Sentimental analysis classification problem

Federico Filì
Politecnico di Torino
federico.fili@studenti.polito.it

***Abstract*—In this report, you will find a possible approach to sentiment analysis (positive or negative) of tweets, using classification techniques. The solution consists of extracting information from the tweet corpus and using data about the user, date and other specific characteristics to classify the text**

I. Problem overview

The proposed task is a sentiment analysis problem on a dataset of tweets. The training set contains 179995 tweets which have the following features:

- **ids**: the identification of the tweet;
- **date**: the date on which the tweet was written. It is expressed in PDT timezone;
- **flag**: the query used to collect the tweet;
- **user**: the user who wrote the tweet;
- **text**: the corpus of the tweet;

The evaluation set contains 44999 tweets and does not contain the label. The feature sentiment is what we will have to predict. It can be either 0 or 1 if the sentiment is negative or positive. In particular the number of positive tweets is 104296, while the number of negative tweets is 75699 The dataset contains 28k more positive tweets than negative ones. We tried to balance the dataset by removing the last 28k positive tweets and subsampling, but this always led to worse results, so the population tends to write more positive tweets. The dataset contains about 2000 duplicates. We will address this later in the preprocessing section. After further analysis, we found that the evaluation and development sets contain 10647 unique users. This can be a strength since the model can also learn what these users tweet and better predict the sentiment of their tweets. We will provide further interpretation of this in the discussion section. The feature date is expressed in Pacific Daylight Time, which is used in western North America. We observed that there are only three unique months in both sets: April, May and June. Since there are limited users, the tweets were extracted from a community of people in a limited time period. Date and user could be good indicators of how a particular user behaves and what type of tweets they write. The dataset does not contain missing values and this simplified the problem since we do not have to deal about inferring values or removing entire records (or columns) that could be critical. Analyzing the length of the tweets, we noticed that there are tweets that exceed the limit of 280 characters, but contain only encoded characters. The information about the length of the tweet does not improve the score, probably because the length is almost equally distributed for both sentiments, as shown in the graph in Fig. 1. The main difference is the number of

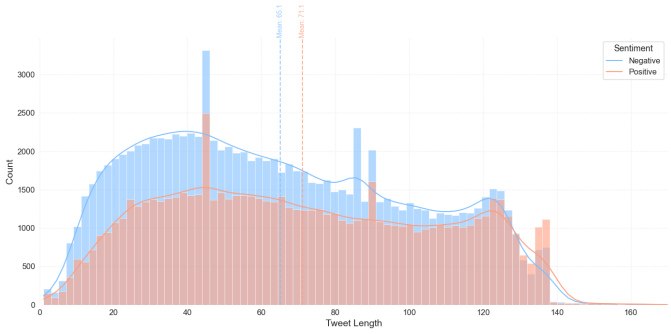

Fig. 1. Tweet length distribution for positive and negative sentiment

tweets, and this is because there are more positive tweets than negative ones. The training set is called: ”developments.csv” and the test set is called: ”evaluations.csv”

II. Proposed approach

A. Preprocessing

We started the data preparation by conducting an in-depth analysis on the dataset, identifying any anomalies, missing values, inconsistencies. In particular, the ”ids” feature was removed from the dataset because it was not relevant for the classification process. Being a simple unique identifier, it does not provide useful information to the machine learning model. Similarly, the ”flag” feature was removed, because in all the records present in the dataset it reported the same value, ”NO QUERY”. This uniformity made the feature devoid of informative utility, contributing only to increase the dimensionality of the data without adding analytical value.
During the preliminary analysis of the data, we found that several duplicate texts, identified by ”ids”, presented discordant sentiment classes. This discordance could have introduced noise in the model and compromised its predictive accuracy. To preserve the integrity of the dataset and ensure greater consistency, we decided to eliminate the 354 duplicates identified. This operation was necessary to avoid potential ambiguities and reduce the risk of overfitting.
Another relevant aspect that emerged during the analysis is the presence of emoticons within the texts. Emoticons, while not verbal, convey important emotional nuances that help enrich text analysis. These symbols can improve the model’s ability to understand emotional context, making learning more accurate. However, HTML characters [1] in the texts could have interfered with the correct recognition of emoticons. To

avoid losing this crucial information, we manually converted the most common HTML entities, as shown in the Table I

