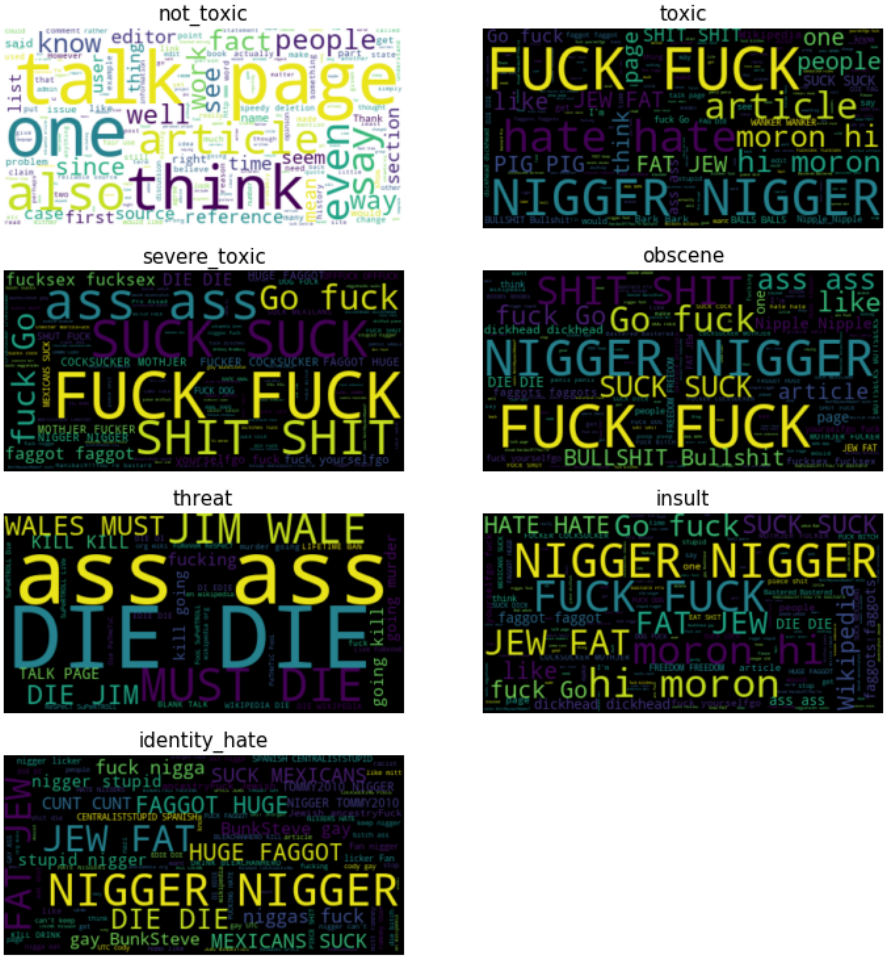
Toxic Comment Classification challenge
Udacity Capstone - Machine Learning Engineering

R. Angeles



Chapter 1

Definition

1.1 Overview

In this project we work on the Kaggle toxic comment classification challenge that aims improving online conversation. In a nutshell, the challenge consist in building an ML model to classify Wikipedia, talk edits page, comments as being toxic, severe toxic, obscene, treat or identity hate.

1.2 Problem Statement

Our aim is to explore the performance of three well models for the analysis of this data set: CNN, GRU and LSTM on top of the GLOVE pre-trained word embedding. We not only aimed for a model that performs well but deployed it on website via Sagemaker.

1.3 Metrics, goals and reasults

The benchmark analysis for comparison is the most voted kernel of the competition: a Naive Bayes - Support Vector Machine model, it reaches 0.9772 auc roc, while the current leader team reaches 0.98856. The main results of the work consist in exploring the performance three popular model for NLP: CNN, LSTM and GRU. When combined with a given preprocessing, and using the Glove pretrained word embedding our model scores only 0.07% below our benchmark model under 10 mins, on a laptop.

In addition, we deployed the CNN model to an interactive website via SageMaker and on the road we create portable utilities to use the benefits Torchtext on Sagemaker pytorch containers. After presenting our analysis we discuss performance and future directions.

