Tweets Sentiment Analysis

Giuseppe Concialdi
Politecnico di Torino
Student id: s294666
giuseppe.concialdi@studenti.polito.it

Christian Montecchiani
Politecnico di Torino
Student id: s303681
christian.montecchiani@studenti.polito.it

***Abstract*—In this report we developed a model that is able to classify the sentiment of a Twitter's tweet. We focused on text cleaning and information retrieval within the tweets employing advanced feature extraction techniques. We exploited high-performance algorithms that could handle such a big dataset and could accomplish the classification in a reasonable amount of time.**

I. Problem overview

The objective of the competition was to build a model that is able to classify whether a tweet contains positive or negative sentiments. The dataset provided is arranged as follow:

- A **development** set composed by 224,994 records of tweets. Each sample has six different features, including the *sentiment* attribute that is the target of the classification.
- A **evaluation** set consisting of 74,999 samples. Its dimension is one-third of the development set and it does not feature the target variable.

Every sample is characterized by:

- *ids*: the unique identifier of the tweet. It is represented by a progressive integer number that is related to the timestamp of the tweet. By digging in the past Twitter documentation, we found out that the tweet id is generated in a snowflake format [1], invented by Twitter for the generation of sequential ids for their tweets. The timestamp is encoded within the id as shown in Figure 1.

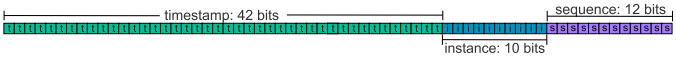

Fig. 1: Structure of a 64-bit snowflake id.

- *date*: the timestamp of each tweet. The date is encoded as a string with the format:

  $$weekday \quad month \quad day \quad hour:min:sec \quad tz \quad year$$

  The dates in the dataset range from April 6 to June 25 of 2009.
- *flag*: a string whose significance is unsure. It is featured in the whole dataset with the unique value of `NO_QUERY`. Given its absence of meaning, this feature will be removed without a second thought, but it is possible that it would report the query used to retrieve the tweets when they were extracted using some Twitter APIs.
- *user*: the username of the creator of the tweet. Even though there are almost 225 thousand tweets, there are only 10,647 different usernames.
- *text*: the text of the tweet. This is the core part of the analysis, it embodies a lot of insights that can be extracted and analyzed to retrieve the overall polarity of the tweet's sentiments. In 2009 the maximum length of a tweet was 140 characters [2]. However Figure 2, shows that some tweets exceed this threshold, so it is likely to have issues with the encoding of the text extracted.
- *sentiment*: the target variable of the classification. It assumes two possible integer values: 0 and 1 that represent respectively negative and positive sentiments. The dataset is fairly unbalanced, as shown in Figure 3: there are more positive sentiments than negative ones.

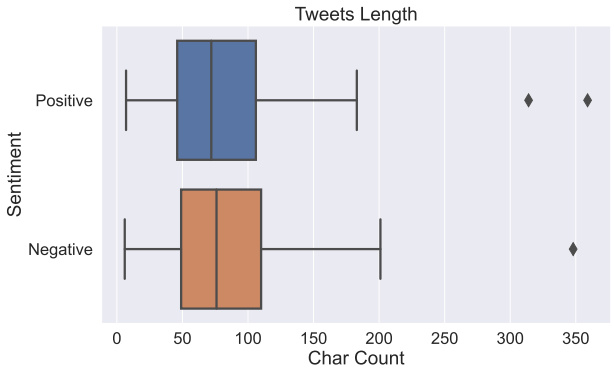

Fig. 2: Boxplot showing the number of chacters of the tweets per sentiment

The dataset does not feature any missing value, but it contains redundant information that is useless for the analysis.

II. Proposed approach

The preprocessing was characterized by the features extraction and the text mining phases.

We decided to extract the time-related information from the *date* attribute and figure out later if their relative importance would matter.

