AML - Sentiment Analysis - Challenge 3
Group 4

Philipp Peron
philipp.peron@eurecom.fr

Khristen Thornburg
khristen.thornburg@eurecom.fr

Hugo Rechatin
hugo.rechatin@eurecom.fr

Ioannis Panagiotis Pitsiorlas
ioannis.pitsiorlas@eurecom.fr

1. Introduction

Dealing with language produced by humans in an informal setting, such as a social media platform, can be challenging due to spelling errors, slang and punctuation. The goal of this challenge is to perform sentiment analysis on a set of tweets, classifying them as posittive, neutral or negative. We started by doing some feature engineering to simplify the text, then we used two different models, an LSTM model and a BERT model, to perform the classification. After fine tuning the hyperparameters, the BERT model acheived 79.4% accuracy on the test data using the $text$ field and 89.7% accuracy using the $selected\ text$ field.

2. Data Analysis

To begin, we went through the training set to ensure that there was no over representation in the classes. Figure 1 shows that although neutral sentiment tweets have a higher frequency, the imbalance is not significant (we do however notice the effects of this imbalance in our confusion matrices later on).

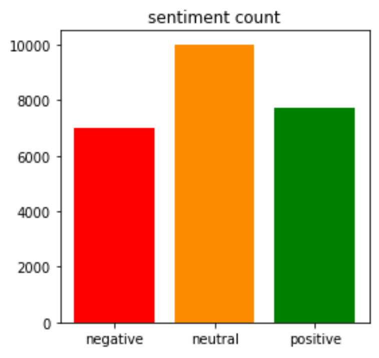

Figure 1. Count of stentiments in training data

To further familiarize ourselves with the data, we traced the distributions of the word count per sample, shown in Figure 2. This shows that negative and positive tweets need very few words to convey their sentiment, where neutral sentiments have a much higher average word count in the $selected\ text$.

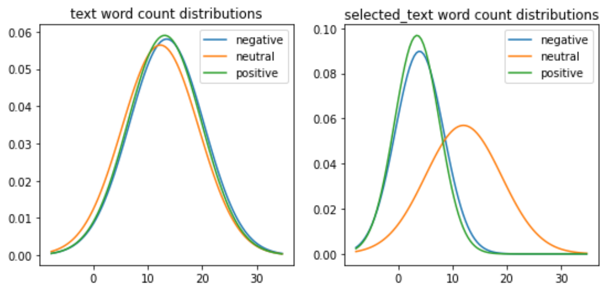

Figure 2. Word count distributions

Seeing that the average $text$ and $selected\ text$ word counts are roughly the same for neutral sentiment samples, we can suppose that :

VOC_{neu}[1] $\in$ VOC$_{neg}$[2] $\cup$ VOC_{pos}[3]

VOC_{neg} and VOC_{pos} probably have specific recurring words that can be used to classify them quickly, making their $selected\ text$ word count much smaller. We hypothesize that positive and negative sentiments are detected by their word use, whereas neutral ones are detected by their lack of typically negative or positive vocabulary.

Appendix A explores the previous hypothesis by looking at the total word count for each sentiment.

3. Feature Engineering

Before passing the data into our different models for training, we first had to process it. Our data processing steps for the LSTM-model and the BERT-model were different from each other and are described in the following subsections.

[1] neutral tweet vocabulary
[2] negative tweet vocabulary
[3] postive tweet vocabulary

3.1. Preprocessing for LSTM

The processing of the tweets for the LSTM model mainly consisted of text simplification and converting the words into numeric representation.

To simplify the tweets, we started by splitting the them into smaller parts using a tokenizer from the *Spacy* Python library. We removed the punctuation, the stopwords like 'afterwards', 'a', and 'the', and the URLs in the tweets. We took the lemma of the remaining words. Lemmatization is the process of replacing conjugated words with their root word, e.g. 'saw' into 'see' or 'horses' into 'horse'. This simplifies the analysis process since the model does not have to learn the different forms of a word and can focus on its core meaning instead.

In the last step of pre-processing, we converted the word tokens into a numeric form. For that, we first built a dictionary from all the words in the training data and then assigned each word an integer representing that word. For padding the sequence, we add zeroes to the end to make all the text feilds the same length.

