can see the tendency of positive tweets to go far beyond the negative ones at night time. All this information brought us to further inspect the effect of adding these four features.

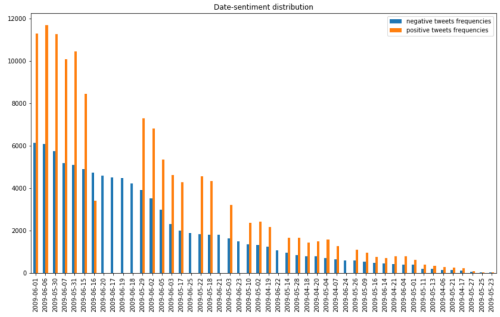

Fig. 2. Sentiment frequencies with respect to the date

II. Proposed approach

A. Preprocessing

Most of the decisions taken for our preprocessing pipeline come up from our previous analysis. First of all, we use the *Beautiful Soup* [1] Python package to parse the remaining HTML references in our texts, in particular, we use the faster lxml's HTML parser version. However, this does not eliminate the non-ASCII characters that were still present in our documents, so a subsequent encoding to *latin1* according to ISO/IEC 8859-1, a series of standards for 8-bit character encodings, and consequent decoding to the UTF-8 standard is needed.

Next, a couple of cleaning steps tailored for the analysis of Twitter data are performed. After lower-casing every letter of the processed text, exploiting our pre-existing knowledge of the social network, we decide to address hashtags and mentions handling. For the former, we remove the hash symbol from every string and we do the same for the latter, replacing every '@' with white space. Another approach is carried out for URLs handling: we replace every link with a keyword, 'URL'; this has shown to slightly improve performance than merely removing URLs. Regular expressions are also implemented to identify repeating characters or letters in a single word, which is something commonly found on social medias' texts. We bring to two the number of repeating letters. Moreover, punctuation handling proved to be unnecessary; one can argue that this kind of cleaning would have removed the emoticons from the text, but, since the frequency of emojis in this dataset is low, it is very likely that some punctuation does impact on the identification of the sentiment of a sentence.

For the sake of completeness, lemmatization and stopword removal were also tested. The first technique considerably worsens the scores. Also using pre-compiled lists of stopwords negatively impacts the performance of our Twitter sentiment classification approach. This method is based on the idea that discarding non-discriminative words reduces the feature space of the classifiers and helps them to produce more accurate results. Regarding sentiment analysis of Twitter data, this technique obtains contradictory results [2]. In our case, we can say that stopwords indeed carry sentiment information and removing them harms the performance of our models.

To prepare our cleaned textual data to be fed to our models, we used the *TweetTokenizer* from the Python's library, Natural Language Toolkit, or more commonly NLTK [3], mainly used for natural language processing for English written text. Tokenizing is a way of separating a piece of text into smaller units called tokens; tokens can be either words, characters, or subwords. This Twitter's slang-aware tokenizer marginally improved the accuracy compared to *WordPunctTokenizer*. Our proposed solution consists on the utilization of this tokenizer together with a term frequency-document frequency (tf-idf) vectorizer provided by sklearn library. The tf-idf is a weighting scheme that assigns to a token t a weight in document d given by:

$$tfidf_{t,d} = tf_{t,d} * idf_t \tag{1}$$

In other words, tf-idf assigns to term t a weight in document d that is:

1) highest when the term occurs many times within a small number of documents
2) lower when the term occurs fewer times in a document, or occurs in many
3) lowest when the term occurs in virtually all documents [4]

In our method we limit the tf-idf vector representation to the first N words, using the *max_features* parameter of the sklearn's *TfidfVectorizer*. To prevent including too frequent and too rare words that would not be so informative, suitable *min_df* and *max_df* parameters will be considered hyper-parameters, together with N and the *ngrams_range*, which is in charge to control the minimum and the maximum number of words for each token, and has been adjusted in the hyperparameters tuning phase.

For which regards the features selection stage, for the reasons we mentioned before, we discard the *flag* column because it does not provide any informative content and the *ids* column because such unique identifiers are able to contribute rather in a time-series context. Therefore, together with the processed text, we use the *user* column and the four columns extracted from the original *date* one: *date* (yyyy-mm-dd format), *day_of_the_week*, *month* and *hour*. For each of these 5 additional columns we apply one-hot encoding; this allows machine learning algorithms to manage the information into categorical values without the confusion caused by ordinality.

Two techniques to cope with the dataset's imbalance were tested: SMOTE [5] and *RandomOverSampler*, but, besides

being computationally expensive, they require additional hyperparameters tuning, without noticeably impacting on the final results.

