Twitter Sentiment Classification

Silva Bashllari
Politecnico di Torino
Student number: [redacted]
[redacted]@studenti.polito.it

Alessandra Borzomi
Politecnico di Torino
Student number: [redacted]
[redacted]@studenti.polito.it

***Abstract*—The aim of this report is to provide an approach in building a binary classification pipeline to predict the sentiments of a given Twitter dataset. In particular, we introduce six different preprocessing formats and two pipelines, using Stochastic Gradient Descent (SDG) and Multinomial Naive Bayes (MNB). Our best pipeline outperforms the given baseline in the leaderboard of 0.753, reaching an F1 score of 0.846.**

I. Problem overview

In recent years, a great attention has been placed on extracting information on opinions and sentiments expressed by people on web-based documents. This has brought to the development of opinion mining and sentiment analysis, a technique to detect and extract subjective information in text documents by determining the sentiment of a writer regarding some aspect or the overall contextual polarity of a given document. [1] This report attempts to present a solution to a binary classification problem of sentiments using a given data set of tweets. Considering that on average 500 million tweets are generated per day, the usage of machine learning algorithms to analyze their content becomes of paramount importance. The classification result for each tweet in our data set is either negative or positive. We have been provided with two data sets, namely:

1) the development set - composed of 224 994 records;
2) the evaluation set - composed of 74 999 records.

In a horizontal perspective, the data sets are composed of the following attributes:

1) ids: a numerical identifier of the tweet;
2) date: the publication date;
3) flag: the query used to collect the tweet;
4) user: the username of the original poster;
5) text: the text of the tweet;
6) sentiment: either 1 or 0, inferring positive or negative sentiment respectively, present only in the development set, as it is the target attribute to be classified in the evaluation set.

The aim of the report is to achieve the highest possible F1 score. While exploring the development set, we immediately came across the fact that it is slightly imbalanced. More specifically, there are 130157 records classified as positive as opposed to 94837 classified as negative. Imbalanced class distribution of a data set has encountered a serious difficulty to most classifier learning algorithms which assume a relatively balanced distribution. [2] However, since we are not using accuracy as the evaluation metric for the algorithms but the F1 score, the harmonic mean of precision and recall, we should be able to manage to a certain extent this aspect of the data. Another observation is that the content of the tweets does not follow a particular logical line or topic. They seem to be randomly selected in semantic terms. In order to create a preliminary idea of our data set, we created two word-clouds with the top negative (Figure 1) and the top positive (Figure 2) words found in the development set. A world-cloud is a visualization technique in which the size of the word's font is proportional to the frequency of occurrence in the text, or tweets in our case. As we can observe, some words like "lol", "love", "one", "twitter", "now", "today" etc are present in both figures, as expected, since they are frequent words that can be used in both contexts.

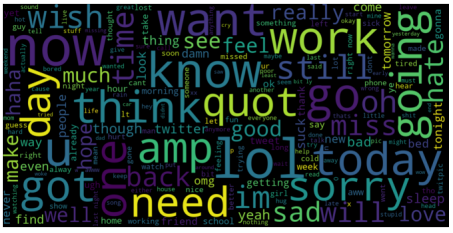

Fig. 1. Top negative tokens.

Fig. 2. Top positive tokens.

Furthermore, we decided to drop the "flag" attribute as it has only one unique value, namely, "no query" and the "id" attribute. Essentially, we are left with four attributes: date, user, text and sentiment.

II. Proposed approach

A. Preprocessing

1. Text attribute: The preprocessing of the "text" attribute was the main and the most time consuming part. We tried a variety

of approaches and techniques. First, we dropped the duplicate values, as we observed that there are 1888 rows which contain the same exact tweets, thus changing the length of the development set to 223106 rows. The three central libraries (besides Pandas, Numpy and Scikit learn, which were used throughout the entire project) used in the text preprocessing were ”Natural Language Processing Toolkit - nltk” , ”Re module - regular expressions operations” and ”Beautiful Soup - BS4” library. NLTK is a suite of open source program modules, tutorials and problem sets, providing ready-to-use computational linguistics courseware. [3] Re or regular expressions module, is a module used for parsing and manipulating text, offering a Perl-like regular expression syntax. [4] Whereas Beautiful Soup allows us to get and manage data from HTML and XML pages. [5] We observed that the content of the tweets, as expected, contained punctuation marks, special characters, HTML entities and URLs. Furthermore, people when tweeting, mention others using the ”@” symbol followed by the name of the user they are mentioning. Using the Re module and Beautiful Soup we cleaned the text from all the above mentioned issues and converted all the characters to lower case to make it more readable for the classification algorithms. Furthermore, using the NLTK library’s modules we handle negations. People, when they post on Twitter, tend to use abbreviations which can be problematic for the classification algorithms. Therefore, we transformed all the abbreviations of negations onto their full structure, for example: ”don’t” to ”do not” etc. In order to manage that, we needed to break down the tweets into tokens using, again, NLTK’s modules. Finally, we merged the tokens back into strings, adding the cleaned text into the development set, having in this manner created the ”cleaner” function. Hence, we created the first version of the preprocessed text attribute called Prep 1. (see Figure 3).

