Machine Learning Engineer Nanodegree

Capstone Project

Jeff Gerlach July 25th, 2019

I. Definition

Project Overview

A major problem in online interactions (such as on message boards, comment threads, and video game chat) is toxic behavior. In this context, toxicity is defined as 'anything rude, disrespectful, or otherwise likely to make someone leave a discussion' [1], which with the anonymity of the internet can be quite common on popular social media platforms, games, and websites. While this can be manually moderated on small scales - say a blog author monitoring a few comments posted regarding an article they wrote - today's large social platforms necessitate automated detection of such behaviors due to the large volume of user-generated content and the desire of platform owners to keep such hostility at bay.

At the close of 2017, a large (around 2 million) comment database was released by the Civil Comments platform [2] when it shut down in an effort to help researchers improve civility in online conversations. Jigsaw, an Alphabet company, adopted this goal and further annotated the data set for research purposes. They also sponsored a Kaggle competition using this data.

This capstone project will focus on the Kaggle competition 'Toxic Comment Classification Challenge', which aims to detect different types of toxicity across a diverse range of conversations from this dataset. This will be achieved by building a machine learning model to determine the probability of a comment from a large test Wikipedia comment dataset as being labeled with one of six different types of toxicity that were manually labeled by human reviewers.

Problem Statement

The goal of this Kaggle competition (and thus this capstone) is to build a machine learning model that can classify toxicity types using machine learning methods. The model must predict the probabilities of a comment being labeled by a human reviewer as one of six types of toxicity for each comment in the test dataset.

Previous research has been performed developing machine learning models to classify toxic interactions [3], and the goal of this Kaggle compeition was to open this particular dataset to the community to see if more accurate

models[4] were possible. This is a multi-label classification problem, as each comment must recieve six probability outputs (one for each type of toxicity), with a value of 0.5 or higher corresponding to a positive label for the corresponding toxicity class.

The classes are as follows:

- `toxic`
- `severe_toxic`
- `threat`
- `obscene`
- `insult`
- `identity_hate`

My attempt at solving this problem will involve experimenting with a recent (relatively speaking) neural network architecture known as a Capsule Neural Network [5]. The first steps will involve exploring the training data for potential features and possible pitfalls, and then developing a baseline benchmark logistic regression model to classify the test set. The next goal will be to develop and refine a Capsule Neural Network and attempt to achieve better performance than the baseline model, and then see how it compares to the top leaderboard scores for this Kaggle competition and see if it is capable of detecting various types of toxicity in online comments.

Metrics

The evaluation for this project will be based off of the metric used by the Kaggle competition: the combined mean ROC AUC of each column (area under the curve of the receiver operating characteristic for each toxicity classification type) - this was introduced towards the end of the of original Kaggle challenge due to changes in the dataset [6].

ROC curves are graphs that show classification performance of a model, and visual two parameters - true positive rate (TPR) and false positive rate (FPR) for varying classification thresholds. Computing this curve at all thresholds would not be efficient, so area under the ROC curve (AUC) is used instead. AUC is an aggregate measure of performance at all thresholds, with a value of 1.0 for 100% correct predictions[7] and a AUC value of 0.5 corresponding to random guessing[8].

The goal of this competition was to achieve the highest averaged ROC AUC value between all toxicity classification types with a machine learning model (with a perfect model having a score of 1.0). This was a deemed a fairer metric for model performance comparisons by the competition sponsors due to the class imbalance in the dataset. Sci-kit Learn's `roc_auc_score` method will be used to calculate the model scores.

II. Analysis

Data Exploration

The *Toxic Comment Classification Challenge* Kaggle competition initially provided a training set to be used for developing and training a machine learning model and a test set was provided to generate predictions for submission onto the leaderboard. As this competition had completed at the time of working on this capstone project, a set of ground truth test labels was provided as well. Here we see some summary facts regarding the datasets, which are provided in 2 CSV files:

Dataset	Rows	Columns
train	159,571	8
test (initial)	153,164	2
test (actual)	63,978	2

Table 1: Dataset dimensions

In the file description, it is mentioned that only a portion of the test data is used for final scoring in the competition - in the provided ground truth test labels, the unused rows are labeled with `-1` values for each of the six classification types. This leaves the final test dataset at 63,978 examples, leaving a 71% training to 29% testing data ratio, which seems to be an ordinary split.

The datasets include an `id` column and a column for `comment_text` which are the Wikipedia comments that will be the source of features for all models. There are six target columns in the training dataset (and also the provided test ground truth labels) representing each of the toxicity categories that were hand-labeled by human reviewers - it is possible for a comment to have more than one label (a `1` in a given column) or no labels at all (`0` in each column). There are no repeated `id` columns, nor any `n/a` values that need to be cleaned from the training dataset.

Again, the comment toxicity labels are as follows:

- `toxic`
- `severe_toxic`
- `threat`
- `obscene`
- `insult`
- `identity_hate`

Below is a listing of some sample rows from the training dataset:

	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
5	00025465d4725e87	"\n\nCongratulations from me as well, use the ...	0	0	0	0	0	0
6	0002bcb3da6cb337	COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK	1	1	1	0	1	0
7	00031b1e95af7921	Your vandalism to the Matt Shirvington article...	0	0	0	0	0	0
8	00037261f536c51d	Sorry if the word 'nonsense' was offensive to ...	0	0	0	0	0	0
9	00040093b2687caa	alignment on this subject and which are contra...	0	0	0	0	0	0

Fig. 1: Sample rows from training dataset

And some summary statistics using Pandas:

	toxic	severe_toxic	obscene	threat	insult	identity_hate
count	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000	159571.000000
mean	0.095844	0.009996	0.052948	0.002996	0.049364	0.008805
std	0.294379	0.099477	0.223931	0.054650	0.216627	0.093420
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Fig. 2: Summary statistics from training dataset

Below we see that nearly 90% of the comments do not have a toxicity type label, and the ones that do are not evenly distributed, with the `toxic` category dominating nearly half the positive labels. While not as bad as say the perennial spam dataset example where 1% of the data is classied as spam and the rest is not spam, care will need to be taken when selecting a validation set to match the positive and negative toxicity distribution along with the toxicity subtype distributions as closely as possible to avoid skewing results.

```
Number of training examples 159571
Number of test examples 63978
Train/Test Ratio: 71.38% / 28.62%
```

Fig. 3: Train/Test Data Ratio

```
Comments with no toxicity: 143346
% of total training dataset marked as not toxic: 0.8983211235124177
```

Fig. 4: Percent of comments with no toxicity label - high imbalance

```
toxic             15294.0
severe_toxic       1595.0
obscene            8449.0
threat              478.0
insult             7877.0
identity_hate      1405.0
none             143346.0
dtype: float64
```

Fig. 5: Distribution of classification labels - again, showing high imbalance

Exploratory Visualization

As seen in the previous section, the most noticeable issue with the training dataset is the class imbalances between all the classes.