3.2. Preprocessing for BERT

To prepare the data for the BERT model, we added special tokens to the beginning and end of each text field and then tokenized the samples using either the cased or uncased BERT tokenizer. This automates the process of manually making a dictionary and converting the words to integers. We added zeroes to the sentences as padding and created attention masks to focus on only the words, not the padding. The BERT tokenizer automatically performs the lemmatizing and the removal of stopwords. BERT works on an attention principle, so it knows to only pay attention to the words that actually affect the sentiment outcome, meaning it knows to ignore stopwords.

4. Model Selection

4.1. LSTM

We started the model selection process for this task by trying an LSTM model. Compared to a traditional recurrent neural network (RNN), an LSTM is better at retaining information over time.

To implement the model we used PyTorch which has a functionality called *packed_sequences* which we can signal to ignore padding, without it having to learn it. This has the benefit of strongly increased classification performance, compared to standard padding, and also increases training speed, since the network gets fewer inputs for short sequences compared to long sequences.

In order to track our experiments, we used the platform *Weights and Biases*[4]. WandB helps with both the HP configuration and the code of an experiment so that it can be easily reproduced later on.

[4]Online, available at https://wandb.ai/

The first layer in our model is an embedding layer. The embedding layer maps the numeric token to a vector representation. The idea is that similar words correspond to similar embedding vectors. There exist many pretrained embedding layers. We decided to use the *GloVe* embeddings. The authors of the paper published different pretrained embeddings trained on different amounts of words from 3 sources. Table 1 lists the different embeddings that we tried with their sources, the number of tokens, the vocabulary size, and the dimensions of the pretrained embeddings.

When loading the embeddings, we assigned random vectors to the tokens which were not included, most of which were slang words or misspellings. These random vectors were sampled from a normal distribution similar to the distribution of all the other embedded vectors. We then tested the base model with the different embeddings. The pretrained embedding with the largest dimension and the smallest vocabulary performed best, which was the one trained on Wikipedia and Gigaword with a dimension of 300.

Sources	Tokens	Vocabulary	Embedded dim.
Wikipedia 2014 + Gigaword 5	6B	400k	50, 100, 200, 300
Common Crawl	840B	2.2M	300
Twitter	27B	1.2M	25, 50, 100, 200

Table 1. Pretrained GloVe-word-embeddings from three sources. From each source there are embeddings with varying dimensions.

We also tried *GRU* as an alternative to the LSTM, which resulted in faster convergence and fewer model parameters, but a slightly worse accuracy overall. Therefore, we continued with the LSTM-units.

We changed the basic LSTM model to allow for bidirectional recurrent units and multiple layers and tested alternative optimizers to Adam. SGD with learning-rate-decay showed very noisy accuracies until the first learning rate decay step, and also converged to a lower final accuracy compared to Adam. We tried using AdamW instead of Adam, but it slightly decreased performance, so we kept Adam as the optimizer in the end.

We considered adding a self-attention mechanism but decided against it since there is too little data for the training and it would likely result in no improvement.

Table 2 shows the tuning ranges and sampling methods for each optimized HP. Since we were training the model from scratch, we had to determine a lot of the model's HPs. That is why we decided to use automated HP tuning. We used the tuning framework *Ray Tune* to conduct the HP search. During tuning, a task scheduler stopped trials early when their preliminary results were not promising,

Hyperparameter	Min	Max	Sampling	Best
Dropout	0.3	0.7	uniform	≈ 0.5
Recurrent dim.			[64, 128, 256, 512]	128
LSTM layers	1	2	discrete	2
Bi-dir. LSTM			[True, False]	True
Batch size			[16, 32, 128]	32
LR	10^{-5}	10^{-8}	log. uniform	$\approx 10^{-4}$

Table 2. HP ranges explored during tuning, the sampling methods, and the values for the best model

and therefore increased tuning speed significantly. The HP tuning was performed using a training and a validation set and ran for 6 hours. In the end, the top HPs, were used to retrain the model with the entire dataset. The final HPs together with their search ranges are listed in Table 2.

Table 3 shows the architecture of the LSTM model after the HP search. It has a rather slim architecture with a recurrent dimension of only 128. A higher capacity is achieved through depth, by having two layers and being bidirectional.

