Machine Learning Engineer Nanodegree
Capstone Project

Yiyan CHEN
July 12th, 2018

Toxic Comment Classification Challenge

I. Definition

Project Overview

Discussing things you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

The Conversation AI team, a research initiative founded by Jigsaw and Google (both a part of Alphabet) are working on tools to help improve online conversation. One area of focus is the study of negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful or otherwise likely to make someone leave a discussion). So far they've built a range of publicly available models served through the Perspective API, including toxicity. But the current models still make errors, and they don't allow users to select which types of toxicity they're interested in finding (e.g. some platforms may be fine with profanity, but not with other types of toxic content).

The main goal of this project is to build a multi-headed model that's capable of detecting different types of of toxicity like threats, obscenity, insults, and identity-based hate with a comparable result to the current best model. In addition to this, 2 other objectives of interest will be shown in the next section

Problem Statement

To notice that, the problem is not to find the best fit category among the 6 labels for a given toxic comments. This problem is in fact a multi-label classification problem rather than a multi-categories classification problem. More precisely, In the submission file, we need to give a probability for the presence of each of the 6 types of toxicity in each test comment. So the output of the predictor will be a vector of 6 dimensions. These probabilities are "independent" so they don't need to be summed up to 1.

Basic CNN:

Classify toxic comments with an acceptable public and private ROC-AUC score(>95%) on the official test set.

Solution by steps:

1. Implement a basic CNN used to classify texts with a ROC-AUC according to this reference: [4]
2. Implement automatic parameters tuning using metrics evaluated by cross validation in order to find the best parameters for the model

Improvements:

Try my best to get a highest possible score, at least outperform the benchmark model using logistic regression[3].

Solution by steps:

1. Improve data preprocessing by considering more features
2. Adding more layers
3. Add batch normalization and drop out
4. Multi-inputs and multi-output model? (To be clarified in later section)

We have two major strategies to handle a multi-label classification problem, one is to make prediction one by one, another is to first extract some common feature then predict them all together. So it’s interesting to compare this two strategies by comparing shared parameters CNN model above with the benchmark model[3] using exactly the same input preprocessing methodology.

Metrics

In the submission file, we need to give a probability for the presence of each of the 6 types of toxicity in each test comment. So the input of the predictor will be a raw text comment. The output of the predictor will be a vector of 6 dimensions. One dimension is the probability of the comment having a toxicity type. The overall Accuracy of each single label binary prediction(if >0.5 : 1 else 0) is not fitted for this model, because of the unbalancy of each class. For example of the “toxic” label, there are 144277 non toxic comments and 15294 toxic comments, if we predict all are not toxic, we can get an accuracy of 90%.

I can think of two possible evaluation metrics:
1. The mean column-wise F2 score giving more importance on recall
2. As proposed by kaggle: the mean column-wise ROC-AUC
To better understand the metric 2, a column is a toxicity type, the predictor will associate each positive and negative sample with a score(probability of being positive), then the column-wise

ROC-AUC is that if we randomly select a positive and a negative sample, the probability of the former having lower score than the later:

$$P\Big(\text{score}(x^+) > \text{score}(x^-)\Big)$$

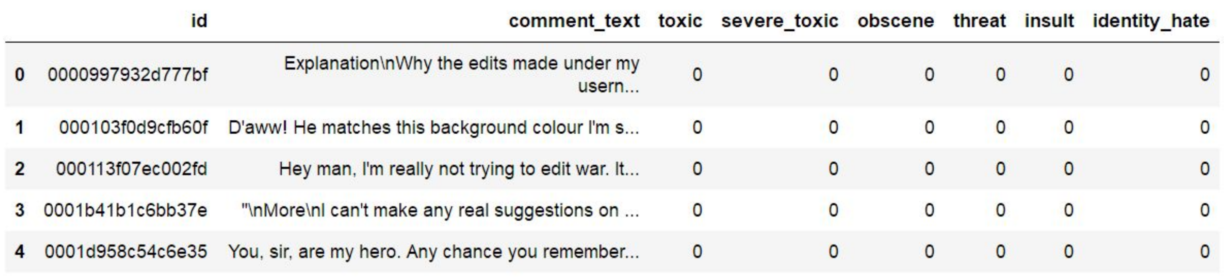
These two are all reasonable and their meaning are quite understandable. I will chose the second one for the ease of comparison with other models since this is required by the competition organizer.

II. Analysis

Data Exploration

Data overview

We are provided with a large number of Wikipedia comments(wiki talk[1]) which have been labeled by human raters for toxic behavior:



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

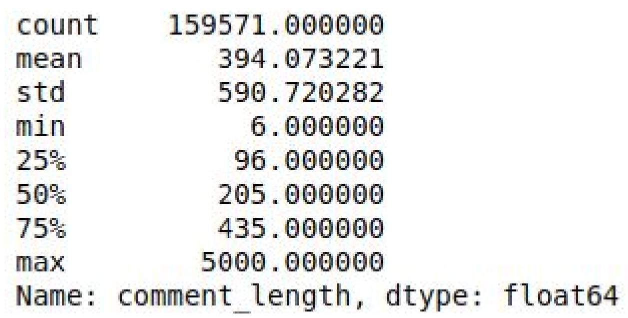
The only feature we have is the comment text in english while we can extract more features from it such as word, characters and length. There are 6 targeted binary variables.
For the toxic label, there are 144277 non toxic comments and 15294 toxic comments in the training set, so the distribution is not balanced and this is true for all 5 other labels. Finally, there are 153164 comments to be labeled in the test set. We can see that there are much more non toxic comments(around 90%) than toxic comment(around 10%), so the class distribution is not balanced. This is the reason why I chose the ROC-AUC metric to measure the performance.