Entity HTML	
&quot;	”
&amp;	&
&gt;	>
&lt;	<

TABLE I
HTML ENTITY CONVERSION TABLE

After replacing the characters, we further enriched the text by replacing the emoticons with detailed textual descriptions, using a specialized dictionary available on GitHub [2]. This step allowed us to preserve the original emotional nuances, without sacrificing the clarity or coherence of the text. Additionally, we further processed the text, removing links, hashtags, and mentions. For text tokenization, we used the TweetTokenizer module from the nltk library, which offers some advantages over the traditional word_tokenize. The main difference, particularly relevant for our case, is that TweetTokenizer preserves the integrity of words containing apostrophes, such as contractions (e.g., ”I’m”, ”won’t”, ”don’t”). Given the frequent use of contractions in the analyzed texts, this approach allowed us to maintain the correct meaning of the words and not alter the syntax of the text.
After tokenization, each word was further subjected to specific transformations, in detail letters repeated more than twice within a word were truncated at the second occurrence. For example, words like ”hey” were reduced to ”heyy” and ”cooooooool” to ”cool”. This technique proved to be effective in the context of the dataset, reducing noise without significantly altering the meaning of the words. Only words that are alphanumeric or contain apostrophes and have more than one character were preserved. The apostrophe, in particular, was considered important to preserve the correct use of contractions, such as in ”don’t” and ”it’s”, which can have a significant impact on sentiment analysis. We decided to keep contractions and not remove the standard stop words from the nltk library, since their removal led to a significant degradation of the model’s performance, due to the elimination of common words important for sentiment analysis and sentence meaning, such as ”not”, ”never”, ”always”, etc.
After the text transformation, adding the author’s name as a feature further improved the model’s performance. This integration provided the model with essential contextual information, allowing it to better adapt to the specific style and language of each author. The writer’s identity, in fact, plays a crucial role in influencing the tone, meaning, and use of expressions in texts, allowing the model to more accurately distinguish different types of sentiment. The author’s name was concatenated to the text and considered as a word, because its encoding generated a large and difficult to manage sparse matrix, as the number of unique users can be very high.
Regarding the publication dates of the texts, we have extracted:

- **day** of the week
- **hour** of publication

The days of the week were then encoded using the technique OneHotEncoder, allowing the model to learn any correlations between the day of publication, the time and the emotional content of the text.
In addition to text cleaning, we generated some new features to enrich the dataset. Among these, the presence of links and mentions (@mentions) in the texts, differentiated between the two classes. We observed that they are a good indicator of sentiment because there are more positive tweets that contain a link, compared to negative ones. This is not explained by the fact that the dataset is unbalanced, since this difference in positive tweets is clearly higher than in negative ones. The same observation was performed for mentions. This suggests that, although the presence of links and mentions is a relevant source of information for the model, it is not sufficient by itself to guarantee accurate predictive capacity. However, combined with other features extracted from the text, it can help enrich the data representation and improve the predictive capacity of the model. Finally, the dataset that will be used for training will be composed of the appropriately processed text, accompanied by Boolean fields that indicate the presence of mentions and links, together with information on the time and day of publication.
However, in order to extract significant information from the text, a text mining technique called tf-idf (term frequency-inverse document frequency) will be used.
The tf-idf technique assigns a score to each word in the text, evaluating the importance of the word within the single document and in the entire collection of documents.
In detail, the term frequency (tf) measures the frequency with which a word appears in a single document. The inverse document frequency (idf) calculates the rarity of the word in the entire corpus of documents. The basic principle is that common words, which are frequently repeated in the corpus, will have a lower tf-idf score, since they do not provide distinctive information. On the contrary, words that appear rarely, but are present in a single document, will have a higher tf-idf score, indicating their importance for the characterization of that particular document.
The result of the application of tf-idf is a sparse matrix, where the columns represent the different words found in the document collection, and their respective values indicate the tf-idf score associated with each word. This representation allows to extract textual features useful for machine learning. To refine the effectiveness of this technique, it will be necessary to optimize some parameters. Among them: max_df and min_df, which allow to filter out terms that are too frequent or too rare. Terms that appear in almost all documents (for example, common articles or prepositions) will be excluded by setting an appropriate value for max_df, while min_df will remove terms that appear too rarely and may not be informative.
The ngram_range parameter, which allows to consider not only single words (unigrams), but also combinations of multiple

Model	Parameter	Values
Preprocessing	ngram_range	{(1, 1), (1, 2), (1, 3)}
	max_df	{0.1, 0.2, 0.5}
	min_df	{2, 5, 10}
	max_features	{5000, 10000, 15000}
LinearSVC	C	[0.1, 1.0], 20 values
	random_state	42
	max_iter	10000
	penalty	{'l2'}

TABLE II
HYPERPARAMETERS CONSIDERED

words (bigrams, trigrams, etc.), to capture broader semantic contexts and meaning structures that emerge from the combination of multiple terms. These preprocessing-related hyperparameters will be discussed in detail in the next steps, during the optimization of the model.
In summary, using tf-idf with the right parameter configuration will ensure that only relevant and distinctive information of the text is considered by the model, thus improving the overall predictive performance.

