Classification models comparison for a Sentiment Analysis task on a Twitter dataset

Ionut Cosmin Nedescu
Politecnico di Torino
Student id: s292495
s292495@studenti.polito.it

Andrea Parolin
Politecnico di Torino
Student id: s291462
s291462@studenti.polito.it

***Abstract*—In this report we present our approach in order to solve a sentiment analysis problem on a Twitter dataset. We try to obtain the best F1_macro score applying a few pre-processing steps and training different classification algorithms. We make a comparison of results between the classification algorithms we used and show how our best result is obtained by applying the LinearSVC.**

I. Problem overview

In this competition we deal with a sentiment analysis problem, which consists in analyzing textual data - in this case tweets - and predicting whether the text has a positive or a negative sentiment. We are given two sets:

- *development.csv*: this file is made up of 224994 data-points and contains the attributes *ids*, *date*, *flag*, *user*, *text*, besides the target column *sentiment* which states if the text is negative (0) or positive (1).
- *evaluation.csv*: this file is made up of 74999 data-points and has the same structure of the development set with the only difference that the target variable has to be predicted.

The main components of a tweet are the following:

- mentions: a mention is a reference to another person. We make a mentions by typing the character ”@” before the username.
- hashtags: including a hashtag gives your Tweet context and allows people to easily follow topics that they’re interested in. You make an hashtag by typing the character ”#” before the word or the phrase.
- retweet: a retweet is a Tweet that you share publicly with your followers. A retweet contains the characters ”RT”.

First of all, we check whether there are missing values in the development set. Fortunately, all the rows are non-null so we have no missing values. We check now if the problem is well balanced between the two classes. As we can see from Figure 1 the development set is unbalanced towards the positive sentiment. For a more understanding we show the exact number of data-points per sentiment in Table I.

TABLE I
NUMBER OF DATA-POINTS PER SENTIMENT TYPE

Sentiment	Number of rows
1 - positive	130157
0 - negative	94837

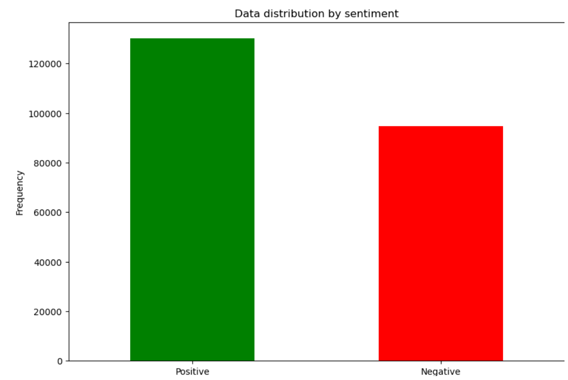

Fig. 1. Class balance of the problem

We have around 2000 rows that contain duplicated textual data, this is probably due to the fact that the data-set is made up of same tweets coming from different sources. We drop them because otherwise it could affect the prediction biasing the results.

By comparing the minimum, the maximum and the medium length of positive and negative tweets, we discover that there are some outliers in our development set. Nowadays, the maximum length of a tweet can be 280 characters, this set is made up of tweets mostly coming from year 2009 where the maximum length was 140 characters. From Figure 2 we can see that there are some tweets that exceed this limit. This could be due to the fact that some tweets contain some unicode characters still not codified in a proper way. By further exploring the *text* field in the development set, we notice that there are some HTML tags. It comes to our awarness also the presence of emoticons that could be crucial in predicting the sentiment of a tweet [1]. On the other hand, from the exploration, we can say that there are no emojis in the text. We have to make a distinction between emoticons and emojis: they have the same purpose, which is expressing emotions, the only difference is that emoticons are made up of symbols and letters, while emojis are Unicode encoded.

II. Proposed approach

In this section we propose our approach to the problem. Since this is a sentiment analysis problem, where our aim is to perform a classification between a positive and a negative

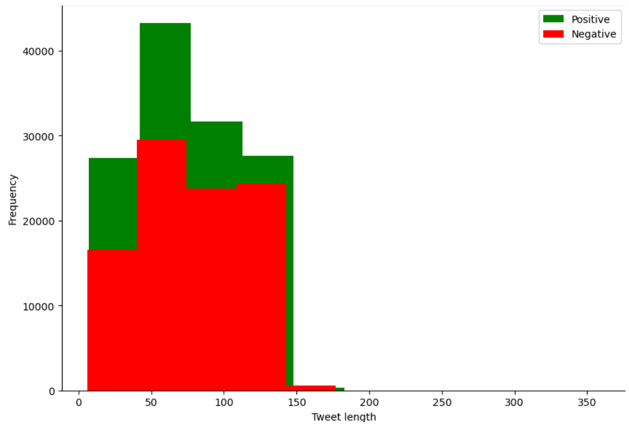

Fig. 2. Tweets length

text, after a short exploratory analysis we decide to drop the columns *ids*, *date*, *flag* due to the fact that these attributes do not add any information.