File descriptions [6]

train.csv - the training set, contains comments(text and ID) with their binary labels
test.csv - the test set, you must predict the toxicity probabilities for these comments.
test_labels.csv - labels for the test data; value of -1 indicates it was not used for scoring;

Comment length

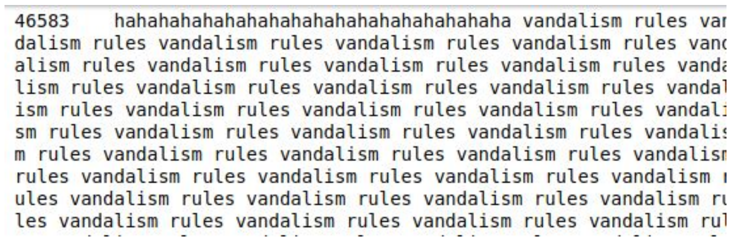
The comment_text field is complete, there is no “null” or empty comment.
As shown in the picture above, the only feature we have is the text comment and there are 6 targeted binary variables.



```
count    159571.000000
mean        394.073221
std         590.720282
min           6.000000
25%          96.000000
50%         205.000000
75%         435.000000
max        5000.000000
Name: comment_length, dtype: float64
```

The median of the length of one comment is around 200 chars and the mean is around 400 chars. The max length is 5000, probably limited by wiki talk backend server. This means that there are some extremely long comments as outliers.

Abnormality



```
46583    hahahahahahahahahahahahahahahahahaha vandalism rules van
dalism rules vandalism rules vandalism rules vandalism rules vand
alism rules vandalism rules vandalism rules vandalism rules vanda
lism rules vandalism rules vandalism rules vandalism rules vandal
ism rules vandalism rules vandalism rules vandalism rules vandali
sm rules vandalism rules vandalism rules vandalism rules vandalis
m rules vandalism rules vandalism rules vandalism rules vandalism
rules vandalism rules vandalism rules vandalism rules vandalism r
ules vandalism rules vandalism rules vandalism rules vandalism ru
les vandalism rules vandalism rules vandalism rules vandalism rul
```

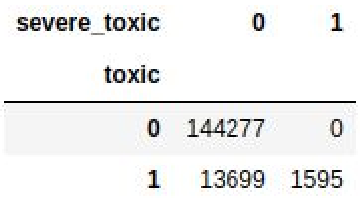
Further investigation shows that most of extremely long comments are due to repeated expression, in this case, we don’t need to consider the entire comment. The 200 - 400 first chars will be sufficient for example.

Exploratory Visualization

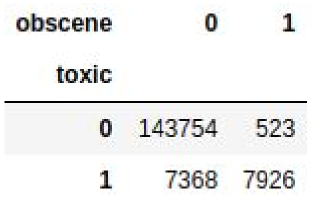
In this section, you will need to provide some form of visualization that summarizes or extracts a relevant characteristic or feature about the data. The visualization should adequately support the data being used. Discuss why this visualization was chosen and how it is relevant.
Questions to ask yourself when writing this section:
- _Have you visualized a relevant characteristic or feature about the dataset or input data?_
- _Is the visualization thoroughly analyzed and discussed?_
- _If a plot is provided, are the axes, title, and datum clearly defined?_

Correlation between labels

In the diagram below we can see that “severe toxic” can imply “toxic” but other 4 toxicities can not imply toxic: for example there are 523 obscene comments which are not toxic.

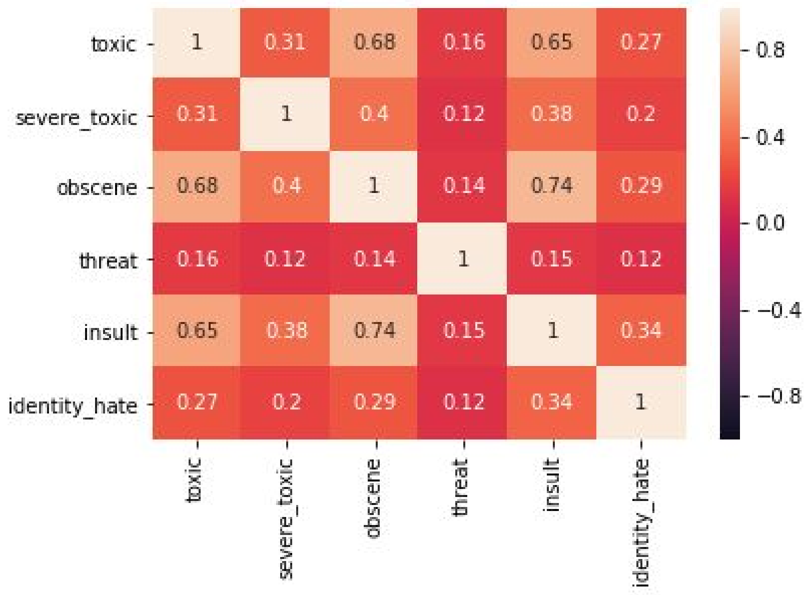


severe_toxic	0	1
toxic		
0	144277	0
1	13699	1595



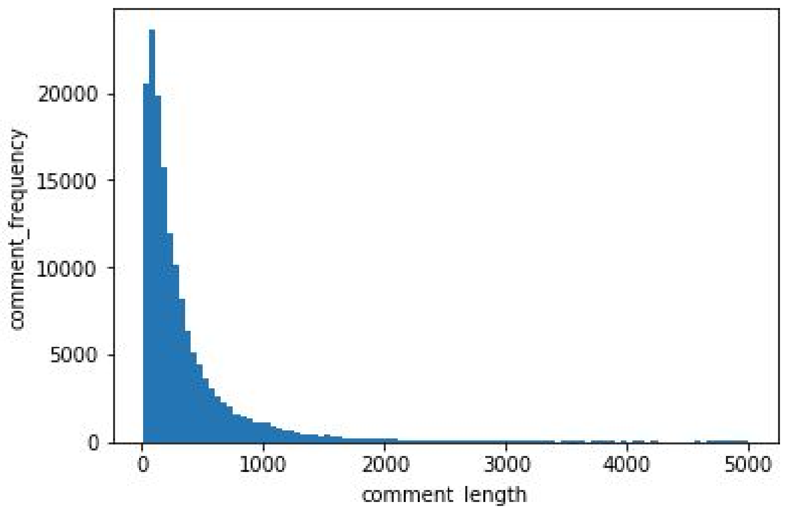
obscene	0	1
toxic		
0	143754	523
1	7368	7926

