Machine Learning Engineer Nanodegree

Capstone Project

Jing Li
December 17, 2018

I. Definition

Project Overview

Toxic comment classification is a Kaggle challenge to identify and classify toxic online comments[1]. The abuse and harassment online discourage many people from expressing themselves and seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments. The challenge is to build a multi-headed model capable of detecting different types of toxicity like threats, obscenity, insults, etc.

Convolutional Neural Networks (CNNs) were responsible for major breakthroughs in image classification and are the core of most computer vision systems today. CNNs have also achieved remarkably strong performance on sentence classification[2]. For this project, I use CNN for toxic comment classification.

The dataset used for training and testing the model is from wiki corpus dataset[3] which was rated by human raters for toxicity.

Problem Statement

Jigsaw and google have built models to study negative online behaviors like toxic comments[4]. The current models do not allow users to select which types of toxicity they are interested in finding. The solution is to build a multi-headed model that can detect different types of toxicity like threads, obscenity, insults and identity-based hate.

Metrics

ROC_AUC metric is required and used by Kaggle to score submitted models.

---

[1] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

[2] Y. Zhang, B. Wallace. A sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification 2016

[3] E. Wulczyn, N. Thain, and L. Dixon. Ex Machina: Personal Attacks Seen at Scale 2017

[4] https://www.engadget.com/2017/09/01/google-perspective-comment-ranking-system

There are arguments on whether this is a good metric for this challenge. It is a good metric since it essentially balances the true positive rate vs. the false positive rate. However, the Precision-Recall may be more appropriate if the true negatives (clean comments) are not meaningful to the problem and huge number of negative examples just dwarf the number of positives. We will use ROC-AUC since we want to compare our results with Kaggle leaderboard.

ROC (Receiver Operating Characteristic) curve is created by plotting the TPR (**True Positive Rate) on the y-axis** against the FPR (**False Positive Rate) on the x-axis** for every possible classification threshold. TPR is also known as sensitivity or recall and FPR as fall-out. The ROC curve depicts relative trade-offs between true positive (benefits) and false positive (costs). AUC (**Area Under the Curve)** is the area under the normalized ROC curve. It summarizes the ROC curve into a single number. ROC-AUC statistics works well with imbalanced classes and is often used in machine learning to compare model performance. Refer to this Wikipedia page[5] for details of the ROC-AUC including AUC integral calculation, its pros and cons.

II. Analysis

Data Exploration

The dataset provided by Kaggle is from wiki corpus dataset[6] which was rated by human raters for toxicity. The corpus contains 63M comments from discussions relating to user pages and articles dating from 2004-2015. There are 159571 samples in train dataset and 153164 samples in test dataset. Only 63978 samples in test dataset are being scored by Kaggle challenge. The dataset has the following fields:

- id: unique identification (object)
- comment_text: comment text (object)
- toxic: toxic comment label (integer, 0 or 1)
- severe_toxic: severe toxic comment label (integer, 0 or 1)
- obscene: obscene comment label (integer, 0 or 1)
- threat: threat comment label (integer, 0 or 1)
- insult: insult comment label (integer, 0 or 1)
- identity_hate: identity hate comment label (integer, 0 or 1)

This is a multi-label classification problem. Based on the value of mean in train data description, most of the categories are labeled 0. We can also see this from The Toxicity Level Breakdown in Exploratory Visualization. The count percentages for severe_toxic, identity_hate and threat are below 5%. So, roc_auc might not be a good metric if we are more interested in capturing toxic comments and not so care about clean comments. However, we'll stick to roc_auc since it is a required scoring for this challenge and we are using Kaggle challenge leaderboard scores to compare our model performance. In practice, we want to consider classification imbalance and

[5] Wikipedia: Receiver Operating Characteristic

[6] E. Wulczyn, N. Thain, and L. Dixon. Ex Machina: Personal Attacks Seen at Scale 2017

other metric such as precision-recall curve might be more appropriate. We observed that the severe_toxic seems to be a sub-category of toxic and we see example comments with multiple positive labels. Both train and test dataset are checked for missing values, outliers. We do not see any abnormal values.

The vocabulary of the comments for both train and test dataset is 394788. After filtering out stop words, it is 393637. The vocabulary size excluding comments not used for scoring is 300258 and 299317 with and without stop words respectively.

