Toxic Comment Classification

NLP Project - Master IASD

Xavier GEFFRIER, Léonard de LA SEIGLIÈRE

April 2022

The code of this project can be found HERE : `https://github.com/leonarddls/NLP_projet`

1 Problem Presentation

1.1 Objective

Our objective in this project is to train a model to detect and label "toxic" text comments. This is semantic analysis. For each comment, written by a human, we want to apply from 0 to 6 labels, the labels being *toxic*, *severe_toxic*, *obscene*, *threat*, *insult*, *identity_hate*.

1.2 Data presentation

Data is composed of English text comments posted on Wikipedia, in the "Discussion" section of Wikipedia pages. It is a very interesting dataset. First it is huge (around 160K rows), moreover it is free and without any copyright, and most importantly it has been labelled by humans, allowing simple supervised learning.

Most comments are shorts (less than 100 words) as we can see on Figure 1. The majority has no toxic label (Figure 2), and among the classified ones, some labels are much more frequent (Figure 3). Note that one comment can be labeled for multiple toxic behaviors, for instance be classified simultaneously as *insult* and *identity_hate*.

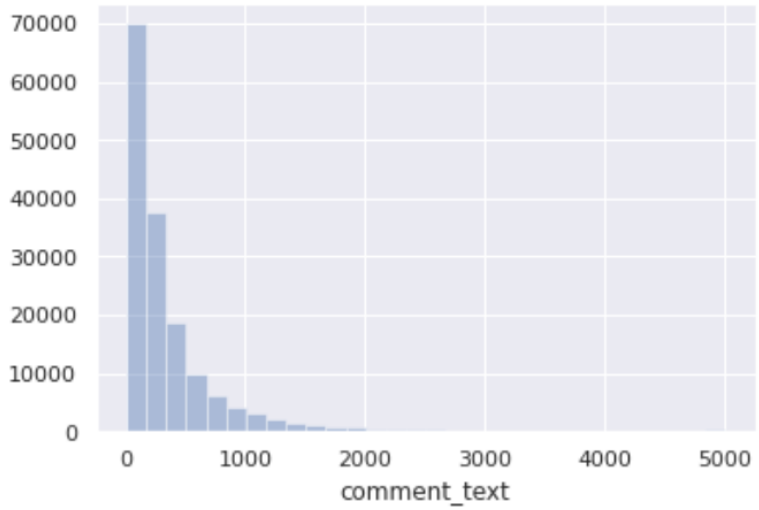

Figure 1: Histogram showing repartition of comments by size.

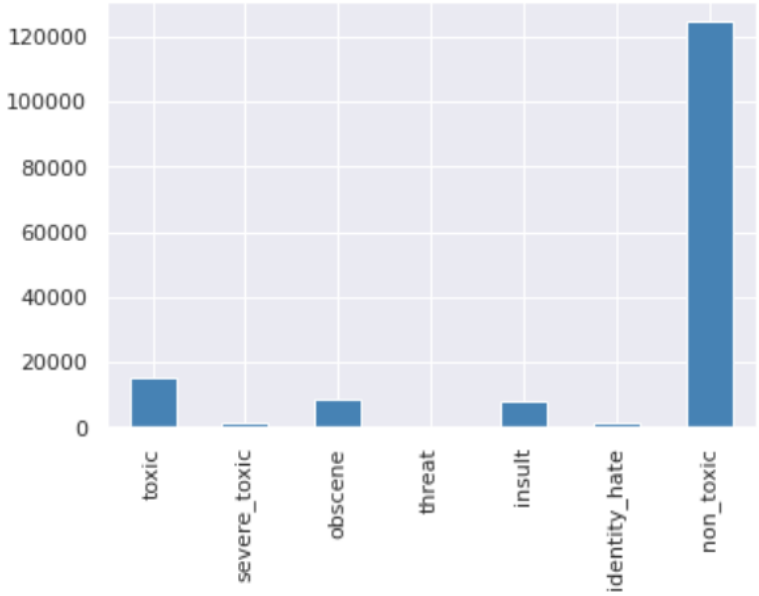

Figure 2: Frequency of each label among comments including non-toxic comments.