The diagram below shows the correlations between each pair of labels:

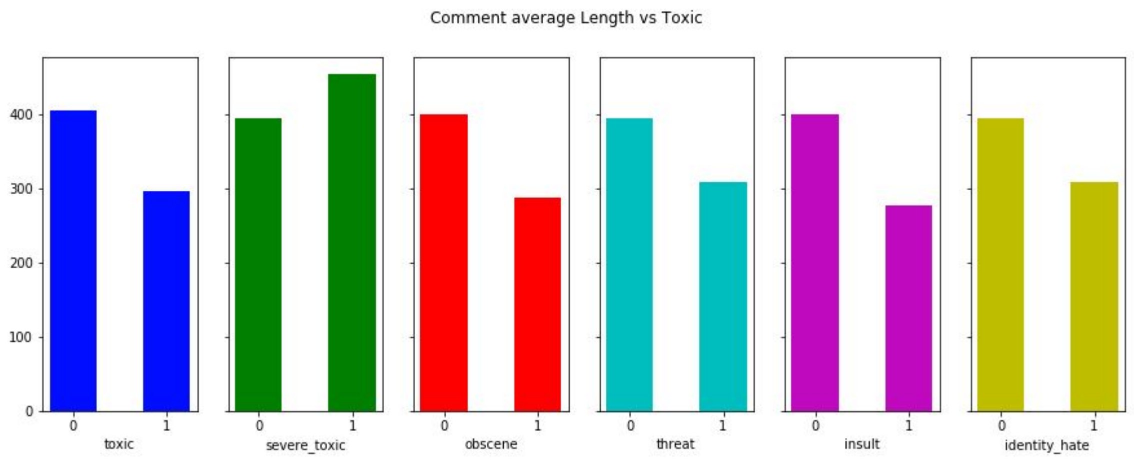


We can see that “toxic”, “insult” and “severe_toxic” are quite relevant, other label are relatively independent.

Length of comments

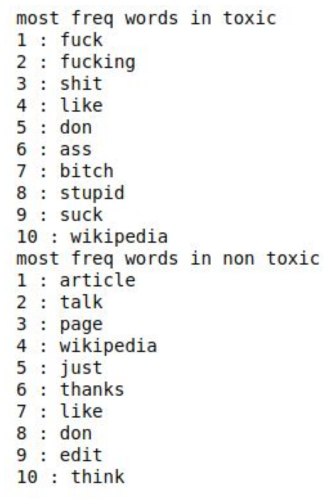


There are few extremely long comments at the right side of the graph. Most of comments length are inferior to 1000. The most frequent length is around 200.
One question, does the toxicity affect the comment length?



The graph above shows an interesting aspect, most toxic comments have shorter length then normal comment except for “severe_toxic”(maybe they need more text to be considered as “severe”?).

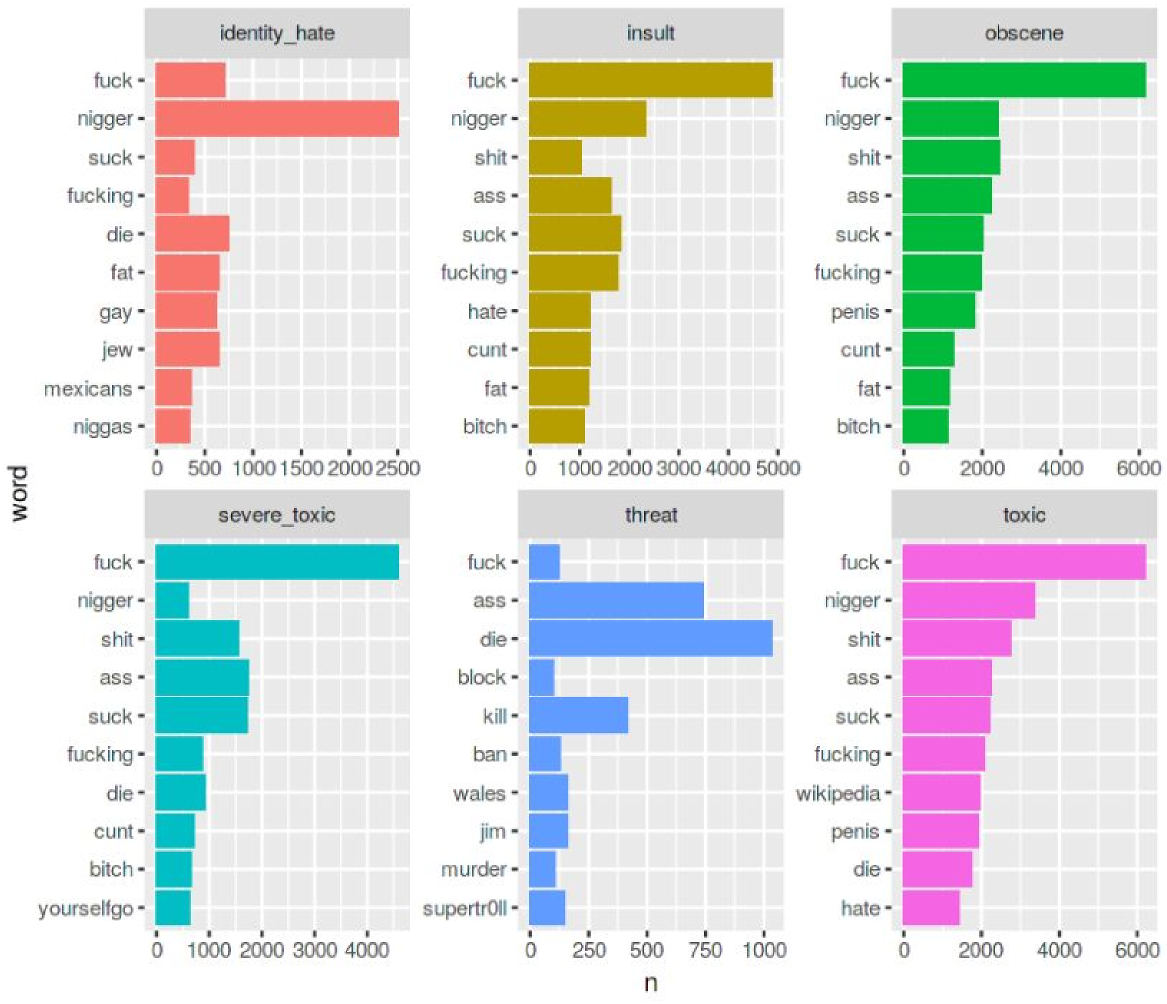
Important terms in the comments



```
most freq words in toxic
1 : fuck
2 : fucking
3 : shit
4 : like
5 : don
6 : ass
7 : bitch
8 : stupid
9 : suck
10 : wikipedia
most freq words in non toxic
1 : article
2 : talk
3 : page
4 : wikipedia
5 : just
6 : thanks
7 : like
8 : don
9 : edit
10 : think
```