Chapter 2

Analysis

2.1 Data Exploration

Jigsaw/Google/Kabble provide a training dataset of 160K comments, with binary labels corresponding to the following classes: *Toxic, Severe toxic, Obscene, Threat, Insult or identity hate*. Additionally, a test dataset with over 153K unlabelled test comments is provided to make a submission to the competition. The dataset is available at [1]. This should be placed under a data directory to reproduce calculations. Fig. 2.1 illustrate entries of such dataset. The statistics of this data will be presented below through a series of plots. The

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation Why the edits made under my userna...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww He matches this background colour I'm se...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man I'm really not trying to edit war It's...	0	0	0	0	0	0
3	0001b41b1c6bb37e	More I can't make any real suggestions on impr...	0	0	0	0	0	0
4	0001d958c54c6e35	You sir are my hero Any chance you remember wh...	0	0	0	0	0	0

Figure 2.1: Training dataset

most occurrent words corresponding to the labels are in fact shown in the tittle page of this document.

2.2 Exploratory Visualisation

The data exploration is presented on */local/preprocessing_exploration.ipynb*. The training set is distributed according to Fig. 2.2. It is extremly imbalanced. Correlations of the labels are shown in Fig. 2.3 Some correlations are large. But since we will predict each class independently this is not a problem. The histogram number of words distribution of the training set comments is presented on Fig. 2.4. There are comments which are clearly much longer but the vast majority are shorter than 80 words as the cat plots in Fig. 2.5.

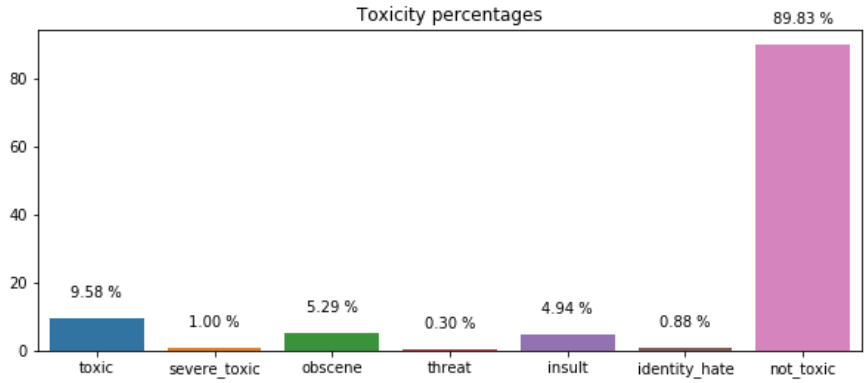


Figure 2.2: Distribution of the different labels on training set

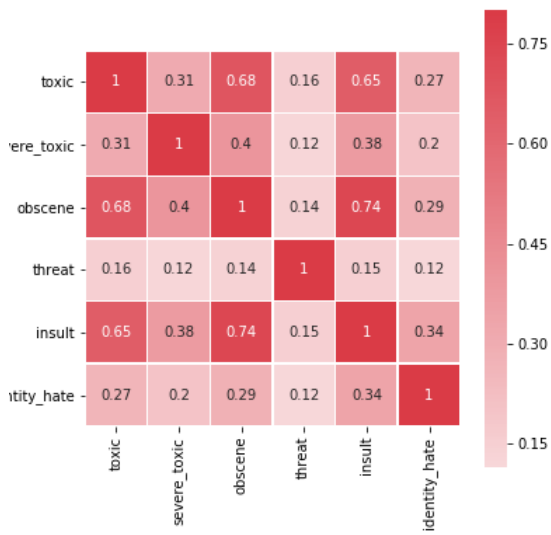


Figure 2.3: Correlations coefficients between labels

2.3 Algorithms and Techniques

The algorithms used to analyse the text are based on dimensional reduce the tokenize representation of words to a pretrained embedding. Specifically, we used the word embedding [2], loaded with Torchtext. In order to solve this classification problem, over this embedding the following deep learning architectures applied:

- Long-Short-Term-Memory. The foundations of this model can be found at [3].
- Gated-Recurrent-Units. See [3] and [4].

