The viability of Character Embeddings in neural toxic Wikipedia comment classification

Lucas Liu

May 6, 2021

1 Introduction

Internet toxicity remains a pervasive and elusive problem in online communication spaces. Quality conversations and meaningful discourse are difficult if not impossible to maintain due to user anonymity and scarcity of human oversight. On December 17th 2017, the Conversation AI team, founded by Jigsaw and Google, released a dataset on Kaggle as part of their *Toxic Comment Classification Challenge* [1]. The stated aim of this competition was to outperform Conversation AI's Perspective API toxic comment classification model by training a model on the provided dataset, with the overall highest achieving team receiving 35,000 $ as a prize. The majority of publicly available code for this challenge utilized recurrent neural models and word embeddings. This choice is curious, as toxic users are not known for their correct usage of the English language nor their consistent spelling, both of which hamper the performance of word embeddings. Consequently, the aim of this report is to investigate whether character embeddings are a viable alternative to word embeddings using appropriate multi-label multiclass evaluation metrics.

2 Challenge Context

2.1 Dataset

The provided dataset for this challenge comes formatted as several CSV files. Each line in the CSV contains a hash ID presumably generated from the comment, the comment in string format, and 6 integers that correspond to the 6 classes of toxicity, serving as something similar to one-hot encoding.The 6 types of toxic comment are as follows: *toxic*, *severe toxic*, *obscene*, *threat*, *insult* and *identity hate*. The classification task is multi-label multi-class classification. Each toxic comment may fall into any combination of the above classes of toxicity. Comments provided in the corpus were sampled from Wikipedia articles and comments from 2004-2015 then classified by crowdsourced human annotators [1].

2.2 Other Submissions

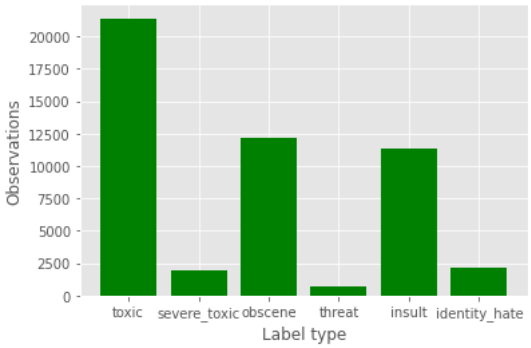
It should be noted that since this competition ended roughly 3 years ago, before PyTorch's rise in popularity, most of the publicly available code is written in Keras. Sorting by the "most votes" filter under the code tab, several patterns and trends in model design can be observed. Many submissions choose to implement either a recurrent neural network (usually LSTMs) trained on word embeddings [2] [3] [4], a linear classifier or logistic regression trained on word n-grams [5], and the relatively scarce newer submissions that use pre-trained models like BERT [6]. These publicly available notebooks typically demonstrate baseline model efficacy with accuracies around 96-98 %. Leading submissions achieve above 98 % in ROC-AUC score [7], but have private code likely due to the sizable prize pool.

[1] Toxic Comment Classification Challenge
[2] Word Embedding & LSTM
[3] Do Pretrained Embeddings Give You The Extra Edge?
[4] Pooled GRU + FastText
[5] Logistic regression with words and char n-grams
[6] Transformers: multi-label classification
[7] Competition Leaderboards

3 Challenges and Motivations

3.1 Dataset Imbalance

In examining the corpus [8], it became clear that the distribution of toxic comments is extremely imbalanced. Of the 223,549 comments, 201,801 were not toxic. The vast majority of comments that were toxic satisfied only 1 form of toxicity.



Furthermore, we can see that distribution of the various classes of toxicity is also extremely imbalanced. The frequency of the *toxic* label is almost double that of the next most frequent label *obscene*. The least common type of toxicity, *threat*, appears in less than 1 % of the total number of toxic comments. Consequently, designing a model that generalizes well to all 6 classes despite being trained on an imbalanced dataset is of practical interest.

3.2 Evaluation Metrics

Accuracy is not a viable evaluation metric for this dataset for several reasons. The first is that, as previously mentioned, the dataset is extremely imbalanced. A high accuracy does not provide much information on how well our model performs on each class individually. Furthermore, what accuracy entails in a multilabel classification task is not well defined. Sklearn's implementation, which I use, considers subset accuracy: for each comment, if our model off by a single label per class or more then that comment is considered mislabeled [9]. It appears that most of the publicly available baselines with previously mentioned accuracies of around 96-98% do not use this metric, as a baseline that used BERT and subset accuracy achieved a maximal subset accuracy of 93% [6]. Instead, it is likely that they consider each comment 6 different classification tasks then compute accuracy over the total number of labels correct. For a dataset of one comment, if the model is off by 1 label, then the reported accuracy would be 5/6. Evaluating a model on a more appropriate metric such as macro and micro multiclass/multilabel f1 scores will give better insights into the performance of candidate models and the overall hardness of this classification task.