In order to reduce the dimensionality of the text attribute we decided to use stemming, hoping this would increase the F1 score. Stemming is a fundamental step in processing textual data whose common goal is to standardize words by reducing a word to its base or stem. [6] For the purposes of our classification problem, we decided to try Porter’s Stemmer and Snowball Stemmer, an updated version of the first, otherwise known as Porter 2. [7] [8] Porter’s Stemmer was proposed back in 1980s but is now a very popular stemmer as an easily accessible module through the NLTK library. Finally, we constructed three versions of the preprocessed ”text” attribute to be used for trainning the models, namely: Prep 1, Prep 2 - same as Prep 1 but stemming the words using Porter’s Stemmer and Prep 3 - same as Prep 1 but stemming the words using Snowball Stemmer. (Figure 3)

2. **Date :** Originally, we considered to drop the date and user attributes, as we assumed that they might not have much added value. However this assumption proved to be inaccurate. The format of the date attribute is: Day-of-week, Month, Date, Time(Hour, Minutes,Seconds), Time-Zone, Year, for example: Mon May 18 01:08:27 PDT 2009. Grouping by the hour and performing one-hot encoding on the sentiment attribute we constructed a temporary data frame to study the frequency

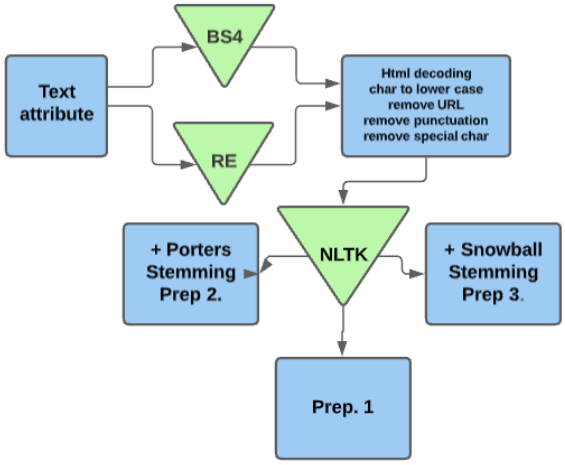

Fig. 3. Preprocessing - Text Attribute

distribution of the sentiments along the day. Then, using matplotlib we visualized the frequency distribution into two graphs (Figures 4, 5). Since we do not have a balanced data set, we constructed two different graphs, as we are not interested in drawing comparisons between the frequency of negative and positive tweets, but rather to see if there is some trend along the 24-hour frame (which we categorized from 0 -23). There is some variance in terms of the number of tweets along the day, thus, adding the date attribute may improve the accuracy of the classification. In order to facilitate the computation process, we removed the day of the week and the time zone and kept the granularity of the time to the hour. In this manner, the final format of the date attribute became: Year-Month-Date-Hour.

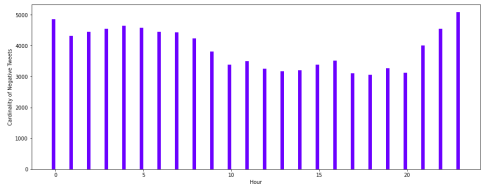

Fig. 4. Frequency distribution of negative tweets along the hour.

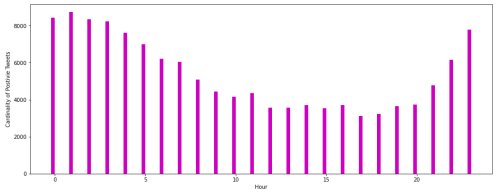

Fig. 5. Frequency distribution of positive tweets along the hour.

