Opinion Mining Evaluation Forum -Team Iconic

Beatriz Gonçalves – m20210695 | Gonçalo Lopes – m20210679 | Guilherme Simões – m20211003

1. Best Approach

After importing the given data, we had to apply some preprocessing to it. The first step was to make the soup, in other words, parsing the provided data with Beautiful Soup to transform the data into a tree of python objects. Then we lowered the text in order to make it easier to analyze. The following procedure was to remove digits and special characters so that there were no incoherencies in the sentences. Removing the stopwords was what followed the previous step. Here we chose to use the stopwords list that comes in the NLTK (Natural Language ToolKit) but after executing a complete preprocessing, we noticed that some of the sentences were empty due to the extensive range of words that come in this list. To solve this problem, we created a list with words that should not be removed, to limit the deleted stopwords. Finally, the last preprocessing step was to use stemming to reduce similar words to a root word.

In order to extract features from the given sentences, our approach was to use CountVectorizer which is the simplest way to convert text to vector and count the word occurrences in the corpus. But using word count as feature value of a word isn't accurate enough to define the word's importance in a document. So, we introduced TF-IDF (Term Frequency-Inverse Document Frequency) to have the ratio of the count of a word's occurrence in a document and the number of words in the document, transforming the count values produced by CountVectorizer in TF-IDF heights, and scaling down the impact of tokens that occur very frequently and the other way around.

At this point we were ready to fit some models to the data and see what worked best for us. The model that gave us the best accuracy and the least overfitting was the XGBoost (Extreme Gradient Boosting).

2. Experimental Setup

As mentioned before, we started by preprocessing the given datasets. After the Preprocessing, the next step was the feature extraction which was implemented in two different ways.

In the first one, we used Bag-of-words and TF-IDF Transformer, where the CountVectorizer was used to convert a collection of text documents to a matrix of token counts and the TF-IDF Transformer computed the words counts to the Inverse Document Frequency (IDF) values and only then compute the TF-IDF scores. The second implementation of the feature extraction was based on TF-IDF Vectorizer since it combines counting and term weighting.

Subsequently, various models were implemented, in order to find the one with the best reasonable accuracy. For that, models such as Random Forest, XGBoost, SVC and KNN were considered.

In the end, XGBoost was the one with the best evaluation according to the metrics: Precision, Recall and F1 (per label and macro avg), Accuracy and Confusion Matrix. Overfitting was also calculated for each model and XGBoost had the best result, although it was not considered a metric for model evaluation.

The list of Python libraries needed to run our code in the notebook can be consulted below:

- OS
- Numpy
- Pandas
- Random
- Re
- NLTK
- Bs4
- Sklearn
- Matplotlib

3. Evaluation of Our Best Approach

Regarding the evaluation of our best approach, we can see, in the tables below, the comparison of the performance between the two models that performed better throughout the project. Keep in mind that XGBoost delivered the best result and had CountVectorizer TF-IDF Transformer has Feature Extraction methods, as mentioned before. On the other hand, the KNN model was our baseline (i.e., a weaker approach we tried) which had TF-IDF Vectorizer has Feature Extraction method and the same preprocessing as the other models. In the following Figures it is possible to compare both approaches.

	precision	recall	f1-score	support
Anger	0.33	0.66	0.44	211
Anticipation	0.41	0.42	0.42	170
Disgust	0.24	0.12	0.16	77
Fear	0.43	0.22	0.29	104
Joy	0.47	0.40	0.43	97
Sadness	0.36	0.24	0.29	87
Surprise	0.35	0.18	0.23	96
Trust	0.43	0.35	0.38	158
accuracy			0.38	1000
macro avg	0.38	0.32	0.33	1000
weighted avg	0.38	0.38	0.36	1000

