AML Spring 2022

Challenge 3: Sentiment Analysis

September 19, 2022

Group 8: Harkati Chiraz Rayene, Senane Zineb, Zheng Estelle

Emails: {Chiraz.Harkati, Zineb.Senane, Estelle.Zheng}@eurecom.fr

Professor: Michiardi Pietro

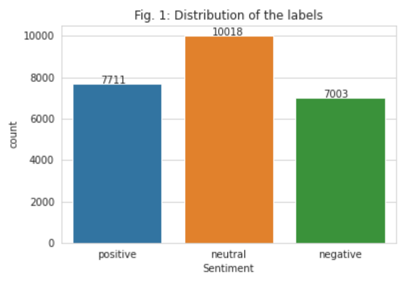
1 Introduction

This Natural Language Processing (NLP) challenge aims to classify tweets as positive, negative, or neutral, reflecting the sentiment of the user writing the tweet. We first preprocessed the tweets to clean them, then extracted features using two different approaches: sparse and dense representations. Finally, we assigned the tweets to one of the three classes using different models: Naive Bayes, LSTM, BERT and BERTweet classifiers. We compared the results and chose the best one in terms of the F1-score.

2 Data Analysis & Preprocessing

2.1 Data Analysis

The data we used is provided by the Figure Eight's Data for Everyone platform. We are given two datasets: a training dataset with 24732 samples and a test dataset with 2748 samples. Each sample has 3 features: 'textID', 'text' (corresponding to the tweet), and 'selected_text' (where some preprocessing has already been done to clean the tweets). We also have a label 'sentiment' for the training set, which determines the class ('positive', 'negative' or 'neutral') of the given tweet. There are no missing values in both datasets.



From Fig. 1, we can see that the distribution is well balanced between positive and negative labels, but slightly skewed between neutral and positive/negative labels. As the skewness is not huge, we kept the distribution like it is. After some analysis, there are texts containing user mentions, hashtags, and URLs, even in the selected_text column. Moreover, after removing stopwords for each class label and when displaying the word clouds (visual representation of popular words in texts based on frequency and relevance), intuitively, we saw that the obtained words make sense. Indeed, frequent words were: 'love', 'thank', 'good', 'great' for the positive class; 'now', 'today', 'time', 'going' for the neutral class; and 'hate', 'sad', 'bad' for the negative class. However, we noticed some words like 'm', 'u', and 're', which are related to chat abbreviations and that need to be converted.

2.2 Data Preprocessing

Some preprocessing has already been done on the tweets to clean them and the output of this cleaning is given in the column 'selected_text'. However, as seen in the data analysis part, this preprocessing is not enough. Thus, we applied additional preprocessing on selected_text.

First, we removed non-ASCII words and replaced '&' characters with 'and'. Then, as there are still some usernames' mentions and URLs, we suppressed them as they are not relevant for the given task. We also removed punctuations only for Naive Bayes and LSTM because they work at the word level. Then, we converted emojis into texts as models can only interpret texts. After that, we did some word replacements, which were essentially chatting words and abbreviations replacements. Except for BERT, we removed stopwords, which are commonly occurring words in the English language such as "the", "a", "an", and "in", because they do not add much meaning to sentences. However, as it was better to keep words like "not" or "no" to detect more negative tweets, we removed them from the stopwords' list of the NLTK.corpus library. We did not convert the tweets to lower case because tweets containing upper cases also have a role to play in the sentiment, and it can be taken into consideration with models like the BERT cased version. For instance, 'BAD' conveys more negative sentiment than 'bad'.

Finally, an important step is to reduce the inflectional forms of each word into a common base or root in order to be able to analyze the meaning behind a word. To do that, we compared lemmatization and stemming methods. Stemming uses the stem of the word, i.e. the part of a word responsible for its lexical meaning, and removes the suffixes (e.g. the stemmed version of "studies" is "study"). Instead, lemmatization uses the context in which the word is being used and returns its base form (e.g. lemmatized version: "study"). Stemming is simpler to implement and faster than lemmatization because it gives shorter vocabulary, but it only takes a word and produces a normalized form, without considering the context. Whereas lemmatization, although taking more time to run, can return either a noun, a verb, or an adjective depending on the context of the text. Thus, we chose lemmatization as it gives more qualitative performance compared to stemming, except for LSTM because we will apply Word2Vec which does not need this improvement as we will see later.

