Algorithmic Machine Learning

Challenge 3 - Technical Report

Group 20

Professor: Pietro Michiardi
Authors: Daniele Falcetta
Simone Papicchio
Massimiliano Pronesti
Federico Tiblias

June 8, 2022

1 Introduction

With the increasing amount of content and debate on social media platforms such as Twitter, there is an interest in automatically extracting meaning and sentiment from users' posts, to be able to evaluate the aggregate opinion of a large number of users.

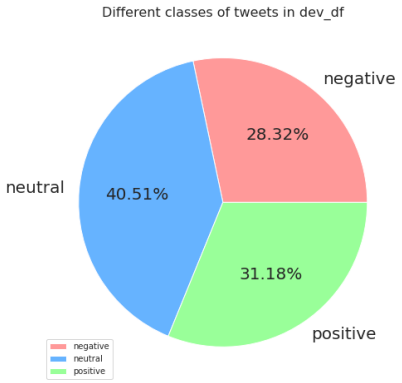
The dataset used for this challenge consist of tweets from Figure Eight's Data for Everyone platform. The main task is to classify tweets with positive, neutral, or negative labels. We compare different solutions: from techniques based on word embeddings, to LSTM models, to state-of-the-art approaches using pre-trained transformers. This challenge proposes the additional task of finding the most important words in a sentence. We propose two approaches: one based on word counts and the other on *Transformers Interpret* and on *Captum* [1], a tool and library, respectively, for model interpretability and visualization.

2 Data Analysis & Pre-processing

2.1 Data exploration

Tweets are, in general, quite far from plain English text, as they usually contain slang, abbreviations, emojis, emoticons and misspellings. Those available in the given dataset are no exception.

We start our data exploration by studying the label distribution. As we can see from the pie chart in Figure 1, the dataset is slightly imbalanced towards the *neutral* class whereas almost balanced between *positive* and *negative* tweets. A further analysis on the text length distribution of each class reveals that almost each category of tweets has the same one, as shown in Figure 1 for the *positive* class. In addition, looking at the Wordclouds for the *positive* and *negative* tweets shown in Figure 2, we can also notice, as expected, that the most present terms are words like *love*, *good* and *thanks* for the former and *sad*, *work* and *miss* for the latter.



Distribution of text length for positive sentiment tweets.

	length
count	7711.0
mean	13.12
std	6.76
min	1.0
25%	8.0
50%	12.0
75%	18.0
max	33.0

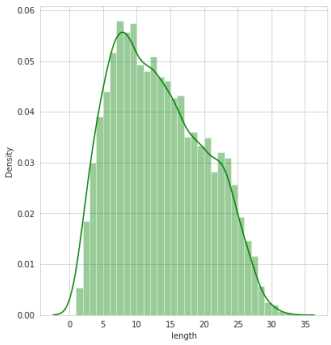


Figure 1: Classes and text length distribution in the training dataset

2.2 Pre-Processing

- **Emojis and Emoticons:** Punctuation marks, letters, and numbers used to create pictorial icons that generally display an emotion or sentiment. We convert them to the equivalent textual representation of the emotion they attempt to convey employing the *emot* library. A complete and more effective parsing required, however, integrating the original dictionary with some missing entries[1].
- **Abbreviations and Slang:** Similarly to emojis and emoticons, we do a raw text preprocessing in order to map the most famous slangs and abbreviations with the correct word in regular English.
- **SpellChecker:** Employing the *SpellChecker* library, we attempt to convert misspelled word to correct English.
- **Tokenization:** Tokenization consists in breaking the raw text into small chunks called tokens. Many algorithms use tokens as input to conduct further analysis.

[1] the "complete" list is taken from Wikipedia: https://en.wikipedia.org/wiki/List_of_emoticons

