Sentiment Analysis - FDS project report

Alessandro Pecchini, Alessio Palma and Davide Santoro

Abstract— **This report focuses on emotion classification from comments written by people on Reddit (a social news aggregation, web content rating, and discussion website). We grouped the different emotions into three main categories: positive, negative and ambiguous and we achieved 62% of accuracy without Deep Learning classification approaches.**

I. INTRODUCTION

Nowadays more and more people convey their emotions through Internet (especially through social network) and express their mood with comments. Consequently, there are a lot of emotions coming out from these comments that can be grouped, and understanding the association between the comment and the feelings can be useful for different aims. For example, with Sentiment Analysis it's possible to recommend an appropriate emoticon for a sentence that we are writing, to measure the level of customers satisfaction from reviews and to filter dangerous comments on websites. So Sentiment Analysis is definitely an important task, and it is also not trivial in fact we need to:

- clean the dataset;
- transform text into numerical values;
- choose a model that is able to learn from these numerical values;
- test this model to evaluate its performances;
- interpret the results to understand its decisions.

II. RELATED WORKS

Sentiment Analysis is a task that is tackled a lot in research, using various text encodings and classification algorithms. Some more traditional approaches like the one proposed by Jaswanth et al. [3] consist of classification using Logistic Regression and Bag Of Words (BOW) model, which is a commonly adopted setup before the advent of Deep Learning. With the explosion of the latter, Sentiment Analysis has become mainly a Deep Learning related task, in fact these algorithms contributed to achieving outstanding performances, like we can see in the work of Badjatiya et al. [4], which performed hate speech detection from online tweets with Convolutional Neural Networks and Long Short-Term Memory. It is also important to underline the fact that this task is highly affected by the quality of the features (i.e. the text itself), as demonstrated by Li et al. in [5]. If the text is very informal and full of slang words, as the dataset chosen by us for this project is, then correctly classifying sentences is more difficult than it would be with homogeneous and more formal datasets.

III. PROPOSED METHOD

To perform Sentiment Analysis we faced with different steps:

A. Cleaning

To clean the dataset we removed the punctuation and we have tried different combinations of the following techniques: stopwords removal, making the text lowercase, removal of words appearing less than 5 times (because they don't contribute to give a sentimental characterization to the sentences and help us to greatly reduce the size of feature vectors) and applying lemmatization (the process of grouping together the inflected forms of a word, replacing them by their dictionary form).

B. Embedding

We experimented different word embeddings, the first two are techniques focused on the words frequency and aren't affected by the order, the third is a neural network based approach:

1) Tf-idf: The document is embedded by a vector that has the same size of the dictionary. Each position contains an index *tf-idf* that indicates how much a single word is important in the specific document (tf) with respect to the inverse of its total frequency among all the documents (idf). This index is computed by:

$$TF-IDF(t,d|D) = tf(t,d) \cdot idf(t,D) \tag{1}$$

Where

$$tf(t,d) = \frac{\sum_{t_i \in d} 1(t_i = t)}{|d|} \tag{2}$$

$$idf(t,D) = log(\frac{|D|}{\sum_d 1(t \in d)}) \tag{3}$$

In this equations D is the set of documents, t is the specific word and d is the single document.

2) Bag of Words: It's a way of extracting features from text that, as the previous techinque, maps each document into a vector with the same size of the dictionary. Each value in these vectors represents the number of occurrences of a particular word in the document.

3) Word2Vec: It is a shallow, two-layer neural networks which is trained to reconstruct linguistic contexts of words. It takes as its input a large corpus of words and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the vector space such that words that share common contexts in the corpus are located in close proximity to one another in the space. Word2Vec is a particularly computationally-efficient