	selected_text	text_pp	text_nonstop	text_stemmed	text_lemmatized
11478	my skin sucks	im getting really spotty and my spots always s...	getting really spotty spots always scar no not...	get realli spotti spot alway scar no not pick ...	get really spotty spot always scar no not pick...
7070	Spending the first night in my new place!	spending the first night in my new place	spending first night new place	spend first night new place	spend first night new place
7183	cramps;	stomach cramps sat in bed with a hot water bot...	stomach cramps sat bed hot water bottle hot mi...	stomach cramp sat bed hot water bottl hot milk...	stomach cramp sit bed hot water bottle hot mil...
23806	_Writer aww no which one?	writer aww no which one	writer aww no one	writer aww no one	writer aww no one
24347	Alaska-so far so good	after 42 hours of not sleeping it time for be...	42 hours not sleeping time bed alaskaso far good	42 hour not sleep time bed alaskaso far good	42 hour not sleep time bed alaskaso far good

Table. 1: Table with the different preprocessing steps

2.3 Feature Extraction

To analyze preprocessed data, a major step in NLP is to find a numerical representation for the input data by extracting features. In this challenge, we have tried different techniques to extract the desirable features: the Bag-of-Words representation for Naive Bayes and the Word2Vec representation for LSTM. The BERT model has its own embedding layer that does the feature extraction internally.

Bag-Of-Words: Bag-Of-Words gives a **sparse** representation of text that describes the occurrence of words within a text. It involves a vocabulary of known words and a measure of the presence of known words. It is the simplest way to vectorize a record, but its main problem is that all the words are weighted uniformly which is not true in all scenarios as the importance of the word differs with respect to the context.

Word2Vec: The Word2Vec algorithm uses a neural network model to learn word associations from a large corpus of text and generate a **dense** representation of a text by also taking into consideration the similarities. Once trained, it can detect synonyms or suggest additional words for a partial sentence. The Word2Vec model is more advanced than Bag-Of-Words since it uses a combination of two techniques: Continuous Bag Of Words (CBOW), which predicts the current word from a window of surrounding context words (where the order of context words does not influence prediction) and the continuous skip-gram architecture. It uses the current word to predict the surrounding window of context words. In our setting, we passed the stemmed text to Word2Vec while working with the LSTM model in order to consider the root of words and to represent all the lexical variants in one vector only. This will help us to reduce the vocabulary size and gain some training time.

BERT: BERT stands for Bidirectional Encoder Representations from Transformers. It is a transformer-based pre-trained language model (PLM). As it is a transformer, it is a deep learning model that adopts the mechanism of self-attention, differentially weighting the significance of each part of the input data. It is very efficient as it is pre-trained bidirectionally (i.e. it learns information from both left and right sides of a token's context) on a large corpus with Masked Language Modeling and Next Sentence Prediction tasks. Thus, BERT can help learn contextual word representation (i.e. dynamically informed by the words around them) instead of a static one for each word like Bag-Of-Words or Word2Vec. It is a strong representation learning model in NLP, usually used in a fine-tuned manner for transfer learning. That is to say, we initialize BERT with pre-trained weights and further fine-tune it on our small dataset to adapt to the downstream task, i.e. here the sentiment classification task.

3 Model Selection

To perform the classification of tweets, we used different models: Naive Bayes classifier, LSTM, BERT and BERTweet. To compare them, we used the F1-score. Before applying the models, we mapped the 'sentiment' column to the 'target' column created by setting negative, neutral, and positive to -1, 0, 1 for Naive Bayes and to 0, 1, 2 for LSTM, BERT and BERTweet because these models cannot have negative values as labels.