We can see that frequent words in toxic and non toxic are quite different, I suppose that the representing words will play important roles when solving the problem.

I have found an interesting visualization in on the following web page: [7]



We can see that the presence of certain representing words may relatively affects the label. For example, “nigger” may lead to “identity_hate” and “die” may refer to “threat”.

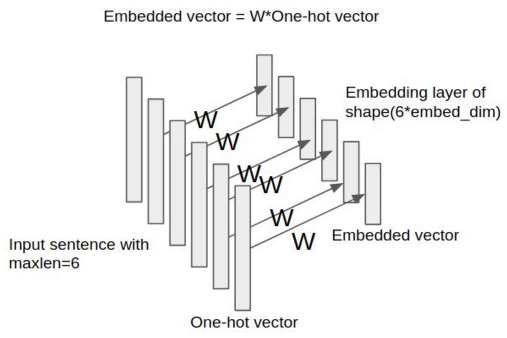
Algorithms and Techniques

The main algorithm I will use is CNN, instead of doing lots of feature engineering for the pattern of representative words I will extract, the CNN can do the job automatically. In addition, most of comments contain some special expressions, CNN can easily detect them using its convolution layers. Finally, compared to RNN: for tasks where the length of text is important, it makes sense to go with RNN variants. These types of tasks include question-answering, translation etc. For tasks where feature detection in text is more important, for example, searching for angry terms, sadness, abuses, named entities etc. CNN works better.[5]

Preprocessing:

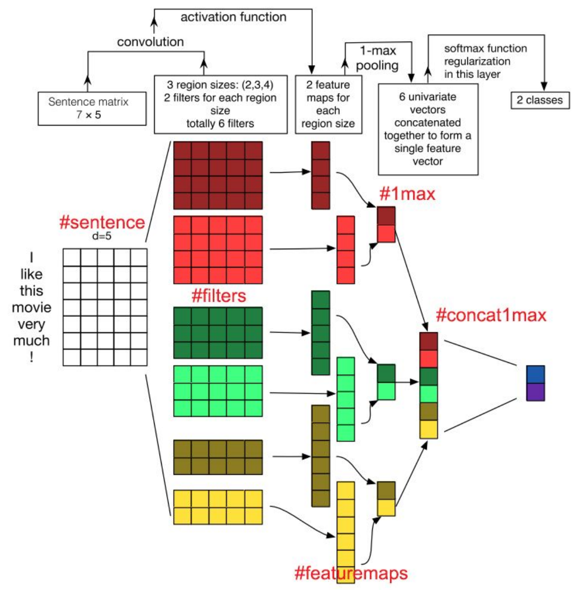
1. I will use tokenizers to extract words or n-grams under some specfication from comments, then build a vocabulary of these words by associating an unique index (from 0 to number of terms) to each of them.
2. I will transform each comment into a sequence of term ids by just replacing the term by its index. Then for each comment, I will cut it or fill it with a special index so that its length is equal to a constant for example 200. This means I only keep the first 200 terms for each comment in order to handle large comment with repeated expressions.

Word Embedding:


Embedded vector = W*One-hot vector
Embedding layer of shape(6*embed_dim)
W
W
W
W
W
W
Embedded vector
Input sentence with maxlen=6
One-hot vector

1. Each word index can be seen as a one-hot vector, the dimension of this vector is the number of total words which is large, the goal is to associate it with another full vector with less dimensions say “emb_dim”. To do so we can multiply the one-hot vector by a matrix of   “emb_dim*nb_word”. This matrix is the embedding matrix “W”. We can learn it by applying backpropagation to the CNN structure below.

CNN Architecture:


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
#1max
#sentence
d=5
I like this movie very much !
#filters
#concat1max
#featuremaps

I borrowed the image from image from [4], because this is exactly what I intend to implement for the first stage. To note that the input of the architecture of dimension 5 can be seen as a embedding layer, for my own project there are 6 output instead of 6. In addition to the architecture, I will add some dropouts on each layer for the first trial.

