Multilingual Comment Toxicity Detection

Final Project on Spoken and Written Language Processing course

Margarita Geleta
margarita.geleta@est.fib.upc.edu

Universitat Politècnica de Catalunya, June 2020

Abstract

A single tiny toxic comment is capable of bursting an online discussion. A succession of toxic comments can lead to cyberbullying. Identifying such offensive contributions and hate speeches would allow us to create safer online communities and social media spaces, a problem which has actually posed a real challenge for tech companies. For this reason, this research aims to design, train and evaluate a natural language processing system capable of classifying toxicity comments on the web. The study has been extended into building a *multilingual* deep learning model with English-only training data, that is, a system capable of identifying the toxicity in several languages. This paper also reports the creation of a tensorflow docker container, the configuration to use personal *Graphic Processing Units* (GPUs) and steps to speed up using *Tensor Processing Units* (TPUs).

Keywords: Toxicity detection · Text classification · Natural Language Processing · Deep Learning · Multilingual models · Offensive language detection · Sentiment Analysis · Docker

Disclaimer: the paper contains text that may be considered profane, vulgar, or offensive.

1. Introduction

Toxicity is defined as anything rude, disrespectful, aggressive - basically, language that hurts. It is the opposite of healthy speech, and the definition was well metaphorized in *Toxic Language* (by A. E. Lavish, 2013): "*Toxic language* [...] *like all toxins, causes an adverse reaction in those who hear, read or see it*".

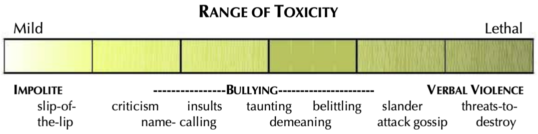
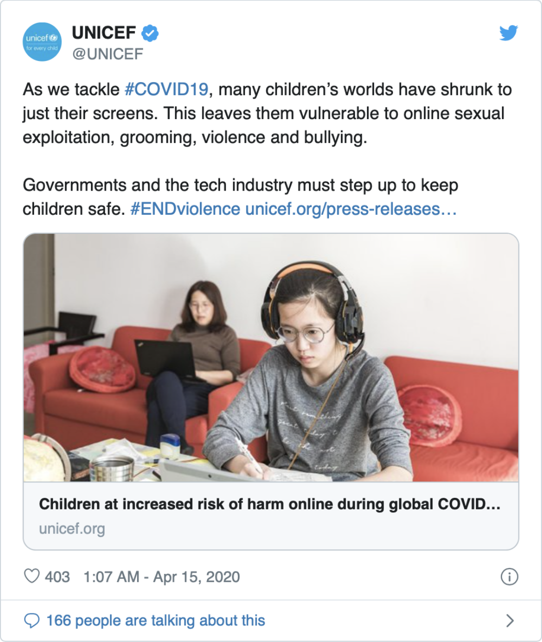


Figure 1: *(courtesy of A. E. Lavish, from Toxic Language, 2013) there are different degrees of toxicity, each of them harms those who hear, read or see this kind of comments.*

And precisely, these potential adverse reactions, ranging from irritation to terror, are what pose a danger in online communities and social media platforms. *Cyberbullying* is a clear example - threatening and abusive messages exchange or spam, which is accomplished through chat rooms, texting, online forums and e-mails. In particular, cyberbullying has been linked to depression and suicide risk among teenagers (S. Bauman et al., 2013). The digital risks of cyberbullying have increased even more during the COVID-19, with schools closing and a tremendous increase in unregulated screen time and online access; according to the *Child Online Safety Index* (COSI)[1], around 60% of 8- to 12-year-olds have been exposed to cyber risk. An interesting quote by S. Hinduja, PhD: "*when smartphones and social media became*

[1] https://www.dqinstitute.org/dqeverychild/

ubiquitous for students, cyberbullying went up", which is quite logical since in the digital world the number of potential targets and aggressors is limitless. He added: "*during this unprecedented time when [children] are all stuck at home,* [...] *being forced to use online platforms for learning*" at the risk of being victims of online toxicity.



UNICEF
@UNICEF

As we tackle #COVID19, many children's worlds have shrunk to just their screens. This leaves them vulnerable to online sexual exploitation, grooming, violence and bullying.

Governments and the tech industry must step up to keep children safe. #ENDviolence unicef.org/press-releases…

Children at increased risk of harm online during global COVID...
unicef.org

403 1:07 AM - Apr 15, 2020

166 people are talking about this

Figure 2: *a tweet by UNICEF explaining the new "cyber-pandemic" and the risk of this shift to digital citizenship.*

Cyberbullying is just one stream from the online toxicity flow but the grave repercussions are obvious. This evidence is an open letter to the tech industry and the policy makers - there is a need of automated systems capable of detecting toxicity in posts and comments.

Actually, this is what really has motivated us to get engaged in this research topic. Think for a moment: what are the features that distinguish a toxic comment from a non-toxic one? How can a machine detect the toxicity without knowing the *meaning* of the words (in contrast to as humans do)? This is not trivial whatsoever. Every word has a meaning. But the meaning can change depending on the context (add also irony and sarcasm to this set). And it can even change over time. A sentence full of neutral words can be hostile ("*only whites should have rights*") whereas a sentence packed with hostile words (*"f**k what, f**k whatever y'all been wearing*") can be neutral when you recognize it as a famous lyric. And notice that we - humans - are pretty good at this kind of parsing, but machines do not capture these semantic or external-context meanings. We need to train and validate NLP models in order to make them capable of discriminating toxicity quite accurately.