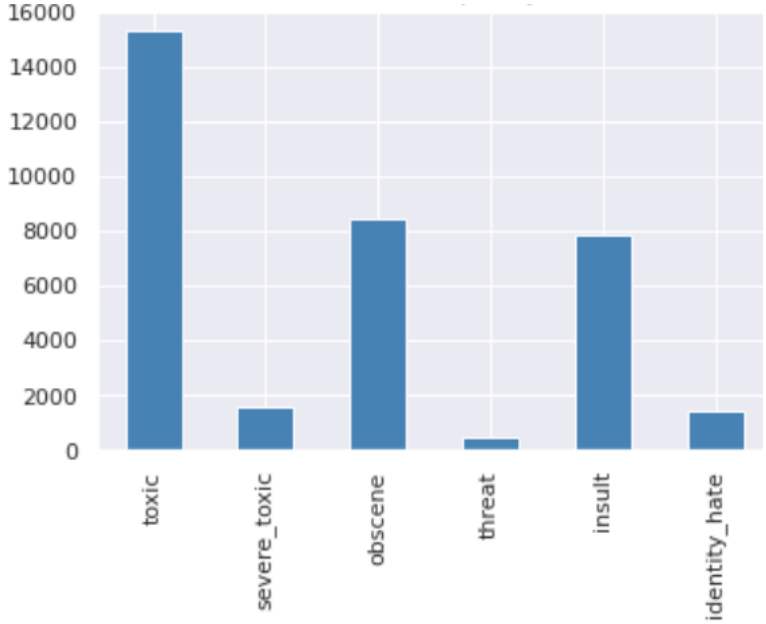

Figure 3: Frequency of each label excluding non-toxic comments.

1.3 Plan and evaluation

We will try three different methods and compare them by using the AUC-ROC (Area Under the Curve - Receiver Operating Characteristics), which seemed a better metric than F1-score that we first planned to use.

F1-score is useful to evaluate classification models with 2 classes or more, and is defined as the harmonic mean of precision and recall. It is especially used when data is unbalanced like in disease or fraud detection, and in our case, that is why we first thought it could be a good metric.

AUC-ROC is also designed for classification problem, and also use the false positive rate (FPR) and

true positive rate (TPR). ROC is a probability curve, plotted with TPR against the FPR (see Figure 4). The area under the curve, AUC, gives us an idea of how model is good to separate data. The best possible value is 1: no false positive. AUC-ROC is good too when dealing with unbalanced classes.

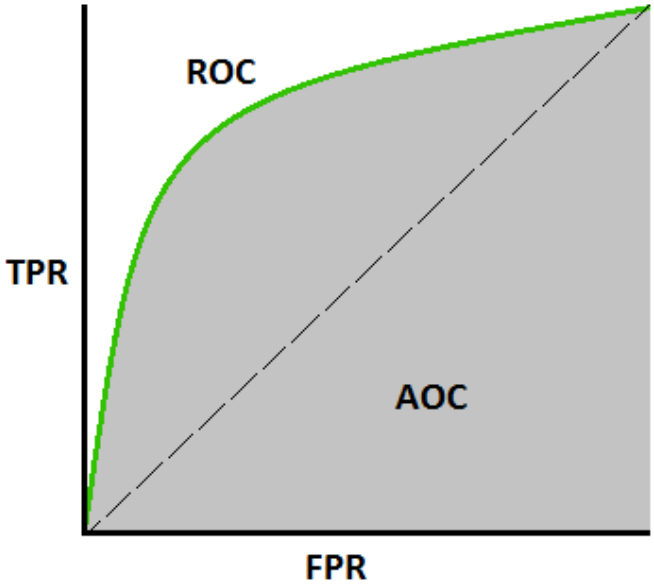

Figure 4: AOC-ROC curve [1]

The main difference between the two is that the F1 score takes predicted classes as an input, and AUC-ROC predicted scores (or probability). So when we want to evaluate a model like a neural network, which can output probabilities, AUC is a better option, since F1-score would require that we set a threshold to actually classify data.

Our first method will be a T**F-IDF associated with simple classifiers like a Decision Tree and a Linear SVC**, which is a basic method that will give us some standard score to evaluate the others. Then we will train and use a **LSTM** neural network. Then, a **BERT** model.

2 The Three methods

2.1 TF-IDF and simple classifiers : Decision Tree and Linear SVC

2.1.1 About TF-IDF

TF-IDF stands for Text Frequency - Inverse Document Frequency. It is a very basic way to have an idea of the "importance" of each word in a text, by just computing its frequency, that is to say the number of occurrences divided by the text length, and compare it to the frequency in the whole document (i.e. in other texts).

2.1.2 Our work

We use a Decision Tree and a Linear SVC Classifier on the output of the TF-IDF to classify a comment and already have small classifiers.

The idea of those network is to have an idea of what a basic network would be capable of in order to better appreciate the good result we expect from our previous models.