Learning

I will first use the entire train set to build the vocabulary, then transform the test and train set comment into a sequence of word index. Someone build the vocabulary by including the test set, I will not do this.
Then I will split the train.csv data set into training and validation set, then train the CNN model above using training set and test the resulted model on validation set. Of course I could also do this multiple times with cross validation to tune the hyper parameters.

Testing

Once a submission file created, I will upload it onto the Kaggle platform to get the testing result.

Benchmark

The benchmark model[3] handles the problem by the following steps:

Preprocessing

1. Extract words and sequence of characters of length ranging from 1 to 4 (1-4 grams)

2. Keep 30000 the most frequent words and 30000 the most frequent 1-4 grams, use both of them to represent each comments as a sparse vector separately. Here, a sublinear tf-idf transformation has been applied.
3. From 2, for each comment we got two tf-idf transformed representation vectors of dimension 30000, now we concatenate them together into a 60000 dimension representation vector.

See file “prepare_benchmodel.ipynb” for benchmark feature extraction implementation
The data preprocessing has been implemented by

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Here, as the original author did, we take maximum 30000 words and 30000 ngrams composed by 1-4 characters as vocabulary, and we transform each comments to a 60000 dimension vector with value of the tf-idf transformation.
The 60000 extracted feature has been stored at the following file:

"temp_res/train_features.npz"
"temp_res/test_features.npz"

Learning and prediction

1. Use the 60000 features for each comment above to learn a logistic regression model for each label.
2. Apply the tf-idf transformation based on the vocabulary to the test comments
3. Apply the 6 logistic regression model separately to predict each label.

The total parameters to train is around 60000*6 = 3.6e5. The model can be trained using SGD like solver, the training can be parallelized.

To notice that the original code use the training and testing set together to extract it’s 60000 features, this is a bad practice, although this might gain higher score on the kaggle platform.

I reimplemented the original code and I added a hyperparameter tuning function with cross validation. The code is in the file “benchmark_model.ipynb”. I have gained better ROC-AUC score comparing to the original one:
0.9790 public score compared to the 0.9788 public score of the original implementation.
So outperform 0.9790 public score will be the goal of this project.

The output submission file is “bench_submission.csv”

III. Methodology

(approx. 3-5 pages)

Data Preprocessing

Basic CNN:

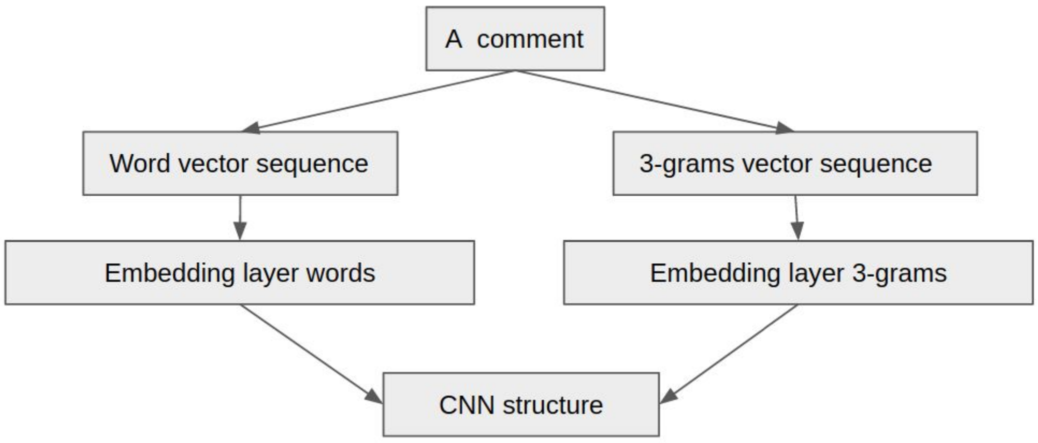
1. Keep 10000 words in total extracted from training set with keras's default tokenizer.

For each comments only keep the first 200 words. Why 200? We can see that 75% of document have less than 500 characters, 200 word is around 800 characters so most of documents can be completely covered, this can solve the abnormality observed from "Data Exploration" section that there are extremely long comments formed by repeated expression.

2. Embedded vector has dimension 300, convolutional layer has filters with size 2,3,4,5, this means that the conv layers can check 2,3,4 or 5 words togethers, to note that the filters size can be parameterized using grid search.
3. The output layer is a 6 unit dense layer using "sigmoid" activation function to approximate a probability. To note that we I won't use "softmax" layer because this is not a multi categories classification problem.
4. Gridsearch and cross validation have been applied for parameters tuning

Improvement:

I used the same methodology as the benchmark model does to extract words and n-grams. But instead of merge all of them together as a whole vector, I distinguished them into multiple independent inputs:



The preprocessing file for the improved CNN is "prepare_adv_cnn.ipynb".

Finally I did choose the 20000 most frequent words and 20000 most frequent 3-grams to represent one comment as two independent sequences. Compare to the 30000 words and 30000 1-3 grams that the benchmark model extracted, What I extracted is only a small subset.

Following the two sequences, I added two independant embedding layers, these two flows of information will be joined together at the end of the new CNN structure that we are going to present later.

To note that for my CNN model, I need the transformation to get a sequence of word_index for each comments but not the if_idf vector, so I defined the following class to do the job in file "prepare_adv_cnn.ipynb":

```
class SeqVectorizer(TfidfVectorizer):
  def transform2seq(self, raw_docs):
```

To notice that the indexing of TfidfVectorizer begin from 0, so the padding value could be "len(vocabulary)".

The output of this module is stored in the file:
"temp_res/train_mulinp250400.npy"
"temp_res/test_mulinp250400.npy"

Implementation

Basic CNN

See file "basic_cnn.ipynb"

1. Preprocessing
   The data preprocessing has been implemented by
   ```
   from keras.preprocessing import text.Tokenizer
   ```
   Which is the most simple one. To notice that the indexing of tokenizer begin from 1, so
The padding value could be 0.