Example data in train dataset:

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
12	0005c987bdfc9d4b	Hey... what is it..\n@ | talk .\nWhat is it.....	1	0	0	0	0	0
16	0007e25b2121310b	Bye! \n\nDon't look, come or think of comming ...	1	0	0	0	0	0
42	001810bf8c45bf5f	You are gay or antisemmitian? \n\nArchangel WH...	1	0	1	0	1	1

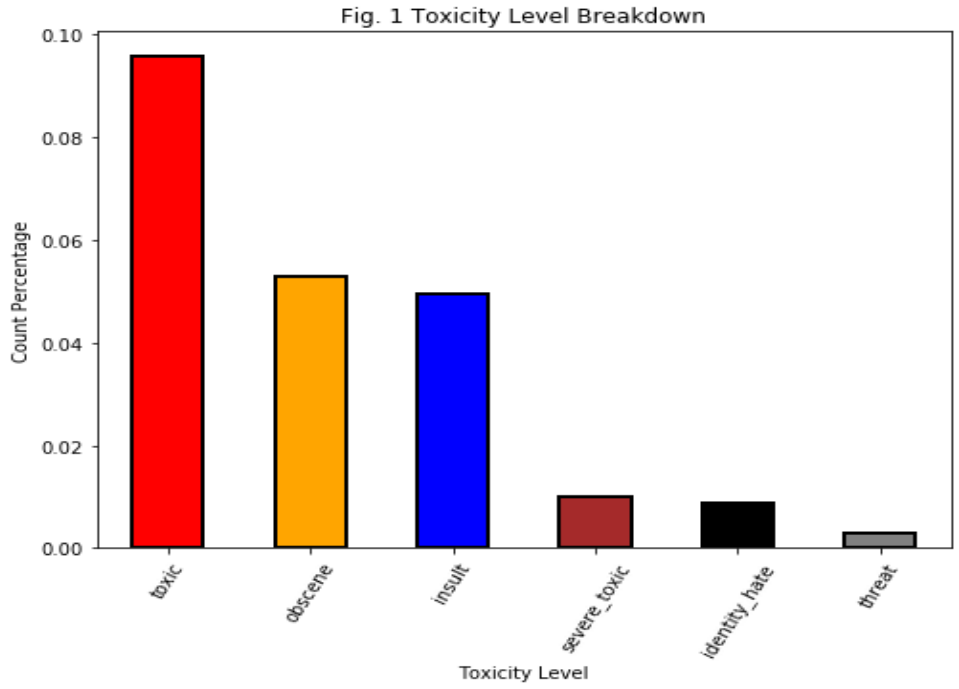
The pre-trained word vectors typically improve the generalization of models learned on limited amount of data. We use pre-trained word vectors based on fastText model trained on Crawl since it allows rare words to be represented appropriately and performs better compare to others[7]. Specifically, we use the 2 million-word vectors trained on Common Crawl (600B tokens) click here to download

The first line of the file contains the number of words in the vocabulary and the size of the vectors. Each line contains a word followed by its vectors. Each value is space separated. Words are ordered by descending frequency.