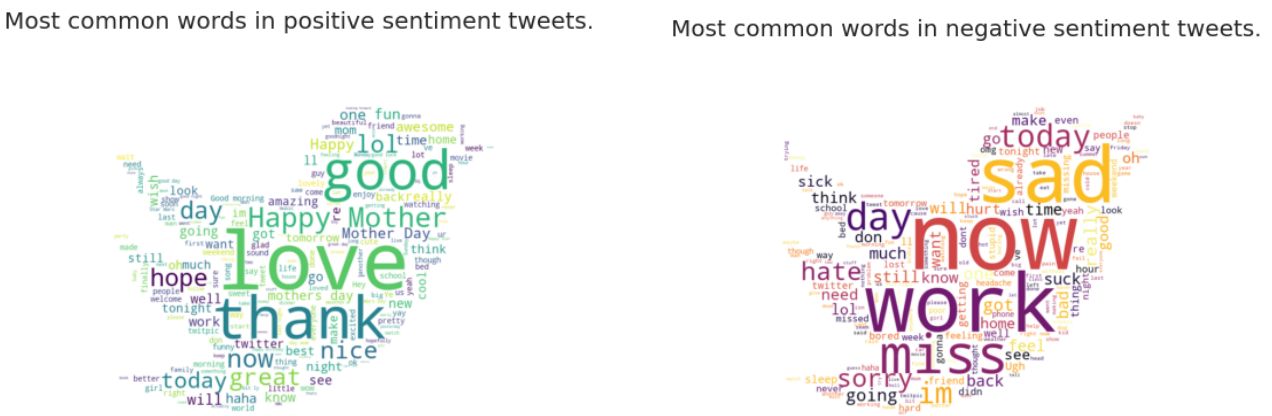


Figure 2: Wordclouds for the training dataset

- **Stopwords removal:** Stopwords are a set of commonly used words in any human language. For instance, examples of stopwords in English are: "the", "is" and "and". It's therefore a common practice to remove them to allow models to focus on the important words. The stopwords dictionary is taken from the *nltk* library.
- **Stemming & Lemmatization:** The goal of both method is to reduce inflectional forms and sometimes related forms of a word to a common base form. Stemming uses the stem of the word, while lemmatization uses the context in which the word is being used.

2.3 Augmentation

While our training dataset contains roughly 25k entries, it still pales in comparison to the huge amount of data transformer models were trained on. To better fine-tune our pretrained models, we try different techniques to increase the number of available datapoints.

- **Concatenation:** Build new tweets by concatenating existing ones. Couples are formed randomly and kept only if the tweets have the same label and the resulting tweet is < 140 characters long. This procedure can be repeated producing datasets of increasing size.
- **Backtranslation:** Transformer models are capable of translating back and forth from a language. Due to linguistic differences and imperfections in the model, the backtranslation is not always a perfect replica of the original sentence but maintains the original meaning. We exploit this phenomenon to build new sentences that have the same meaning of the original one, but a different structure. German is chosen for this task as it's the closest language to English that also has readily available pre-trained models. In particular, we rely on Hugging Face's `Helsinki-NLP/opus-mt-en-de` and `Helsinki-NLP/opus-mt-de-en`. Generated sentences are kept only if they are different than the original ones.

2.4 Word Embeddings

Word embeddings are a type of representation that allows words with similar meaning to have a similar representation[2]. Before embedding our data we perform all preprocessing steps described in Section 2.2.

- **Word2Vec:** Two-layer neural network that processes text by "vectorizing" words. Word2vec groups the vectors of similar words together in a vector space by detecting similarities. For this task, we train[3] the neural network[4] on the processed tweet, creating a feature space of size 18964 unique tokens x 663 vector size. The main drawbacks of this representation are that it is unable to model the sequences of word (solved by LSTM), it ignores the global word co-occurrence (solved by GloVe), not designed for encoding contextual representation (solved by Transformers)

[2] Distributional Hypothesis: given a word, we look for the semantically related words that are nearby
[3] *context window 5, negative sampling 10, skip-gram model 1, vector size 663*
[4] *https://radimrehurek.com/gensim/models/word2vec.html*

- **GloVe[2]:** Unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase linear substructures of the word vector space. We download the embedding containing the glove vectors trained on the Wikipedia and GigaWord dataset. The feature space size is 400000 unique tokens x 50 vector size. The main drawback of this embedding is that it is not trained on our tweets, i.e. most of the words are completely unknown to the space.

3 Models

Transformers are undoubtedly the gold standard for many NLP tasks including sentiment analysis. Nevertheless, we also try different approaches to see how they compare with the state of the art.

Scores are computed by first splitting the labelled data in development and validation datasets (80/20 random split), then training on the development part, and finally assessing performances on the unseen validation subset. If the model reaches satisfying results is then applied on the unlabelled test dataset and predictions are uploaded on Kaggle.

- **KNN/Random Forest/CatBoost:** We exploit the embedded text to attempt tackling the sentiment analysis problem in a "classical" machine learning fashion, using it as training data to be fed to a KNN, a Random Forest and a Catboost classifier.

- **Convolutional LSTM:** This architecure is used with both Word2vec Tab.1 and GloVe Tab.2. In both cases, the architecture takes as input the pre-processed sentence already tokenized. Each token is mapped into the respective feature vector. The output of the embedding is passed to a Spatial Dropout with different percentages for the architectures. A convolutional layer with 64 kernels size 5 creates the input for the LSTM. Before the classification, there are several fully connected layer. The tables below show the differences between the architectures.

- **Transformer:** A transformer is a deep learning model which adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data. Unlike RNNs, transformers process the entire input all at once. They do so through the attention mechanism, a technique able to enhance some parts of the input data while diminishing others, mimicking how real brains focus attention on specific aspects of information. In particular we use "Twitter-roBERTa-base for Sentiment Analysis", a 125M parameter transformer model pretrained on the TweetEval dataset[3], which we further fine-tune for our task.