First we see provide a bar plot to drive home just how imbalanced the data is in regards to positive and negative toxic labels. The `none` category had to be calculated separately as it did not exist in the training set.

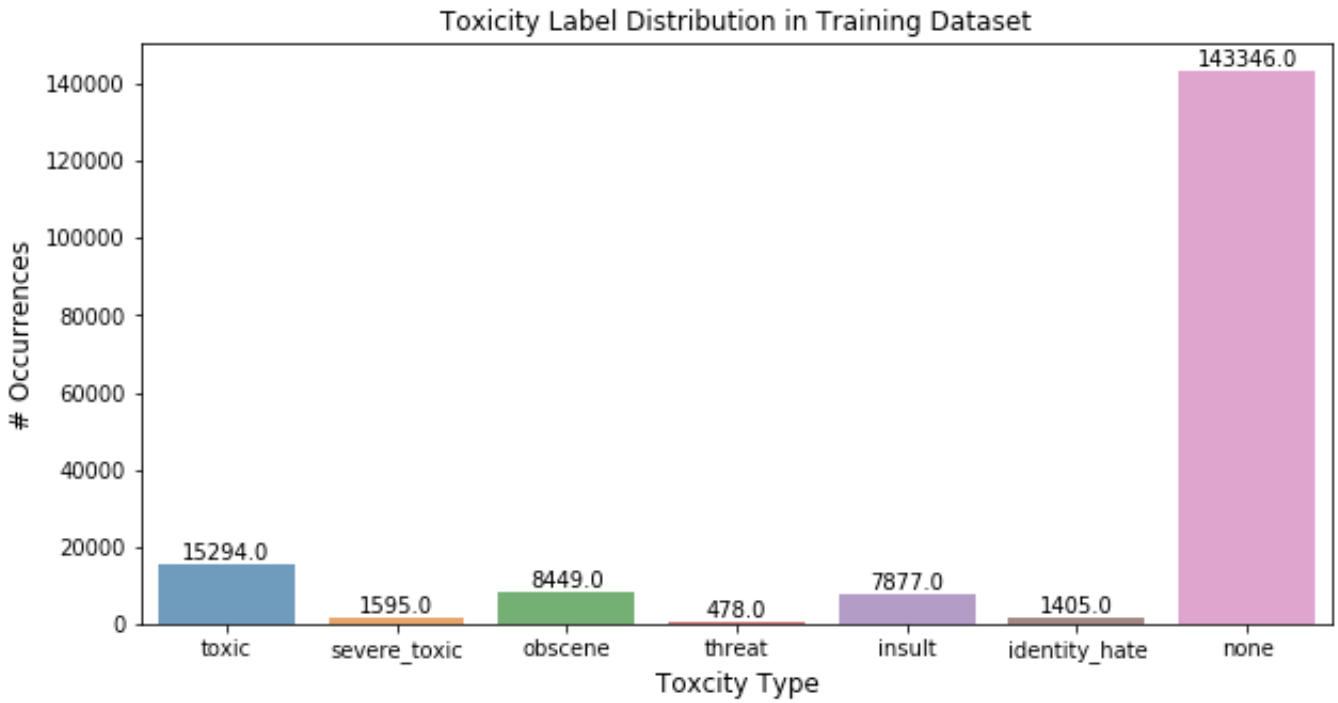

Fig. 6: Bar plot showing distribution of classification labels

To further break down the imbalance in the data, we visualize the distribution of positive labels - again, some comments can have multiple labels.

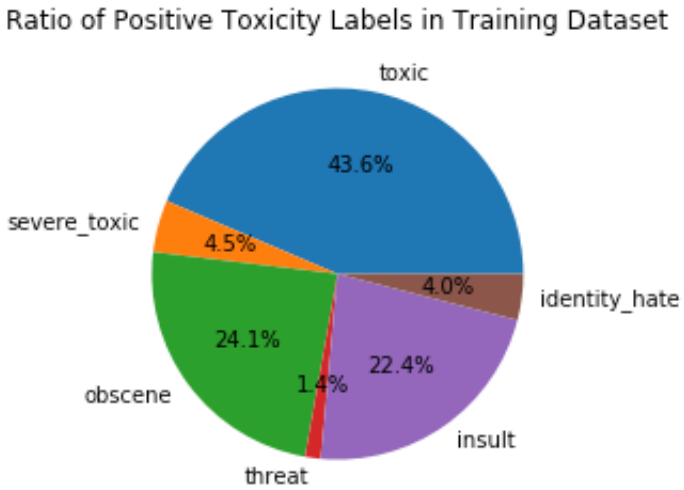

Fig. 7: Pie chart showing distribution of positive toxicity labels

Here we can see `toxic` accounting for nearly half the positive labels, with `obscene` and `insult` overshadowing the last 3 categories. One way of dealing with this imbalance could be to use stratified sampling while splitting this dataset into training and validation sets for the models.

Algorithms and Techniques

The approach I initially intended to take on this project was to use a Capsule Net (CapsNet) with a Gated Recurrent Unit (GRU) neural network architecture. The CapsNet was originally developed by Geoffrey Hinton et. al. [9] for computer vision purposes, and getting rid of the pooling layer being successfully used in Convolutional neural networks (CNNs) due to the loss of information this entails. The original goal was to be able to capture object pose information in a neural network model, and the CapsNet achieves this by adding "capsules" which re-use output from one another in a hierarchal fashion to form more stable representations as they are passed up the chain. This is thought to more closely mimic brain organization in biology [10].

While the CapsNet was originally developed for computer vision applications, it has been found to have success in other areas, and contributors to this Kaggle competition were able to achieve top-performing results using this architecture. I figured this was an interesting model to implment and learn from, and so initially chose to use it by referencing some implementations that were submitting on the Kaggle competition platform [11].

In order to extract features to feed into the models used in this project, the comments provided in the datasets need to be translated into a format that is expected by the models. This will be accomplished using a smaller TFIDF vectorization using the existing word corpus for the benchmark, and using pretrained word embeddings (a form of transfer learning) from Facebook's 2 million 300-dimensional fastText word vectors for the neural network models [12]. These dense word embeddings put words with similar meanings nearby one another in vector space, and have much larger coverage than could be achieved generating features from the dataset comment corpus.

I also desired to experiment with model ensembling, and so planned to develop a third (Pooled GRU) neural network model - in addition to a "simple" logistic regression benchmark model - and combine the outputs together to ideally create a more accurate model with a higher resulting score [13]. The input features would be extracted from the dataset comments and fed into the models which would produce 6 classification probabilities for each label.

