A Lexicon-based approach to Twitter sentimental analysis

Marco Sorbi
Politecnico di Torino
marco.sorbi@studenti.polito.it

Gabriele Spina
Politecnico di Torino
gabriele.spina@studenti.polito.it

***Abstract*—In this report we propose a possible solution for a short text sentiment analysis problem by means of classification techniques. The proposed approach implements lexicon construction and ensemble features combined with bag of words to enhance the performance of text classification. Different models were compared, with some of them obtaining good results.**

I. Problem overview

A. Aim

The objective of the performed Natural Language analysis is to retrieve sentiments from short texts (tweets) collected from Twitter, i.e. to measure whether the tweet is positive or negative. Two datasets were provided: development and evaluation.

B. development dataset

is composed by 224994 records, each one described by 6 parameters:

- **id**: *nominal*, integer code identifying each tweet.
- **date**: *interval*, date and time at which the tweet written.
- **flag**: *nominal*, query used to retrieve the tweet from Twitter.
- **user**: *nominal*, twitter user who wrote the tweet.
- **text**: *nominal*, text of the tweet.
- **sentiment**: *nominal*, target variable, with value 0 or 1 representing negative and positive sentiment.

C. Evaluation dataset

The evaluation dataset contains 74999 records, described by the same attributes used in the development one except for the sentiment, which is the unknown.

II. Proposed approach

A. Preprocessing

Fig. 2 shows the applied workflow.

1) Data cleaning:

- *Duplicates* removal: in the development dataset, 1888 entries with repeated attribute *text* were found and removed, because otherwise they would have gotten a higher influence with respect to other texts.
- *Attributes removal*: some fields were dropped:.
  - **ids**: the id of the tweet should be uncorrelated with its sentiment.
  - **flag**: this attribute has only one unique value "NO_QUERY", thus it is useless for the analysis.
- *Transformations*:
  - Dates were converted from strings to timestamps and then normalized with a min-max scaler to fit in range $[0, 1]$.
  - Texts were preprocessed in various ways:
    - HTML entities were unescaped.
    - URLs, targets ("@user") and leading '#' in hashtags were removed (after twitter-specific features extraction).
    - Non-ASCII characters were replaced with similar ASCII ones (using `unidecode` python library [1]).
    - Capitalized letters were replaced with lowercase ones.
    - Tokenization was done by means of NLTK's `TweetTokenizer` [2].
    - Emoticons were replaced with english words explaining them (e.g. ":)" $\rightarrow$ "smiley") using an emoticon dictionary [3].
    - Stemming was performed using Porter algorithm to reduce a word into its stem [2].
    - Sequences of three or more times the same letter were truncated to length one or two [4].
    - Negations have been handled during lexicon-based features extraction [4].

2) Feature extraction: The features used during classification are the normalized timestamps and other features extracted from text.

- *Twitter-specific*: twitter gives the opportunity to add hashtags, user tags and external urls to each tweet. These characteristics have been summarized in the following features:
  - *#urls*: number of urls.
  - *#hashtags*: number of hashtags (identified by '#').
  - *#targets*: number of user tags (identified by '@').
- *Textual*: features representing simple statistics extracted from texts:
  - *#words*: number of words.
  - *#esclamation_marks*: number of esclamation marks.
  - *#question_marks*: number of question marks.
  - *#quotes*: number of double quotes.
  - *#capitalized_words*: number of words composed only by capital letters.

- *average_word_length*: average length of words.

- *Lexicon-based*:
  - *n-grams*: from every tweet, weighted monograms and bigrams were extracted.
    Monograms' weights are affected by:
    * Repeated letters: if a word contains a sequence of at least three times the same letter, the related monogram's weight is multiplied by 1.5.
    * Negations: if a monogram is inside a negation scope[1], its weight is multiplied by -1.
    * Number of repetitions inside the tweet: the final result is the sum of the weights of each occurrence of the term in the tweet.

    Bigrams' weights are the average of the weights of the monograms that compose them.