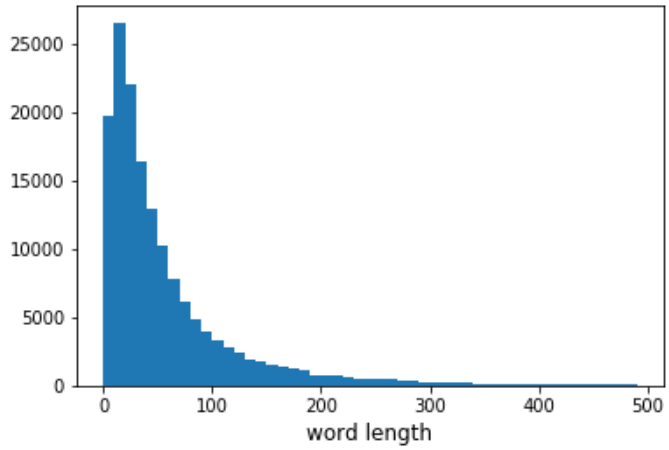


Figure 2.4: Number of words

- Convolutional approach. This scored the best evaluation metric and was most efficient. An introductory, yet thorough, introduction to this architecture can be found at Ref. [5].

Finally, the model performing the best was deployed to a website via SageMaker. This required creating utilities to make torchtext preprocessing objets run on pytorch containers.

The last needed technique for the analysis of this work is the area under the curve (AUC) of receiver operating characteristic (ROC) metric, or rather the average between the different classes. The ROC is defined as the curve generated by plotting the true positive rate vs the false positive rate for each possible threshold probability value used to predict the classes, and the AUC ROC is simply the are under this curve. In fact, since the present problem is a multi-label problem the metric is the average AUC ROC obtained for the different classes.

The ROC curves predicted for the convolutional model are presented in Fig. 2.6. The AUC ROC is simply the areas under this curves. AUC ROC equal to 1 corresponds to having true positive rate equal to 1 and false positive rate equal to 0, each probability chosen. As a consequences probability predicted for false and positive classes are completely separated.