B. Model selection

The following algorithms have been tested:

- *Random forest*: this algorithm creates decision trees on randomly selected data samples, gets a prediction from each tree and selects the best solution by means of voting. This typically prevents the overfitting problem of decision trees, despite losing some degree of interpretability.
- SVM: this algorithm identifies the maximum-margin hyper-plane that separates two classes. We look for the closest points to the separating hyper-plane. These points are called support vectors. Computing the distance between the plane and the support vectors, we calculate the margin. The goal of the algorithm is to maximize this margin. Using kernels you can also make use of SVM for non-linear classification, mapping model inputs into a multidimensional space.
- *LinearSVC*: since the dataset is very large and the algorithm provided by *sklearn* to fit a support vector machine requires a huge training time, we evaluated also the LinearSVC model that should better scale to large numbers of samples.
- *Logistic Regression*: this is a nonlinear regression model used when the dependent variable y is dichotomous, i.e. it takes two values. The goal of the model is to establish the probability that an observation can generate one or the other value of the dependent variable, estimating the probabilities through a logistic function, or sigmoid function. The function is an S-shaped curve that can take any real value number and map it to a value between 0 and 1, excluding boundaries:

$$y = \frac{e^{b_0+b_1*x}}{1+e^{b_0+b_1*x}}. \tag{2}$$

  The coefficients b_0 and b_1 are estimated with the Maximum Likelihood Estimator, finding the values that minimize the error of the probabilities predicted by the model with respect to the class in the dataset.

Comparisons between models' results are illustrated in Table2. Besides its training being remarkably slow, we saw that Random Forest's performance are not comparable to the ones achieved by Logistic Regression, even though they increase as we rise the number of estimators. In addition, SVM's training time, given the dimension of our dataset, is too large for performing an hyperparameters tuning stage on it. Given these considerations, we chose Logistic Regression because, besides having the best results, it permits us to rapidly asses our choices.

C. Hyperparameters tuning

The whole hyperparameters tuning stage has been undertaken performing grid searches with sklearn's *GridSearchCV*. Even though this is a hugely time-consuming operation, a cross-validated grid search over a parameter grid is one of the most powerful techniques to estimate the skill of the model on unseen data. In our pipeline, we decide to run a 5-fold cross-validation for each of our candidate models, setting the *cv* parameter to 5. This phase can be divided into two steps, each of them corresponding to a different purpose:

- *Preprocessing tuning*: as we said before, we tune the *max_feature* (N), *min_df*, *max_df* and *ngram_range* parameters of *TfidfVectorizer*. The first one is the most important because it allows building a features vocabulary that only considers the top N tokens ordered by term frequency across our whole corpus. When dealing with huge datasets and computational power limitations are present, this parameter guides the trade-off between performance and memory consumption. To assess the performance of this preprocessing pipeline we will fit a *Logistic Regression* model on top of the features generated with *TfidfVectorizer*.
- *Model tuning*: After finding the best set of hyperparameters for the first group, on the basis of these we maximize the scores tuning the second group. As concerns Logistic Regression, it does not have any critical hyperparameters to tune, however, regularization sometimes might be useful. Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. It generally refers to the concept that there should be a complexity penalty for more "remote" parameters (b_0 and b_1 for Logistic Regression). The idea is that just looking at the training data and not paying attention to how far the involved parameters are, leads to overfitting. The parameter C of the sklearn's *LogisticRegression* classifier balances this trade-off.

Results for the tuning stage are summarized in Table1.

III. Results

All our experiments are 5-fold cross-validated using *cross_val_score*; the results into tables are the mean of the test scores of every partition. Since the metric with whom our proposed approach is evaluated is the F1 score, this is also the only benchmark we take into account in the report.

TABLE I
HYPERPARAMETERS TUNING

Step	Parameter	Candidate values	Best value
tf-idf	max_features	{40k, 60k, 80k, 100k}	60k
tf-idf	ngram_range	{(1, 2), (1, 3)}	(1, 3)
tf-idf	max_df	{0.3, 0.5, 0.7}	0.5
tf-idf	min_df	{1, 2, 3}	1
LR	C	{np.logspace(-4, 4, 20)}	1.624

The first table summarizes the grid search conducted for our preprocessing step. As we can see, the best value for the size of our vector representation of text is not the highest one, which characterizes the most memory demanding option and takes into account more words to discriminate between sentiments,

but 60k. This happens because sometimes it is not effective to transform the whole vocabulary, as our data may have some exceptionally rare words. The second best parameter tells us that our model will look at the frequency with which up to 3 words occur together. The value for the maximum document frequency, 0.5, claims that words with a document frequency above this threshold are most likely misleading. The best resulting value for the minimum document frequency threshold is the default one.