2. CNN structure
   For CNN structure, I use the keras lib to define all layers:

   ```
   from keras.models import Model
   from keras.layers import Input, Embedding, Dense, Conv2D, MaxPool2D
   from keras.layers import Reshape, Flatten, Concatenate, Dropout, SpatialDropout1D,
   BatchNormalization
   ```
   The function that creates embedding layer and CNN is "create_cnn"
3. Learning
   I used sklearn to do grid search for parameters and cross validation:
   ```
   from sklearn.model_selection import GridSearchCV
   from sklearn.model_selection import ShuffleSplit
   ```

Improved CNN structure

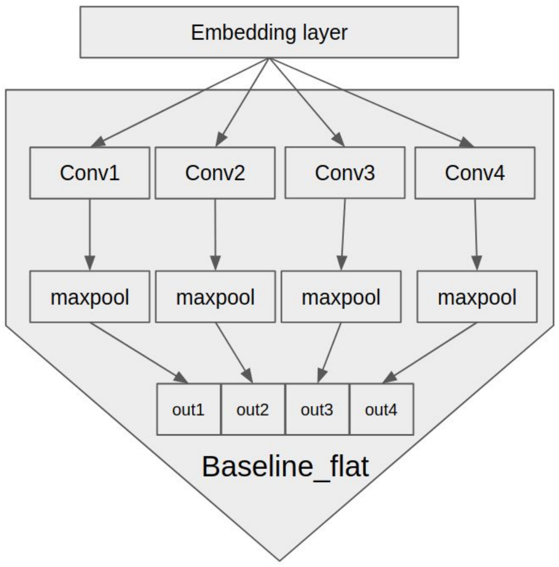
This part is implemented in "adv_cnn.ipynb". The input of this part is

“train_mulinp250400.npy”
“test_mulinp250400.npy”

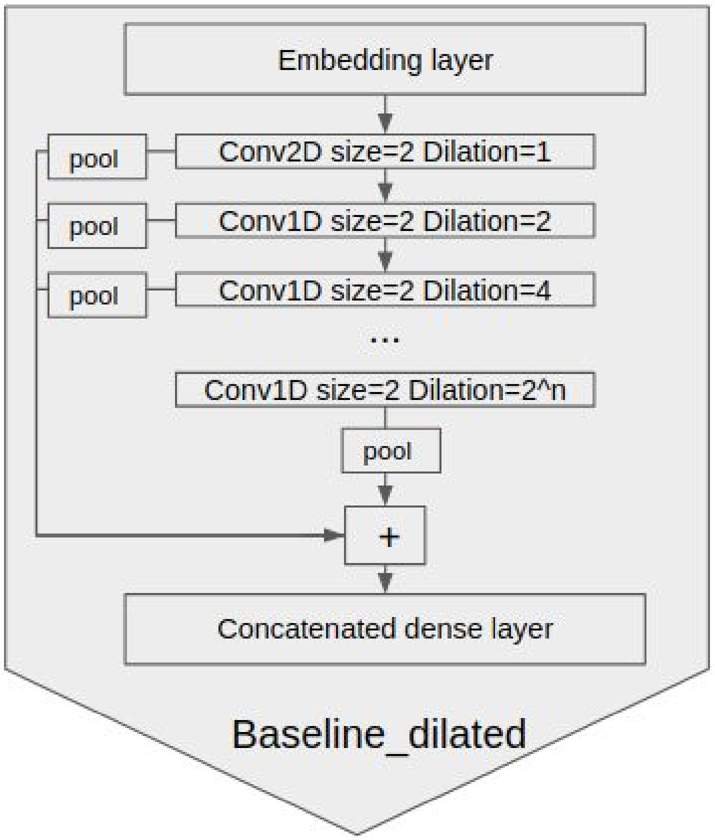
To notice that we need to first split the concatenated input sequence into 2 independent sequences.

Flat baseline:

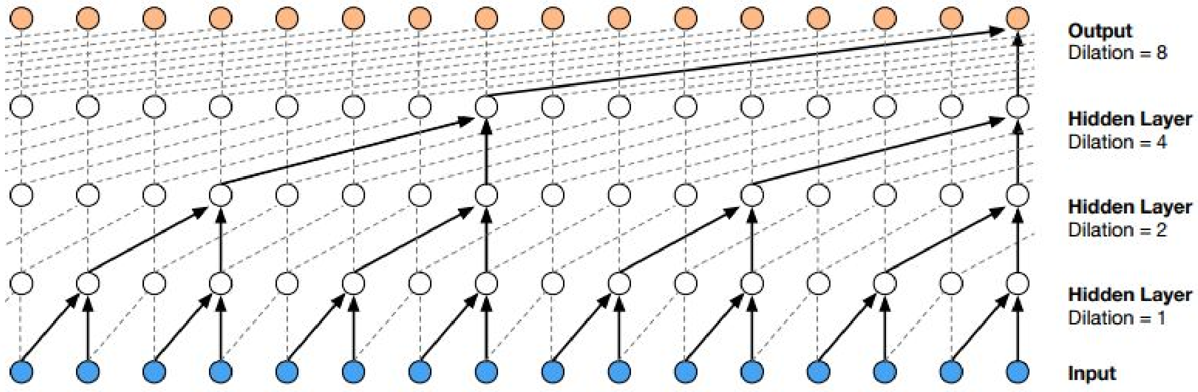
Shown as the figure below, the flat baseline is the same as the basic CNN model.



Dilated baseline:



Each filter in dilation layer has 2 input units from the previous layer, unless traditional convolutional layer, these 2 input units has a distance between them. The distance grow exponentially as the network grows deeper.