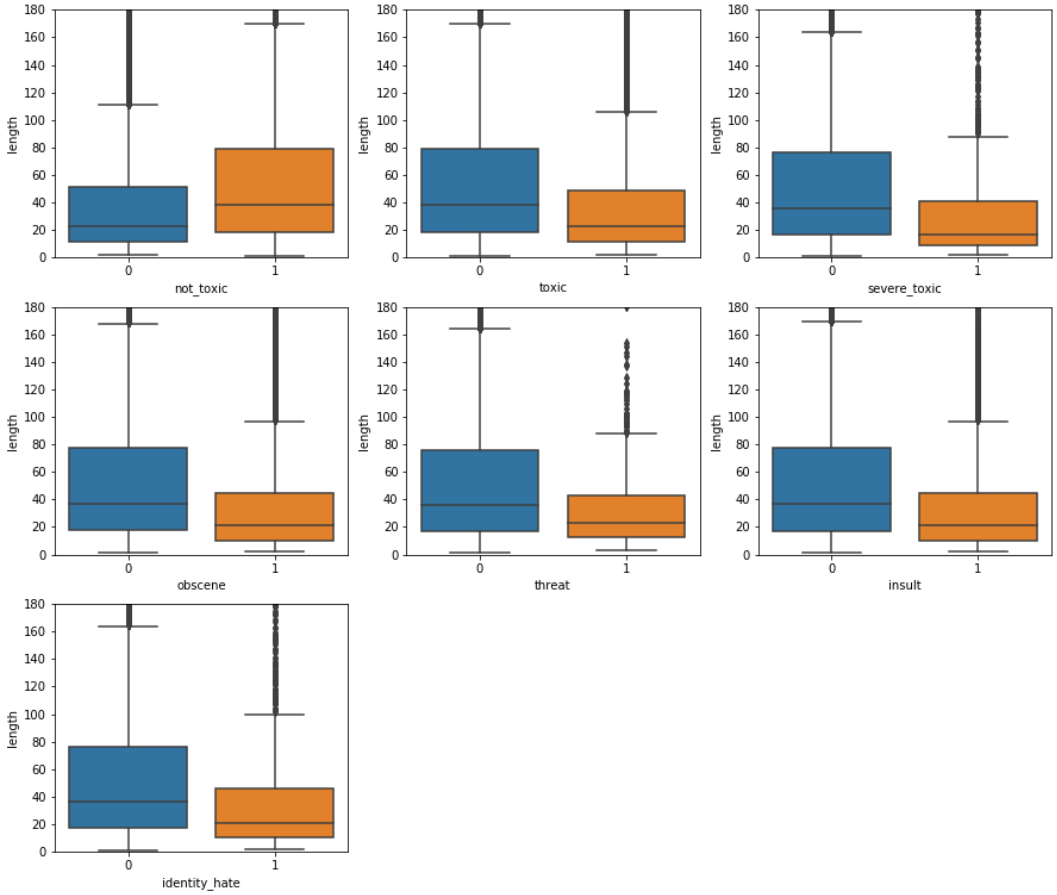


Figure 2.5: Number of words per comment

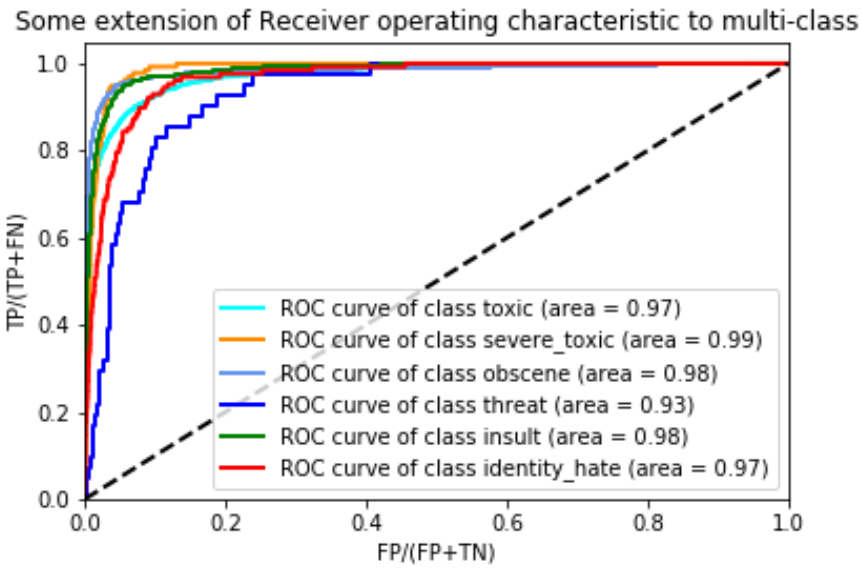


Figure 2.6: ROC curve for the model based on convolutions

Chapter 3

Methodology

3.1 Data Preprocessing

We three deep learning architectures for natural language processing. Before applying these, we perform the following text preprocessing (found at the local/preprocessing folder if the git repo):

1. Removed special characters except for numbers and apostrophes, which were kept according to the english grammar rules.
2. Substitute any numerical character by letter n.
3. Reduce long comments to 80 words. This choice improved our metric and it was justified by observing the catplots of Fig. 2.4. Indeed, most comments have 80 or less words. Interestingly most toxic comments tend to be shorten than 40 words long.

The next steps were implemented using the [6] and can be found on the /local/building-models folders of the git repo

4. Stopwords removal was initially implemented but in fact this slightly reduced the accuracy obtained.
5. Reducing words to their stemmed also slightly lower our evaluation metric.
6. Tokenization was implemented using the Spicy package.
7. Comments were fed to models in batches of 256 comments to all models. Comments with similar number of words were grouped together, and padded, such that each batch had elements with identical number of words. It is worth remarking that this is easily achieved by Torchtext and that we choose models compatible with this choice. Different batches sizes were attempted but such batch size delivered the best evaluation metric performance.
8. Finally, we embedded the training vocabulary into the glove.6B.100d embedding provided by torchtext, with a vocabulary of the 20000 most frequent words on the training set. Of course plus the unknown and pad tokens, which were set to zero vector before trainig. Other vocabulary lends were tried and also the different vocabulary fasttext.en.300d.