B. Model selection

The models we chose to work on and solve the task are:

- **Random Forest**: is an ensemble classification algorithm that builds and combines multiple independent decision trees during training. Each tree votes for the final prediction, and the most voted class is chosen. This algorithm was immediately discarded because it takes a long time to run (more than 2h) without feature elimination.
- **Logistic Regression**: is a classification algorithm that predicts the probability that an observation belongs to a specific class, transforming a linear combination of features through the sigmoid function to obtain an output between 0 and 1. The algorithm did not have poor performance but despite an optimization and a search for the best hyperparameters, the final score did not go beyond F1 $\approx$ 0.7528
- **Support Vector Machines**: are a classification algorithm that tries to find an optimal hyperplane (decision boundary) that separates the data into different classes with the widest possible margin. In particular, we used the LinearSVC classifier because it does not take much time to run and provides very good results.

For all models except Random Forest, we searched for the best hyperparameters to further improve the initial score, this will be discussed in the next session.

C. Hyperparameters tuning

Hyperparameters can be divided into two main groups:

- **Preprocessing**: ngram_range, max_df, min_df, max_features for TfidfVectorizer
- **Prediction**: C and penalty for LinearSVC

To optimize these hyperparameters, we adopted a random search approach on a specific range of values using RandomizedSearchCV that simultaneously considers both preprocessing and prediction parameters. This method allows exploring the hyperparameter space in a single optimization run. We chose RandomizedSearchCV [3] mainly because it drastically reduces the execution time, unlike GridSearchCV that performs an exhaustive parameter search by trying all possible combinations of values in a specific range at the expense of execution time. The optimization run was structured in such a way as to define a pipeline that includes both the feature extraction stage (TfidfVectorizer and other transformations) and the classifier (LinearSVC). We specified a parameter distribution that covers both the pre-processing and classification aspects and split the dataset into 80% train-validation and 20% test, using a stratification based on the target variable to maintain the class distribution. The LinearSVC classifier was defined using the class_weight = 'balanced' parameter, which will assign larger weights to the minority class during the learning process, thus balancing the influence of the two classes. We applied RandomizedSearchCV with a 5-layer stratified cross-validation, using StratifiedKFold, on the training set. This approach allows us to evaluate different hyperparameter combinations across multiple data splits. The randomized search process explored 50 different hyperparameter combinations, evaluating each based on the F1 score. After identifying the best hyperparameter combination, we evaluated the performance of the optimized model on both the training and test sets, calculating the F1 score for both. Complete representation of hyperparameters in Table II

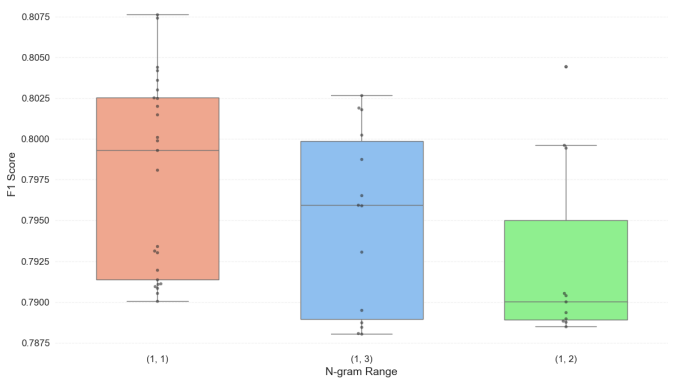

Fig. 2. F1 scores for different n-gram ranges

III. RESULTS

The tuning of the pre-processing hyperparameters was performed on LinearSVC and it was found that the best configuration is the following:

- **TfidfVectorizer**: max_df = 0.2, min_df = 2, ngram_range = (1, 1), max_feature = 15000

Instead the best configuration for the classifiers is the following:

- **LinearSVC**: C = 0.336, penalty = l2

These parameters led to a value of F1 $\approx$ 0.8110 As for the LinearSVC classifier, we used random_state: 42 for consistency and max_iter: 10000 to manage the convergence of the algorithm to a solution. During the hyperparameter