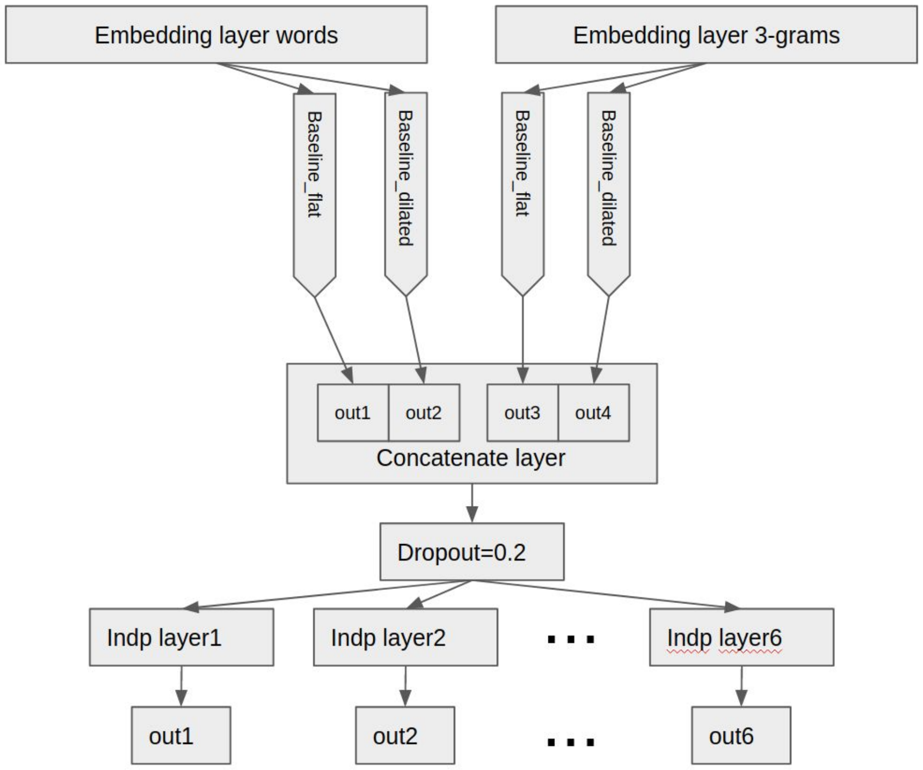


I borrowed the above picture from Sergei Turukin's blog[6].

We know that the natural language sentence can always be decomposed into a binary grammar tree. This inspired me that why not use a convolutional binary tree to recognize the hierarchical structure of the given sentence. Technically, the dilated convolution can easily grow the receptive field of each unit exponential to its depth.

The final output layer is a concatenation of all the global max pooling of each convolutional layer. The reason why I need all of them is that, firstly I don't know a priori the grammar depth of a arbitrary given sentence. Secondly I think the detail marked on each level is important to the classification problem.

Global CNN structure:



To notice that there are batch normalization after almost every layer, see that the neural net is quite deep, This can help for solving gradient explosion and gradient vanishing problem.

After concatenating outputs from different embedding layer and different baselines, I added a dropout layer to prevent overfitting. Thus, the share feature for all the 6 classification problem can be seen as the vote of 4 different classifiers

Finally instead of generating the 6 dimension output vector from one common layer, I added 6 independant layer so as to generate 6 outputs independently.

Refinement

Batch normalization, dropout, parameter tuning with cross validation are all implemented at the beginning for the basic CNN method. The improved CNN model can be seen as a refinement of the basic CNN model. The comparison of the result will be shown in the “Results” section.

IV. Results

Basic CNN

After applying the 3-fold Cross validation and parameter tuning, I found the best parameters with final score is:

batch size	128
epochs	3
filter sizes(size of filter for each conv layer)	1,2,3,5
num filters(number of filters for each conv layer)	50
CV scores	0.9812,0.9864,0.9841
AVG CV score	0.9839
Testing score (private, to check overfitting)	0.9743
Testing score (public, to be ranked)	0.9749

The submission file is “submission_basic_cnn.csv”

A score of around 98% means that if we randomly select a comment from non toxic comments and another comment from toxic comments, the later has 98% chance to get a greater predicted proba being toxic than the former.

This is already an acceptable result, although it can not outperform the benchmark model yet. Note that the best result on the leaderboard is around 98.6%.

Private and public testing score are computed from kaggle platform using different distribution of the given testing set in order to prevent test set overfitting. They are quite similar means that the model is stable.

For the sensibility of the model, I trained the model on 3 different part of training data by applying Cross Validation, the 3 score are closed to each other. So the model is quite robust and stable.

Advanced CNN

After applying the k-fold Cross validation and parameter tuning, I found the best parameters with final score is:

batch size	64
epochs	3
'Filter_nums': number of filters in dilated convolutional layers	[30, 50, 80, 100]
'Filter_sizes': size of filters in convolutional layer of the baseline flat	[1, 2, 3, 4, 5]
'Idpt_size': the size of the 6 final independant Dense layers	128
CV scores	0.9861 0.9831 0.9868 0.9851
AVG CV score	0.9858
Testing score (private, to check overfitting)	0.9806
Testing score (public, to be ranked)	0.9805

The submission file is “subm/submission_char_word_cnn_adv.csv”
The 4 validation scores are near to each other, so the model is quite robust. The private testing score and the public testing score are also closed, with the public score of 0.9805 higher than the benchmark public score of 0.9790. This means that the new CNN model is more stable and stronger than the benchmarked model.

I found that smaller batch size (64 versus 128) can help preventing overfitting. Bigger independent dense layer size(128 versus 64) at the end can also help augmenting the score.

Justification

The new CNN architecture takes only a small subset of words and n-grams from the benchmark model’s vocabulary. But finally, the new CNN architecture gets higher score and is more robust due to the fact that:

1. The new CNN architecture consider the order of each words or n-grams in the comment as a sequence.
2. The new CNN architecture can explore the hierarchical structure of the sentence.
3. The new CNN architecture can share the common useful information among all the 6 classification problems and learn the embedded representation of each words.