3.1.1 Implementation

Finally, we trained models on 90%, randomly chosen, of the training dataset and left the remaining 10% for cross-validation. The following three model architectures were attempted,

- Long Short Term Memory (LSTM. We used the bidirectional version with hidden dim of 30 for each direction, on top of a dense layer of 30 neurons, which were connected to the 6 different classes. Trained for 7 epochs.
- Gated Recurrent Unit (GRU) Same configuration as LSTM. Trained over 4 epochs.
- Convolutional approach. This scored the best evaluation metric and was most efficient. So we describe it in more detail. In fact, it is a pleasure to acknowledge credit of this models to the tutorial on sentiment analysis with torchtext and pytorch in Ref. [5].

The package *PytorchSummaryX* can be use to provide a summary of the model architecture provided in Tab. 3.1. This table shows that after embeding an input of comments with 15 words each, there are three groups of 100 convolution filters acting with dimensions [2,3,4] times 100. This can be tough as examining the 2 grams, 3 grams and 4 grams to extract features. There is actually a bug in this program as a max pool layer that acting on the output convolutions named 1-conv, 2-conv and 3-conv. Such max pool layer outputs a tensor of size (batch_size, (number of convolutions) x (embedding size)). Finally,

```
The model has 2,092,306 trainable parameters
torch.Size([15, 256])
==============================================================================
                      Kernel Shape       Output Shape   Params Mult-Adds
Layer
0_embedding           [100, 20002]     [256, 15, 100]  2.0002M   2.0002M
1_convs.Conv2d_0  [1, 100, 2, 100]  [256, 100, 14, 1]    20.1k    280.0k
2_convs.Conv2d_1  [1, 100, 3, 100]  [256, 100, 13, 1]    30.1k    390.0k
3_convs.Conv2d_2  [1, 100, 4, 100]  [256, 100, 12, 1]    40.1k    480.0k
4_dropout                         -         [256, 300]        -         -
5_fc                      [300, 6]           [256, 6]   1.806k      1.8k
------------------------------------------------------------------------------
                         Totals
Total params          2.092306M
Trainable params      2.092306M
Non-trainable params        0.0
Mult-Adds                3.152M
==============================================================================
```

Figure 3.1: NLP based on convolutions

This model 2M parameters to train. Most of them correspond to the embedding. This model trained on 9 mins on a 2017 laptop, and under 4 mins on GPU via SageMaker. Finally it is worth mentioning that in all three cases we used dropout regularization, with probability 0.5, for the connected layers.

3.2 Refinements

Some refinements were already mentioned above here we summarize briefly:

- Tried removing stopwords and stemming words but our models work best by removing all special characters and punctuation leaving only aphostrophes.
- Boxplots show that the vast mayority of comments are shorter than 80 words so we take this a the word limit to analyse. A Future improvement should perhaps break each comment into slices of 80 words.
- Used Glove embedding, the best performance was achieved with the 100 dim version tokenizing the 20,000 most common words appearing on the training data set. Higher numbers seem to reduce accuracy.
- The dimension of the dense layer dimension was choosen to run under 1 hour on cpu reaching the highest score.
- The number of convolutional filters and its dimensions were also tuned.

3.3 Results

3.3.1 Model Evaluation and Validation

Figs. 3.2 show the learning and loss curves at half train steps, for the corresponding training batch and over the complete cross validation dataset. From each pair of curves one can infer that training is long enough without overfitting our model to the training set.