Although the *date* attribute could hide some useful insights about the sentiment of the tweet, the major knowledge lies within the *text* attribute. The information extraction from this feature can be performed in different ways and with different

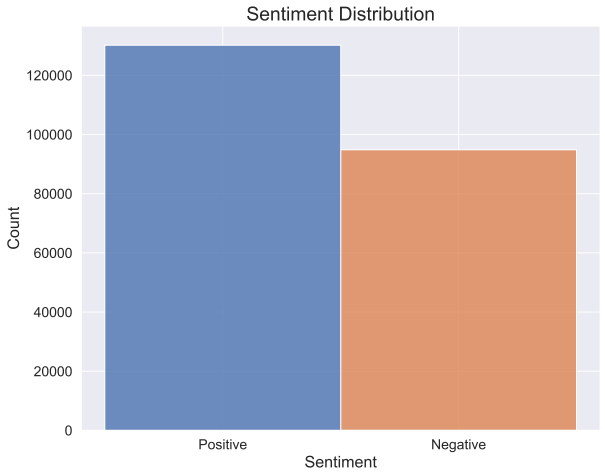

Fig. 3: Distribution of the tweets' sentiments in the training dataset

processes. We decided to tackle the problem from different points of view and we managed to embed all this data into the model. Firstly, we cleaned up the tweets' text, then we exploited the text performing:

- A **Sentiment Intensity Analysis** [3] on the text, obtaining new features on the polarization of the sentiments.
- A **TF-DF** of the text in order to get the words that mainly influence the sentiments of the tweets.
- A **Word Embeddings** [4] approach with the FastText [5] library to retrieve the morphological relationships between the words in a sentence.

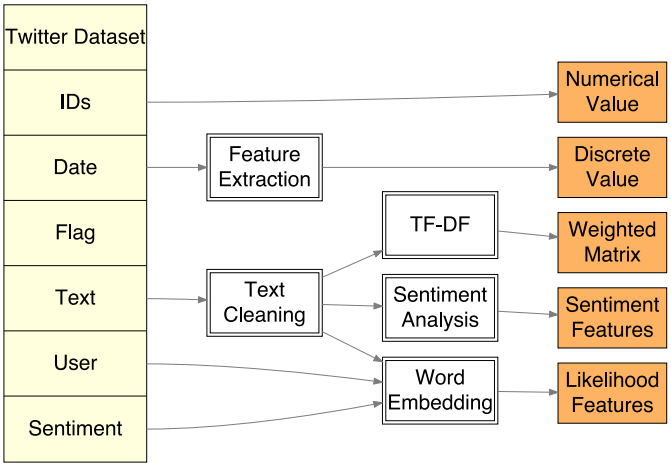

Fig. 4: Overall schema of the problem approach

Figure 4 shows a summary of our approach. The last technique resulted in a very powerful tool, that is able, on its own, to perform the classification of the sentiment of the evaluation set with a macro F1-score[1] higher than 0.8.

A. Preprocessing

The preprocessing step is the core phase of the entire analysis. Here we performed the extraction, transformation and normalization of the features. As discussed in Section II, we extracted from the *date* attribute all the relevant information and we discarded the properties that are redundant like *year*, *timezone* and *flag*. Then we decided to add a new feature based on the number of characters of the tweet. The new *char_count* attribute is useful to troubleshoot the tweets' texts that are longer than 140 characters and it is also valuable for the classification itself.

Afterwards, we addressed the tweet duplication issue. There are three kinds of duplicated tweets:

1) Same *text*, **different** *author*, **same** *sentiment*
2) Same *text*, **same** *author*, **same** *sentiment*
3) Same *text*, **different** *sentiment*

For the first kind of duplicated tweets, no action is required. Those are very common and eventually short sentences that different authors twitted. They share the same sentiment so it enforces the model to learn that those phrases belong to one specific class. The second kind are tweets that have been posted more than once by the same author. This could be a mistake or maybe the user wanted to retweet the same post. Either way, we decided to keep only one copy of the duplicated post. The most controversial kind of duplication is the last one: regardless of the author, if the same text is labelled differently, means that the data is mistaken. For this reason we decided to drop all the records with these conflicting properties.

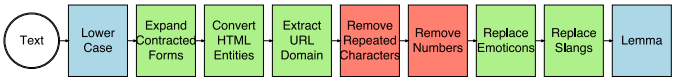

Fig. 5: Text cleaning pipeline

Then, we tackled the cleaning of the tweets' text. To properly perform this step, we took into consideration a lot of alternatives. Different elements characterize a Twitter post and we tried to draw every possible bit of information from the text. The tweets' length issue, led us to inspect the text in search of an encoding-related problem. We found out that the text extracted seems to be encoded in UTF-8 [6] but some special symbols were parsed as HTML entities[2]. We employed the html library to clean up the text. After that, we lowered the case of the words and we extracted the domain name from the urls contained in the tweets. We exploited two vocabularies[3] to expand into clear words the emoticons and the slang terms. By substituting these pieces of text in addition with some regex to regularize the phrases we obtained a much more coherent and

[1]The F1-score is the harmonic mean of the precision and recall therefore it is defined for a single class. With macro F1-score we refer to the mean of all classes' F1-scores.