A. Preprocessing

We start by removing the duplicates and solving the HTML tags because, in an NLP problem, tags and characters that are not proper words do not add any further information that can be useful to the classifier. We remove the tweets that still have more than 140 characters and plot again the distribution without outliers, as in Figure 3. Both the negative and the positive tweets follow the same distribution, with the difference that positive tweets are more due to unbalanced data.

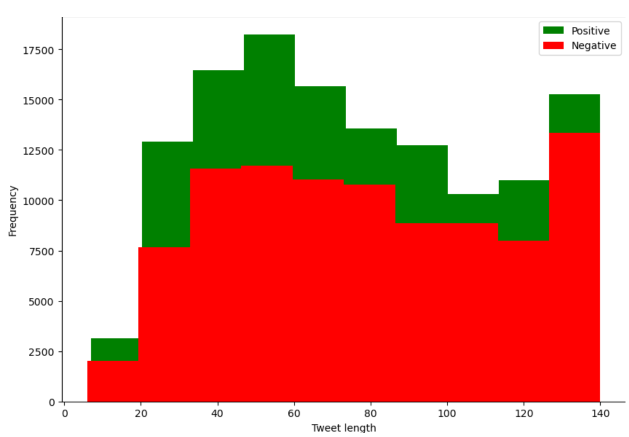

Fig. 3. Tweets length without outliers

Moving forward, we started the processing of the *text* column. The aim is to convey to a clean text, free of symbols. Those symbols can encode sentiment and by not processing it we lose information.

We replace the emoticons - the ones made with brackets and commas - with *EMO_POS*, if the emoticon is positive, and *EMO_NEG* if the emoticon is negative. We take care of the slang words used on social media and translate them to words that belong to the English vocabulary (e.g. "idk" is short for "I don't know"). We use a library called *contractions* which allows us to replace the contracted forms in sentences and transform them to full length (e.g. "I'd like to" becomes "I would like to").

In our pre-processing pipeline, we convert all the characters to lower case, we remove the characters that identify a mention (i.e. @), an hashtag (i.e #) and a RT (i.e. re-tweet). We remove the numbers from the tweets because they cannot help us to classify a text. We remove links and replace them with an empty string.

A crucial step in sentiment analysis is the handling of negation. A negation can completely change the meaning of the words that surround it. By ignoring it or by applying a stop word removal (that most of the time contain expressions like "not", "neither") we lose information. We added the tag 'NEG_' to every word that came after a negation until punctuation is found (e.g. "I'm not feeling so good today, I will stay at home" becomes "I am not NEG_feeling NEG_so NEG_today, I will stay at home"). We replaced laughs and words like 'lol' with a standardized form (e.g. 'hahaha' replaced with 'strong_laugh'). We also remove all the characters that are not belonging to the English alphabet. In Figures 4 and 5 the wordcloud of the most appearing words for positive and negative sentiment are displayed.

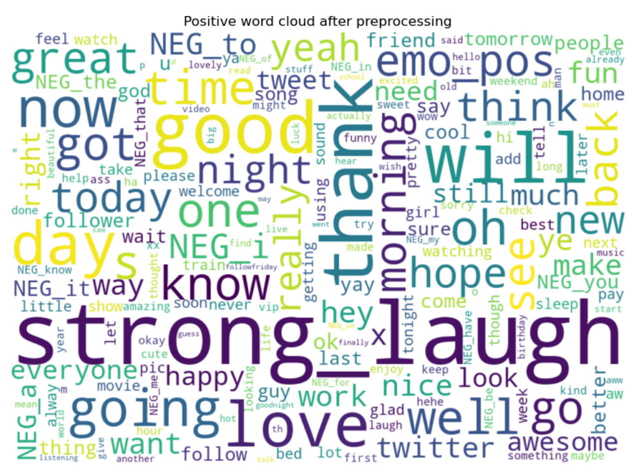

Fig. 4. Word Cloud of the most appearing words for positive sentiment

Columns *user* and *text* are merged by separating them with a blank space and allowing the encoding of the username as part of the text. Similar results could be obtained by encoding the username as a column. It has been decided to add also the column *user* to the features used by the classification models because it helps improving the score and also the accuracy. This should be due to the fact that some users tend to write tweets that are mostly negative, while other users tend to write tweets mostly positive. In a generic sentiment analysis, the focus should be only on the text, but in this case tweets are connected to usernames and can influence the prediction towards a better result. We use the library *nltk* that has useful functions for the pre-processing steps. We apply stemming which aims to reduce the inflectional forms of each word