Up to this point it seems like we have solve the problem, reaching the ROC AUC of the benchmark model. The final scores rated by kaggle are:

Model	AUC ROC score on test set
GRU	0.9655
LSTM	0.9639
NBSVM benchmark	0.9772
CNN	0.9765
Kaggle Leader	0.9885

Two conclusions can form this table. Firstly, our benchmark is a tough competitor as it achieves pretty similar results without the complexity of neural networks. Nevertheless, we deploy the convolution model to SageMaker reducing the training time to have using a ml.xlarge.p2 contained for training. Future work can easily boost the accuracy of our model but we leave this as a pending task. Secondly, and more importantly, tables in Fig. 3.3, show that the accuracy other score metrics are not great for the extremely imbalanced classes $\{severe_toxic, threat, identity_hate\}$.

Normally, a high AUC ROC would mean that we have a robust model whose predicted probabilities for the positive and negative classes are well separated. However, the histograms shown in Fig. 3.4, for the probability distributions for each fo the true positive and negative classes, show that this is not the case for all classes. Clearly, for the extremely unbalanced classes more work is need as the probabilities for positive and negative classes have a lot of overlapping. Future work should address this problem making an Error analysis, addressing imbalance with resampling methods or focal loss [7].

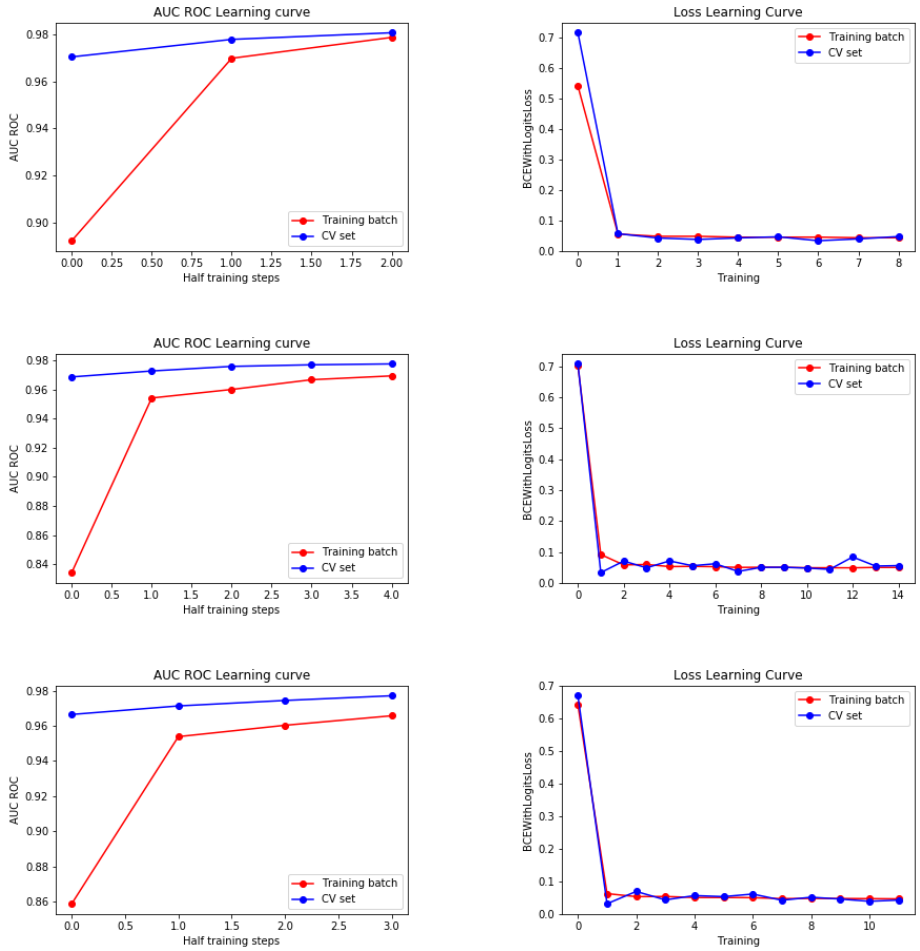


Figure 3.2: Learning curves. The first, second and third rows correspond to the CNN, LSTM and GRU models respectively. The x axis runs over half epoch steps.

3.3.2 Justification and deployment

Our results for for the CNN architecture are comparable to the benchmark, just 0.011% percent from the team leading this competition. Furthermore, our model is a good predictor has f1 scores above 70%, on the cross validation datasets for obscene, insults, toxic comments. We decided to take this model to SageMaker, retrained there and deploy it to the website, see Fig. 3.5.