3.3 Toxicity and Word Embeddings

Toxic comments, from a grammatical perspective, are not likely to utilize consistent spelling nor commonly understood interpretations of what various words mean. Training word embeddings instead of using pre-trained word embeddings for this classification task may help alleviate the latter challenge. Though the spelling of the words *stupid* and *sttupidd* are different, a human classifier would have little to no difficulty recognizing the semantic and qualitative similarity of both words. However, word embeddings and fixed word-token vocabularies may struggle to learn consistent embeddings between semantically identical but misspelled words during training. This limitation poses several obstacles. Toxic users often attempt to disguise their vulgar language by deliberately misspelling them so as to avoid simple automated moderation programs. During evaluation, a misspelling of an otherwise common and semantically meaningful word may not be recognized as part of the vocabulary and be replaced with an unknown. During training, a word and its misspellings might end up having wildly different embeddings due to the contexts they are observed in and the frequency of their appearance. This motivates investigation into character embeddings as an alternative, as while individual characters may be misspelled, the overall character level structure of toxic words should remain intact.

[8] Challenge Dataset
[9] sklearn.metrics.classification_report

3.4 Inconsistent Labels

Cursory examination of the dataset reveals many comments which have confusing or inconsistent labels. Several examples follow. The longest comment in my training corpus is "SCHOOL SUCKS" followed by several hundred exclamation marks. This was considered by the human annotators to be toxic and obscene, which suggests an unusually low bar for toxicity. The comment "SHUT UP, YOU FAT POOP, OR I WILL KICK YOUR ASS" is considered identity hate yet several comments with racial and anti-semitic slurs were not. It is a general trend that the less common labels of *severely toxic*, *threat* and *identity hate* do not have consistent definitions and are difficult, even for humans, to infer from comments.

4 Project Implementation

My project evaluates 4 of the neural model designs we've learnt about in class on word and character level embeddings for a total of 8 different models. These designs include neural bag of words, convolutional neural network, recurrent neural network (GRU) and convolutional recurrent neural network (LSTM). The models are evaluated on accuracy, subset accuracy, micro f1 and macro f1 scores.

4.1 Evaluation Metrics

The *accuracy* metric is what is used by most of the publicly available Kaggle code. It considers each comment to be 6 distinct classification tasks, and so getting one label wrong out of 6 does not result in a 0 accuracy for that comment. This was used to serve as frame of reference and means to draw comparisons to Kaggle's models.

The *subset accuracy* metric considers a comment mislabeled if its predictions for any of the 6 classes of toxicity do not align with the true label.

The *Micro F1* score "averages the total true positives, false negatives and false positives" and is similar to the *accuracy* metric [9]. This helps avoid the pitfalls of accuracy when a large proportion of the dataset is nontoxic.

The *Macro F1* score averages the model's f1 scores on each of the 6 different labels with equal weights assigned to each class [9]. This gives better insight than micro f1 and subset accuracy into how well the model generalizes to all 6 classes.

4.1.1 Metric Baseline

The baseline performance for these 4 metrics was obtained using a model that exclusively predicts 0 for each of the 6 classes, in effect considering every comment regardless of content to be non-toxic. It achieves 96.3 % on *accuracy* and 89.9 % on *subset accuracy*. *Micro* and *Macro f1* are 0 as per their definition. This suggests that most of the Kaggle models are not substantially better than this baseline in regards to accuracy.

4.2 Dataset

For the roughly 230,000 comments, 70,000 of them were left out for the dev set with the remaining being used for training. The text corpus is first cleaned by removing most forms of punctuation except relevant ones like the exclamation and question marks and periods, as well as replacing uppercase with lowercase letters. Individual comments are then word tokenized using space as the delimiter. The word token vocabulary consists of 50,000 of the most popular tokens plus the padding and unknown word tokens for a total of 50,002 tokens which defines the number of embeddings we train. The character token vocabulary consists of 31 different character token, including punctuation and padding and unknown tokens. Many of the comments tend to be long, so we only keep the first 128 word tokens and the first 640 character tokens. Models are trained from 5-30 epochs using batch size of 64. At the end of each epoch, models are evaluated on the dev-set using SKLearn's accuracy and F1 score reporting methods[9].

5 Model Implementation

5.1 Loss

As our task requires multi-label multi-class classification, we will use binary cross entropy loss. This necessitates the usage of a sigmoid function as our nonlinearity in the final layer for all models. For

numerical stability, pyTorch's BCEwithLogitsLoss loss function is used as it uses the logsumexp trick [10].

5.2 Neural Bag of Words

Input indices are converted to word/character embeddings. The resultant tensor is of shape sequence length by embedding dimension. We then taken the mean of the embeddings and pass it through a linear layer with 6 output dimensions before applying the sigmoid nonlinearity. For word embeddings, optimal results were achieved with embedding size of 64. For character embeddings, optimal results were achieved with embedding size of 32. This and all other models were implemented in pyTorch.