2. Approach

Our approach to tackle this NLP problem can be summarized as using cross-lingual models trained on the *Jigsaw Multilingual Toxic Comment Classification* dataset, first on personal GPUs and then we upgraded to TPUs provided by the host team. Let's discuss each of those steps and components separately.

2.1 Dataset

The *Jigsaw Multilingual Toxic Comment Classification* is the 3rd annual competition organized by the Jigsaw team. The competition aims to use English-only training data to run toxicity predictions on many different languages by means of multilingual models. The data can be found on the competition platform *Kaggle* and the source of the data is either Wikipedia talk page edits or "*Civil Comments* plugin" comments. Overall the corpus contains 63 million comments from discussions relating to user pages and articles dating from 2004-2015.

Every piece of comment is classified as *toxic*, *severe toxic*, *obscene*, *threat*, *insult* or *identity hate*. This labelling was done via crowdsourcing which means that each comment was rated by different people and the tagging might not be 100% accurate, for instance, a comment which might be considered as *toxic* and *threat*, may be just labelled as *toxic*. Or, consider the following ASCII artwork:

```
            /´¯/)
          ,/¯  //
         /   / /
     /´¯/'  '/´¯¯`•¸
   /'/   /   /     /¨¯\\
  ('(   ´(  ´     ,~/'  ')
   \\              \\/   /
    \\            _•´
     \\           (
      \\          \\\
```

Figure 3: *ASCII artwork representing an offensive sign but classified as non-toxic in the Jigsaw Multilingual Toxic Comment Classification dataset*.

It has not been classified as *toxic* in the training dataset, although it is obvious to a human what it is. Perhaps, if you were asked to rate the comment:

```
"/´¯/) \n ,/¯// \n // / \n/´¯/''/´¯¯`•¸ \n
/'////¨¯\\ \n('(´(´,~/'') \n\\\// \n\\_ •´
\n\\( \n\\\\"
```

You would probably not classify it as toxic, because it looks like a series of random characters. Yet, this string is exactly the ASCII artwork shown before.

Even in some *Kaggle* discussions, *Kagglers* agreed that some labels seem to be inconsistent, and there are many examples of identity hate, threats and insults which are not marked as toxic, and they wondered how the inconsistency in labelling is going to impact the ability of classifiers to get higher scores. We highlight we are not going to do cleaning in terms of correcting the inconsistent labels, since this is not the purpose of this project, however, it is a good point to keep in mind.

2.2 BERT and Tokenizers

Google's BERT (*Bidirectional Encoder Representations from Transformers*) is a powerful deep learning algorithm for NLP for text representation. It learns useful text representations by considering the full context of a word, by looking at the words that come before and after it. We cannot feed BERT directly with raw text data - before that, we need to pre-process it coherently. That is, via *tokenizers*.

A tokenizer is in charge of preparing the inputs for a model. For that purpose, `@huggingface` provides a fast and efficient tokenization library. An implementation of a tokenizer consists of the following pipeline of processes, each applying different transformations to the textual information:

- ***Normalizer***: it executes all the initial transformations over the initial input string. Examples include: to lowercase, to strip text, or to apply *unicode normalization*.

Observation: unicode normalization is important because characters that look the same can have different encodings, which can lead to unexpected bugs. To illustrate: the unicode character `é` can be represented either as `U+00e9` (single code point, in `UTF-8`) or as a combination of `e + ´`, i.e. `U+0065` and `U+0301` (two code points, in `UTF-16`). Thus, at machine level `\u00e9 != \u0065\u0301` an issue we would like to avoid.

- ***Pre-tokenizer***: it is in charge of splitting the initial input string. It can split the text into its different word elements using *whitespaces*, or *char delimiters*, for instance.
- ***Model***: this is the algorithm that handles all the sub-token discovery and generation, this part is trainable and dependent on input data. Among all the tokenization algorithms, we can highlight a few sub-token algorithms used in Trans- formers based state-of-the-art models: *Byte Pair Encoding* (BPE), *Word Piece*, *Sentence Piece*, among others.

Note: the pretrained BERT Chinese from Google is character-based, i.e. its vocabulary is made of single Chinese characters. Therefore it makes no sense if we use the word-level segmentation algorithm to pre-process the data and feed it to such a model. Yet, since we will be using English-only training data, we are going to use the BERT tokenizer, which was trained using the WordPiece tokenization.

- ***Post-processor***: it takes care of incorporating any additional useful information that needs to be added to the final output. For instance, for BERT it would wrap the tokenized sentence around `[CLS]` and `[SEP]` tokens.
- ***Decoder***: in charge of mapping back a tokenized input to the original string.

Note: we are going to pick pre-trained BERT weights. However there are two kinds: *cased* and *uncased*. We have to choose whether to pick a *cased* or an *uncased* model depending on whether we think letter casing will be helpful for the task we are trying to solve.

2.2 Cross-lingual Language Models (XLMs)

Let us introduce the cross-lingual language models. The last two years we have seen the rise of Transfer

Learning approaches in Natural Language Processing (NLP) with large-scale pre-trained language models becoming a basic tool in many NLP tasks. While these models lead to significant improvement, they often have several hundred million parameters and current research on pre-trained models indicates that training even larger models still leads to better performance on downstream tasks.