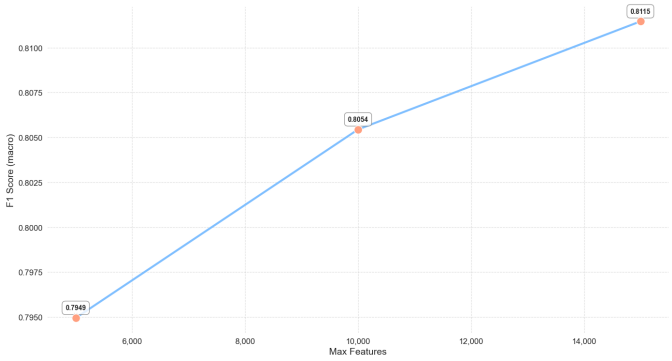

Fig. 3. Model performance as a function of the max_feature parameter

tuning phase, the influence of different values of ngram_range (the choice between unigrams, bigrams and trigrams) on the model's ability to correctly classify the data was evaluated. To verify the effect of the choice, the model's performance was compared using different configurations, as shown in the box plot in Fig. 2. The use of unigrams proved to be more effective because it allowed to capture the essential information without introducing noise and complexity to the model. Another hyperparameter that was optimized is max_feature, which controls the maximum number of terms considered by the model in the vector representation of the texts. As shown in the graph in Fig. 3, as the number of maximum features increases, a constant improvement of the F1 score is observed, allowing the model to capture more relevant information from the text, thus improving its ability to generalize. The value of max_feature = 15000 proved to be optimal because it allowed the model to acquire a richer and more complete representation of the text, without introducing excess complexity or noise. The parameter we investigated in more depth, as shown in the graph in Fig. 4 is the key parameter C for the LinearSVC model that controls the degree of misclassification penalty during training. We started by examining a range of C values of [0.1, 1.0, 20] and this led to a result of C = 0.3. We narrowed the new range to a final optimal value of C = 0.336.
We trained the LinearSVC model on the entire training set using the best configuration found through hyperparameter search. The optimized model was then used to label the evaluation set. Given the results obtained, the model should not suffer from overfitting or underfitting.

IV. Discussion

In conclusion, the presented approach represents a solution to sentiment analysis and adds improvements compared to using only the raw text of tweets. Moreover, there are several opportunities for improvement. First, a crucial element is text preprocessing. Refining the automatic correction of typos, using more accurate libraries for emoji handling, and improving the transformation of slang expressions and acronyms could lead to significant results. These interventions would allow the model to work on cleaner and more informative texts, thus increasing the quality of its predictions. Furthermore, it is important to consider the dependence of the model on the names of the users present in the dataset. This dependence risks limiting the ability of the model to generalize across different datasets, making it less flexible. Therefore, to ensure greater adaptability to different contexts, it could be useful to reduce the influence of these specific variables or make the

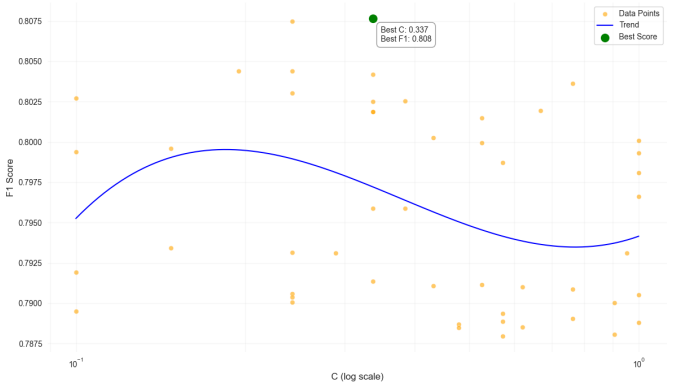

Fig. 4. F1 Score by varying the parameter C

model more robust to these characteristics. Finally, while optimizing hyperparameters can certainly improve performance, one must take into account the risk of overfitting, that is, that the model adapts too much to the training data, thus losing the ability to generalize effectively to new data. For this reason, it is essential to find a balance in optimization, using techniques such as cross-validation and regularization, to keep the model performant and generalizable. In summary, improving text preprocessing, balancing the influence of dataset-specific variables, and carefully optimizing hyperparameters are key strategies to obtain a more robust, versatile, and effective model.

References

[1] Wikipedia contributors, "List of xml and html character entity references — wikipedia, the free encyclopedia," 2024. Ultimo accesso: 22 settembre 2024.
[2] N. Shah, "emot: emo_unicode.py — github repository," 2024. Ultimo accesso: 22 settembre 2024.
[3] Scikit-learn developers, "sklearn.model_selection.randomizedsearchcv," 2024. Ultimo accesso: 22 settembre 2024.