Toxic Comments – N gram analysis

HW3

Harsha Vardhan, Khurdula.

CS59000-08 Natural Language Processing

Instructor: Jon Rusert

15th Feb 2023

Problem

The current task at hand requires us to create three different language models. On the toxic comments corpus, and then calculate scores for each of these comments. And then analyze the results for each data set.

Pre-processing

The goal of preprocessing is to normalize all the comments. The goal is to remove all special characters, punctuations and repetitive comments and as well as empty strings. I intend to use only two columns from the train.csv, namely “comment_text” and “toxic”. Where comment_text is the comment made by a user, and toxic is an encoded label which determines if that particular comment made by an user is rather toxic or otherwise not.

Padding also is necessary in order to determine the start and end of a sentence, hence each comment is tagged with ‘<s>’ at the beginning of each comment and ‘</s>’ at the end of each comment.

Approach

I tried to keep the approach simple, and divide the entire data of 150k+ comments into three classes of data.

1) Complete data. (Data which contains both toxic and non-toxic comments.)
2) Only toxic comments.
3) Only non-toxic comments.

Now that I’ve divided to split the data, the goal now is to train the models.

Initially, the score determined by a simple MLE() model is absolutely 0 or negative INF! When we use ln to determine the score. However, I decided to use Laplace() model which seems to work well, although we still have very low scores.

Working with such low probabilities, might still lead to underflow exception, or just get a lot of zero probabilities. Programmatically, I used two classes within the script, A normalize class and LanguageModels class which implement the task.

The goal is to split each comment into Bigrams and then calculate the probabilities using these bigrams for each comment.

The method LanguageModel.train_LM() method trains 3 different Models. Model 1 called LM_full is trained on entire data, LM_toxic is trained on only toxic comments, whereas a third model LM_not is trained on non_toxic data.

Observations:

1) **LM_full:**

This model is fitted on entire normalized comments data, a total close 150,000 comments. Initially, I would never seem to get any significant probabilities even with log probabilities.

- This language model, gives very low probabilities, such that they are represented in exponents, after extracting the score. When tested against entire comment corpus.
- When tested against the non_toxic comments, we yield close to same probability scores, as most of the comments within the comments are non_toxic.
- However we see better numbers whenever we test the model against toxic comments.
- These increase in probabilities, does not mean that a comment can be classified as toxic, as these models fail to capture sentiment, rather they determine the probability based on the vocabulary that they've learned.
- The problem with this model is that, since we have way too many non-toxic comments, and very few toxic comments, when this model is tested against toxic comments, we get very low scores.
- These scores where slightly improved, by manipulating the input data to the Language Model, when trained instead providing it entire corpus, I fed it with equal number of toxic and non-toxic comments. However the change noticed is not significant enough to make this change permanent for better scores. But was a useful insight.

2) **LM_not:**

This language model is fitted on comments which are not toxic at all. It contains close to 80,000+ comments, and 166673 items within its vocabulary. LM seems to have marginally better scores than LM_full, these could be for some of the following reasons:

- Maybe, having lesser number of comments, prevents this language model from being overfitted, which leads to very small probabilistic values.
- The vocabulary better captures the comments, as it is easier to work with most of the known words.
- One of the interesting facts about these Models is that all of them seem to generate a same score for a comment, across each data set.
    - For example a toxic comment X has same scores within: Entire_corpus test, and Toxic_corpus test.
- The scores that are generated for non_toxic comments are 10 times greater than the scores that are generated for non_toxic comments, this is primarily because it has been trained on non-toxic corpus.
- It is also biased towards vocabulary that it has seen previously, meaning: most of the slurs used within a toxic comment, have been encountered by this Language model, and having those slurs as <UNK> gram, for calculating the score, reduces the score by times 10, for a score determined against toxic comment, that was not previously known to the Language Model.
- This model tends to generate lower scores on entire corpus as there are a lot more UNK comments with un encountered vocabulary.
- However, it is safe to assume this model is skewed towards non-toxic comments, as it has been only trained on such dataset.

3) LM_toxic:

The Language model that has been trained on only toxic data, is the third model that is returned by the train_LM() method. This model has been trained only on 28000+ comments, and contains 31269 items within its vocabulary. This model seems to perform poorly out of the three. Except for the toxic data set. Here are my observations:

- One of the reason for this model to perform very poorly against entire corpus dataset is the fact that it has never seen a lot of these comments, its vocabulary is very limited compared to the rest of the models. For instance, this model has vocabulary which is less in size by 4-5 times when compared to LM_full and LM_not.

- This model also performs very well when tested against toxic corpus, we get to see better scores, and this is mainly because it has been fitted against these comments previously.
- Out of the three language models, this model generates scores which are very low for most of the corpus. It performs well on the comments that it was trained on.
- But when induced or tested with additional toxic comments, even those comments are determined to have lesser scores, because the model does not have most of the slurs used within these newer toxic comments. Hence it does not do well with outliers.
- This model could do well with newer toxic comments, only if it had more toxic data to be trained on. This model seems to be under fitted.

Using the script

The script that I've written for this HW3, has two classes within it:

1) Normalize(), which normalizes the entire dataset as discussed under section, *pre-processing.*
2) **LanguageModels(),** this is the class that contains two required methods, i.e. train_LM() and test_LM().

Note:

The test_LM method does not take any file_path as the data is stored within the class itself, it is not required, it only takes one parameter i.e. the language model and generates three csv files for each test.

The following is a sample of how this script can be executed within python idle.

```
>>> from HarshaVardhanKhurdulaHW3 import LanguageModels

>>> obj = LanguageModels()

>>> language_models = obj.train_LM("trainingData/train.csv")
```

Note: here we are storing a list that is returned by the train_LM() method, this list contains objects to the Three trained Language models i.e.

1) Language_models[0] is **LM_full**
2) Language_models[1] is **LM_not**

3) Language_models[2] is **LM_toxic**

>>> obj.test_LM(language_models[0]) //tests LM_full against datasets.

>>> obj.test_LM(language_models[1]) //tests LM_not against datasets.

>>> obj.test_LM(language_models[2]) //tests LM_toxic against datasets.

The output is stored within newly created csv files.

References:

Daniel, J., & H. Martin, J. (2023). *Speech and Language Processing*. Stanford University Press. https://web.stanford.edu/~jurafsky/slp3/3.pdf

Khurdula, H. V. (2023, February 14). *Toxic-Comments-N-gram-analysis*. Github. Retrieved February 15, 2023, from https://github.com/Khurdhula-Harshavardhan/Toxic-Comments-N-gram-analysis.git