2.2.1 Multilingual BERT (mBERT)

Deep, contextualized language models provide powerful, general-purpose linguistic representations, but the focus on previous to 2019 literature was on models trained on English to capture information about English. The question that has risen is to which degree do these representations generalize *across* languages. In particular, the The monolingual BERT algorithm has been extended to multilingual BERT (mBERT) language model for pretraining (Devlin et al., 2018), as a single language model pre-trained on the concatenation of mono-lingual Wikipedia corpora from 104 languages with a shared word piece vocabulary.

Before going any further, let us make explicit the term *zero-shot translation* - this means that a model can implicitly translate for the language pairs it has never seen during training time. For example, if we train a model on Portuguese to English and English to Spanish, the model is able to generate a reasonable translation for Portuguese to Spanish directly, without seeing any data for the language pair <Portuguese, Spanish> during training time. *mBERT* enables a very straightforward approach to zero-shot cross-lingual transfer, i.e., we can fine-tune the model using task-specific supervised training data from one language, but evaluate that task on another language, and see how it does generalize information across languages.

mBert does not use any marker denoting the input language, and does not have any explicit mechanism to encourage translation, that is, equivalent pairs to have similar representations.

2.2.2 DistilBERT

Transformer models have transformed NLP. They swept all the previous benchmarks, at a computational cost. Those models have become monsters. This trend toward bigger models raises several concerns. First of all, the environmental cost of their exponentially scaling computing requirements. Secondly, while operating these models on-device in real-time has the potential to enable novel and interesting language processing applications, and thanks to this low-parameter model we can execute predictions on machines with low resources.

That is where *Knowledge Distillation* appears - which is a compression technique in which a small model is trained to reproduce the behaviour of a larger model. DistilBERT is the distilled version of BERT, with its size reduced to 40%, retaining 97% of BERT's performance on language under- standing benchmark GLUE, and is 60% faster. This version of BERT was first introduced in the paper "*DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter"* in late 2019.

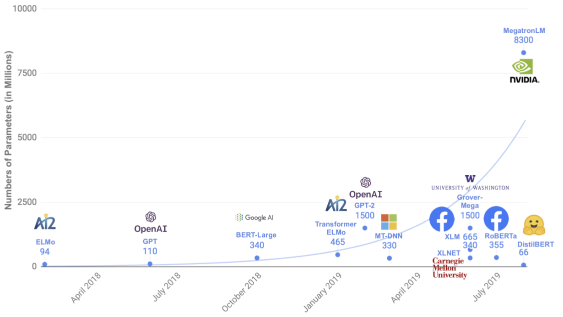


Figure 4: *latest largest Transformer models and their size in millions of parameters.*

2.2.3 XLM

Later, the cross-lingual language model (XLM) was proposed in *Cross-lingual Language Model Pretraining* (Lample and Conneau, 2019). It is a transformer pre-trained using BERT-like extended objectives. It introduced a new unsupervised method for learning cross-lingual representations. These models take advantage of the shared vocabulary - created through *Byte Pair Encoding* (BPE) - across different languages that share the same alphabet, leading to an improvement in the alignment of embedding spaces across languages. In the original paper an interesting fact was explained: a model of a low-resource language can be improved by transferring context from a higher-resource language that shares a significant fraction of their vocabularies.

2.2.4 RoBERTa

The same year, the RoBERTa model was proposed (Liu et al., 2019). It has been built on BERT with modifications in key parameters, training with much larger mini-batches and learning rates.

2.2.5 XLM-RoBERTa

Finally, all those models have been outperformed by a state-of-the-art model called XLM-RoBERTa by Facebook AI (Conneau et al., 2019). Based on RoBERTa, it has been trained on 2.5TB of filtered *CommonCrawl* data (trained on 100 different languages), that has achieved state-of-the-art results both in classification tasks and in translating tasks. *Byte Pair Encoding* (BPE) is used to split the input into the most common sub-words across all languages, thereby increasing the shared vocabulary between languages.

2.3 *Tensorflow* docker container

Since we aim to train many models, and *Kaggle* platform has many constraints, to illustrate, competitors have limited time per GPU and per TPU; and as we have an own machine available with GPU - precisely, a *TS-877 Ryzen-based NAS* with 8 cores and 16 threads, with a *GeForce GTX 1060* 6GB graphics card, we decided to create a `Tensorflow:2.1.1-gpu-notebook` *docker container* to train our models.

First, we login to the server via `ssh`. Then, we use the `docker pull` command to ensure the desired image is installed. Next, we create a docker container, specifying the GPU support, with the following command: `GPU=nvidia0 gpu-docker run -it --rm -v (PATH):/tf/notebooks -p 8888:8888 tensorflow/tensorflow:2.1.1-gpu-jupyter`, where `(PATH)` is the path to the folder with our data. This automatically opens a jupyter notebook environment in the folder specified in `PATH`. To check the number of available GPUs, use `tf.config.experimental.list_physical_devices('GPU')`.

2.4 Tensor Processing Units (TPUs)

A *Tensor Processing Unit*, or TPU, is an AI accelerator application-specific integrated circuit (ASIC) developed by Google specifically for neural network machine learning, particularly using Google's own *TensorFlow* software that we are going to use from this point on.