All of these techniques and algorithms are implemented using Jupyter notebooks, heavily leaning on the SciKit Learn and Keras libraries to implement the machine learning models. Keras was used due to the smaller size of these datasets, and training was done locally.

Benchmark

The benchmark model used for this project will be a straightforward logistic regression model based off of a Kaggle submmission as a reference for data formatting and submission generation and scoring [14]. Data preprocessing (i.e. feature generation using vectorization) was done using default English language stopwords, stripping accentsfrom characters, and 1-grams and the top 10,000 words and 50,000 2-gram to 6-gram

character splits for this benchmark.

After a very short run time, it can be seen that with the appropriate features, even a basic model can achieve very good performance - a `0.9765` ROC AUC score averaged fromeach of the classification label AUCs on the test set. This seems to be a good benchmark, as it is a tried and true, easy to follow model and implementation that runs quickly and is easy to experiment with - almost all of the functionality is baked right into SciKit Learn, and performance is already within 1% of the top submissions.

III. Methodology

Data Preprocessing

Comment vectorization was accomplished using vectorization; Scikit-Learn's `TfidVectorizer` was used for the benchmark model using the tokenized word and character corpus from both the training and test sets. This approach should not cause leakage between datasets due to it only being used for feature generation. For the two neural networks, pre-trained word embeddings are used to provide a much higher fidelity model for word relationships. For both approaches, the comments from the dataset have to be tokenized and split into words. The Keras `text.Tokenizer` object is used here to split out the words and convert them to lowercase and strip out punctuation, with a maximum of 100,000 words, which are converted to integers. Then the training and test comments are converted to sequences and then either truncated or padded with zeroes to a length of 200 elements.

Each tokenized sequence of words is then looked up in the pre-trained fastText model and the corresponding 300 dimension vector is inserted into an embedding matrix which will then be fed as an Embedding layer in the Keras neural network models. Thus a vector of length 200 (tokenized comment) will be converted into a 200 x 300 matrix when exiting the embedding layer allowing word relationships to be used as features.

Implementation

After using Sci-kit Learn to implement the benchmark model, I switched over to using Keras to build the neural network architectures. I decided that because the given datasets for this competition were small enough, I did not necessarily require GPU processing to train the models in a reasonable amount of time, so Keras would suffice. There were also plenty of example models submitted on Kaggle using Keras to reference while developing and tweaking my CapsNet and Pooling GRU models.

The figure below lists the layers and input and output dimensions in the CapsNet architecture, which was fairly straightforward to implement with Keras. I referenced the setup in [15] which was based on a Github repository implementation which attempted to recreate the setup from Hinton et. al.'s paper [16]. First the tokenized input

(normalized to 200 elements) is fed into the Embedding layer, which either adds in the 300-dimensional fastText word embeddings or initializes the row with zeros if the word is not found. This is followed by spatial dropout (with a rate of 0.25) and bidirectional GRU (with lenghth of 128) layers which feed into the Capsule layer (10 capsules with 16 dimensions each). This output is then flattened and a final dropout layer (rate of 0.28) is added before using a Dense layer to output the final 6 probabilities of each toxic classification label for the comment.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         (None, 200)               0
_________________________________________________________________
embedding_4 (Embedding)      (None, 200, 300)          30000000
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 200, 300)          0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 200, 256)          329472
_________________________________________________________________
capsule_1 (Capsule)          (None, 10, 16)            40960
_________________________________________________________________
flatten_1 (Flatten)          (None, 160)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 160)               0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 966
=================================================================
Total params: 30,371,398
Trainable params: 371,398
Non-trainable params: 30,000,000
_________________________________________________________________
```

Fig. 8: CapsNet Architecture

The model is trained with the binary cross entropy loss function and Adam optimizer working to optimize the ROC AUC for each classification label. I trained this CapsNet for 3 epochs. I performed several runs of this model while varying the parameters to get the highest score possible on the test set, which will be described in the Refinement section below.

The Pooled GRU neural network I used for ensembling was derived from another high-performing model submitted in one of the Kaggle kernels[17] and was also developed in Keras. This model was mainly just to get another input for ensembling, and so only minimal refinements were made to it. The main factor to note is that this model uses pooling, while the CapsNet does not. The architecture is detailed in the figure below:

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_5 (InputLayer)         (None, 200)               0
__________________________________________________________________________________________________
embedding_5 (Embedding)      (None, 200, 300)          30000000    input_5[0][0]
__________________________________________________________________________________________________
spatial_dropout1d_2 (SpatialDro (None, 200, 300)       0           embedding_5[0][0]
__________________________________________________________________________________________________
bidirectional_2 (Bidirectional) (None, 200, 256)       329472      spatial_dropout1d_2[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)            (None, 198, 64)           49216       bidirectional_2[0][0]
__________________________________________________________________________________________________
global_average_pooling1d_1 (Glo (None, 64)             0           conv1d_1[0][0]
__________________________________________________________________________________________________
global_max_pooling1d_1 (GlobalM (None, 64)             0           conv1d_1[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)  (None, 128)               0           global_average_pooling1d_1[0][0]
                                                                   global_max_pooling1d_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)              (None, 6)                 774         concatenate_1[0][0]
==================================================================================================
Total params: 30,379,462
Trainable params: 379,462
Non-trainable params: 30,000,000
__________________________________________________________________________________________________
```

Fig. 9: Pooled GRU Architecture

This model was run for 3 epochs as well and used the same training settings as the CapsNet implementation.

Finally the outputs from the 3 different models - logistic regression, Capsule Network, and Pooled GRU were combined with an ensembling method, reminiscent of the random forest classifier approach. This allows for several lower-performing models to combined their results together and achieve higher accuracies than any one model on its own. In most cases, the more dissimilar models are (technically the less correlated the model results are), the better the ensembled performance tends to be. Most of the top Kaggle submissions involve ensembles of dozens (or even more) models, which is not very production-friendly, but allows you to win competitions with those few extra decimal places of accuracy/score. In my case a simple average of the 3 sets of predicted probabilities for each of the six toxicity classes did in fact result in increased test set scores, so I did not use anything more complicated such as blending. Results are detailed in a following section.

Refinement

The models went through several iterations before arriving at the final CSV file generated for submission. The initial benchmark model at first used just 1-gram word splits resulting in a test set score of 0.965, but after some experimentation and research, I found that adding in 2- to 6-gram character split vectorization (using the Scikit-Learn `TfidfVectorizor` method, an easy addition) resulted in nearly a percent jump in test set score of 0.9765. This was just the benchark model, so I did not spend much time investigating further improvements.