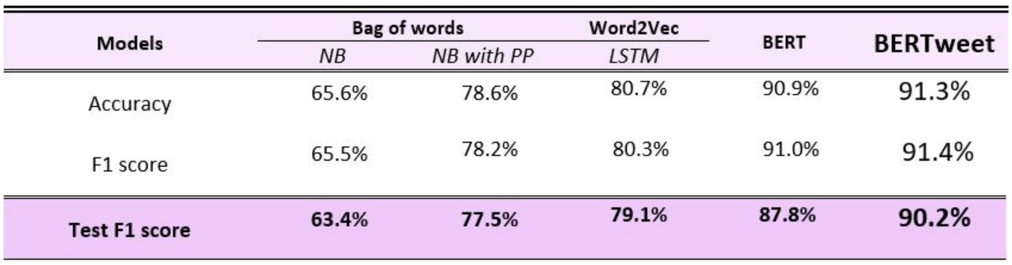
Naive Bayes classifier: A naive Bayes classifier is a probabilistic machine learning model based on the Bayes theorem and used for classification with discrete features (e.g., word counts for text classification). It is among the fastest and simplest models but its biggest disadvantage is that it requires features to be independent which is not guaranteed in most real-life cases. However, we chose to use this model for its simplicity and its efficient time complexity, which is useful to refer to as a baseline. Here, we applied a Bag-Of-Words algorithm to the lemmatized text and passed the result to the multinomial naive Bayes classifier.

LSTM: LSTM stands for Long Short Term Memory. It is a recurrent neural network that overcomes the problem of vanishing gradients. It has a memory cell at the top which helps to carry the information from a particular time instance to the next time instance in an efficient manner as it has feedback connections. We used this model in our case since it will enable us to treat sequences of data such as texts (sequence of words).
The architecture we implemented is composed of: an embedding layer with weights set from the Word2Vec model's learned word embeddings; three consecutive structures of an LSTM layer with 128 nodes and 13158 parameters followed by a dropout layer to avoid overfitting and finally a dense layer with the softmax activation function, 129 parameters, and 3 neurons since we want to predict the sentiment of the text and we have 3 different classes. Before passing the labels in the fitting part, we used one-hot encoding as we have 3 nodes in the output layer, one for each class. We used the categorical cross-entropy as the loss function, and the Adam optimizer to optimize the algorithm. We trained the model with 10 epochs and with batches of size 32.

BERT and BERTweet: BERT consists of a variable number of encoder layers and self-attention heads. Through the self-attention mechanism, each word learns to attend to the words in the sentence to better represent its meaning. The main advantage of BERT is that it decreases the sequential operations needed to connect two words in a sentence (from O(n) in LSTM to O(1) in transformer). Another advantage is the amount of computation that can be parallelized in the transformer by using masks. Thus, it is suitable for designing large language models and so for our task.
We trained a BERT classifier with the following hyperparameters: the Adam optimizer with a learning rate of 2e-5 and batch size of 32, according to best practices and GPU memory concerns. Since our text data are tweets, we thought that a PLM trained on tweet corpus may perform better. Thus, we chose to also use the BERTweet model which is trained based on the RoBERTa (improved PLM of BERT) pre-training procedure. The corpus used to pre-train BERTweet consists of 850 million English tweets. The hyperparameters used are similar to the ones of BERT, but with a learning rate of 1e-5. Both BERT and BERTweet have more than 110 million parameters, which take a long time to train. Thus, we used them directly without fine-tuning for a good trade-off between resource and performance.

4 Model Performance

To compare the models and choose the best model, we used the F1-score. We also used accuracy and plotted the confusion matrix for validation to better understand our models' performance. F1-score is balancing precision and recall (better metric when considering False Positive and False Negative respectively) on the positive class while accuracy looks at correctly classified observations both positive and negative.



Models	Bag of words		Word2Vec	BERT	BERTweet
	NB	*NB with PP*	*LSTM*		
Accuracy	65.6%	78.6%	80.7%	90.9%	91.3%
F1 score	65.5%	78.2%	80.3%	91.0%	91.4%
Test F1 score	**63.4%**	**77.5%**	**79.1%**	**87.8%**	**90.2%**

Table 2: Comparison between the different models' performance

For Naive Bayes classifier, we implemented two models based on the simple representation given by the Bag-Of-Words method: one without the preprocessing on raw tweets and the other with the preprocessing we defined above. As expected, the model with preprocessing worked better on both validation and test sets as shown in Table 2, highlighting the importance of cleaning tweets. For validation, the accuracy and F1-score were both around 66% for the one without preprocessing and around 78% for the one with. For the test F1-score, it was

63.4% and 77.5% respectively. Looking at the confusion matrix in Fig. 2, the classifier trained on lemmatized texts worked quite well as the true positives are more numerous (diagonal values) but it struggles to differentiate between neutral/positive or neutral/negative (quite important numbers for these cases).