Initially Google began using TPUs internally in 2015, and in 2018 made them available for third party use. Compared to a *Graphics Processing Unit* or GPU, the TPU is designed for a high volume of low precision computation with more input/output operation per joule, and they are mounted in a heatsink assembly, which can fit in a hard drive slot within a data center rack, this last one is important for usability in data centers and servers all around the world.

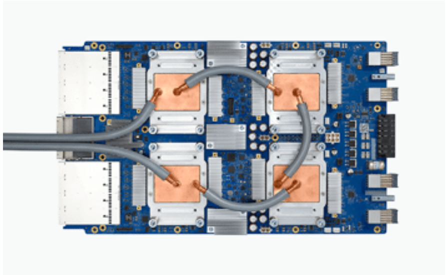


Figure 5: *a liquid-cooled TPU v3 device with 4 chips.*

Architecturally the GPUs and TPUs are very different. A GPU is a processor in its own right, just one optimized for vectorized numerical code, whereas a TPU is a co-processor, it cannot execute code in its own right, all code execution takes place on the CPU which just feeds a stream of microoperations to the TPU.

Wondering why the TPUs can only execute *TensorFlow*, we searched and found that there is really no particular reason a TPU could not run something other than a *TensorFlow* model, it is just that nobody has written the compilers to do so yet. It would be hard, because it is not a completely generic processor, and some of the strange-looking restrictions in *TensorFlow* are there to make TPUs possible.

This information was from 2018 and from then there have been many other experiments to use TPUs outside *TensorFlow*, but only a few of them actually work. The TPUs have been very important in the development of new models and it is now widely available through platforms like *Google Colab* or *Kaggle* for us, data scientists, to try them.

3. Data Exploratory Analysis

We cannot build good models without having a minimal understanding of the data. In total, in the training dataset there are 223,459 comments, each one labelled with the corresponding toxicity class.

Toxic	9.56%
Severe toxic	0.87%
Obscene	5.43%
Threat	0.30%
Insult	5.06%
Identity hate	0.94%

Table 1: *distribution of the toxicity classes.*

During the analysis, we had to define several functions for preprocessing, for instance, a method to replace all the "`\n`" metacharacters by spaces. A classic method for getting insight on text data is by word cloud visualizations, for that, we have used the `WordCloud` library and have imported the English stopwords to get rid of them during the visualizations. They can be seen in figures 6 and 7.

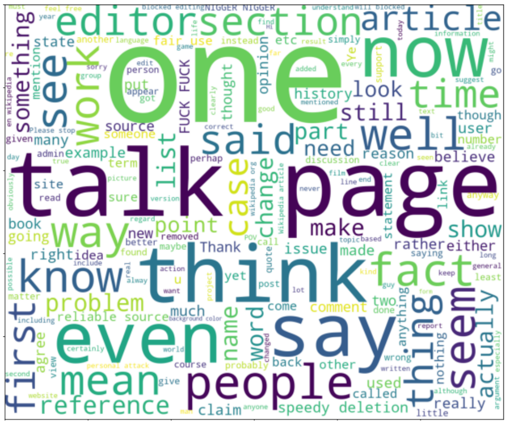


Figure 6: *word cloud visualization of the entire dataset. We can see that words of talk edit pages dominate, like "talk", "page", "editor", "section", and so on. However, we can also spot some toxic words, such as "f**k" or "d*e".*

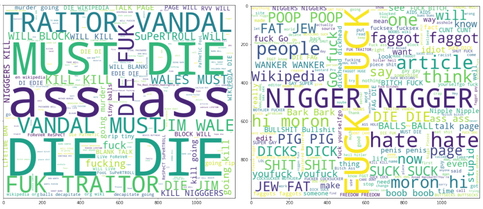


Figure 7: *word cloud visualization filtering the comments for threat and just toxic comments. The dominant words are toxic.*

The big question that we have left hanging in the air at the beginning is what are the discriminatory features that distinguish toxicity. To find those characteristics, we have defined three hypotheses, which we will test next.

The first hypothesis we have formulated is about the length of the comments and the mean length of the words of which a comment is composed of. Even though the distribution description at the concentration region (figure 8) does not give sharp differences, looking at the extremes, i.e. the outliers, has given us some insight. See figures 9 and 10.

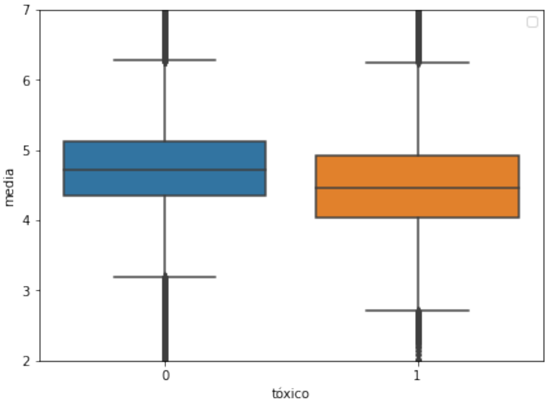
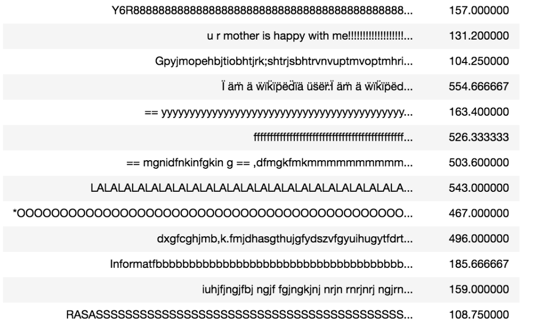
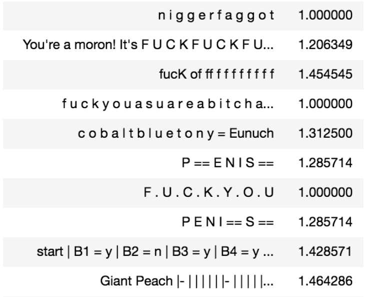