The deployment of this file involved following similar steps as those presented on [4]. However, to use pre-trained embeddings and torchtext (for removing stopwords, stem wods, tokenize, create iterators) a series of utilities were created. Furthermore, the model is compable to run on GPU significantly reducing training time to a half only ml.xlarge.p2 container. All the implementation code for this deployment can be found on /sagemaker of the

	Label	accuracy	recall	precision	f1	roc_auc
0	toxic	0.963402	0.735178	0.859784	0.792614	0.974601
1	severe_toxic	0.991414	0.175182	0.500000	0.259459	0.988882
2	obscene	0.982328	0.792593	0.849206	0.819923	0.984397
3	threat	0.997431	0.024390	0.500000	0.046512	0.973423
4	insult	0.975497	0.667519	0.799387	0.727526	0.984595
5	identity_hate	0.993044	0.289062	0.649123	0.400000	0.977930

	Label	accuracy	recall	precision	f1	roc_auc
0	toxic	0.965720	0.721344	0.898277	0.800146	0.977577
1	severe_toxic	0.991414	0.000000	0.000000	0.000000	0.989878
2	obscene	0.978693	0.822222	0.772622	0.796651	0.985101
3	threat	0.997431	0.000000	0.000000	0.000000	0.961309
4	insult	0.973178	0.739130	0.720698	0.729798	0.984267
5	identity_hate	0.991978	0.000000	0.000000	0.000000	0.967836

	Label	accuracy	recall	precision	f1	roc_auc
0	toxic	0.966096	0.729908	0.894270	0.803772	0.976511
1	severe_toxic	0.991414	0.000000	0.000000	0.000000	0.991037
2	obscene	0.979131	0.818519	0.780919	0.799277	0.984754
3	threat	0.997431	0.000000	0.000000	0.000000	0.958836
4	insult	0.972551	0.723785	0.718274	0.721019	0.984265
5	identity_hate	0.991978	0.000000	0.000000	0.000000	0.967122

Figure 3.3: Metric scores. The first, second and third row correspond, respectively, to the best CNN, LSTM and GRU setting found. A probability threshold of 0.5 was set for classifying probabilities into labels.

github project. I could not find similar utilities form the NLP community so I consider this a valuable contribution for people using torchtext on SageMaker or those wanting to use TorchText tools on TensorFlow.