2.2 LSTM

2.2.1 About LSTM

Long-short term memory (LSTM) is a very well-known architecture of recurrent neural networks (RNN), proposed by Hochreiter et al. (1997) [2]. Their intention was trying to analyse sequence of data, while dealing with the vanishing gradient problem, that affected previously existing RNNs. The long-term gradients which were propagated could tend to zero and vanish, or to infinity and explode, due to finite-precision numbers of machines. LSTM networks can still have this exploding gradient problem, but vanishing is partially solved thaks to a distinction between the hidden state (similar to the output of a simple RNN cell) and the cell state, useful to keep track of long-term memory.

Since we learned in class how an LSTM work, we will not give more details and directly say how we used it.

2.2.2 Our work

We used a Embedder that takes 250 words long tokenized sentences and embeds each word in 128 dimensions. Each embedded sentence is then passed to an LSTM layercells with 60 hidden units. The output of the LSTM is in turn fed into 2 dense layers of 50 and 6 neurons. The last layers of 6 neurons uses sigmoid function and correspond to the probability of a sentence to be of a given category. (see Figure 5)

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 250)]             0

 embedding (Embedding)       (None, 250, 128)          2560000

 lstm_layer (LSTM)           (None, 250, 60)           45360

 global_max_pooling1d (Globa  (None, 60)               0
 lMaxPooling1D)