Figure 8: *non-toxic comments (blue) are composed of longer words on average, than toxic ones (orange). However, the difference is slight. The 25% percentile, median and 75% percentile of non-toxic and toxic comments, respectively, are: (4.35, 4.71, 5.12) and (4.05, 4.46, 4.92).*



Y6R8888888888888888888888888888888888888888...	157.000000
u r mother is happy with me!!!!!!!!!!!!!!!!!!...	131.200000
Gpyjmopehbjtiobhtjrk;shtrjsbhtrvnvuptmvoptmhri...	104.250000
Ï äṁ ä ẅïḱïṗëḋïä üṡëṙ.Ï äṁ ä ẅïḱïṗëḋ...	554.666667
== yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy...	163.400000
fffffffffffffffffffffffffffffffffffffffffffff...	526.333333
== mgnidfnkinfgkin g == ,dfmgkfmkmmmmmmmmmm...	503.600000
LALALALALALALALALALALALALALALALALALALALALALALA...	543.000000
*OOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO...	467.000000
dxgfcghjmb,k.fmjdhasgthujgfydszvfgyuihugytfdrt...	496.000000
Informatfbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb...	185.666667
iuhjfjngjfbj ngjf fgjngkjnj nrjn rnrjnrj ngjrn...	159.000000
RASASSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS...	108.750000

Figure 9: *comments with a mean word length too high have shown to be spam, not useful. But they are not toxic, in general.*



n i g g e r f a g g o t	1.000000
You're a moron! It's F U C K F U C K F U...	1.206349
fucK of ff f f f f f f f f	1.454545
f u c k y o u a s u a r e a b i t c h a...	1.000000
c o b a l t b l u e t o n y = Eunuch	1.312500
P == E N I S ==	1.285714
F . U . C . K . Y . O . U	1.000000
P E N I == S ==	1.285714
start | B1 = y | B2 = n | B3 = y | B4 = y ...	1.428571
Giant Peach |- | | | | | |- | | | | |...	1.464286

Figure 10: *comments with a mean word length too low have shown to be, in general, disguised toxic words.*

The second hypothesis we formulated is that the readability complexity can be a toxicity indicator. In particular, *Flesch-Kincaid readability* tests are used. They are designed to indicate how difficult a passage in English is to understand. There are two kinds. One of these is the "*Flesch Reading Ease*", defined as:

$$206.835 - 1.015\left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6\left(\frac{\text{total syllables}}{\text{total words}}\right)$$

In the Flesch reading-ease test, higher scores indicate material that is easier to read; lower numbers mark passages that are more difficult to read. The visualization can be seen in Figure 11.

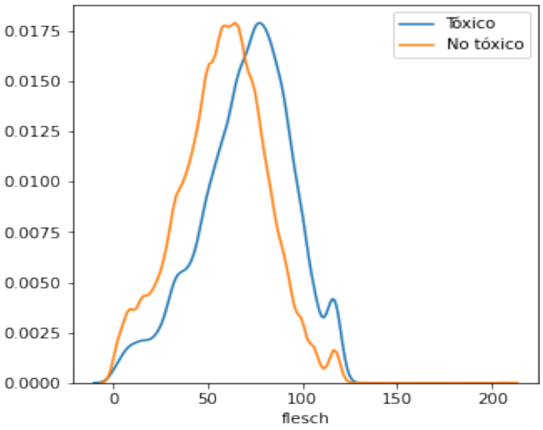


Figure 11: *non-toxic comments (orange) and toxic comments (blue) have a noticeable difference regarding readability.*

Finally, the third hypothesis we formulated is about sentiment. We believed that toxic words should have a tendency to negative sentiment in *Sentiment Analysis*. And we have decided to check that. To perform *Sentiment Analysis* we have used a toolkit from *TextBlob* library, which is a pre-trained Naive Bayes model, based on Stanford NLTK. After applying this model on each comment, we obtain a polarity value ranging from -1 (negative) to 1 (positive), passing through 0 (neutral).

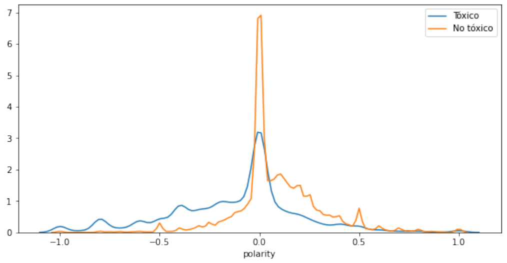


Figure 12: *non-toxic comments (orange) and toxic comments (blue) have a noticeable difference regarding sentiment polarity. Yet there are many words falling to the neutral zone.*

We can observe that toxic comments are more shifted to the negative side polarity, whereas positive comments the other way round. Nevertheless, there is still a strong overlap in the neutrality sector. It was an issue we have not understood until we retrieved a list of the positive words filtered by toxicity.