Layer	Output shape	Param #
InputLayer	[bs, 33]	0
Embedding	[bs, 33, **663**]	**131,532,570**
Spatial Dropout **(0.6)**	[bs, 33, 663]	0
Conv1D	[bs, 29, 64]	212,224
Dropout (0.6)	[bs, 29, 64]	0
Bidirectional LSTM	[bs, **128**]	**98.816**
Dropout **(0.4)**	[bs, 128]	0
Dense	[bs, **512**]	**131,584**
Dropout **(0.4)**	[bs, 512]	0
Dense	[bs, 512]	524,800
Dense	[bs, 3]	1,539

Table 1: LSTM Architecture with Word2Vect embedding, 132,501,533 trainable parameters

Layer	Output shape	Param #
InputLayer	[bs, 33]	0
Embedding	[bs, 33, **50**]	**9,919,500**
Spatial Dropout **(0.8)**	[bs, 33, 50]	0
Conv1D	[bs, 29, 64]	**16,064**
Dropout (0.6)	[bs, 29, 64]	0
Bidirectional LSTM	[bs, **256**]	**197,632**
Dropout **(0.6)**	[bs, 256]	0
Dense	[bs, **1024**]	**263,168**
Dropout **(0.6)**	[bs, 1024]	0
Dense	[bs, 512]	524,800
Dense	[bs, 3]	1,539

Table 2: LSTM architecture with the GloVe embedding, 10,922,703 trainable parameters.

4 Results

- **KNN/Random Forest/CatBoost:** As expected, these approaches proved unsuccessful, with a validation score below 60%.

- **Conv LSTM:** The results are heavily influenced by the selected feature space. The Word2vec LSTM is able to obtain almost 88% of training accuracy but only 68% for validation accuracy. We try addressing the overfitting problem by introducing several dropout layers. However, this solution combined with GloVe produces even worse

results, achieving a score in the ballpark of 40% both in training and validation sets. This low accuracy is mostly caused by the domain adaptation problem, as we download a feature space pretrained on Wikipedia + GigaWord dataset, which is not similar to our dataset.

- **Transformer:** As expected, transformer models outperform all other approaches. This can be attributed to their internal complexity and extensive training[3]. We also point out that preprocessing to remove emoticons, abbreviations, slang, misspellings and URLs make the model perform slightly better than without.

Interestingly enough, data augmentation does not provide any meaningful improvements to any model.

5 Bonus Task

With the increase in model complexity and the resulting lack of transparency, model interpretability methods have become increasingly important. We address two methodologies to extract the important words in a piece of text:

- **Count-based scoring:** For each word, count its occurrences in positive tweets and subtract its occurrences in negative tweets. Keep only words with a high or low enough score as they correlate more with one class of tweets. Note that this method is only able to explain positive or negative words, but not neutral ones. It results in a 0.13 mean Jaccard similarity score.
- **Transformers Interpret and Captum:** libraries providing state-of-the-art algorithms, including Integrated Gradients, to perform model explainability. For each tweet, the model is able to recognize, through the use of a threshold, the words that best explain the sentiment for each class. The higher the score, the more important is that term. By analyzing the results for our tweets we can notice that the model is able to select the right terms when the sentence is short and the sentiment is clearly defined while it struggles when the tweet is not written in plain English but it contains some expression typical of a message. In Figure 3 is shown a negative example. Even if this approach uses the power of the self attention layers, it is not able to outperform the Count-based scoring. This happens because some words are divided in chunks to increase the accuracy of the model which leads, for instance, to the split of the word *"belgium"* in *"bel-g-ium"* so that we end up with high manually checked results but poor Jaccard score (only 32%)

True Label	Predicted Label	Attribution Label	Attribution Score	Word Importance
Negative	(0.94)	Negative	1.10	#s We can ` t even call you from bel g ium sucks #/s
Negative	(0.05)	Neutral	-1.07	#s We can ` t even call you from bel g ium sucks #/s
Negative	(0.01)	Positive	-2.10	#s We can ` t even call you from bel g ium sucks #/s

Figure 3: An example of transformers explainability. The greener the background, the more important the word is for the predicted label. The other way around with red. Namely, the word "sucks" heavily moves the prediction towards the negative class and away from the neutral class. Label attribution score represents the confidence of the model interpreted as: high positive or negative values as confident and values close to zero as not confident.

6 Conclusions

We showed an interesting comparison of various methods for sentiment analysis, confirming transformer models as the state-of-the-art. Our results show that, while the classical ML approaches are too simple for a complex task like the *Tweets Sentiment Analysis*, different types of embedding with ConvLSTM perform very well during the training phase but are not able to generalize over unseen tweets. On the other hand, transformers outperform all the other approaches but they need to be fine-tuned and require a good preprocessing to achieve an higher accuracy. This is due to the fact that our dataset is not written in plain English but contains lots of slang words and abbreviation typical of the internet world. To overcome the overfitting problem, present in all the tested models, a possible solution could be to increase the training dataset or add some regularizing techniques such as a dropout.

References

[1] Narine Kokhlikyan et al. *Captum: A unified and generic model interpretability library for PyTorch*. 2020. arXiv: 2009.07896 [cs.LG].

[2] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: http://www.aclweb.org/anthology/D14-1162.

[3] Hila Chefer, Shir Gur, and Lior Wolf. "Transformer Interpretability Beyond Attention Visualization". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 782–791.