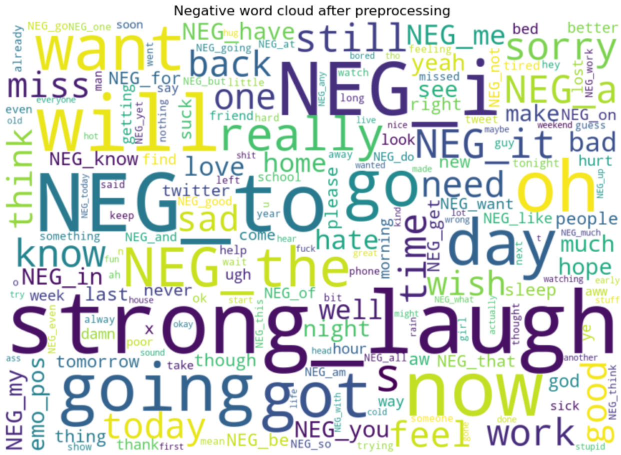

Fig. 5. Word Cloud of the most appearing words for negative sentiment

into a common base or root [2]. More specifically, we apply the function *SnowballStemmer*. The main peculiarity of this function is that it not only retrieves the stem out of a word but it also takes into account that the stem comes from the right sense of the word. We finally remove the rows that are empty after the stemming procedure.

B. Model selection

After obtaining a clean text in the preprocessing step we move forward with the model selection. The aim of the project is to obtain the best score in the evaluation step and reach it. We implemented a comparison between classical ML algorithms.

To allow the encoding of the text feature we choose to implement two vectorizers that allow the conversion of meaningful text into a numerical vector that can be fed to the ML models. The vectorizers we used are Bag-of-Words and Term Frequency–Inverse Document Frequency: both the algorithms have the same purpose to convert words into numbers, the difference is the way they work. BoW creates a binary matrix where every word is encoded with 1 if it's present in the respective document, 0 otherwise. TF-IDF evaluates how relevant a word is to a document in a collection of documents by weighing how much a word appears in a document compared to how many times that word came across in the whole set of documents.

We proceed with the training of raw ML models:

- **Logistic Regression**: is a supervised model that can be used both for binary classification and regression. It is based on logits and it models the probability that the response Y belongs to a particular category. To do this a sigmoid function is applied to the linear regression output and the Gaussian distribution for Y is replaced with a Bernoulli.
- **Decision Tree Classifier**: is a supervised learning algorithm that aims to predict the classes in a simple way. The classes are predicted by segmenting the predictors space into a number of simple regions.
- **Random Forest Classifier**: it ensembles multiple decision trees that are combined to yield a single consensus prediction. When we build these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. Typically we choose $m \simeq \sqrt{p}$ as the number of predictors.
- **Multinomial Naive Bayes**: computes $P(C|X)$ for all classes and assigns a probability that the record X belongs to class C.
- **Support Vector Machine**: SVM has the aim to perform classification by finding a hyperplane that separates at its best the classes in the feature space. Due to long computational times of SVC from *sklearn* library, we use instead LinearSVC which is a faster implementation of the Support Vector Classifier.
- **XGB Classifier**: is a classifier that uses a decision tree and a process called boosting to improve performance. Boosting is an ensemble technique where new produced models have errors corrected by existing models.

We will see in the next sections which are the classification models that perform the best on this data-set.

C. Hyperparameters tuning

After seeing the results of the raw models on the development set, we decide to fine-tune only two of the previous explained models: Logistic Regression and LinearSVC. We proceed to split the development set into training and test with a proportion of 80%/20%. By applying a gridsearch the training set is further divided into training and validation with the help the k-Fold Cross Validation: at each cross validation step, the training set is split into five slices, four of them are used for training the model, the remaining one is used for validating it.

In the Tables II and III there can be seen the hyperparameters that we decide to tune for the Logistic Regression and the LinearSVC, respectively.