  - *Sentiment orientation*: for each n-gram, the sentiment orientation is computed using [4]

$$SO(w) = \\ = \text{sign}\left(\log\left(\frac{P[+,w]}{P[+]\cdot P[w]}\right) - \log\left(\frac{P[-,w]}{P[-]\cdot P[w]}\right)\right) \quad (1)$$

    where (referring to the training dataset)
    * $P[+]$ is the number of positive tweets divided by the number of tweets.
    * $P[-]$ is the number of negative tweets divided by the number of tweets.
    * $\frac{P[+,w]}{P[w]} = P[+|w]$ is the sum of the absolute values of weights of w related to a positive sentiment (considering both positive weights in positive tweets and negative weights in negative tweets) divided by the sum of the absolute values of all w's weights.
    * $\frac{P[-,w]}{P[w]} = P[-|w]$ is as $P[+|w]$, but considering discordant signs for weights and sentiments.

    $SO(w)$ equals $+1$ for positive oriented n-grams, -1 for negative oriented n-grams and 0 for neutral n-grams (in case they have the same precence in both negative and positive tweets).
  - *Natural entropy*: is a measure of how much oriented is an n-gram [4], it has a value in range $[0, 1]$ and it is 1 for strongly oriented n-grams.

$$NE(w) = \\ = 1 + P[+|w]\cdot\log\left(P[+|w]\right) + P[-|w]\cdot\log\left(P[-|w]\right) \quad (2)$$

    Previous lexicon features are not used directly as features by the classifier, but they are used as basis for the following ones, that are summaries of sentiment orientations and natural entropy scores for words in a tweet [4]:

[1]A negation scope starts with a negation word (e.g. "not") and ends with some kind of punctuation or conjuction (e.g. ";" or "but").

  - #positive_ngrams: sum of absolute values of weights of n-grams with concordand weight and sentiment orientation (positive weights with positive orientation and negative weights with negative orientation, they both contribute to the positiveness of the tweet).
  - #negative_ngrams: sum of absolute values of weights of n-grams with discordant weight and sentiment orientation.
  - sum_positive_entropies: as #positive_ngrams, but with terms weighted by natural entropies.
  - sum_negative_entropies: as #negative_ngrams, but with terms weighted by natural entropies.

  Fig. 1 shows some examples of word stems with high natural entropy, highlighting their sentiment orientation.

- *PoS-oriented*: features extracted considering Parts of Speech tag of each word, combined with polarities (sentiment orientations) computed during sentiment lexicon analysis. As proposed in [5], only the following PoS were considered: nouns, verbs, adverbs, adjectives, interjections. For each of those, the following features were extracted:
  - $\forall PoS \in \{\text{NOUN, ADV, VERB, ADJ, INTJ}\}$,

$$\sum_{w\in PoS} SO(w) \quad (3)$$

    that is the sum of all the sentiment orientations of the words belonging to each PoS. It is equivalent to difference between the number of words with positive SO (sentiment orientation) and the number of words with negative SO.
  - $\forall PoS \in \{\text{NOUN, ADV, VERB, ADJ, INTJ}\}$,

$$\sum_{w\in PoS} SO(w)\cdot NE(w) \quad (4)$$

    that is the sum of all the sentiment orientations of the words belonging to each PoS, weighted by natural entropy score.

- *Bag of Words*: BoW representation is used to encode words into fixed-length vectors by means of a spe-

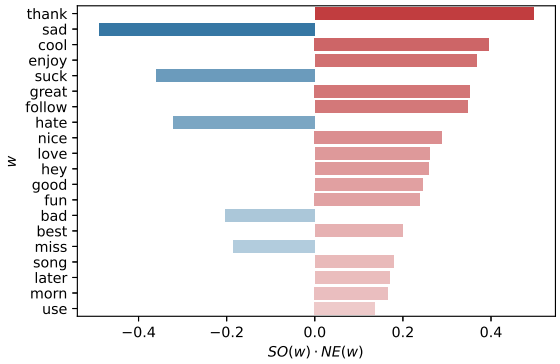

Fig. 1: Examples of strongly-oriented word stems.

cific weighting scheme. It has been shown that combination of ensemble features and BoW ones can enhance the results of classification [6]. Vectorization was performed by means of tf-idf weighting scheme, considering its good classification performances. Through `TfidfVectorizer` from `sklearn` library [7], unigrams and bigrams from positive and negative tweets were extracted and used as features. The number of features to extract in this step was decided during the hyperparameter tuning phase.

- *Non-textual* `date` and `user` attributes of the dataset were used to extract the following features:
  - *timestamp*: integer value representing the time the tweet was posted. Even if this attribute should be uncorrelated with the text and the sentiment of the tweet, it improves the results. This could be explained considering that there are intervals of time in which is more likely to have positive/negative tweets, as shown in [8].
  - *#user_tweets*: number of tweets posted by each user.
  - *user_avg_sentiment*: average sentiment of the user. It ranges in interval $[0, 1]$.