3. **User:** While exploring the user attribute we note that there are 10646 unique users (after dropping the duplicate values on the ”text” attribute, as previously mentioned), inferring that the same people have written multiple tweets. Using one-hot encoding on the sentiment, grouping by the user and sorting once

by the number of negative and then of positive sentiments, we created temporary dataframes (through Pandas) to study more in-depth the metrics of the user attribute. On average, the same user would post around 8.8 negatively categorized tweets and 12.1 positively categorized tweets. However, there are some outlier users and the top 5 for each class are displayed in Figure 6 and 7, respectively. These outliers are also very polarized, but since some of them are present in the evaluation set, we decided not to omit them. Initially, we did consider to implement the homologous versions of Prep 1, Prep 2 and Prep 3 (see: Section II.A) to the user attribute as well. However, that meant that for example, user "lost-dog" would be broken down to two separate tokens and it would not only lose it's meaning but also would be equivalent to another user whose username might be "lost!dog". Thus, we decided to not preprocess the user attribute in order to preserve its uniqueness and value.

user	sentiment_0	sentiment_1
lost_dog	412.0	0.0
webwoke	195.0	63.0
mcraddictal	156.0	49.0
SallytheShizzle	142.0	69.0
_magic8ball	101.0	40.0

Fig. 6. Top 5 users on negative tweets.

user	sentiment_0	sentiment_1
what_bugs_u	0.0	184.0
DarkPiano	5.0	171.0
VioletsCRUK	43.0	166.0
keza34	4.0	160.0
tsarnick	30.0	156.0

Fig. 7. Top 5 users on positive tweets.

To sum up, we had 6 different variations of the preprocessing of the development set:

- *Prep 1:* Cleaned text attribute as described in Section II.A.1
- *Prep 2:* Cleaned text attribute with Porter Stemming.
- *Prep 3:* Cleaned text attribute with Snowball Stemming.
- *Prep 4:* Prep 1 merged onto one column with the User and the preprocessed Date attribute (as mentioned in Section II.A.2).
- *Prep 5:* Prep 2 merged onto one column with the User and the preprocessed Date attribute.
- *Prep 6:* Prep 3 merged onto one column with the User and the preprocessed Date attribute.

The same preprocessing procedures are implemented to both the development and the evaluation data sets with the exception of removing duplicate values, which is done only to the development set.

B. Model selection

The models we have considered for the classification problem are:

- ***Stochastic Gradient Descent (SGD)***: a gradient descent optimization iterative method which minimizes a given loss function. [9] It can fit the classifier by using convex loss functions such as (linear) Support Vector Machines(SVMs) and Logistic Regression.
- ***Multinomial Naive Bayes (MNB):*** A classifier which is suitable for classification with discrete features (e.g., word counts for text classification). [10]

We divided the development set into a training set and a test set using the "train-test-split" function of Sklearn library, in order to avoid over-fitting. After some trials, the best test size for Multinomial was 0.01 and for SGD was 0.2. Considering that the algorithms cannot extract information out of textual data, we had to convert those into numerical values. We created a pipeline which included 3 components. The first two however, can also be considered as a final type of preprocessing, as they transform the text into numerical values:

- ***CountVectorizer****:* creates a dictionary of words and counts the number of occurrences of all the words in a given tweet and in this manner, transforms the tweet in an array of integers.
- ***TfidfTransformer:*** is an abbreviation for Term Frequency Inverse Document Frequency. Applied after the CountVectorizer, Tfidf takes into account the overall weight of a word in a document, not just its frequency. In this manner, we manage to deal with the most frequent words, by penalizing them.
- ***The Classifier Algorithm:*** Finally, here we plug in either Multinomial Naive Bayes or SGD, as described above.

Since we have 6 different versions of the development set after the preprocessing and 2 different Pipelines (we will refer to the Pipeline with SGD as Pipeline 1 and the one with Multinomial as Pipeline 2), we have in total 12 different solutions. The F1 score for each of these solutions after feeding the evaluation set to the pipeline is reported in Table I. The top score seems to be achieved using Pipeline 1 and preprocessing 4.
Furthermore, the classification algorithms themselves have different hyperparamenters to be taken into account and that would increase drastically the number of combinations. Thus, the configuration of the hyperparameters is automated using grid search, explained in Section C.

C. Hyperparameters tuning

Once the Pipeline is created, it is possible to search in the hyper-parameter space using the GridSearchCV function to exhaustively consider all the hyperparameter combinations and find the optimal ones which would eventually produce the most

TABLE I
F1 SCORE RESULTS FOR EACH COMBINATION(PREPROCESSING, PIPELINE).

	Pipeline1 (SGD)	Pipeline2 (MNB)
Prep1	0.757	0.749
Prep2	0.760	0.744
Prep3	0.762	0.745
Prep4	0.789	**0.796**
Prep5	0.793	0.795
Prep6	0.793	0.795

accurate predictions. The hyperparameters we considered for each algorithm in Pipeline 1 are:

- ***CountVectorizer:*** max-df: either 0.5, 0.75 or 1; max-features: either None, 5000, 10000 or 50000; ngram-range: either (1,1), (1,2) or (1,3).
- ***TfidfTransformer:*** use-idf: True or False.
- ***SGD:*** loss: either hinge (inferring SVM) or log (inferring Logistic Regression); alpha: either 0,00001 or 0.000001; max-iter: either 1000, 5000 or 10000.