Embedding layer (dimension=300)
Dropout layer (probability=0.5)
LSTM layer (dimension=128, bi-directional)
LSTM layer (dimension=128, bi-directional)
Dropout layer (probability=0.5)
Dense layer (input dimension=512(=4*128), output features=3)

Table 3. LSTM model architecture

After retraining the model with the found HPs on the entire dataset, it achieved 73.0% accuracy on the preliminary test set on the Kaggle platform. This is a very good result considering no transformer was used.

4.2. Pretrained BERT

A classic way to perform sentiment analysis is to use a pretrained transformer model that converts the text into vector representations used in a deep learning model. For this task, we chose to use a pretrained BERT model.

Initially, we trained the model using the $text$ and the $selected\ text$ and found that the $selected\ text$ gave a much higher performance. Most of our testing was then done using $selected\ text$. It was only later that we realized that this field was supposed to be used for the bonus task and not for the actual sentiment analysis model.

Once the model was working well, we tried changing the hyperparameters to see if we could improve the accuracy. The hyperparameters that we changed were : split of training/validation data, number of training epochs, learning rate, batch size, using text vs $selected\ text$, and cased vs uncased BERT. A summary of the hyperparmeters tested and the resulting accuracies is shown in Table 4.

Epochs	Learning Rate	Batch Size	Comments	Accuracy
1	2E-05	32	on text	74.00%
1	2E-05	32	on select_text	88.81%
1	2E-04	32		19.39%
1	2E-05	32		89.42%
1	2E-06	32		87.19%
2	2E-06	32		88.06%
2	2E-05	32		89.74%
3	2E-05	32		88.44%
2	2E-05	16		88.86%
2	2E-05	64		88.43%
2	5E-05	32		87.80%
2	2E-05	32	cased	88.49%
2	2E-05	32	on text	79.37%

Table 4. Hyperparameter tuning for BERT model

Below is a description of the different parameters tested and the conclusions :

- Number of epochs (1 to 4) : Figure 3 shows how the model performed for 1 to 4 epochs of training. This shows that running the model for multiple epochs makes the training loss decrease but increase the accuracy. This is a classic sign of overfitting. The maximum accuracy was after either one or two epochs.

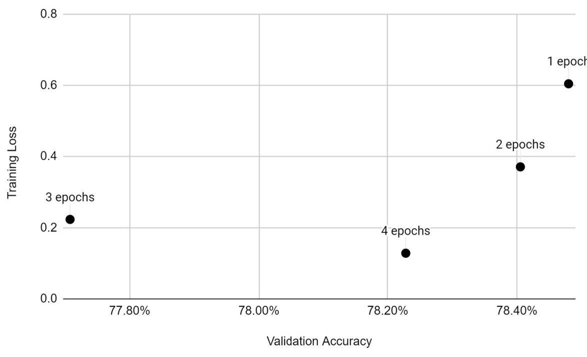

Figure 3. Accuracy vs. Number of epochs

- Learning rate (between 2e-4 and 2e-6) : When the learning rate was increase to 2e-4, the predicted output was all zeroes for the test data (all text was classified as neutral), indicating that this learning rate was too large and the model did not converge. The best accuracy was acheived with a learning rate of 2e-5.

- batch size (16, 32, 64) : A batch size of 32 gave the best performance between the three batch sizes tested, but ultimately this did not have a significant impact on accuracy.

- Training/validation split : in order to test the model, we split the data into training/validation 90/10. However, when submitting the model, we used all the training data to train. This increased the accuracy by 0.6%.
- Cased vs uncased BERT : The uncased BERT model gave about 1.3% better accuracy than the cased. We were a bit surprised by this and assume it is because in tweets, correct capitalization is not used, so where the words are capitalized does not provide much useful information to the model.

4.3. Classification Errors

Figure 4 shows the confusion matrix for the BERT model with a 60/40 training/validation split. This matrix shows that the tweets that are most likely to be misclassified are positive or negative sentiment classified as neutral or neutral tweets as positive or negative. Predictably, there is very little overlap between the positive and negative tweet classification. When looking at these numbers, we have to remember that there are more neutral sentiment tweets overall so it is normal that the neutral tweets have more errors. But the skew is still significantly greater than the ratio of neutral to positive/negative misclassified tweets.