For LSTM, we fit the model by the stemmed text of the training set directly as it has its own embedding layer that uses the Word2Vec weights. The accuracy and F1-score for the validation set were around 80.4% and the F1-score for the test set was 79.1% which is a slight improvement compared to Naive Bayes. Moreover, the confusion matrix in Fig. 2 shows that labels are predominantly on the diagonal meaning that the LSTM classifier classified well (high number of true positives) and it shows noticeably fewer false positives and negatives (around 10%) than Naive Bayes. Given these results, we noticed that LSTM performs quite well on this sentiment analysis problem as expected, even though it could have performed a lot better. This is due to the nature of the architecture used for our LSTM model, as it involves deep learning neural networks that work well for this kind of NLP tasks, and also because it includes dropout layers to prevent any overfitting scenario. Overall, the moderate results we had can be explained by the fact that there is not much to remember from the sequences to be able to classify. Furthermore, the scores obtained can be improved by hyperparameters tuning.

From our experiments, we can see that both BERT and BERTweet provided significantly better results. The accuracy and F1-score for validation were both around 91% (91%) and the test F1-score was 90.2% (87.8%) for BERTweet (for BERT). The results confirmed that BERTweet gave a better result on this task than BERT as BERTweet has been trained specifically on tweets. However, due to its large size and our relatively small dataset, we find during training that both transformers overfit easily. Thus, we introduce the early stopping procedure to avoid this problem. When the performance on the validation set no longer improves in a few epochs, we interrupt the training process. During the training, we saw that the validation loss was decreasing overall which is a good sign to reduce overfitting. Overall, we can clearly see that BERT (and so BERTweet) outperformed the other methods because it learns bidirectionally a contextual word representation, unlike other methods we tried. Moreover, looking at the confusion matrix in Fig. 2, we can see that the labels are predominantly on the diagonal meaning that the BERTweet classifier (and also the BERT one) worked well on classifying almost all of the true positives with negligible false positives and negatives, much less than Naive Bayes classifier and LSTM.

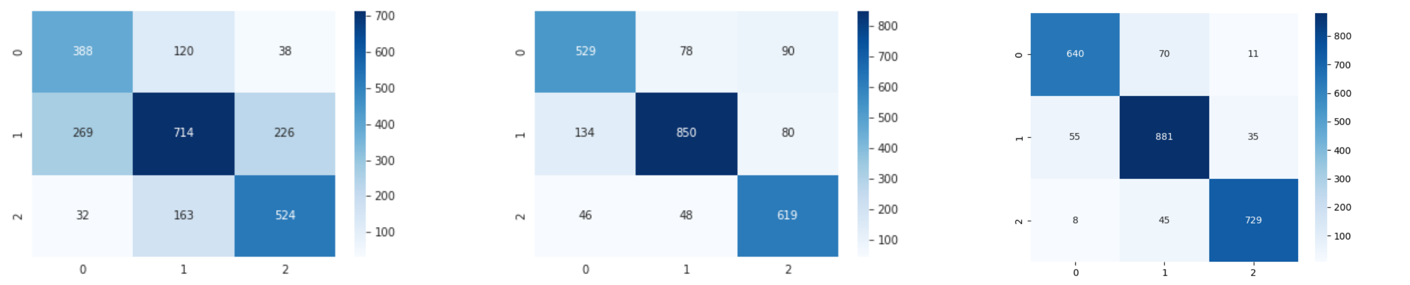


Fig. 2: Validation confusion matrix for Naive Bayes (left), LSTM (middle) and BERTweet (right)

5 Conclusion & Further Improvements

In this challenge, we implemented basic NLP techniques starting with tweet preprocessing. We compared lemmatization and stemming techniques and chose the first one as it gives more qualitative results except for LSTM for which we chose stemmed texts. Then we did two methods of feature extraction from preprocessed tweets that gave a static word representation: a sparse distribution using Bag-Of-Words for Naive Bayes classifier and a dense distribution using Word2Vec for LSTM. We also implemented BERT and BERTweet which have their own efficient embedding as they learn bidirectionally. Thanks to the self-attention mechanism, the BERTweet model achieved the best performance with a test F1-score of 90.2%. As a future work, we can try to determine which words are the reason for each classifier to determine the classification (negative, neutral, or positive) by using the Jaccard coefficient to evaluate the overlap between the selected words and the ground truth.