Next we show the examples and explain this phenomenon:

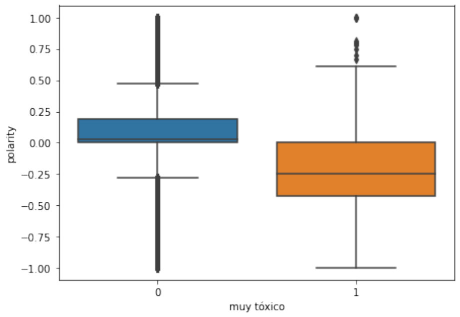


Figure 13: *non-toxic comments (blue) and very toxic comments (orange) have a noticeable difference regarding sentiment polarity. Yet there are many words falling to the neutral zone.*

(1)"**Thank you** for your response. You have answered my questions **perfectly.**"

(2)"**Merry** Christmas Season's **greetings** and **best wishes** for 2012! **Thanks** for all you do here"

These comments have given positive polarity and, in fact, they are positive ones. There are good words and it shows gratitude and wishes.

(3)"i am **awesome**!!!"

The above comment, even though it was not useful whatsoever, it outputs positive sentiment. In the model, each "!" exclamation mark is considered as an emphasizer, and it adds positivity. Since there are so many exclamation marks, no wonder why it has yield really positive polarity.

(4)"Your censorship adds nothing. The statistics I posted come from an **excellent** source, the U.S. Justice Dept. Why does your **highness** not like the statistics? Who made you king?"

We clearly see that the author of this comment was annoyed and he used irony in his reply. This comment should not be positive, yet since he is using "positive" words disguising his real mood, the model has not been able to see the true polarity.

```
(5)"Awesome as fu** !!!"
(6)"porn is the best thing since sports"
```

In the examples above we see that the sentiment polarity is getting biased by "positive" nouns, non-negative structures and emphasizers, such as the exclamation mark.

To conclude the tests of our hypotheses, we can extract some take-home ideas:

- Toxic comments are generally *composed of shorter words*. Toxic words are generally not sophisticated.

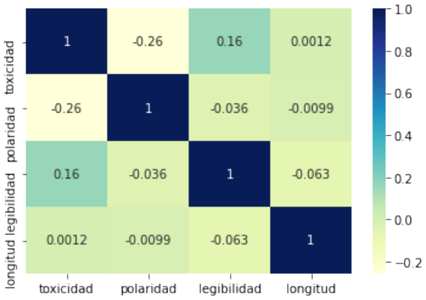


Figure 14: *correlation matrix represented as a heatmap with features {toxicity, polarity, readability, mean length of words}.*

- Toxic comments *are easy to read*: the readability index is higher for toxic words. There is a positive correlation of the readability index and the toxicity of a comment.
- Toxic comments are *dominated by negative sentiment polarity*, if resources such as irony or lexical disguise are not used.

4. Experiments

We have tried different model architectures, including variations in tokenizers and hyper-parameters. In this section we will explain the settings (pre-processing steps), model structures, results and findings.

4.1 Undersampling

As we can see in figure 15, we find ourselves with a huge imbalance in the data. Nearly 90% of the samples are negative and approximately only 10% corresponds to toxic comments. We have to solve this issue because the model could just always predict negative and get 90% of accuracy and that is clearly not what we want.

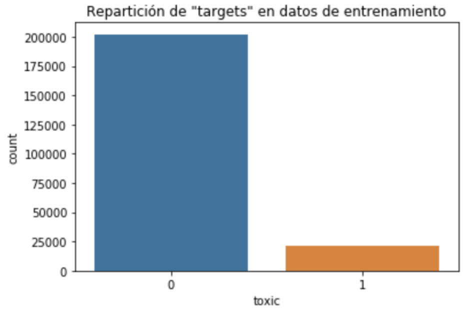


Figure 15: *distribution of non-toxic (blue) and toxic (orange) comments in the dataset.*

One method that helps to overcome this is *undersampling*. Undersampling consists in randomly taking out samples of the largest class, with or without replacement. This technique is widely used but we have to take into account that this could increase variance and could discard potentially good or important samples. After undersampling we have 43.4% toxic comments and 56.7% non-toxic ones, owing to a 6.88% reduction in size of the original dataset. After that, we can freely choose *Accuracy* as our main evaluation metric, as both classes have a balanced distribution.

4.2 Model architectures

As data preprocessing, we have just performed string cleaning via *RegEx*, to get rid of consecutive spaces, # and @ symbols and *https://* links, which could confuse our model. The generic architecture of the toxicity detection model is composed of three main blocks: first, a *Tokenizer*, that prepares the input in a form that the model can ingest; next comes the *Transformer* block, that embeds the text extracts into vector representations (embeddings).