 dropout (Dropout)           (None, 60)                0

 dense (Dense)               (None, 50)                3050

 dropout_1 (Dropout)         (None, 50)                0

 dense_1 (Dense)             (None, 6)                 306

=================================================================
Total params: 2,608,716
Trainable params: 2,608,716
Non-trainable params: 0
```

Figure 5: Keras summary of our LSTM based network

2.3 BERT

2.3.1 About Transformers and BERT

The Transformer is a kind of neural network architecture that has been created by Vaswani et al. (2017)[3] initially to perform translation. Until that, recurrent neural networks like LSTM were used but they come with a few drawbacks:

- First, they are slow to train. Since each word has to be passed sequentially, it is not possible to parallelize the learning and to accelerate it by just adding more GPUs.
- Then, they can only treat limited sequences at once, and can "forget" the beginning of a long paragraph when parsing the end of it.
- For each word, they do not take surrounding context into account, but only sequence up to this word. Bidirectionnal LSTM attempt to fix that by reading the sequences backward and forward (see Figure 6) but they only manage to concatenate both informations, and fail to treat them as one more compelte information.

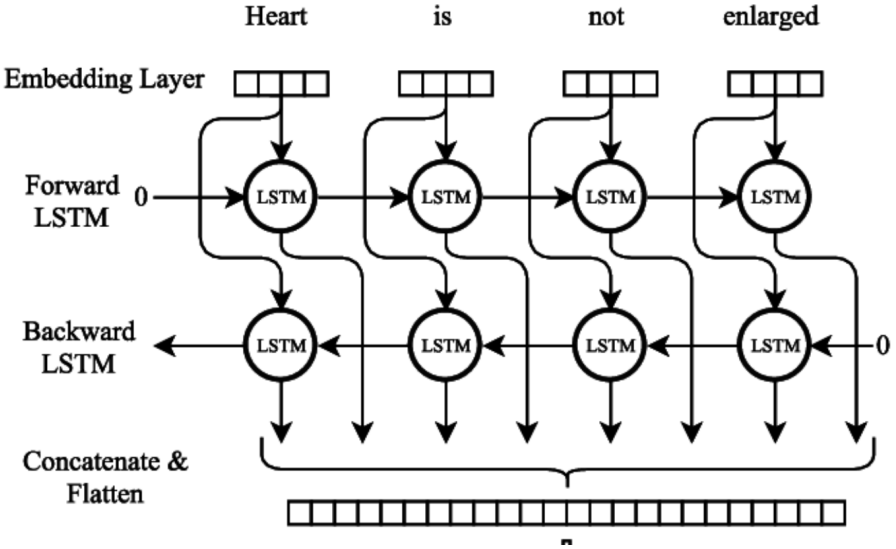

Figure 6: Schema of a bidirectional LSTM: both directions are taken into account, but in a parallel way with concatenation at the end, and not as one global information.

Transformer are designed to fix these flaws with three main changes:

1. **Positional encoding:** each word is enriched with its position (index) as an input, so that model can have information about the order of the sequence.
2. **Attention:** the words that we must pay attention to in the source language when translating a word on the destination language, an idea proposed by Bahdanau et al. (2014)[4] that uses learning over examples.
3. **Self-attention:** the internal representation of a language, by using context of words.

The Transformer consists in an encoder and a decoder (see Figure 7). The encoder create embeddings from input words simultaneously, and the decoder use them and the current output translated words to continue translation, one word at a time.

This separation in tasks is useful. The encoder learns what the source language is, what the context is. The decoder learns how words of one language relate to the other ones. By interesting ourselves

only on the decoder and stacking some of them, we obtain a GPT. In the same way, if we stack the encoders, we get BERT.

BERT stands for a Bidirectional Encoder Representation from Transformers. BERT can be used for translation, as well as text summarization or, what is important to us, sentiment analysis.

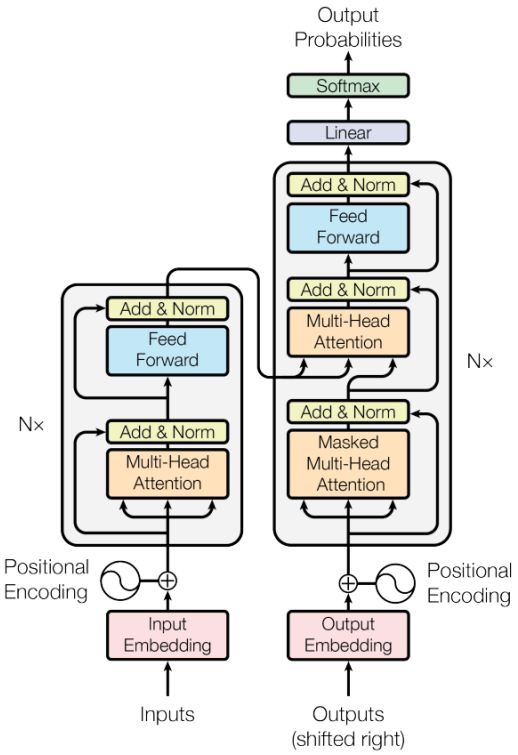

Figure 7: Schema of a Transformer from the original paper of Vaswani et al.[3]

2.3.2 Our work

Our network take 250 length of padded or shortened tokenized sentenced ans feeds then to a transformer layer. We used a pretrained small size BERT model called *DistilBERT* which has about 66 million parameters (vs 110 milion in full-sized BERT). The outputs of DistilBERT are fed into 3 dense layers of 256, 64 and 6 neurons each. The last layers of 6 neurons uses sigmoid function and corresponds to the probability of a sentence to be of a given category. (see Figure 8)

The transformer layer is set to non trainable, only the dense layers are trained. However, because of the size of DistilBERT, the training takes a lot of time, mainly because the forward pass is very computationally intensive.

3 Results

Method	AUC (validation data)
Decision tree	0.81
LSTM	0.981
BERT	0.983

Table 1: Results obtained on validation data for each method.

Layer (type)	Output Shape	Param #	Connected to
input_ids (InputLayer)	[(None, 250)]	0	[]
attention_mask (InputLayer)	[(None, 250)]	0	[]
tf_distil_bert_model (TFDistilBertModel)	TFBaseModelOutput(last_hidden_state=(None, 250, 768), hidden_states=None, attentions=None)	66362880	['input_ids[0][0]', 'attention_mask[0][0]']
global_max_pooling1d (GlobalMaxPooling1D)	(None, 768)	0	['tf_distil_bert_model[0][0]']
batch_normalization (BatchNormalization)	(None, 768)	3072	['global_max_pooling1d[0][0]']
dense (Dense)	(None, 256)	196864	['batch_normalization[0][0]']
dropout_19 (Dropout)	(None, 256)	0	['dense[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 256)	1024	['dropout_19[0][0]']
dense_1 (Dense)	(None, 128)	32896	['batch_normalization_1[0][0]']
dense_2 (Dense)	(None, 6)	774	['dense_1[0][0]']

Total params: 66,597,510
Trainable params: 232,582
Non-trainable params: 66,364,928

Figure 8: Keras summary of our transformers based network

These results are not exactly what we expected. We thought that TF-IDF + DT would be the less precise classifier, and it is. However, LSTM and BERT are almost indistinguishable, while we figured out that BERT would outperform LSTM. In fact, the difference in their score is so tiny that in some of our trys, it was LSTM that was first: we can not conclude that one of them is better for this particular problem.

How can we explain this? First, it is quite hard to train layers added to a pre-trained BERT model. The forward pass takes a lot of time, simply because the number of parameteres is huge. So we may have not trained enough our model, due to this constraint. But this is unlikely: the AUC was not growing anymore.

A more likely hypothesis is that both models perform well and that we can not really expect any model to perform better. The drawbacks of LSTM that we mentioned in section 2.3.1 are not problematic in this dataset. Most comments are short enough for LSTM memory. A simple sequential view of the words is enough to determine their toxicity, even if it is not a real context observation like in BERT. If the hypothesis is right, then LSTM may be preferred, because it required less computation power x time to train it.

References

[1] Sarang Narkhede. Understanding auc - roc curve, Jun 2018.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.