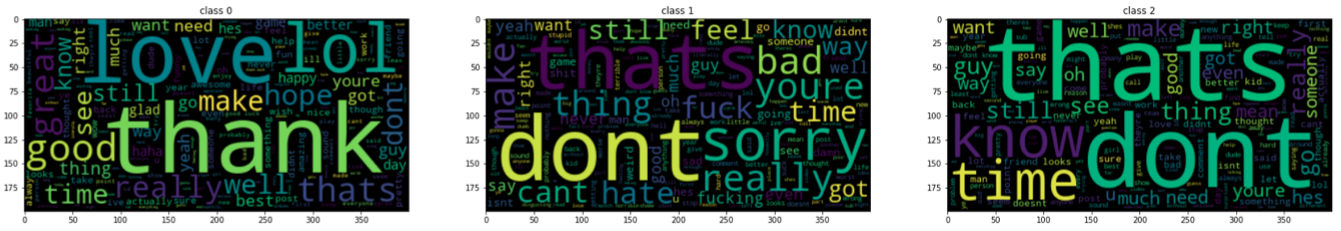

Fig. 1: Word clouds

predictive model for learning word embeddings from raw text. It comes in two flavors, the Continuous Bag-of-Words (CBOW) architecture and the Skip-Gram (SG) architecture. These models are quite similar and are explained in the notebook. Like all neural networks Word2Vec has weights, and during training its goal is to adjust those weights to reduce a loss function. However, Word2Vec is not going to be used for the task it was trained on, instead, we will just take its hidden weights, use them as our word embeddings, and toss the rest of the model.

C. *Models*

All the models we used are part of the scikit-learn Python library [2] and are the following:

- Multinomial/Gaussian Naive Bayes
- Logistic Regression (with multi_class = 'multinomial')

Gaussian Naive Bayes was used only for Word2Vec embeddings, since we have continuous features. Multinomial Naive Bayes can fit well on BOW and Tf-idf encodings as reported in the scikit-learn documentation.

IV. DATASET

"GoEmotions: A Dataset of Fine-Grained Emotions" [1] is an human-annotated dataset of 58k Reddit comments extracted from popular English-language subreddits and labeled with 28 emotion categories (12 positive, 11 negative, 4 ambiguous and 1 neutral). In particular, in addition to the columns related to emotions, this dataset includes an id, the text content of the comment and an indicator of the clarity of the comment. In order to work with this dataset we removed the duplicated comments, the useless columns and the samples marked as unclear. We also mapped the different emotions in three main categories (0 = positive, 1 = negative and 2 = ambiguous), so the resulting dataset (which is still quite balanced) only contains information about the text and the relative group of emotion. In figure 1 we can see the most common words for every class.
Hence we went from a dataset with 31 columns to our final dataset with only two columns, one containing the comment and the other containing the emotion category which the comment belongs to.

V. EXPERIMENTAL RESULTS

A. *Setup*

We have evaluated all the embedding techniques over the two chosen models, in order to compare them. The metrics that we've used to evaluate their performances are: precision, recall and f1-score for each class and accuracy for the overall comparing. We also used confusion matrices in order to better appreciate the obtained results.

B. *Comparison and results*

Since we've tested different combinations of text cleaning techniques, we'll report only the best results obtained by each encoding on the two models. Many other tests can be found on the project's notebook, each test is replicable thanks to the preprocessed datasets that you can find in our shared Google Drive folder.

1) *Tf-idf:*

Here it's difficult to find the real best preprocessing combination, but surely the lemmatization doesn't bring any improvement (it actually worsen the overall accuracy) and it's interesting how stopwords presence bring a great improvement, this is probably due to the fact that they give more meaning to the sentence and in task of this kind (where we need to link the presence of specific group of words to some emotion) even stopwords are important.

Multinomial Naive Bayes with stopwords and lowercase:
Overall accuracy: 57%
Classification report:

Emotion	Precision	Recall	F1-score
Positive	0.63	0.68	0.65
Negative	0.54	0.32	0.40
Ambiguous	0.52	0.61	0.56

Confusion matrix:

Real values	Predicted values		
	Positive	Negative	Ambiguous
Positive	0.63	0.16	0.25
Negative	0.13	0.54	0.23
Ambiguous	0.25	0.29	0.52

Logistic Regression with stopwords and lowercase:
Overall accuracy: 59%
Classification report:

Emotion	Precision	Recall	F1-score
Positive	0.66	0.69	0.68
Negative	0.56	0.39	0.46
Ambiguous	0.54	0.61	0.57