5.3 Convolutional Neural Networks

Input indices are converted to word/character embeddings then convolved with a set number of filters which span a set number of input tokens. Each convolution between filter and index span produces a scalar activation. The resultant tensor is of shape sequence length minus filter width by number of filters. We then maxpool over each of the kernel activations to get a tensor, per comment, whose dimensionality is the number of filters. This is then fed into a linear filter then sigmoid. Character embeddings of 32 dimensions with 1024 filters of width 4 were used. Word embeddings of 96 dimensions with 2048 filters and filter width 2 were used. Dropout with p of 0.2 and 0.5 is used between layers.

5.4 Recurrent Neural Network / GRU

Input indices are converted to word/character embeddings then fed into a Generalized Regular Unit. The GRU units are bidirectional and have 2 layers for a total of 4 layers across word and character based approaches.The final hidden state of the uppermost layer is fed into a linear layer then sigmoid. The GRU used for the word embedding based approach has 256-dimensional hidden state, whereas the one for the character embedding has 128. Dropout with p of 0.2 and 0.5 is used between layers.

5.5 CONV-LSTM

Input indices are converted to word/character embeddings then convolved with a set number of filters that span a certain width of input tokens. The convolved activations are then treated like embeddings and input into a bidirectional LSTM with 2 layers and a 256 dimensional hidden state. The hidden states of the LSTM are then max and average pooled along each dimension, then concatenated to get a tensor of size 1024. This tensor is fed into a linear layer of output size 128 with a relu activation then a final linear layer before the sigmoid activation. Dropout with p of 0.2 and 0.5 is used between layers.

6 Results

Model	Accuracy	Subset Accuracy	Micro F1	Macro F1
Baseline (return 0)	96.28	89.90	0	0
NBOW(word)	**97.68**	**90.64**	**63.54**	**46.43**
NBOW(char)	96.36	89.72	10.17	7.07
CONV(word)	96.66	85.88	64.04	54.38
CONV(char)	**97.32**	**89.44**	**67.79**	**55.96**
GRU(word)	98.12	91.85	71.98	59.02
GRU(char)	**98.15**	**91.87**	**73.51**	**59.85**
CONV-LSTM(word)	**98.05**	**91.72**	**71.73**	44.30
CONV-LSTM(char)	97.88	90.97	71.10	**50.37**

7 Analysis and Conclusion

As one would expect, a character based bag of words system is unlikely to perform well in classifying natural language. Word embeddings outperform character embeddings in a bag of words type model. Individual character embeddings by themselves provide fairly little context and make it difficult to interpret the meaning of a sentence. Wildly different sentences can have similar character bag of words representations. The character bag of words model has an unexpectedly high precision but very low

[10] BCEWithLogitsLoss

recall. This could be because certain types of overtly toxic comments can be characterized by aggressive punctuation and repetitions of exclamation marks.

Character level embeddings outperform word level embeddings when used in convolutional and recurrent neural networks. However, word embeddings do better when used in the CONV-LSTM model. One reason for this may convolution over word embeddings provides a wider context, spanning multiple words, whereas convolution over character embeddings spans only portions of words. This suggests that though character embeddings provide more reliable insight into the meanings of individual words or small spans of subwords, this benefit is outweighed by the vastly greater context being captured by convolution followed by recurrent neural networks. Another reason may be that the complexity of the CONV-LSTM model combined with character index sequences being on average much longer than word index sequences make a character embedding based CONV-LSTM more difficult to train, overshadowing any possible benefits. The adverse affects of model size can be observed in how the performance for the word CONV-LSTM is not that much better than the word embedding convolutional network.

Subset accuracy is lower than that of micro accuracy likely for the same reason that macro F1 is lower than that of micro f1. The *severe toxic*, *threat* and *identity hate* labels are not only rare but inconsistently defined. Subset accuracy considers a comment correctly labeled if all of its class labels were correct, which is extremely difficult to do given the noisiness of the aforementioned three labels. Macro F1 weighs the F1 scores for each class equally. The difference in F1 score between *toxic* and *extreme toxic* would usually be around 20 points, with some models like the CONV-LSTM having up to 50 points, with similar discrepancies for *threat* and *identity hate* with *obscene* and *insult*.

From the table, we can see that character embeddings are in general about on par with word embeddings presuming that we're not using a bag of words or other similarly context-less model. Additionally, the character embedding based GRU network has the best performance on all 4 metrics. It is clear that character embeddings are at least a viable alternative to word embeddings and can do very well, outperforming word embeddings, when used in conjunction with certain model designs. However, representing text as characters is relatively expensive as the input index sequence becomes much longer, which causes difficulties stemming from model size and performance.

The notebook can be found here