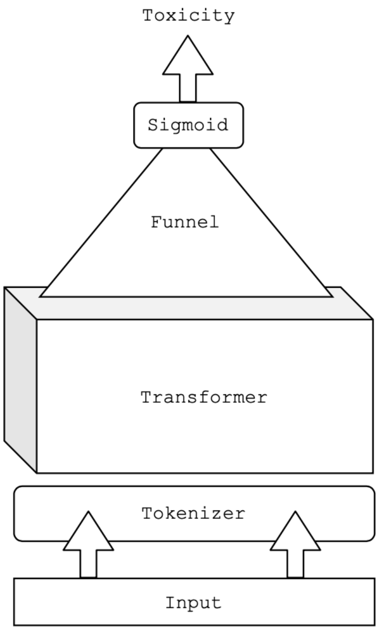


Figure 16: *schematic general architecture we used for our toxicity detection models.*

Finally, we have the *Funnel* component, which models (non-)linear combinations starting from the embedding up to the final node, which contains a neuron with a sigmoid activation function that predicts the toxicity for the given input. The *Funnel* can be just one dense layer, connecting the *Transformer* outputs to the single neuron with the sigmoid, or can be expanded to several layers with non-linearities in between.

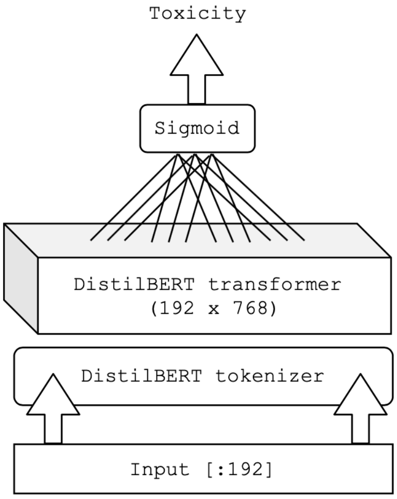


Figure 17: *the skeleton of the best-performing model in our experiments: One-layer Funnel DistilBERT with 192 max input length, with 50% dropout.*

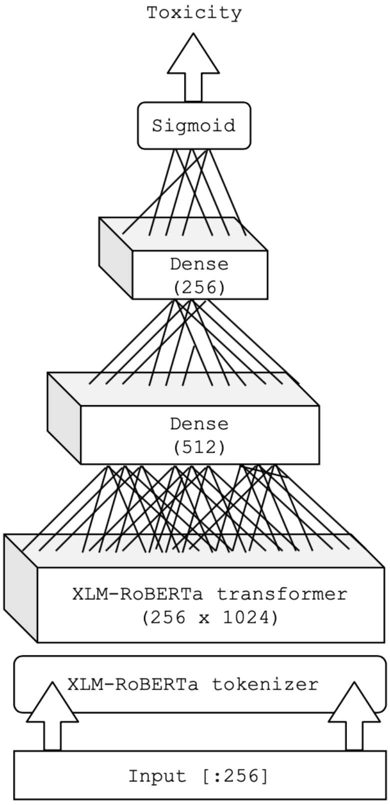


Figure 18: *XLM-RoBERTa model: Two-layer Funnel XLM-RoBERTa with 256 max input length, with 50% dropout and sigmoid non-linearities between hidden layers.*

At figures 17 and 18 we show the most successful architectures in our experiments, even though we have tried some others too. For example, we have also tried deeper models, with more hidden layers in the *Funnel* (up to 5 hidden layers), with *Rectified Linear Unit* (ReLU) non-linearities. Those models performed very badly, at the end they output the same prediction for any entry. What happened is what is called "the dying of ReLUs" - the *Funnel* part, which was responsible for discriminating the inputs, was dead. To use the *leaky version* of ReLUs (with alpha = 0.01) was an attempt to address this issue and give them a chance to recover. However, it did not help; therefore we decided to stick to shallower networks.

You may notice that the input is not ingested entirely, but it is truncated up to some length (either 192 or 256). It is known that BERT has a maximum length of tokens which is 512, thus, for any text with more than 512 tokens we have basically two options:

- Cut the longer texts off and only use the first 512 Tokens. The original BERT implementation truncates longer sequences automatically. For most cases, this option is sufficient.

- Split the text in multiple subtexts, classify each of them and combine the results back together (choose the class which was predicted for most of the subtexts, for instance). This option, obviously, is more expensive.

Yet we can truncate any part of the text, not just the beginning. Some *Kagglers* even suggested taking the beginning combined with the end, because toxic comments had a tendency to appear at the end of the comments. In our case, we have used the first approach - cutting longer texts off. But not 512 tokens, less. Why? More tokens implied more parameters to train and store. This produced *Out-Of-Memory* (OOM) issues and we had no other option than to define a "smaller" maximum length. On our own GPU training, we defined 192 tokens and in TPU training - 256 tokens.

4.3 Hyperparameters

4.3.1 Batch size

On GPUs we have tried batch sizes 16 and 8. On TPUs batch size 128 and 256. We have encountered a problem with batch size 256 and higher, the RAM provided by *Kaggle* was not enough to perform the computations. But, as the TPU guide says, if you want to go faster on a TPU, simply increase the batch size, the absolutely best option would be 128 elements per core, because the 128x128 hardware matrix multipliers of the TPU are most likely to be kept busy. In our case, we are going to scale the batch size with the core count. But we only have been able to use 128 total batch size, although we assume with more RAM capacity we would see an increase in performance with higher batch sizes.

4.3.2 Non-linearities in hidden layers

Sigmoids have shown to work well. ReLU and leaky ReLU activations, as explained in the previous section, have *died* during training time. The hyperbolic tangent activation function has not excelled in results.