For the CapsNet model, I first started by experimenting with varying parameter values in the network (number of capsules and their dimensions, gru length, input vector length, and dropout rates), but only noticed very very small test score changes. So instead I took a step back and looked for any particularities in the data and ways to minimize their impact. The most obvious issue was the large class imbalance in the training dataset as noted in the data exploration. After doing some research, I found that stratified sampling could help improve training, and one library that is derived from sci-kit learn is `scikit-multilearn` which provides an `iterative_train_test_split` method[18] which attempts to keep the class distributions equal between training and validation sets. This one change increased the test set scores for both the CapsNet and GRU models a whole percentage point (from ~0.96 to 0.97), so definitely had an impact on model performance.

The next big refinement to the models involved switching from the 840 billion, 300-dimensional GloVe word embedding[19] to Facebook's 2 million word, 300-dimension fastText word embedding. It is hard to say just why this increased the test set scores (yet another whole percentage point from around 0.97 to slightly over 0.98), perhaps the word relationships in this model are more relevant to the training and test sets and detecting toxicity in this competition. Luckily the code was encapsulated enough that this change was as simple as just pointing to the fastText data file instead of the GloVe file.

I did not originally include the Pooled GRU model or ensembling when I first was developing the CapsNet for this project, but after reviewing several high-performing kernels on Kaggle for this competition, I determined that in order to push my final submission score on the test set even higher, this would be a straightforward approach to take with a high change of success. After researching several ensembling methods, I decided to try the approach of averaging the predicted probabilities for each classification label from from both the logistic regression model and the CapsNet model. This did indeed result in a slightly higher test score, and at this point I decided to implement the Pooled GRU model to see if I could average it's predicted outputs into the ensemble and raise the final submission test score even higher. After implmenting the baseline Pooled GRU and incorporating it with the ensemble, I immediately saw a 0.001 increase in test score, which at this level (0.984) was quite respectable.

IV. Results

Model Evaluation and Validation

My final model used in this report is an ensemble of a logistic regression model, a Capsule Network model, and a Pooled GRU neural network with their outputs averaged together to produce the final predicted probabilities for each classification label. The models were trained with 90% of the training dataset provided, and validated using the remaining 10% of the training set. Stratified sampling was used to ensure that each of the highly imbalanced classes were evenly spread among these training and validtion sets. Once training was complete with the 90% set and model was settled upon, the full training dataset was used to train the final model in each

of the 3 cases. None of the test data was used to train the models, so the ensemble should be generalized and not be overfit to the training data according to the final test score on the unseen test dataset (which now happens to have truth labels available for to provide a quicker feedback loop for model development). The only thing to keep in mind is that the word corpus used both training and test data to generate the features that are used by the models, so that would need to be account for when presented with completely unseen data.

The final ensembled model results produced a test score of 0.984156, which places it in the top third of submissions on the Kaggle leaderboard, with the top score being 0.98856, so less than half a percent better ROC AUC score - a lot of effort and optimization is required to get those last few points, and these approaches oftentimes are not very production-friendly.

Justification

The final results of each of the 3 models (including the benchmark logistic regression model) and the final ensembled model are presented below:

model	Test set ROC AUC score
Logistic Regression (benchmark)	0.9765
CapsuleNet	0.9815
Pooled GRU	0.9829
Ensemble	0.9842
Best on Kaggle	0.98856

Table 2: Final Test Set Scores

It can be seen that while each of the models perform very well on their own, when combined they achieve an even better result, which highlights the appeal of ensembles of models, and in this case it is computationally trivial to achieve (after training each model that is). The final scoring metric, the averaged ROC AUC for each toxicity label as described previously, is a good generalized measure of multi-class classifiers across a range of threshold values taking into account False and True Positive rates. In fact, this competition was updated partway through to make the average ROC AUC the metric used to score submissions.

As these test scores are very close to 1.0 (a perfect model) and my solution is within 0.0044 of the top submission (and much higher than a score of 0.5 achieved from random guessing), I am confident in stating that the final model has in fact solved the problem at hand.

V. Conclusion

Free-Form Visualization

As the final test score for the model only provides an averaged ROC AUC for all six labels in the datasets, I have generated the ROC curves for the final model for each of these labels. The diagonal line represents a score of 0.5 (random guessing), and the closer the curve is to the top-left corner, than higher the AUC is for the model classifying that particular label.