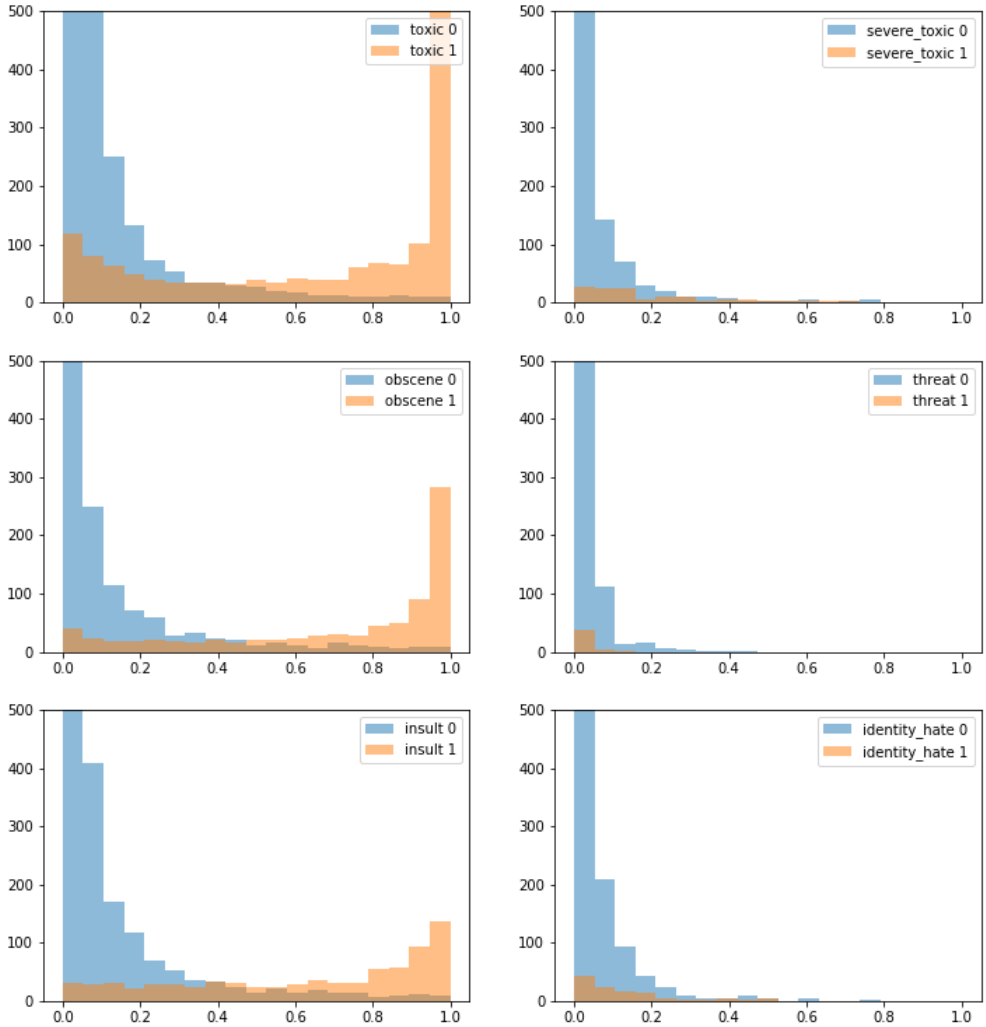


Figure 3.4: Histograms of predicted probabities for the negative an positve classes.

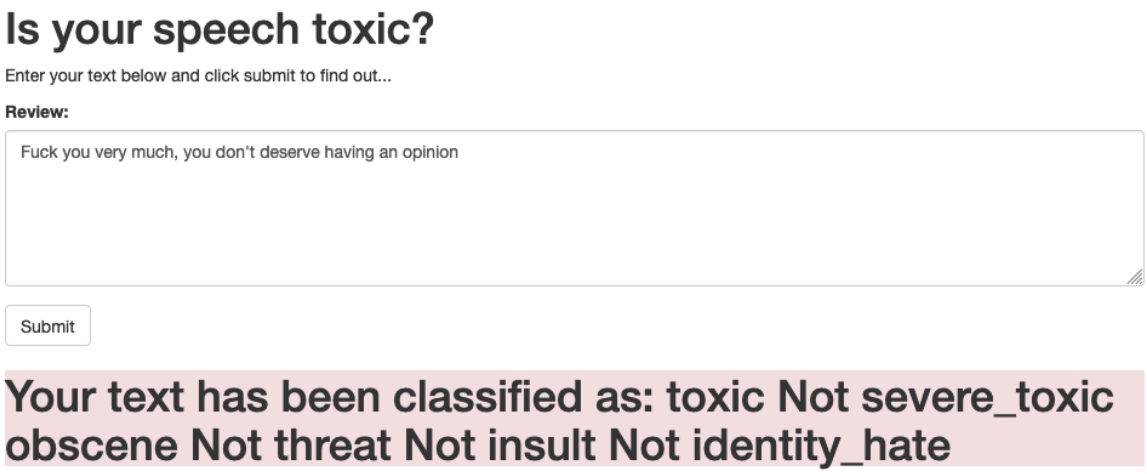


Figure 3.5: Deployed website

Bibliography

[1] Google Jigsaw and Kaggle. Toxic comment classification challenge.

[2] Google. Glove.

[3] Andrew NG. Sequence Models.

[4] Udacity. Machine Learning Engineering Nanodegree.

[5] Ben Trevett. Tutorials on PyTorch and TorchText for sentiment analysis.

[6] Pytorch. Torchtext package.

[7] et. al. Tsung-Yi Lin. Focal Loss for Dense Object Detection.