Figure 1 - Precision, Recall and F1-Score of XGBoost

	precision	recall	f1-score	support
Anger	0.50	0.36	0.42	292
Anticipation	0.35	0.36	0.35	164
Disgust	0.10	0.17	0.13	47
Fear	0.24	0.27	0.25	93
Joy	0.36	0.32	0.34	111
Sadness	0.22	0.28	0.25	68
Surprise	0.23	0.28	0.25	79
Trust	0.35	0.38	0.36	146
accuracy			0.33	1000
macro avg	0.29	0.30	0.29	1000
weighted avg	0.35	0.33	0.34	1000

Figure 2 - Precision, Recall and F1-Score of KNN

After analyzing the results obtained by both models, we can see that the f1-score accuracy of the XGBoost model is better than the KNN one, but both models perform well when it comes to the sentiment analysis (considering the datasets used). However, the bigger difference can be spotted in some of the sentiments that the model analyses, mostly when it comes to Anger, where the XGBoost clearly outperforms the KNN model.

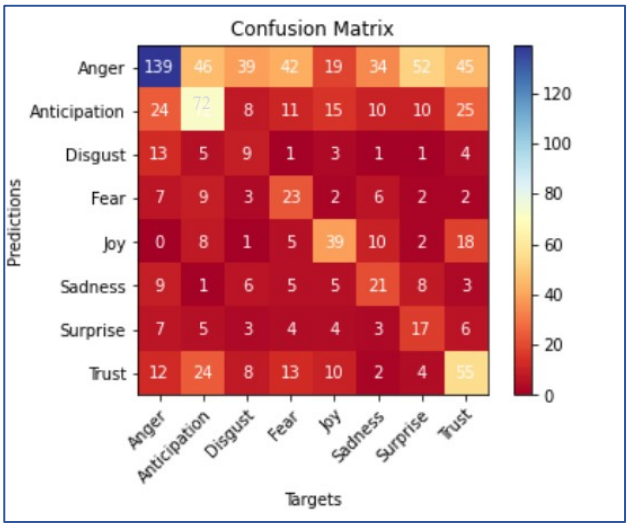

Figure 3 - Confusion Matrix of XGBoost

Moreover, and considering the Confusion Matrix of our chosen model, we can clearly see that there are a good number of sentences that are a fair number of sentences that are wrongly classified as Anger. Besides that, the models classify well enough sentiments such as Anger, Anticipation, Joy and Trust.

To conclude, the given datasets were not easy to work on because the sentences sometimes lack words for the model to identify the sentiments associated to each sentence. Despite that, the preprocessing had a fundamental influence in the results, as well as the feature extraction and the best model. With this project, we were able to consolidate the knowledge transmitted in the Text Mining classes.

4. Future Work

Since the given dataset was not that complex, we decided that implementing a Neural Network wouldn't be beneficial to improve the result we had. But if we were to work in a more complex one, we would implement a RNN with an Embedding layer to solve the classification problem. We could also train our RNN to detect spam sentences, for example if some sentences don't have an emotion associated, the network would classify it as spam.

5. References

www.datacamp.com. (n.d.). Python Sentiment Analysis Tutorial. [online] Available at: https://www.datacamp.com/tutorial/simplifying-sentiment-analysis-python

Stack Overflow. (n.d.). python - what is the difference between tfidf vectorizer and tfidf transformer. [online] Available at: https://stackoverflow.com/questions/54745482/what-is-the-difference-between-tfidf-vectorizer-and-tfidf-transformer

kavgan (2022). nlp-in-practice/TFIDFTransformer vs. TFIDFVectorizer.ipynb at master · kavgan/nlp-in-practice. [online] GitHub. Available at: https://github.com/kavgan/nlp-in-practice/blob/master/tfidftransformer/TFIDFTransformer%20vs.%20TFIDFVectorizer.ipynb

Readthedocs.io. (2021). Python API Reference — xgboost 1.5.0 documentation. [online] Available at: https://xgboost.readthedocs.io/en/stable/python/python_api.html

Scikit-learn (2018). 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.20.3 documentation. [online] Scikit-learn.org. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html