[2]Character entities are used to display reserved characters in HTML

[3]We reworked existing libraries to be more suitable for the preprocessing. The `slangs`, `contractions` and `stopwords` dictionaries derive respectively from **Deffro**'s repository [7], **pycontractions** and **NLKT** libraries.

convenient format for the analysis. Ultimately, we applied the lemmatization [8] to extract the lemma from each word. We preferred lemmatizing to stemming [9] in order to preserve the morphology of the words [10].

Figure 5 shows the cleaning pipeline for the *text* attribute.

At this stage, the *text* feature is ready to be processed. We heavily relied on the tools offered by the NLKT library [11] to carry on the analysis. We employed the Vader Sentiment Intensity Analyzer [12] to gain the relative frequencies of *positive*, *neutral* and *negative* terms. Besides, also the *compound* value is stored, which is a single value that summarises the three frequencies. Following, we used the TextBlob library [13] to extract additional features: the *subjectivity* score and the *polarity* index. These variables are correlated among them, but they describe the same phenomenon under different points of view and complement each other.

Going forward, we applied a TF-DF to extract words with a relative high minimum support. Our aim was to retrieve the words that were present in a lot of tweets and which meaning was heavily polarized toward one class. In this case, we did not use a normal stop words vocabulary, because there are a lot of common term that are very important for the classification, like the negations. So we created a custom vocabulary that contained only conjunctions and some adverbs. The most frequent word resulting from the TF-DF are shown in Figure 6.

Fig. 6: Word Cloud generated by the results of the TF-DF. The figure has been created using the wordcloud library [14]

Finally, we leveraged the word embeddings approach through Meta's FastText library. One of the strengths of this technique is that it does not suffer the presence of unfamiliar terms, because it treats the words as multi-dimensional vectors. The outputs of the neural network are the predictions and the softmax likelihoods for each class. We extracted these probabilities and we integrated them into the dataset.

We tried to operate with the *user* attribute to retrieve some useful information within it. We thought that due to the relative low cardinality of this attribute (only 10,000 different usernames) the tweets posted by the same author may reflect its personality, thus it is likely that a sentiment is predominant to the other one. Nonetheless, the One-Hot-Encoding [15] of this features would have resulted into a huge dimensionaly increase of the dataset. So, we chose to incorporate the user author of the post at the beginning of the text of the tweet. The text already contains mentioned user whose name is preceded by an at-sign (@). In this way, the word embedding classification will retrieve information about the author of the post like if he was mentioned within it.

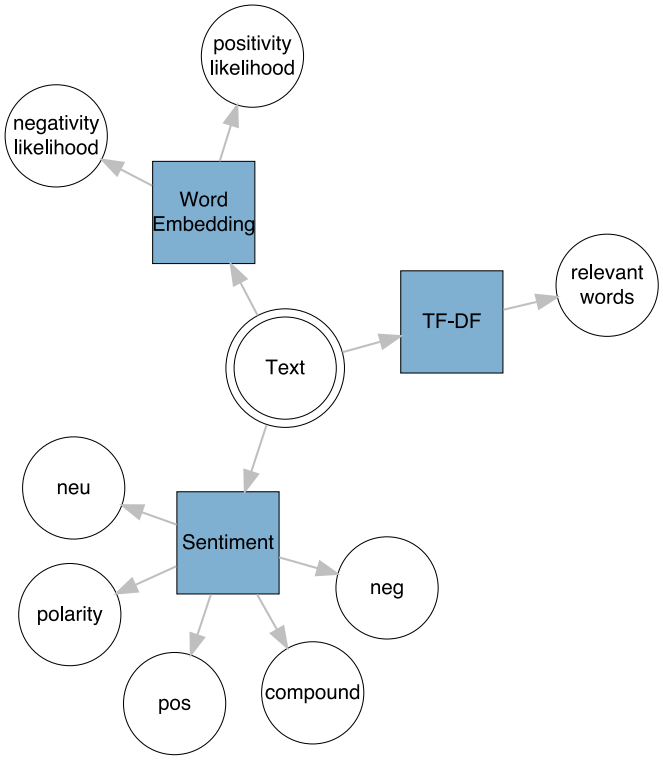

Fig. 7: Feature extraction schema of *text* attribute

Figure 7 summarises the operations performed on the *text* attribute.