  #user_tweets and *#user_avg_sentiment* were computed on the training dataset. Then, using `user` attribute, each record in evaluation dataset was mapped to the corresponding values. For those users not present in the training dataset, *0* and the *average_training_set_sentiment* were used as default values.

These features were tested using a *Random Forest Classifier* with default parameters, splitting the development dataset in 80% for training and 20% for testing. Each group of features, one by one, was removed from the classifier input and tested against the whole group of features. Results are shown in Fig. 3.

B. Model selection

Various classification models were trained to predict sentiments [7]:

- *Random Forest*: an estimator that fits an ensemble of decision tree classifiers and uses averaging to improve the predictive accuracy and control over-fitting.
- *K-Nearest Neighbors*[2]: classification is computed from a simple majority vote of the nearest neighbors of each point.
- *Gaussian Naive Bayes*: Naive Bayes classifiers are algorithms based on applying Bayes' theorem with the "naive" assumption[3] of conditional independence between every pair of features given the value of the class variable. In this case the likelihood of the features is assumed to be Gaussian.
- *Static Vector Machines* ensemble[4]: a set of SVM classifiers, that use the hyperplane with greatest gap between two classes to separate them.

[2]features were normalized in this case

[3]our features aren't pairwise independent, but we decided to try it anyway

[4]using *Bagging Classifier* with Random Patches

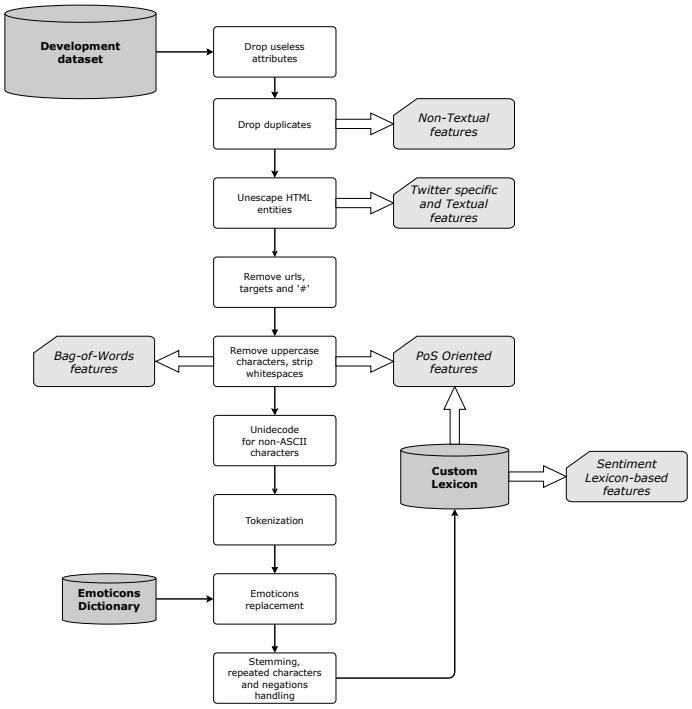

Fig. 2: Preprocessing workflow.

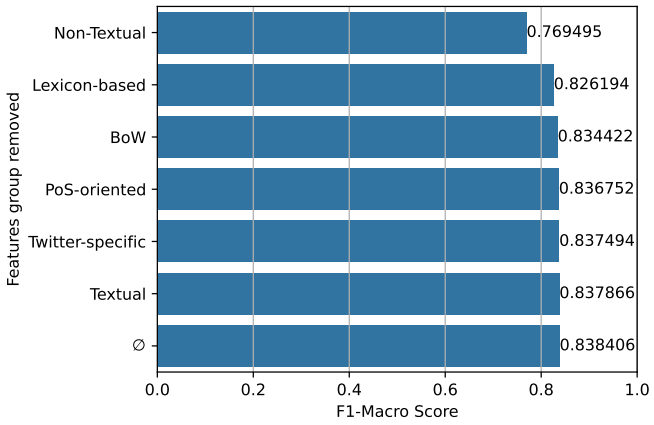

Fig. 3: Features testing results.