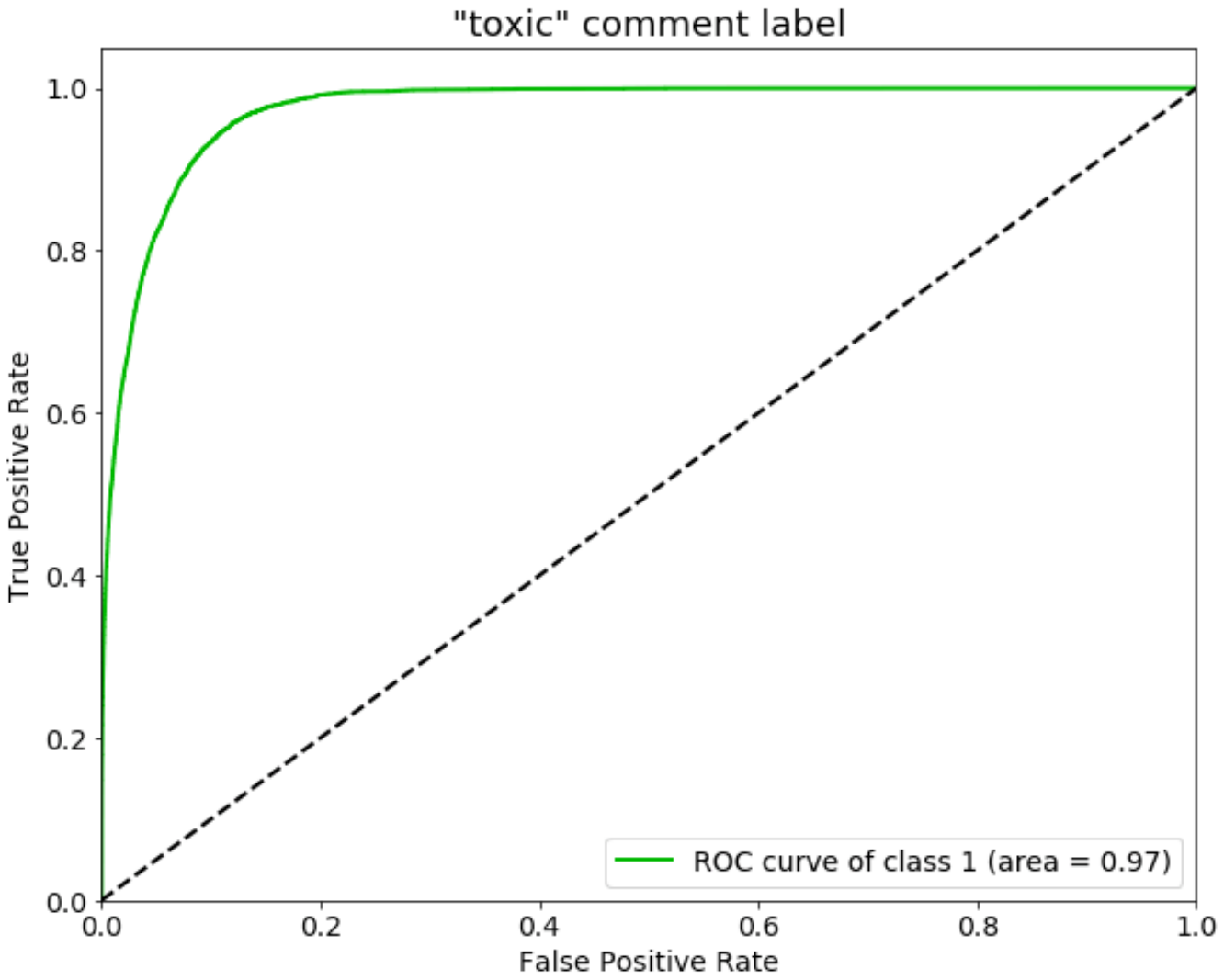

Fig. 10: `toxic` *ROC Curve plot*

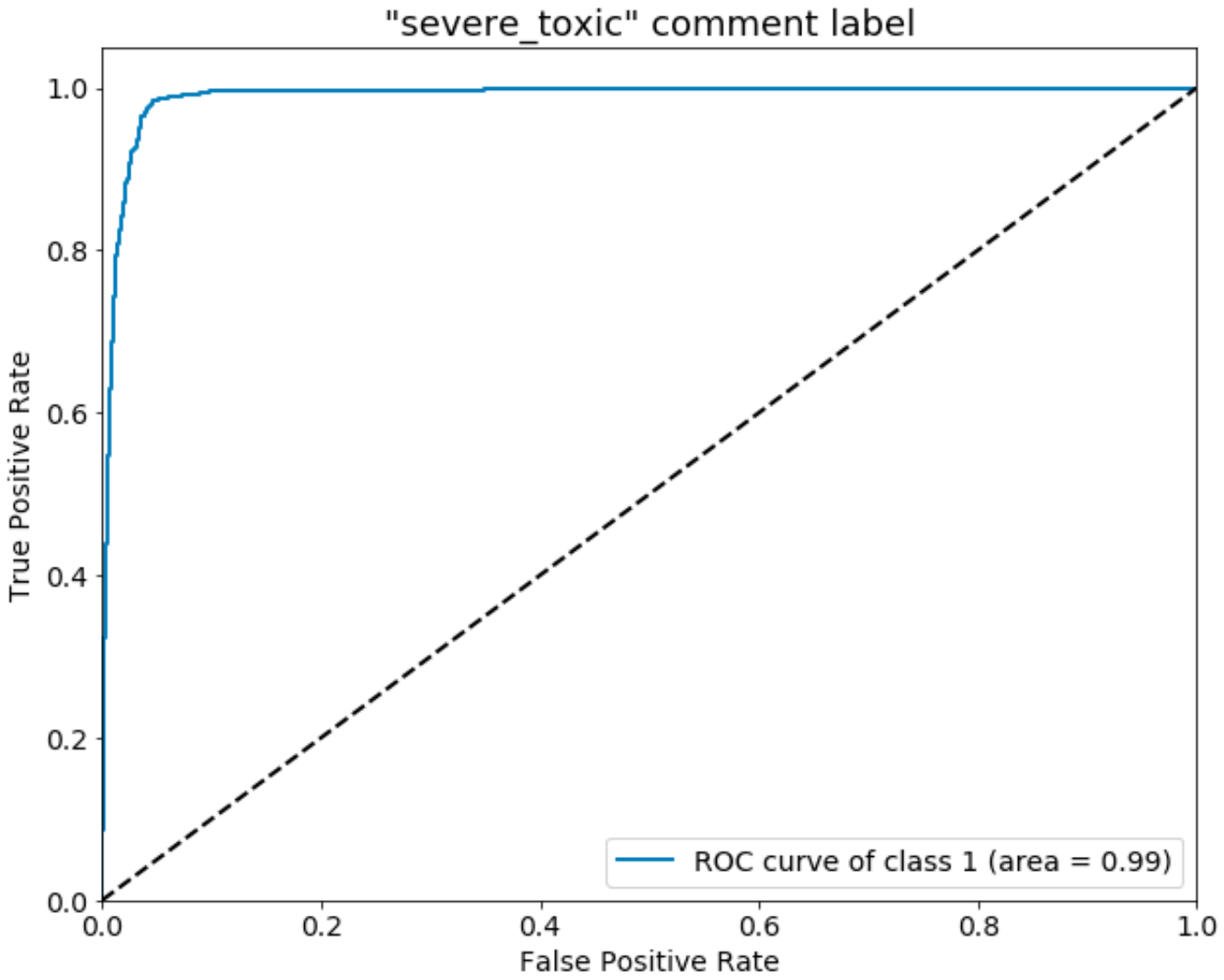

Fig. 11: `severe_toxic` *ROC Curve plot*

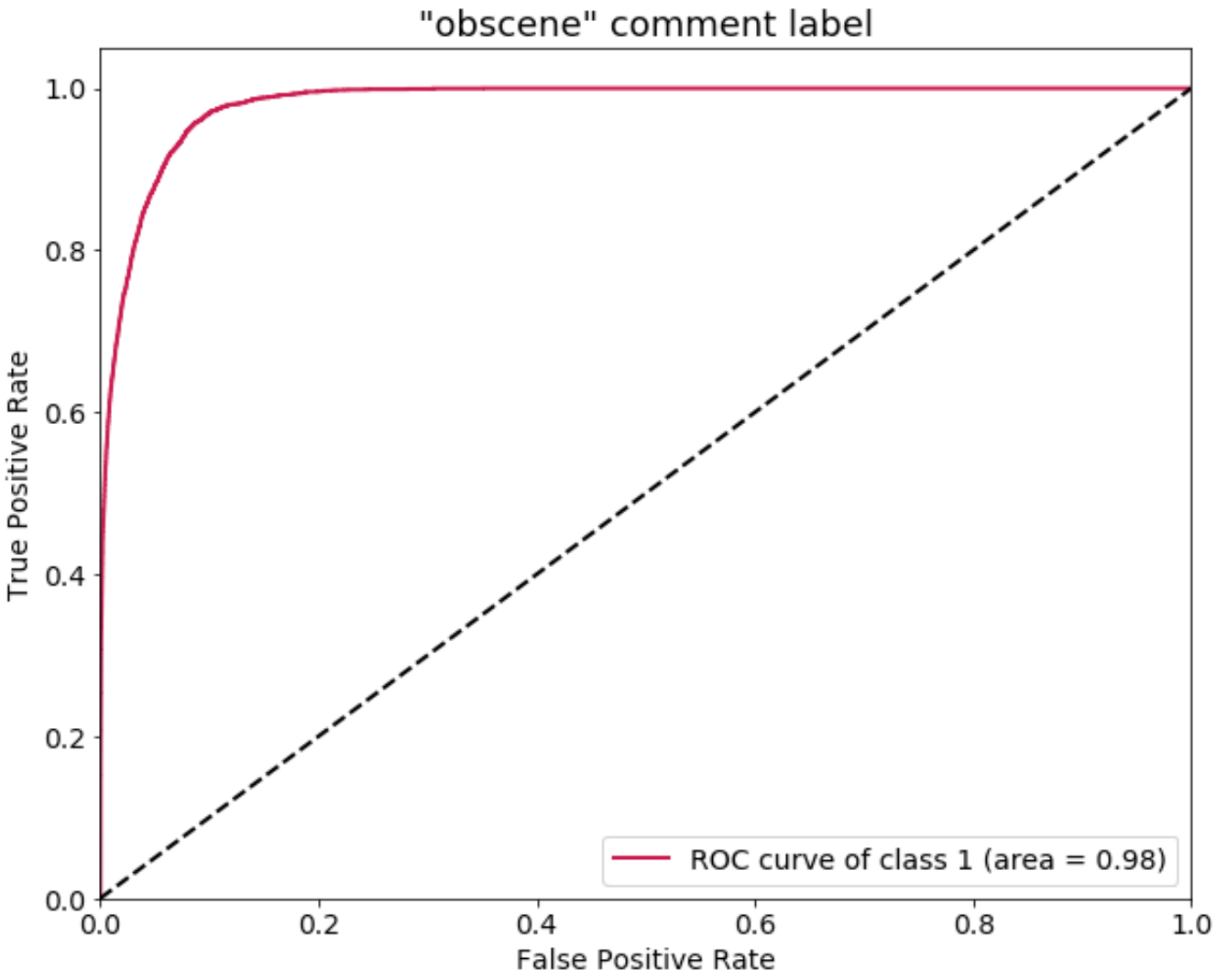

Fig. 12: `obscene` *ROC Curve plot*

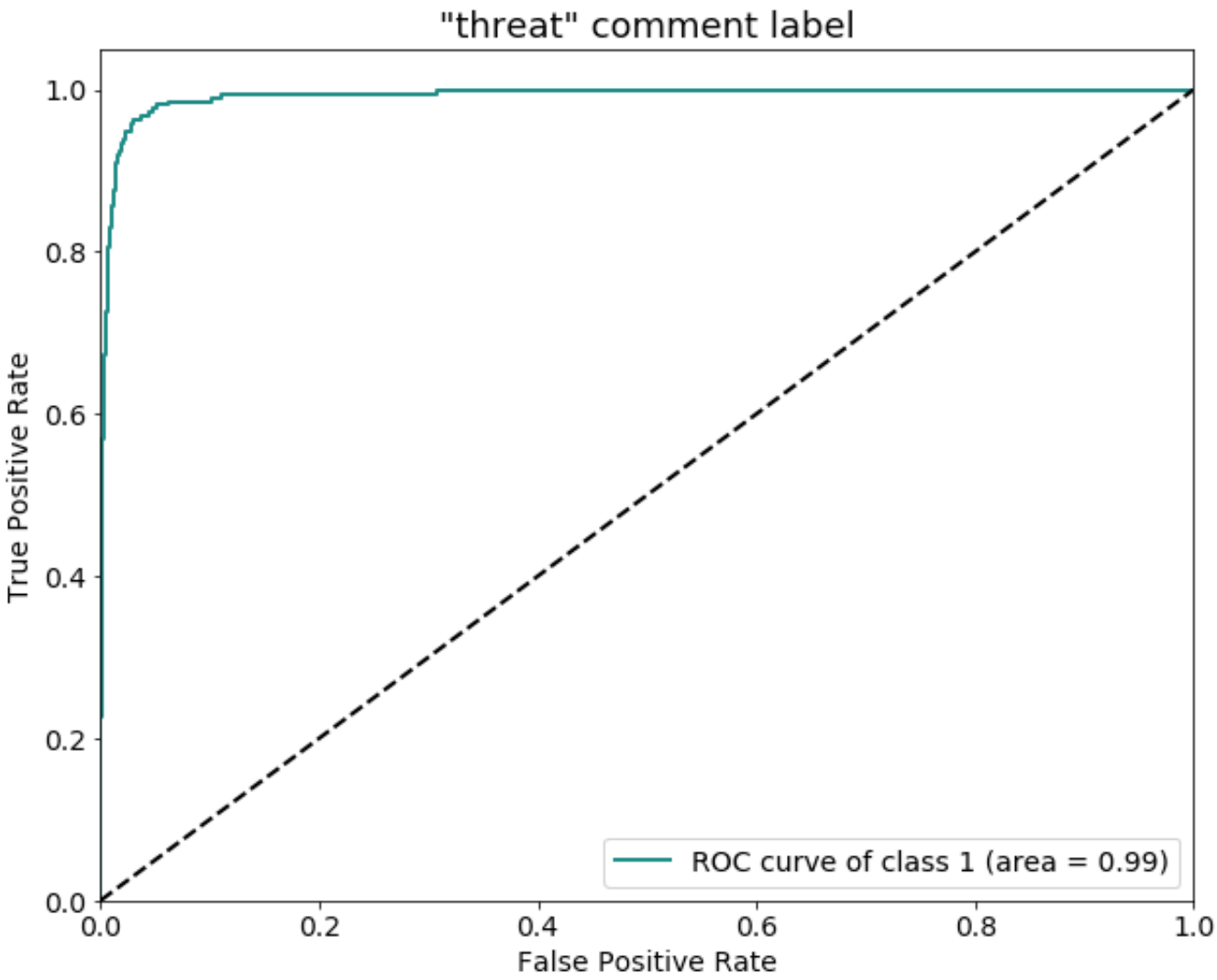

Fig. 13: `threat` *ROC Curve plot*

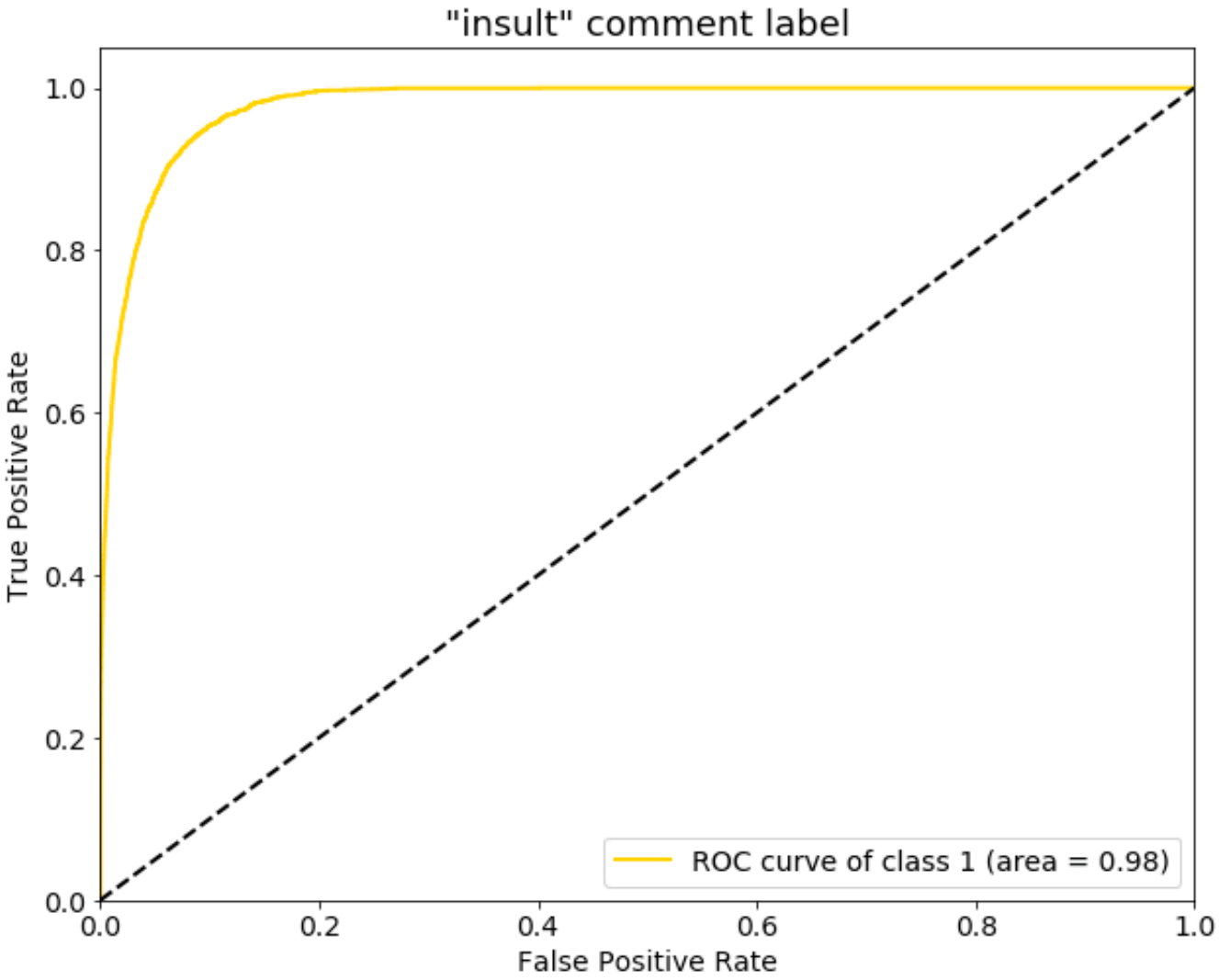

Fig. 14: `insult` *ROC Curve plot*

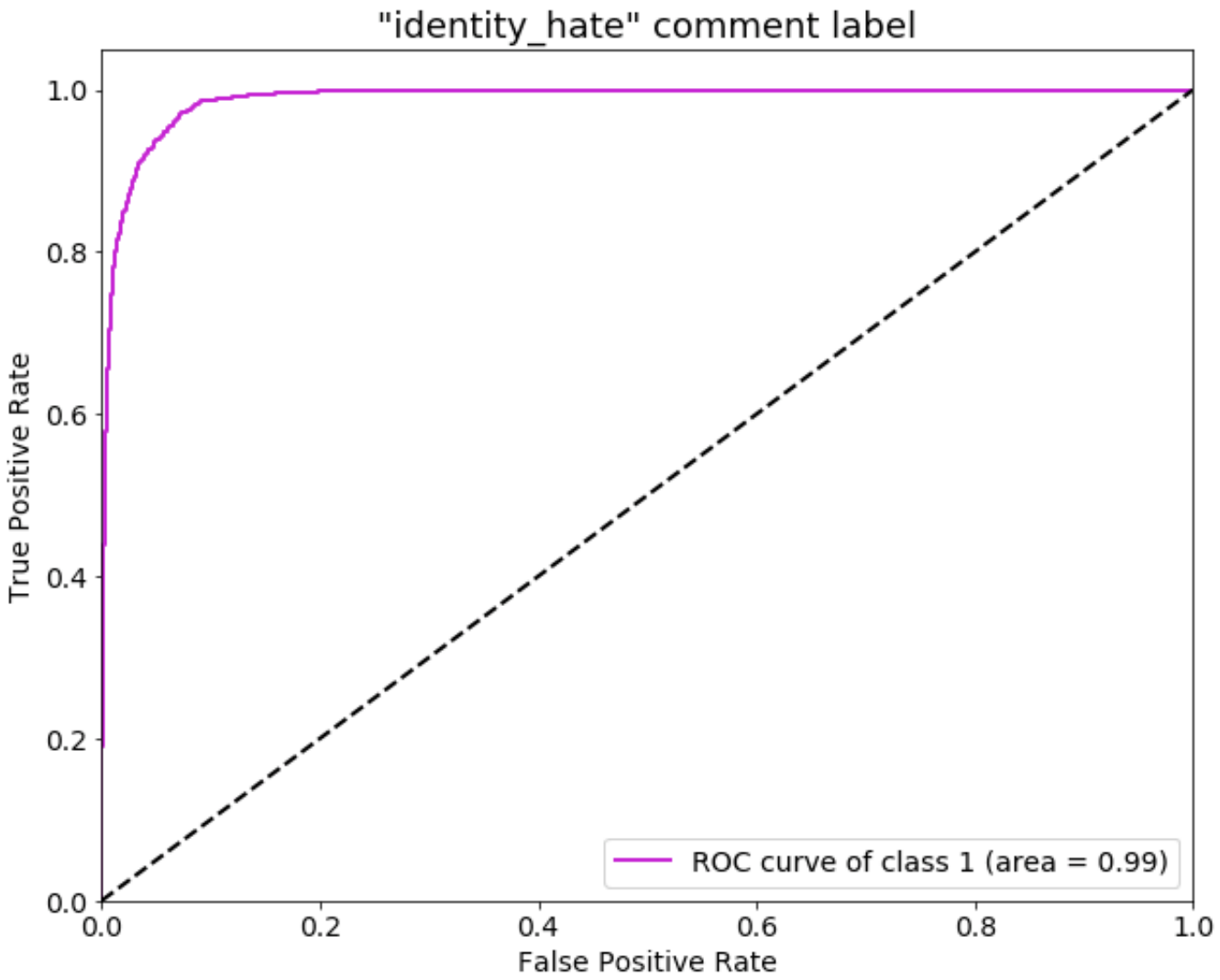

Fig. 15: `identity_hate` *ROC Curve plot*

Seeing each individual AOC, we can see that the model appears to classify `severe_toxic`, `threat`, and `identity_hate` best according to this metric.

Reflection

In this capstone project, I investigated the requirements for a Kaggle competition, did exploratory analysis of provided datasets, developed a 'simple' benchmark model using non-computationally intensive (i.e. 'traditional') methods, and developed and trained a neural network using a state-of-the-art architecture from a recently released paper. I made improvements to this model to increase the scoring metric results on the test data used for the competition, which ultimately involved developing a third neural network model (Pooled GRU network) and then ensembling the classification predicted probabilities from each model together into a final result. I also utilized transfer learning by experimenting with and using the GloVe and fastText word embedding models to

feed into the neural networks for better scores. The final ensemble model was a mutli-class classifer which produced a score of 0.9842, which was within 0.0044 of the top score on the Kaggle leaderboard, where submissions often come from machine learning experts.

This comparative performance (and the fact that a score of 0.5 corresponds to random guessing) leads me to believe this solution could be used in any setting to detect toxic comments, and could be useful in helping moderate online discussion forums for these types of interactions in a production environment and make these forums more welcoming for all users.