The data is not yet ready to be processed by a classification algorithm. The categorical attributes need to be addressed. We removed the *text* and the *user* features because we have already extracted the information needed and then we proceeded to map the remaining features into numerical ones. All the columns are normalized with a MinMax scaler [16].

B. Model selection

We chose four different algorithms to perform the classification task and we compared their performances to assess which are the best classifiers:

- **Random Forest Classifier [17]**: ensemble of decision trees.
- **Bernoulli Naïve Bayes [18]**[4]: classifier for multivariate Bernoulli [19] models.

[4]We tested the Naïve Bayes Classifier and, as we expected, it was the worst classifier. This shows that the features are indeed correlated

- **Linear Support Vector Classifier [20]**: linear kernel version of the more general SVC.
- **Histogram-based Gradient Boosting Classification Tree [21]**: high-performance implementation for big datasets of the Gradient Tree Boosting Classifier [22].

To test the effectiveness of our preprocessing, we tried to test the performances of the models by increasingly combining the results of the different techniques applied to the *text* attribute. As it is possible to see in Table I, at first we applied only

Techniques	Classifier	F1-score
Sentiment	Linear SVC	0,723
	Random Forest	0,737
	Bernoulli Naive-Bayes	0,663
	Hist Gradient Boosting	0,749
Sentiment + TF-DF	Linear SVC	0,762
	Random Forest	0,780
	Bernoulli Naive-Bayes	0,713
	Hist Gradient Boosting	0,786
Sentiment + TF-DF + Word Emb.	Linear SVC	0,896
	Random Forest	0,903
	Bernoulli Naive-Bayes	0,713
	Hist Gradient Boosting	0,904

TABLE I: Results of the model selection phase.

the sentiment extraction technique, then we joined it with the TF-DF outcomes and in the last trial we added the word embeddings' likelihoods into the equation. It is possible to see an increase in the performance by matching the features. The highest score is reached by bringing together all the extracted features with the different methods.

During this phase, we exploited the Random Forest feature importances to understand what are the most relevant features upon which the algorithm is splitting the samples. Accordingly to our results, we found out that the most significant attributes are the likelihood resulting by the word embeddings classification, following there are the sentiment features and lastly some very polarized word coming from the TF-DF. We also discovered that the *ids* attribute was always more relevant than the features extracted from the date. However, we decided to keep those features because their relevance was still high.

C. Hyperparameters tuning

From the model selection phase, the best dataset turned out to be the *word embedding + sentiment extraction + TF-DF*, therefore the models have been working with it.

We decided to tune just the two best performing models from Table I: **Random Forest Classifier** and **Histogram-based Gradient Boosting Classification Tree**. The hyperparameters that we have taken into account for the tuning phase are reported in Table II.

To tune our models we used a hyperparameter optimization technique faster than the Grid Search, called Halving Grid Search [23]. It tries all the candidates with a small amount of records of the training set and iteratively selects the best candidates, using more and more records [24]. To highlight the speed of this technique we employed the same parameters grid with three folds. The results are exposed in Table III

Classifier	Parameters	Values
Random Forest	*max_features*	{log2, sqrt}
	criterion	{gini, entropy}
	min_samples_leaf	{10, 20, 30, 40, 50, 60}
	min_samples_split	{2, 3, 4, 5, 6, 7, 8, 9}
Hist Gradient Boosting	*max_iter*	{150, 200}
	max_leaf_nodes	{None, 20, 30, 40}
	min_samples_leaf	{2, 4, 10}
	early_stopping	{True, False}
	l2_regularization	{0, 0.1, 0.2, 0.3}

TABLE II: Hyperparameters configuration considered

Tuning technique	Time	Best private score	Best public score
Halving Grid Search	12m 24s	0,9036	0,844
Grid Search	4h 23m 5s	0,9039	0,845

TABLE III: Comparison of the performances of the tuning techniques

III. Results

In this section are reported the main outcomes of the hyperparameters tuning phase. Table IV showcases the best

Classifier	Parameters	Values	F1-Score
Random Forest	n_estimators	500	0,845
	max_features	log2	
	min_samples_leaf	10	
	min_samples_split	9	
Hist Gradient Boosting	max_iter	150	0,846
	max_leaf_nodes	30	
	min_samples_leaf	4	
	loss	binary_entropy	
	early_stopping	False	
	l2_regularization	0.3	

TABLE IV: Results of the models with the best hyperparamters configuration

configuration and the public F1-scores results for each model. We decided to submit the evaluations performed by the Hist Gradient Boosting Tree Classifier.

