toxicity-filter

Github: https://github.com/minhd-vu/toxicity-filter

Authors: Vu Nguyen, Minh Vu

Introduction

In many online environments, toxicity can be a pressing issue regarding inclusivity, cyberbullying, and comfort. Toxicity can be defined as behavior that negatively impacts others and potentially makes the online environment less enjoyable. In order to remedy the issues that toxicity presents, a filter can be created to mitigate the impact of toxicity by detecting whether a statement is toxic or not and censoring it accordingly. Additionally, the filter should be able to discern content that is constructive criticism or from toxic content.

Input/Output

For example, we would like it so that when people are in a chatroom or a video game, if someone decides to type in a toxic statement, we would like to be able to detect that and immediately block it. However, we would not like to censor constructive criticism, so we must also consider differentiating between toxic speech and criticism.

Input	Envisioned Output	Actual Output
"You should kill yourself"	"you ***"	0.8508837223052979
"Play more defensively"	"play more defensively"	0.013167029246687889
"You're trash!"	"you're ***"	0.8076990246772766

Typically, words such as "kill" and "trash" are not censored by the filtering systems in place; however, because these tokens are deemed to make the statement toxic, they are censored.

Impact

Toxicity is most daunting in online gaming. Currently, most systems to deal with toxicity are simple curse word filtering, but the systems in place don't go beyond to detect whether something is toxic or not. Moreover, many times some keywords may be censored when they should not be due to the lack of depth in the censoring software. By implementing a filter toxicity option, developers could utilize NLP to decrease the amount of toxicity.

Methodology

At first, we tried utilizing tools such as PyTorch to train a BERT model, however, we ran into many errors that were very difficult to fix. We wanted something simple to use, so we found a library called Simple Transformers, which makes the model training process simple for beginners. It is based on the HuggingFace and Transformers library. We also decided to use roBERTa instead of BERT since roBERTa was used in the examples provided by Simple Transformers.

We started working on preprocessing the data first. We lowercased all of the sentence data and removed all unnecessary newlines so that each sentence is on one line to prevent errors when trying to train on the text. When we were trying to do binary classification, we decided to take the toxicity column and rounded the values to `0` or `1` for each sentence to classify if a sentence was toxic or not. This isn't the most accurate way to classify whether a sentence is toxic or not, so we later scrapped the idea of rounding and decided to not change the toxicity column when we began trying to train a regression model. This means we wanted values from `0` to `1` to classify how toxic a statement was because some sentences are more toxic than others.

We then created a new CSV file to include our sentence data and their corresponding toxicity value. We will pass this CSV file into Simple Transformers to train our model. This same process was used for the test data as well. We utilized Simple Transformer's lazy loading feature to overcome the obstacle of exhausting system memory when training large models. Lazy loading just loads data from the disk instead of putting the data into RAM. Training will be slower but you can train very large models using this feature. We also utilized Google Collab to train our model since training on our hardware will take a long time. Once you set up the model parameters, you are ready to train and evaluate your model!

Experiments

When we first started with binary classification, we had an accuracy of around 92%, but this was misleading as it turns out that our model classified everything as not toxic. Our test data comprised of 92% of non-toxic statements and the rest were toxic, which explains why we had such high accuracy. We found out that our training data had a large non-toxic majority, which made the model classify everything as non-toxic to get a high accuracy rate. We decided to multiply the number of toxic sentences to even out the test set by duplicating the toxic sentences by a certain factor. After doing this, we got an accuracy rate of 77%.

When we were training our regression model, we calculated a loss of `0.01`. We didn't get to make an evaluation script, but if we did make one, we would probably go through the test data and evaluate the sentence using our model. Then, we would accept a certain margin of error. If the evaluation result was within the margin of error, we would classify it as a success.

Dependencies

```
mkdir data
cd data
pip3 install kaggle simpletransformers
kaggle competitions download -c jigsaw-toxic-comment-classification-challenge
kaggle competitions download -c jigsaw-unintended-bias-in-toxicity-classification
```

Training

```
python3 baseline.py
```

When training the model a simple change of the `regression=True` to `regression=False` will change whether the model is regression or binary classification. You will also have to adjust the input data accordingly. We utilized the `class_labels()` function in `functions.py` to map the floating-point data to `0` and `1` when doing the binary classification.

API

```
python3 server.py
```

The API utilizes the regression model. Sending a POST request to the API will return a value from `0` to `1` depending on the level of toxicity that is detected. Create a `cache_dir` directory in the root directory or simpletransformers will throw an error. Now, `curl` a POST request to the API, just specify the message.

```
curl -X POST -H "Content-Type: application/json" \
    -d '{"message": "why are you so bad!"}' \
    http://localhost:5000/
```

Shortcomings

We attempted to deploy to Heroku but it seems that it's a little limiting for what I'm trying to do. Because the model files are so large, we had to use Git LFS and they can be cloned on Github. We believe I might have reached the cap of the Github LFS free tier (which is around 1GB so if everything doesn't download and you want to test, we can always upload to another service and provide you with the link).

The Discord bot was unable to be created because the API was never deployed. We believe that deploying would be rather easy if we were using something like a VPS (through something like Digital Ocean with an Nginx reverse proxy) rather than Heroku, but we didn't want to incur the cost of a VPS at the moment.

The filtering was never completely accomplished. Although there was a way to determine if an entire sentence was toxic or not, we wanted the user to define whether something was deemed toxic depending on their application. For example, if they required a toxicity filter in a chat application among elementary students, then the threshold for whether something was toxic would be fairly low, and vice versa for a chat app among adults. Therefore, to accomplish the desired filtering, the application side would have to censor the entire sentence.

Conclusion

We learned that our data needs to have an even ratio to train or else we will have inaccurate evaluations. We also learned that a regression model was probably a better choice in our case since classifying toxicity isn't black and white as some statements are more toxic than others.

We also set up an API that would give you the toxicity value when you make a POST request with a sentence, so you can see how toxic a statement is with a value from `0` to `1`.