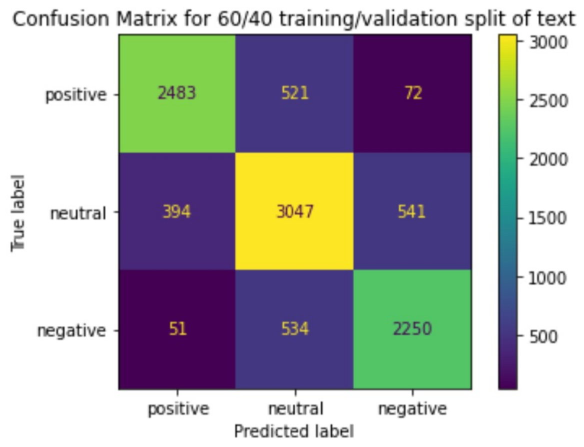

Figure 4. BERT Confusion Matrix

In order to understand why the tweets were misclassified, we tried to see if there were any patterns in the sentences that were misclassified. Unfortunately, our brains were not able to draw any better conclusions than the BERT model. For example, the most frequently misclassified sentences were ones that had a neutral label and a negative prediction. Some of the examples of *selected text* from this category are ”he won’t reply”, ”is stuck in traffic” and ”fiona is sad because there wasn’t enough room in megan’s suitcase for her”. All of these sentences sound pretty negative to us, but the ground truth classifies them as neutral.

In order to account for the issue of having more neutral tweets than either positive or negative, we tried implementing an oversampling technique called weighted random sampler. This attempts to solved the problem of having imbalanced classes and not have more neutrals than the negatives and positives. Unfortunately even when accounting for the imbalanced classes, the accuracy did not improve.

5. Bonus Task

For detecting the relevant words inside the tweet which are most responsible for the attributed sentiment, we used the Captum library, which is specifically designed for interpreting models like BERT. This kind of interpretation helps to better understand embeddings, BERT and attention layers. We constructed helper functions to create references and baselines for each word token, token type and position IDs. After that, we defined pairs of questions and categories as the input to our BERT model in order to understand what the model is focusing on when it classified our tweets. Although widely used in practice, we were not able to achieve satisfactory results.

6. Conclusion and Further Work

Our model with the best results was an uncased pretrained BERT that gave an accuracy of 79.4% on the *text* of the preliminary Kaggle test set. It clearly outperformed the traditional LSTM method, which only yielded 73.0% accuracy.

Further work could include trying other models in the BERT family, like RoBERTa or DistilBERT. It would also probably be beneficial to do more preprocessing on the sentences before they are fed to the model, possibly autocorrecting the misspelled words or including the punctuation in the analysis. For example, if there are a lot of exclamation points, the sentiment is more likely to be either positive or negative, not neutral.

A. Word Sum Graph

Figure 5 shows the word count across all samples ($word\ sum$) for each sentiment. Each column graphs the $word\ sum$ array of each sentiment in function of the sorting index of one of the three sentiments.

For instance, the first column graphs the $word\ sum$ arrays according to the sorting index of the negative sentiment $word\ sum$. For that reason, the negative $word\ sum$ graph of the first column (in red) is strictly increasing.

It is important first to note that there are stop words that all 3 sentiments share, hence the common spikes for the highest $word\ sum$ in every graph. These most common words dwarfed the other, more relevant words (words that characterize each vocabulary). To make the graphs more readable, we limit the range of the the highest $word\ sums$. From there we could pull conclusions on the discrepancies between the vocabulary of the three sentiments :

- In the first and last columns (ordered with regard to negative sentiment $word\ sum$ and positive respectfully), we can note that the neutral graph (blue) does not differentiate its vocabulary as much as the positive and negative sentiments do. For instance, in the first column, the positive $word\ sum$ graph has many spikes where the negative $word\ sum$ are close to 0, showing the differences between VOC_{neg} and VOC_{pos}. However, the neutral $word\ sum$ graph shows far less of these spikes. We observe the same tendencies in the graphs of the last column. This complements well the hypothesis made that VOC_{neu}∈VOC$_{neg}$ $\cup$ VOC_{pos}.

- The second column shows how VOC_{neg} and VOC_{pos} have a vocabulary that distinguishes itself much less from the neutral tweets than from the opposite sentiment.