For which regards Logistic Regression tuning, the tested values for the inverse of regularization strength are numbers spaced evenly on a log scale, because variations in results tend to vanish as C increases. The best value is really near the default one (1) so it does not improve the score.

TABLE II
MODELS' RESULTS

Model	Training time	BL	TC	PP	HT
Random Forest	439s	0.758	0.759	0.782	0.791
Logistic Regression	71s	0.783	0.789	0.846	**0.856**
SVC	6h	0.792	/	/	0.849
LinearSVC	56s	0.780	0.786	0.841	0.844

The second table shows the f1 score achieved by our tested models with different settings:

- BL: baseline, results without any text cleaning and using only the text attribute
- TC: results with tokenization and using only the cleaned text column
- PP: results after the whole preprocessing stage, including the selected features
- HT: results after the hyperparameter tuning step

We decided not to test SVC with the TC and PP settings for time reasons. Random Forest has been trained with 50 estimators.

Despite outperforming other models, as we can see from the confusion matrix in Figure4, Logistic Regression's recall for the negative class is quite low (0.80), while the precision for the negative class is high (0.86). This means the classifier is very picky and does not think many things are negative.

The public score on the competition leaderboard, 0.858, is in the top ten scores and it is not so far from the cross-validated result; this proves that cross-validation is the best option to asses the true performance of our models and we are confident that our private score will be comparable.

IV. DISCUSSION

With the shown results, we have proven that every model benefits both from our text cleaning. We demonstrated that the assumptions made during the explorative analysis were legitimate and that our feature selection is a crucial factor to improve the f1 score. What strikes at first glance is that using the precisely formatted date as a feature helped so much; this happened because the dataset we have been dealing with was restricted to a three months period, whilst for bigger dataset things can be different. Moreover, interesting insights

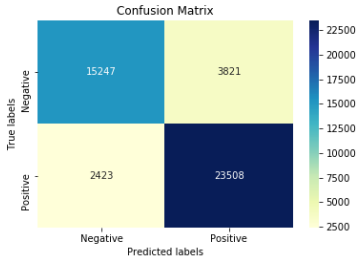

Fig. 3. Confusion matrix for the final Logistic Regression classifier

turned out from the hyperparameters tuning stage, in particular from the vectorization of texts: our models benefit from a limited number of features, which prevents the highest level of overfitting that we reach using all the words met when fitting the vectorizer; most frequent words, according to tf-idf, instead, resulted to be misleading in our analysis, even though removing the commonly used 'stopwords' leads to poorer results.

From the comparison between models, we have shown the power of Logistic Regression and that one can achieve these results with very short training and inference time. However, our model is still affected by overfitting, but this is inevitable for two main reasons:

1) The size of our dataset and the fact that it is not "general" but it related only to a three months period
2) This is the natural behaviour of Logistic Regression. The more data we have, the more likely it is that it will be possible to perfectly separate the two sets of values. As a result, overfitting becomes more of a problem when you have many "predictors".

From Figure4 we can see how our model would benefit from more data, since the overfitting diminishes when we use an higher percentage of the original dataset.

Further performance optimization might be achieved with word embeddings, like Word2Vec [6], or exploiting transformers; the latter are neural network based approaches that introduce an 'attention' mechanism [7] that takes into account the relationship between all the words in the sentence and create weightings indicating which other elements in the sentence are most critical to the interpretation of a problem.

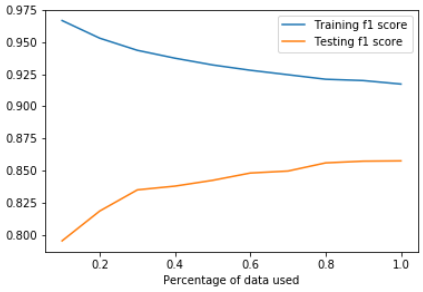

Fig. 4. Logistic Regression's scores with respect to percentage of data used

REFERENCES

[1] L. Richardson, "Beautiful soup documentation," *April*, 2007.

[2] H. Saif, M. Fernandez, Y. He, and H. Alani, "On stopwords, filtering and data sparsity for sentiment analysis of Twitter," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, (Reykjavik, Iceland), pp. 810–817, European Language Resources Association (ELRA), May 2014.

[3] E. Loper and S. Bird, "Nltk: The natural language toolkit," *CoRR*, vol. cs.CL/0205028, 2002.

[4] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press, 2008.

[5] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[6] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.