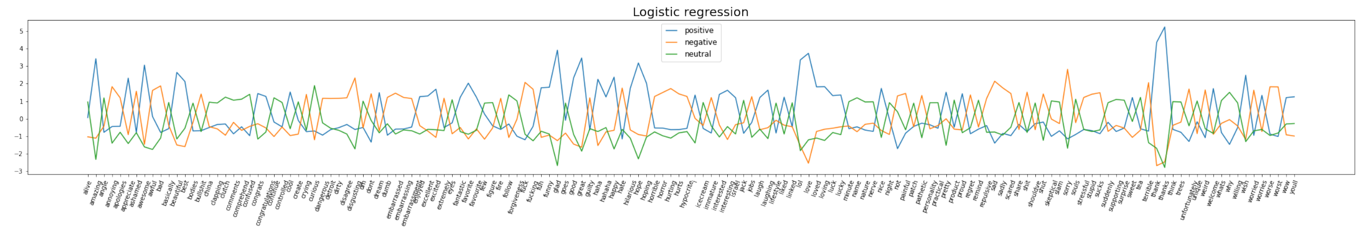

Fig. 2: Tf-idf Logistic Regression weights

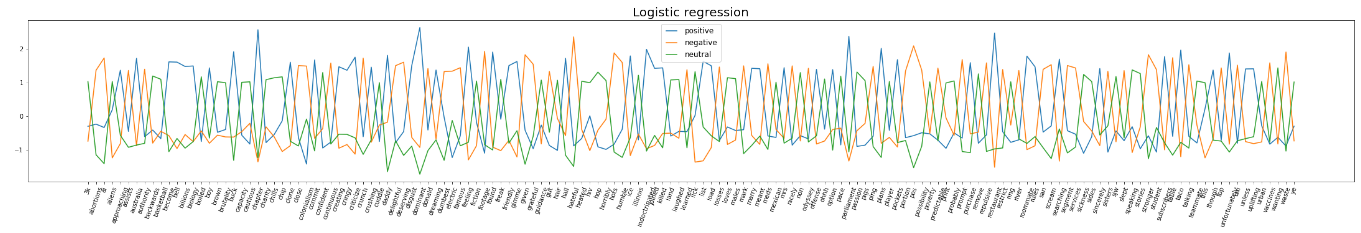

Fig. 3: BOW Logistic Regression weights

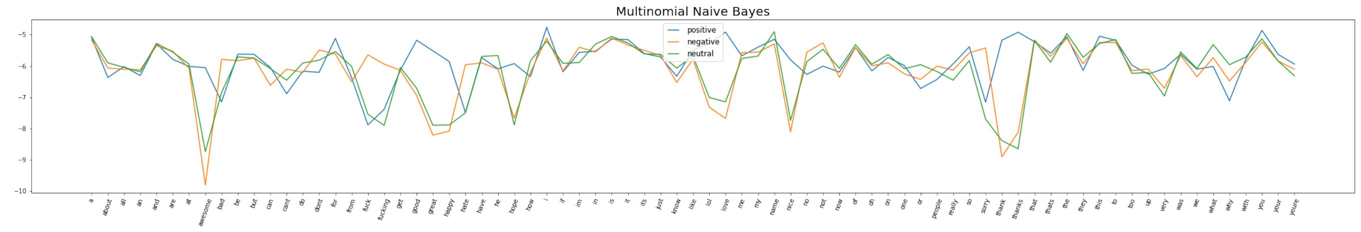

Fig. 4: Tf-idf Naive Bayes likelihoods

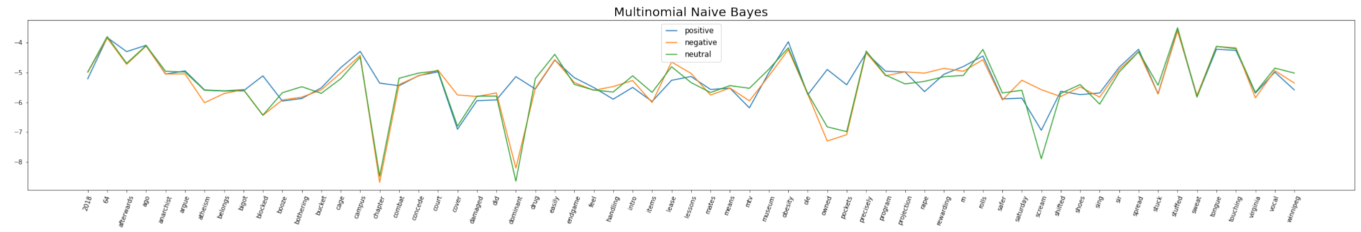

Fig. 5: BOW Naive Bayes likelihoods

Confusion matrix:

Real values	Predicted values		
	Positive	Negative	Ambiguous
Positive	0.66	0.15	0.23
Negative	0.10	0.56	0.23
Ambiguous	0.24	0.29	0.54

2) Bag of words: The best results with Naive Bayes method are achieved by maintaining stopwords and with lowercase text. Of course, lowercase text reduces variation and the number of different words in the dataset, and although one might think that the same word written in uppercase and lowercase represents different emotions in the end the tests tell us that bringing everything into lowercase improves performances, and moreover it has the advantage of reducing the size of the corpus used to train the model.

Multinomial Naive Bayes with stopwords and lowercase:
Overall accuracy: 61%
Classification report:

Emotion	Precision	Recall	F1-score
Positive	0.66	0.73	0.69
Negative	0.55	0.42	0.48
Ambiguous	0.57	0.59	0.58

Confusion matrix:

Real values	Predicted values		
	Positive	Negative	Ambiguous
Positive	0.73	0.065	0.21
Negative	0.21	0.42	0.37
Ambiguous	0.27	0.14	0.59

Logistic Regression with stopwords and lowercase:
Overall accuracy: 62%
Classification report:

Emotion	Precision	Recall	F1-score
Positive	0.71	0.70	0.70
Negative	0.56	0.46	0.50
Ambiguous	0.57	0.65	0.61

Confusion matrix:

Real values	Predicted values		
	Positive	Negative	Ambiguous
Positive	0.7	0.071	0.23
Negative	0.17	0.46	0.38
Ambiguous	0.21	0.14	0.65

3) Word2Vec: The best combination of preprocessing techniques turns out to be: lowercase, removed stopwords and removed infrequently occurring words for NB; lowercase, removed infrequently occurring words and lemmatization for LR. Removing rare words here seems to always help, both in obtaining a lighter corpus to train the Word2Vec model on and in increasing accuracy. Skip-gram architecture performs better than the CBOW one, and for LR we gained in accuracy with increasing the embeddings size from 100 to 1000, meanwhile for NB this didn't help. Due to the fact that this method produces the shortest embeddings and is not computationally too expensive, we here also performed 5-fold cross validation and hyperparameter tuning for Logistic Regression (since scikit-learn's Gaussian Naive Bayes doesn't have any relevant hyperparameter to tune). The best hyperparameters are max_iter = 2000 and penalty = "None".

Gaussian Naive Bayes with lowercase, removed stopwords and removed rare words. Skip-gram architecture with embedding size = 100:
Overall accuracy: 45%
Classification report:

Emotion	Precision	Recall	F1-score
Positive	0.61	0.41	0.49
Negative	0.32	0.51	0.39
Ambiguous	0.46	0.45	0.45

Confusion matrix:

Real values	Predicted values		
	Positive	Negative	Ambiguous
Positive	0.41	0.28	0.31
Negative	0.17	0.51	0.32
Ambiguous	0.18	0.37	0.45

Logistic Regression with lowercase, removed rare words and lemmatization. Skip-gram architecture with embedding size = 1000:
Overall accuracy: 60%
Classification report:

Emotion	Precision	Recall	F1-score
Positive	0.69	0.68	0.68
Negative	0.57	0.31	0.40
Ambiguous	0.53	0.69	0.60

Confusion matrix:

Real values	Predicted values		
	Positive	Negative	Ambiguous
Positive	0.68	0.046	0.28
Negative	0.19	0.31	0.49
Ambiguous	0.21	0.097	0.69

C. Models interpretations

Due to Word2Vec model's architecture it's impossible to extract one scalar weight associated with each unique word of the corpus, for this reason we've extracted only the parameters of the simplest encoding techniques. The best accuracy is achieved with BOW model and Logistic Regression, reaching **62%**, which is not at all a bad result considering the fact that we are using "traditional" classification techniques on a dataset which is made for Deep Learning and fine grained predictions. With every encoding and preprocessing technique Naive Bayes performs worse than Logistic Regression, and that's due to the fact that Naive Bayes makes a strong assumption: it assumes that each feature is independent from the others, but of course that's not true in a task like Sentiment Analysis, where the words used are dependent on each other in order to determine a context that produces a specific emotion. We can see that features are actually correlated in figure 6, although correlation does not always imply dependence.