To select the model used for predicting evaluation sentiments, we trained them with default parameters on 60% of the development dataset and tested them on other 20%, obtaining the results showed in Table I. According to these results, Random Forest Classifier was chosen, as it outperformed all other classifiers by at least 0.05.

C. Hyperparameters tuning

To fit the chosen classifier in the best possible way, we splitted the development dataset in 3 (pairwise disjoint) subsets[5]:

- *Training* set: 60% of the dataset, used to train the model with various hyperparameters.
- *Validation* set: 20% of the dataset, used to validate the goodness of hyperparameters and select them.
- *Testing* set: 20% of the dataset, used to test the selected hyperparameters.

Table II shows the sets of values used to perform the tuning. Table III shows best 10 validation results.
According to these results,

- *criterion* = "gini"
- *max_features* = 5
- *min_samples* = 5

were chosen.

III. Results

Chosen hypermarameters were tested with the testing set described above, obtaining an F1-Macro of 0.8409. The classificator was then trained with the whole development dataset, using the same settings, to predict setiments of records in the evaluation dataset. The obtained public score was 0.842, 0.089 more with respect to the baseline 0.753 score.

IV. Discussion

During the implementation of the task, we thought about some other things that we could do, but that were not done, mainly due to time availability. Some of them are:

- Considering different weight factors for multiple letter repetitions in a word, instead of only 1.5.
- Expanding slang by means of a slang dictionary: an exhaustive and open source dictionary was not found.
- Implementing so called z-score features, introduced to us by [4].
- Using pre-trained model for text vectorization Word2Vec (GloVe, fastText).
- Trying other different hyperparameters for Random Forest during tuning phase.
- Hyperparameter tuning of other classifiers besides Random Forest, even if it seems to be a lot better than the others.

Anyway, the work that we did seems to be quite good: the computed features allowed every tried classifier, except for GaussianNB, to beat the baseline without hyperparameter tuning (considering local testing), and also comparing the obtained public score with scores higher than ours, they are for the majority less than 0.02 above us (at time of writing). We also note that the public score obtained is higher than the local testing score, which should be an indicator of non-overfitting of the classifier: this achievement is obtained also thanks to random forest intrinsic characteristics.

[5]holdout validation was done instead of cross-validation due to the high computational cost of preprocessing and the big size of the dataset

TABLE I: Models testing results

Classifier	**F1-Macro Score**
Random Forest	0.8336
KNN	0.7687
GaussianNB	0.7183
SVM ensemble	0.7758

TABLE II: Hyperparameters' values

Hyperparameter	**Values set**
n_estimators	100
criterion	"gini" "entropy"
max_features	1 2 3 4 5 6 "auto" "log2"
min_samples_split	2 3 4 5

TABLE III: Hyperparameters tuning results

criterion	max_features	min_samples_split	F1-Macro Score
"gini"	5	5	0.8379
"gini"	5	3	0.8378
"gini"	5	2	0.8377
"gini"	"log2"	5	0.8376
"gini"	5	4	0.8376
"gini"	4	6	0.8376
"gini"	3	3	0.8375
"gini"	3	4	0.8374
"gini"	3	5	0.8374
"gini"	5	7	0.8373

References

[1] T. Solc, avian, and B. Bangert, "Unidecode." https://pypi.org/project/Unidecode/, 2021.

[2] S. Bird, E. Loper, and E. Klein, *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[3] N. Shah, tarikaltuncu, and A. Singh, "emot." https://github.com/NeelShah18/emot/blob/master/emot/emo_unicode.py, 2021.

[4] H. Hamdan, P. Bellot, and F. Bechet, "Sentiment lexicon-based features for sentiment analysis in short text," in *16th International Conference on Intelligent Text Processing and Computational Linguistics*, 2015.

[5] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau, "Sentiment analysis of twitter data," *Proceedings of the workshop on language in social media (LSM 2011)*, pp. 30–38, 2011.

[6] M. R. Irfan, M. A. Fauzi, T. Tibyani, and N. D. Mentari, "Twitter sentiment analysis on 2013 curriculum using ensemble features and k-nearest neighbor," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 8, no. 6, pp. 5409–5414, 2018.

[7] 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.

[8] P. S. Dodds, K. D. Harris, I. M. Kloumann, C. A. Bliss, and C. M. Danforth, "Temporal patterns of happiness and information in a global social network: Hedonometrics and twitter," *PLoS ONE 6(12)*, vol. e26752, 2011.