4.3.3 Optimizer

For all our experiments we have used the *Adam* optimizer. We have also tried to use *Adadelta*, but it did not work because *Adadelta* optimizer has no support for GPU nor TPU in *Tensorflow*.

4.3.4 Learning rate

On GPU machines we have used small learning rates, the best results have been using 0.0001. On contrast, on TPU machines it is recommended to use larger learning rates, the same as with batch size, even scale the learning rate according to the batch size. So, the best value tried was 0.001.

4.3.5 Dropout proportion

We have been doing experiments with (a) no dropout whatsoever, (b) 35% dropout proportion, and (c) 50% dropout proportion. We discovered that with more dropout, we obtained better accuracy. Thus, the best performing models, in both cases - with Distilbert and XLM-RoBERTa transformers - had 50% dropout proportion.

4.3.6 Epochs

We have tried epochs in range from 3 to 10. In general, with 3 epochs it was enough. Further, >3 epochs, we have not seen significant improvements in terms of accuracy. With 10 epochs we had OOM issues, even though we have been using checkpoint files to store intermediate weights during training.

4.4 Results

The table of results of the best performing models can be seen in the Appendix.

The best model was the one-layer *Funnel* with *Distilbert* transformer, with a little bit of parameter tweaking to extract its full potential, i.e., with maximum length 192, batch size 16, Adam optimizer and 0.0001 learning rate, and 50% dropout proportion.

4.5 Deployment

We have used the *Flask framework* to serve the model. First, we declared all the necessary functions to be able to process an input string and pass it to the *Tensorflow* model. For that, we loaded the Distilbert tokenizer, the encoder, and the model itself. Then, we declared the route for prediction API. Since we wanted to update the toxicity value in the template in real-time, we have used *AJAX*. According to the toxicity value obtained, we change the *CSS* background color through *jQuery*.

All code is shared in `github.com/margaritageleta/multilingual-toxicity-detector/` repository.

5. Conclusions

After looking at the results we extract some knowledge that we did not have before, and we have completed our hopes and objectives successfully.

The first interesting thing that we learnt is about the data, we have worked with real data and we have seen the best and the worst that people can be online. There are a lot of hateful comments online, and you would not think that there are hate comments in Wikipedia, but there are. We can only imagine the amount of toxicity in social media platforms such as Twitter or Instagram. Lately this last one, Instagram, has been asked to deal with all the cyberbullying that occurs in their app, and this "toxicity-meter" that we created could be useful to detect and auto eliminate all the hate towards people.

Then, after dealing with the different characteristics of the dataset, we started doing a bit of research about the topic and how other people confront it. We saw that Bert was still the base for all of the solutions, but other sophisticated models were created to deal with data like ours. Models such as *DistilBert* and especially *XLM* and *XLM-RoBERTa*. This last one is very interesting to us because it was published a few months back only, so it is the newest and greatest in the business.

Another important thing that we learnt, it is the use of TPUs. We have explained what they are in this very document, but we did not discuss how the experience was. And it was great honestly, they work as intended, better than GPUs and it was interesting being able to use them for the exact purpose that they were created.

References

- Anne E. McTavish. Toxic Language (2013).
- Bauman, S., Toomey, R.B., Walker, J.L.: Associations among bullying, cyberbullying, and suicide in high school students. Journal of adolescence 36(2), 341–350 (2013).
- Devlin J. et al.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018).
- Pires T., Schlinger E., Garette D.: How multilingual is Multilingual BERT? (2019).
- Lample G. and Conneau A.: Cross-lingual Language Model Pretraining (2019).
- Sanh V., Debut L., Chaumond J., Wolf T.: DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter (2019).
- Liu Y. et al., RoBERTa: A Robustly Optimized BERT Pretraining Approach (2019).
- Conneau A. et al. [Facebook AI]: Unsupervised Cross-lingual Representation Learning at Scale (2020).

Appendix

Model description (+ hyperparameters)	Number of parameters	Trained on (+ execution time)	Training accuracy	Validation accuracy
One-layer Funnel with DistilBERT transformer. 3 epochs, batch size 16, Adam optimizer with learning rate 0.0001, dropout 50% and maximum length 192.	134,734,849	GPU (3.42 h)	98.24%	84.62%
One-layer Funnel with DistilBERT transformer. 3 epochs, batch size 16, Adam optimizer with learning rate 0.0001,	134,734,849	GPU (7 h)	98.20%	84.54%

dropout 35% and maximum length 192.				
One-layer Funnel with DistilBERT transformer. 3 epochs, batch size 16, Adam optimizer with learning rate 0.0001, dropout 0% and maximum length 192.	134,734,849	GPU (2.48 h)	98.20%	84.51%
Two-layer Funnel (sigmoid activations) with XLM-RoBERTa transformer. 5 epochs, batch size 16*TPU str. = 128, Adam optimizer with learning rate 0.001, dropout 50% and maximum length 256.	560,546,817	TPU (1.76 h)	97.14%	84.63%
Two-layer Funnel (sigmoid activations) with XLM-RoBERTa transformer. 5 epochs, batch size 16*TPU str. = 128, Adam optimizer with learning rate 0.001, dropout 35% and maximum length 256.	560,546,817	TPU (1.08 h)	97.18%	81.50%