The aspects of this capstone project that I found most difficult was learning a new machine learning development framework (Keras - though this was much more straightforward than starting off with Tensorflow!) while implementing a cutting-edge neural network architurecture (CapsNet) based on a paper from 2 years ago, so online resources on implmentation were comparatively limited. The final issue I had was that due to using Keras and non-GPU optimized code, I had to wait nearly an hour per training cycle (about 3 epochs) on each of the neural network models, which really slowed down experimentation with changing parameter values.

The parts of this project that I highly enjoyed go hand in hand with the difficult aspects. While it was tough to learn Keras on the fly, I really enjoyed its succinctness compared to straight up using Tensorflow (though it has a bit less flexibility). I also enjoyed the challenge of learning a new, not-so-common model architecture (CapsNet) versus the CNNs and LSTM models that are prevalent. Researching about and implementing both transfer learning with the word embedding models and ensembling methods was also quite interesting to me, and were very rewarding learning experiences for me.

Improvement

As with any machine learning model, there are always aspects that could be improved to produce better scores (hence the purpose behind Kaggle competitions in the first place), and this model can (however slightly) be improved as evidenced by the top Kaggle leaderboard scores.

The major area of improvements I would make center on further exploring ensembling methods. Blending methods in particular appear in most of the top public kernel submissions, and reading regarding several Kaggle competitions often point towards this approach to achieve the highest results and allow for high leaderboard placement. These ensembles can get as complex as you want them to be and can include dozens of models - which while not production friendly (i.e. take a long time to run) have the potential to hit the cutting edge of scoring performance. I only approached this technique by averaging, but there are methods of weighted averages, stacking, and blending, where different models are trained on different parts of the dataset (of particular interest is out-of-fold training) and predictions are even used as features on the next layer of models (blending/stacking) [20].

Other improvements which could help involve converting the models to TensorFlow so GPU architectures could

speed up model training, and experimenting with adding more pre-processing of the training comments to include hand-picked features to feed into the models, or possibly using both the GloVe and fastText embeddings together (or different embeddings on different models or parts of the dataset to follow the ensembling theme). Finally experimenting with non-zero entries in the embedding matrix for words that are not found (i.e use averaged values) may be beneficial as well.

References

1. "Toxic Comment Classification Challenge - Overview" https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/overview ↩
2. "Saying Goodbye to Civil Comments" Bogdanoff, Aja. https://medium.com/@aja_15265/saying-goodbye-to-civil-comments-41859d3a2b1d ↩
3. Wulczyn, Ellery, Nithum Thain, and Lucas Dixon. "Ex machina: Personal attacks seen at scale." Proceedings of the 26th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 2017. ↩
4. Georgakopoulos, Spiros V., et al. "Convolutional neural networks for toxic comment classification." Proceedings of the 10th Hellenic Conference on Artificial Intelligence. ACM, 2018. ↩
5. Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." Advances in neural information processing systems. 2017. ↩
6. "Toxic Comment Classification Challenge - Overview | Evaluation" https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/overview/evaluation ↩
7. "Classification: ROC Curve and AUC" https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc ↩
8. "Assessing and Comparing Classifier Performance with ROC Curves" https://machinelearningmastery.com/assessing-comparing-classifier-performance-roc-curves-2/ ↩
9. Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." Advances in neural information processing systems. 2017. ↩
10. "Capsule neural network" https://en.wikipedia.org/wiki/Capsule*neural*network ↩
11. "Capsule net with GRU" https://www.kaggle.com/chongjiujjin/capsule-net-with-gru ↩
12. "Facebook - English word vectors" https://fasttext.cc/docs/en/english-vectors.html ↩

13. Hendrik Jacob van Veen, Le Nguyen The Dat, Armando Segnini. 2015. Kaggle Ensembling Guide. [accessed 2018 Feb 6]. https://mlwave.com/kaggle-ensembling-guide/ ↩

14. "Logistic regression with words and char n-grams" https://www.kaggle.com/tunguz/logistic-regression-with-words-and-char-n-grams ↩

15. "Capsule net with GRU" https://www.kaggle.com/chongjiujjin/capsule-net-with-gru ↩

16. Sabour, Sara, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic routing between capsules." Advances in neural information processing systems. 2017. ↩

17. "Bidirectional GRU with Convolution" https://www.kaggle.com/eashish/bidirectional-gru-with-convolution ↩

18. "Multi-label data stratification" http://scikit.ml/stratification.html ↩

19. "GloVe: Global Vectors for Word Representation" https://nlp.stanford.edu/projects/glove/ ↩

20. "Kaggle Ensembling Guide" https://mlwave.com/kaggle-ensembling-guide/ ↩