The hyperparameters we considered for each algorithm in Pipeline 2 are:

- ***CountVectorizer****:* max-df: either 0.5, 0.75 or 1; ngram-range: either (1,1), (1,2) or (1,3).
- ***TfidfTransformer:*** no tunning considered.
- ***MNB:*** alpha: any value from 0 to 1 with a step of 0.05.

We ran the grid for both pipelines 6 times, due to the fact that we have 6 different preprocessings, which impacted the output of the grid, as it varied depending on the type of preprocessing the training set had. Tunning the hyperparameters significantly improved the F1 score when we ran the pipelines feeding the evaluation set, as it can be observed in Table II.

III. RESULTS

As displayed in Table II, the best F1 score is achieved using Pipeline 1, so SGD classifier, after tunning it using GridSearchCV and choosing either Preprocessing 4, 5 or 6. The score, as it can be observed in Table II, is 0.846. The best combination of the hyperparameters for Pipeline 1 combined with either Prep 4, 5 or 6, was almost the same, specifically:

- *CountVectorizer:* max-df: 0.5; max-features: None; ngram-range: (1,3).
- *TfidfTransformer:* use-idf: True.
- *SGD:* loss: hinge (inferring SVM); alpha: 0.000001; max-iter: 5000 (for Prep 4 and 5) and 10000 (for Prep 6).

This result is expected in terms of the Pipeline itself, however it is surprising in terms of the preprocessing. Using stemming the performance of Pipeline 1 improved slightly from Prep 1 to Prep 2 and 3 but it displayed no difference after merging the user and the preprocessed date attributes with the text from Prep 4 to Prep 5 and 6. Whereas in Pipeline 2, tunning the hyperparameters did mark a relevant difference but stemming in itself seems to have not, keeping the top values of Prep 4, 5 and 6 exactly the same.

TABLE II
F1 SCORE FINAL RESULTS AFTER HYPERPARAMETER TUNNING FOR EACH COMBINATION(PREPROCESSING, PIPELINE).

	Pipeline 1(SGD)	Pipeline 2(MNB)
Prep1	0.798	0.777
Prep2	0.801	0.774
Prep3	0.802	0.774
Prep4	**0.846**	0.823
Prep5	**0.846**	0.823
Prep6	**0.846**	0.823

IV. DISCUSSION

The proposed approach for this problem and the choice of algorithms is very oriented towards achieving the best possible F1 score. Thus, other things like interpretability, efficiency, incrementality or scalability are not taken into account. Considering those could be a way to further develop this problem. The position in the leader-board obtained by our best score is in the top 20. This is quite good considering that the baseline is 0.753 but the variation of range from the baseline to the ceiling is between 0.1 and 0.2, so further improvements are possible. One would be to consider other algorithms, such as: neural networks and deep learning. Recently, deep neural networks are emerging as the prevailing technical solution to almost every field in NLP [11]. Furthermore, considering that the grid search is computationally expensive, we could not explore a lot in terms of inputs, but more hyperparamenters could have been taken into account. Finally, in preprocessing, besides stemming, other methods like lemmatization could have been tested. However, the top score obtained, specifically 0.846, is very promising and far outperforms the baseline.

REFERENCES

[1] M. S. Hajmohammadi, R. Ibrahim, and Z. A. Othman, "Opinion mining and sentiment analysis: A survey," *International Journal Of Computers Technology*, vol. 2, no. 3, p. 171–178, 2012.

[2] Y. Sun, A. K. C. Wong, and M. S. Kamel, "Classification of imbalanced data: A review," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 04, p. 687–719, 2009.

[3] S. Bird and E. Loper, "Nltk: The natural language toolkit.," *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics*, p. 1–4, 2004.

[4] T. Stubblebine, *Regular Expression Pocket Reference*. O'Reilly, 2 ed., 2007.

[5] V. G. Nair, *Getting started with Beautiful Soup: build your own web scraper and learn all about web scraping with Beautiful Soup*. Packt Publishing, 2014.

[6] E. T. Al-Shammari, "Towards an errorfree stemming," *Proceedings of ADIS European Conference Data Mining*, p. 160–163, 2008.

[7] M. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, p. 130–137, 1980.

[8] M. F. Porter, "Snowball: A language for stemming algorithms," 2001.

[9] T. Günther and L. Furrer, "Gu-mlt-lt: Sentiment analysis of short messages using linguistic features and stochastic gradient descent," in *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pp. 328–332, 2013.

[10] "Sklearn.naive$_b ayes.multinomialnb$."

[11] L. Mou, Z. Meng, R. Yan, G. Li, Y. Xu, L. Zhang, and Z. Jin, "How transferable are neural networks in nlp applications?," *arXiv preprint arXiv:1603.06111*, 2016.