- This graph matrix shows us, as we supposed earlier, that the neutral vocabulary uses words that are commonly used by both positive and negative samples, but it does not use words that make its own vocabulary stand out on its own.

- Comparing these graphs to our confusion matrix4, we can observe that where a $word\ sum$ plot shows important differences to the ordered plot, the confusion matrix shows less false classifications. For example, the positive $word\ sum$ graph of the first column differentiate its self from the negative one much more than the neutral graph, and in our bert confusion matrix4, the negative tweets classified as positive aren't numerous compared to those classified as neutral.

- The same can be said for the 2 other columns, there is a lot of confusion around the neutral tweet predictions, and the positive and negative graphs of the middle column show how VOC_{neg} and VOC_{pos} aren't very distinct from the VOC_{neu}, and this shows in the number of positive and negative tweets wrongly classified as neutral.

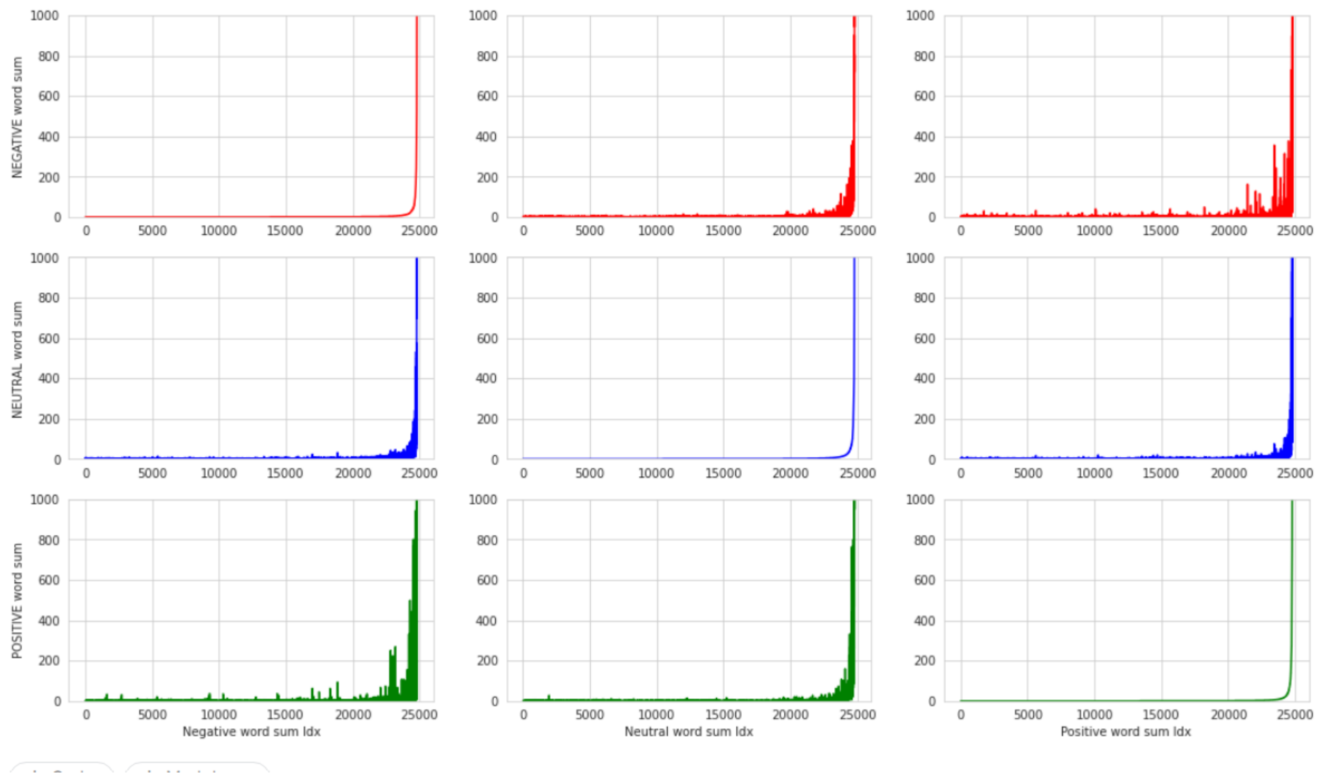

Figure 5. Sentiment Word Sum