IV. Discussion

This classification task extensively relied upon Natural Language Processing libraries. Each library offers a wide range of tools that allows to effectively retrieve information from the text. We decided to cover more NLP libraries to benefit from the variety of materials at our disposal. Almost every library provides advanced features, such as Part Of Speech Tagging, that could have been beneficial to the classification but we decided not to include in this analysis.

All the classification algorithms, and the majority of utilities employed in Section II-B and II-C come from the scikit-learn packages [25]. Alongside the classical classification algorithms, we decided to try the Histogram-based Gradient Tree Boosting Classifier because, like the Random Forest Classifier, it is an ensemble of decision tree. We wanted to explore how the boosting technique would have performed for this job, and it led to very satisfactory results.

REFERENCES

[1] J. Burgess, "Snowflake generation,"

[2] K. Gligorić, A. Anderson, and R. West, "Adoption of twitter's new length limit: Is 280 the new 140?," 2020.

[3] B. Liu *et al.*, "Sentiment analysis and subjectivity.," *Handbook of natural language processing*, vol. 2, no. 2010, pp. 627–666, 2010.

[4] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[5] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[6] Wikipedia contributors, "Utf-8 — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=UTF-8&oldid=1064072349, 2022. [Online; accessed 6-January-2022].

[7] D. Effrosynidis, S. Symeonidis, and A. Arampatzis, "A comparison of pre-processing techniques for twitter sentiment analysis," in *Research and Advanced Technology for Digital Libraries* (J. Kamps, G. Tsakonas, Y. Manolopoulos, L. Iliadis, and I. Karydis, eds.), (Cham), pp. 394–406, Springer International Publishing, 2017.

[8] J. Straková, M. Straka, and J. Hajic, "Open-source tools for morphology, lemmatization, pos tagging and named entity recognition," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 13–18, 2014.

[9] D. A. Hull, "Stemming algorithms: A case study for detailed evaluation," *Journal of the American Society for Information Science*, vol. 47, no. 1, pp. 70–84, 1996.

[10] T. Korenius, J. Laurikkala, K. Järvelin, and M. Juhola, "Stemming and lemmatization in the clustering of finnish text documents," in *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pp. 625–633, 2004.

[11] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media, 2009.

[12] C. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," vol. 8, pp. 216–225, May 2014.

[13] S. Loria, "textblob documentation," *Release 0.15*, vol. 2, p. 269, 2018.

[14] A. Mueller, J.-C. Fillion-Robin, R. Boidol, F. Tian, P. Nechifor, yoonsubKim, Peter, R. Rampin, M. Corvellec, J. Medina, Y. Dai, B. Petrushev, K. M. Langner, Hong, Alessio, I. Ozsvald, vkolmakov, T. Jones, E. Bailey, V. Rho, IgorAPM, D. Roy, C. May, foobuzz, Piyush, L. K. Seong, J. V. Goey, J. S. Smith, Gus, and F. Mai, "amueller/word_cloud: Wordcloud 1.5.0," jul 2018.

[15] D. Harris and S. Harris, *Digital Design and Computer Architecture*. Engineering professional collection, Elsevier Science, 2013.

[16] S. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *arXiv preprint arXiv:1503.06462*, 2015.

[17] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[18] A. McCallum, K. Nigam, *et al.*, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752, pp. 41–48, Citeseer, 1998.

[19] B. Dai, S. Ding, and G. Wahba, "Multivariate bernoulli distribution," *Bernoulli*, vol. 19, no. 4, pp. 1465–1483, 2013.

[20] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," 2004.

[21] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, pp. 3146–3154, 2017.

[22] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002.

[23] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Artificial Intelligence and Statistics*, pp. 240–248, PMLR, 2016.

[24] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6765–6816, 2017.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.