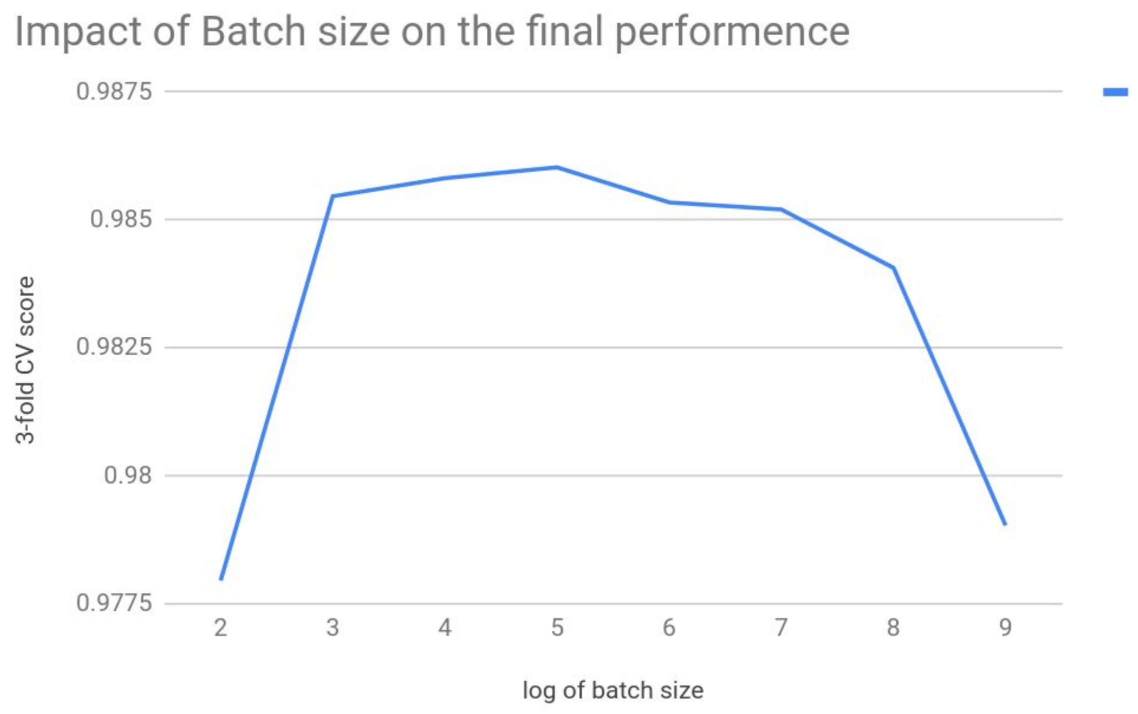
Furthermore, the private and public score of the new CNN model is not only higher but also more stable, because they are quite close to each other(0.9805 and 0.9806).

V. Conclusion

Free-Form Visualization

During the parameter tuning of the new CNN model, I noticed that by varying the batch size from 128 to 64 I have got a significant improvement on the final score although the training take a little bit longer.

This is why I tried different batch sizes in order to see the impact of batch size on the final performance of the CNN model for the toxic comment classification problem.



We can see that the CV score grows dramatically from 8 to 16 and decrease significantly from 128 to 512. A batch size of 32 yield best CV score. I stored the submission file with 32 as batch size into the file “subm/submission_char_word_cnn_adv_varing_batch.csv”, I get a private score of 0.9809 and a public score of 0.9818.

Reflection

First of all, I begin with reimplementing the benchmark model using k-fold cross validation and parameter tuning, I have got higher score than the initial benchmarked model.

After, I implemented a basic CNN based on 10000 words vocabulary and the first 200 words of each sentence. The flat baseline has gained good results capable of solving the problem.

Later, I improved the CNN model by adding a multi-task learning approach and a deep dilated convolutional mechanism.

Finally, by varying the batch size, I have got higher performance score when the batch size is adjusted to 32.

Interesting aspects:

1. The word embedding can be learned directly from the data
2. We can combine different outputs and different inputs together to train a single model with shared parameters to get better results than training them separately, this is the charm of multitask learning.
3. Dilated convolutional layer has similar binary tree form as the grammar tree of the natural language.
4. Batch size is very important for hyperparameters tuning.

Difficult aspects:

1. Make use of my GPU for the training of the CNN model. it’s quite difficult to handle the incompatibility problem between different package of different version.
2. Parameter tuning take long time.
3. Having spent long time on CNN structure conception.

The model can definitely solve the problem with a stable test result of around 98%. This can also be a general solution for this kind of problem, especially when the number of label about a comment grows, the learned embedding vector for each word will be more accurate to the word’s own meaning.

Improvement

Additionally, we can use transfer learning to initiate the embedding layer with word vector pretrained on other similar english text corpus using word2vec or GloVe.

Another possible approach is to use RNN combined with attention mechanism which is able to remember the meaning of a long read content and focus on important aspects that can help detecting toxic comment.

For the extremely long comments or for a toxic comment without remarkable expression that indicate the toxicity, the CNN approach might not be sufficient because it can not remember lots of past content as a vector and infer from them.

References:

[1] https://en.wikipedia.org/wiki/Talk:Deep_learning
[3] https://www.kaggle.com/thousandvoices/logistic-regression-with-words-and-char-n-grams
[4]http://www.joshuakim.io/understanding-how-convolutional-neural-network-cnn-perform-text-classification-with-word-embeddings/
[5]https://arxiv.org/pdf/1702.01923.pdf
[6] http://sergeiturukin.com/2017/03/02/wavenet.html
[7] https://www.kaggle.com/madcap/toxic-exploration