TABLE II
HYPERPARAMETERS FOR THE PIPELINE BOW + LOGISTICREGRESSION

Model	Parameters
BoW	max_df: 0.2, 0.3, 0.4, 0.5 ngram_range: (1,3)
LogisticRegression	C: 0.8, 0.9, 1, 1.1, 1.2 multi_class: 'ovr'

TABLE III
HYPERPARAMETERS FOR THE PIPELINE TF-IDF + LINEARSVC

Model	Parameters
TF-IDF	max_df: 0.2, 0.3, 0.4, 0.5 ngram_range: (1,3)
LinearSVC	C: 0.8, 0.9, 1, 1.1, 1.2 loss: 'hinge'

III. Results

We run a preliminar training without fine-tuning the parameters in order to decide whether an algorithm is worth the fine-tuning or not. In Table IV are shown the results in terms of F1_macro score on the test set by using feeding the algorithms with BoW or TF-IDF.

TABLE IV
F1_MACRO SCORES ON THE TEST SET FOR EACH CLASSIFIER AND VECTORIZER

Model	Test f1_macro BoW	Test f1_macro TF-IDF
LogisticRegression	**0.825**	0.811
DecisionTreeClassifier	0.573	0.574
XGBClassifier	0.696	0.697
RandomForestClassifier	0.367	0.371
MultinomialNB	0.802	0.753
LinearSVC	0.824	**0.829**

We choose to proceed with fine-tuning the two models that have the best result on the test set: LinearSVC with TF-IDF and Logistic Regression with BoW. After the retrieval of the best parameters from the two grid-search of the pipelines, we proceed with the training of the two models. We obtain that the best parameters in the pipeline TF-IDF + LinearSVC are *max_df*=0.3, *ngram_range*=(1,3) *C*=1.2 and *loss*='hinge'. With this configuration, we reach a private F1_macro score equal to 0.854, while on the public leaderboard we obtain F1_macro equal to 0.830. On the other hand, the parameters that lead to the best performance of the pipeline BoW + Logistic Regression are *max_df*=0.3, *ngram_range*=(1,3), *C*=1.2, *multi_class*='ovr'. We obtain a private F1_macro equal to 0.857 and a public F1_macro equal to 0.826. In Figure 6 there can be observed the ten most influential positive and negative words for the LinearSVC classifier.

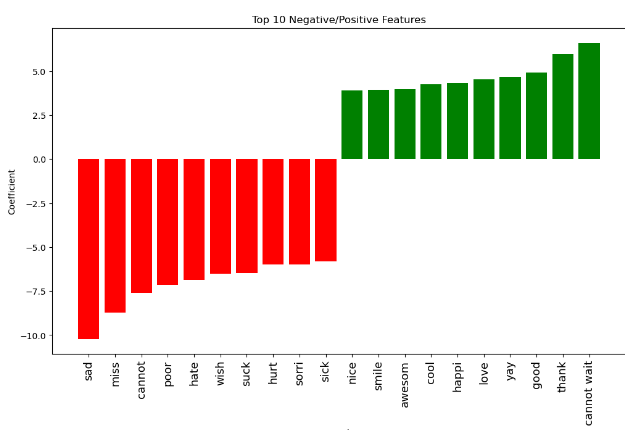

Fig. 6. Top 10 positive and neagative words

IV. Discussion

As far as we have seen from the Table IV, we can state that Decision Trees and Random Forests do not perform well on text classification tasks. In fact, Decision Trees need several key nodes that are hard to find in text classification and Random Forests work bad for high sparse dimensions. Naive Bayes classifier is a good compromise between training speed and accuracy, but in our case it does not reach the best results as other models. This can be maybe due to the choices that we made in the pre-processing step. We avoided using K-Nearest Neighbors due to its long computational times, also because the dataset we were given was high in dimensionality. We can state that on this problem the best performing models were the ones using linear kernel (i.e. LinearSVC and Logistic Regression).

Further implementations could have been, for example, dealing with the unbalanced dataset: we could have implemented an oversampling method to overcome the difference between classes. Oversampling is preferred in these types of problems due to the fact that with undersampling we could lose important information. Regarding dimensionality reduction, it has been decided to avoid it because of the lack of memory to execute it.

This is a Natural Language Processing problem and in order to obtain better results it would have been a good idea to use neural networks in combination with pre-trained models (e.g. BERT).

References

[1] Wegrzyn-Wolska, Katarzyna & Bougueroua, Lamine & Yu, Haichao & Zhong, Jing. (2016). Explore the Effects of Emoticons on Twitter Sentiment Analysis. 65-77. 10.5121/csit.2016.61006.

[2] A language for stemming algorithms. Snowball. (M.F. Porter). Retrieved January, 2022, from http://snowball.tartarus.org/texts/introduction.html