1) Tf-idf and BOW: Here is interesting to note that higher weights are given to bad words for the negative class, as for example we can see in figure 4 with the word "f*ck". The vice versa happens with the positive words in the positive class, as we can see in figure 2 with the words "haha" and "hahaha", representing a laugh. In figure 3 we can see that the dataset contains bias, in fact the word "abortions" has high negative weight and low positive and neutral weights, also "australia" has an higher positive weight even if it is a completely neutral word like "illinois", which in fact has balanced weights among classes. This is surely due to the context of the sentences in which these words appear in the dataset, since reddit comments are unfiltered and everyone can write. Of course different methods give us different parameters shapes.

2) Word2Vec: Feature dimension in Word2Vec can be made custom, respect to BOW and Tf-idf where it is equal to the vocabulary size. It is worth noting that as it is trying to capture only necessary things to describe the words it is a good compromise between accuracy and computational complexity. We also noticed that there was a 9% accuracy

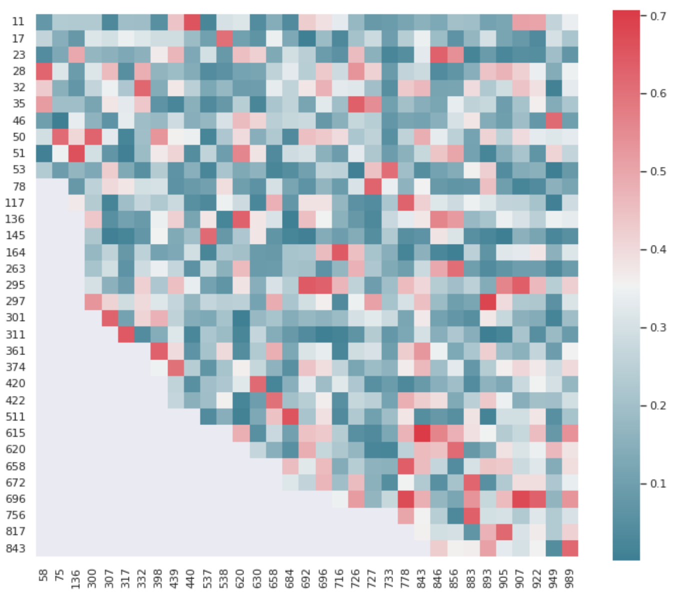

Fig. 6 A part of the Pearson's correlation matrix of the features produced by Word2Vec skip-gram architecture with embedding size = 1000

increase for words of negative class when we increased embedding size from 100 to 1000. This is because in a higher dimensionality space we can better separate vectors belonging to three different classes, although by further increasing the embedding size the results will probably start to be worse due to the fact that data becomes too sparse and dissimilar in many ways (i.e. curse of dimensionality). We also gave a try to Logistic Regression with Polynomial features, but it reached 55% accuracy on a model that reached 57% accuracy with linear features, other than the fact that it takes much more time and memory to train. So we have chosen not to continue with this path. We also performed Principal Component Analysis, as can be seen in the notebook.

VI. LEVERAGED SOURCES

For the text cleaning and preprocessing techniques we took inspiration from here. For the BOW model we reused a code snippet from this page. For the Tf-idf model we learnt the theory from this link, meanwhile for the Word2Vec model we used a code snippet from here.

REFERENCES

[1] "GoEmotions: A Dataset for Fine-Grained Emotion Classification." Google AI Blog. October 28, 2021. Accessed December 22, 2021, https://ai.googleblog.com/2021/10/goemotions-dataset-for-fine-grained.html

[2] "Scikit-learn: Machine Learning in Python", Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

[3] Jaswanth, Sandeep Kumar et al. "Sentiment analysis using logistic regression algorithm". European Journal of Molecular & Clinical Medicine, 7, 4, 2020, 2081-2086

[4] Badjatiya, Pinkesh et al. "Deep Learning for Hate Speech Detection in Tweets". Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion. (2017), https://doi.org/10.1145/3041021.3054223

[5] Li, Goh et al. "How textual quality of online reviews affect classification performance: a case of deep learning sentiment analysis". Neural Comput. & Applic. 32, 4387–4415 (2020), https://doi.org/10.1007/s00521-018-3865-7