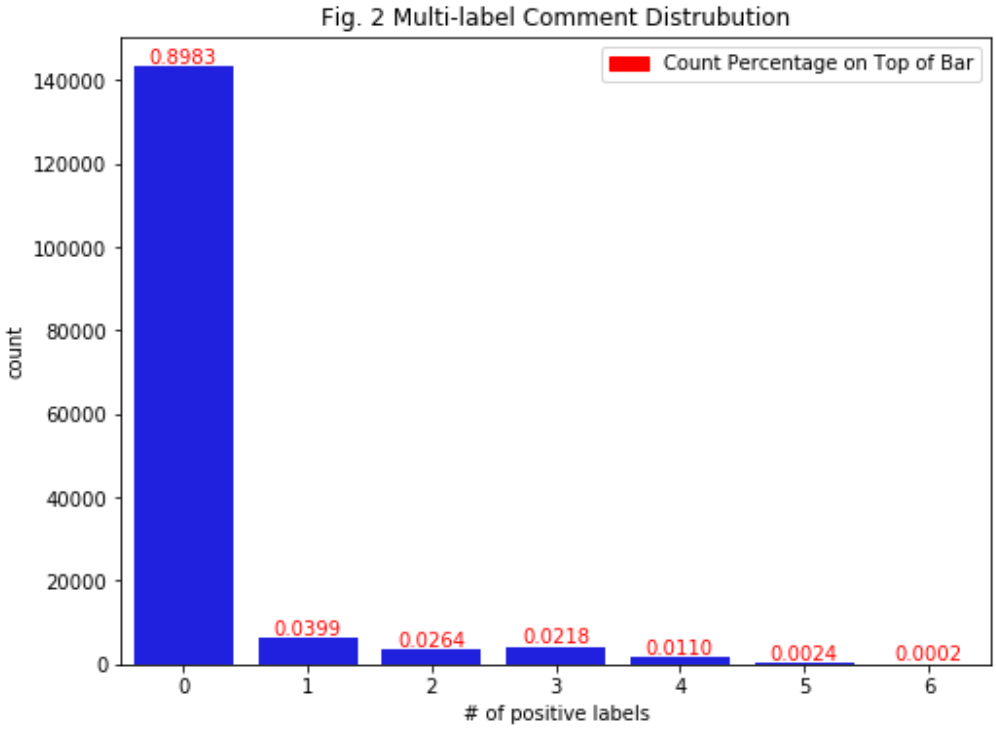
Exploratory Visualization

The toxicity distribution among the 6 categories in train dataset can be seen from Fig. 1. This is helpful for understanding the imbalance in classes. The count percentage is relative to the number of total comments in the train dataset. Here we see about 9.6% comments in train are 'toxic'; about 5% are 'obscene' or 'insult'; about 1% are 'severe_toxic' or 'identity_hate'; about 0.3% is 'threat'. So, most of the comments are clean.

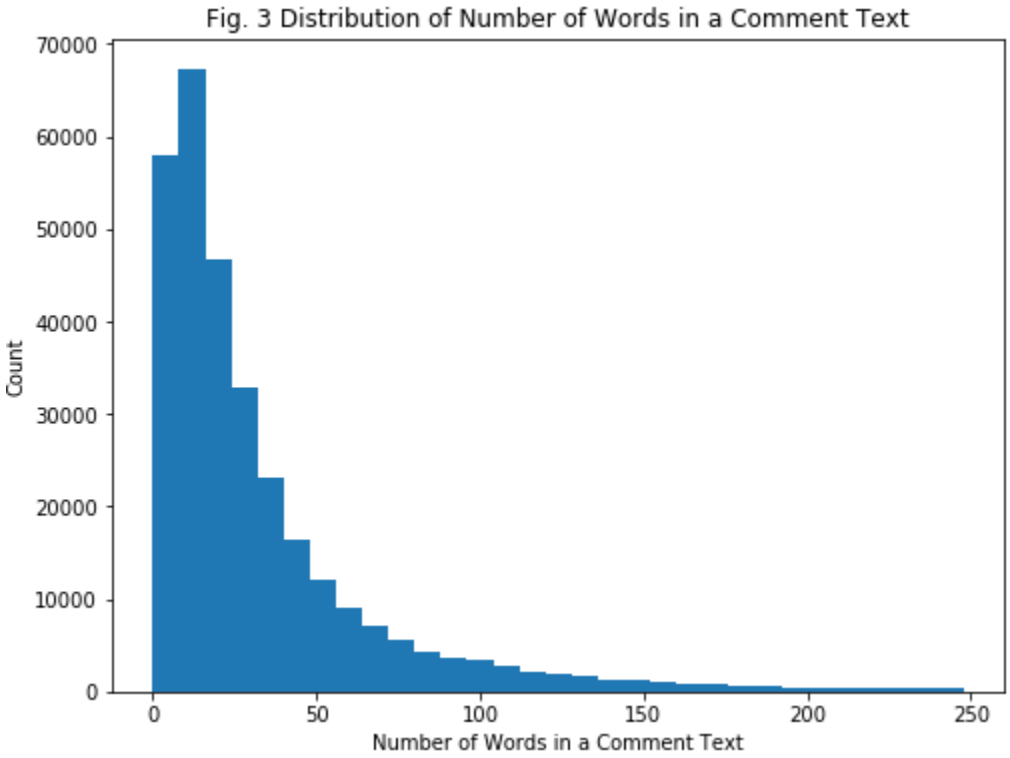
[7] T.Mikolov, E. Grave, P. Bojanowski, C.Puhrsch, A.Joulin. Advances in Pre-Training Distributed Word Representations



Toxic comments can have multiple positive labels. For example, we see comment #42 in the example data is 'toxic' and 'obscene' and 'insult' and 'identity_hate'. Fig. 2 shows multi-label comment distribution. Count percentage is shown on top of the bar. Here we see most comments (about 90%) do not have any positive label. These are clean comments. About 4% comments have 1 positive toxicity label and about 6% have 2 or more positive toxicity labels.



Comments varies in number of words. Fig. 3 shows distribution of number of words in a comment text. The longest comment has 2142 words. However, about 97% of comments has words less than 200.



Algorithms and Techniques

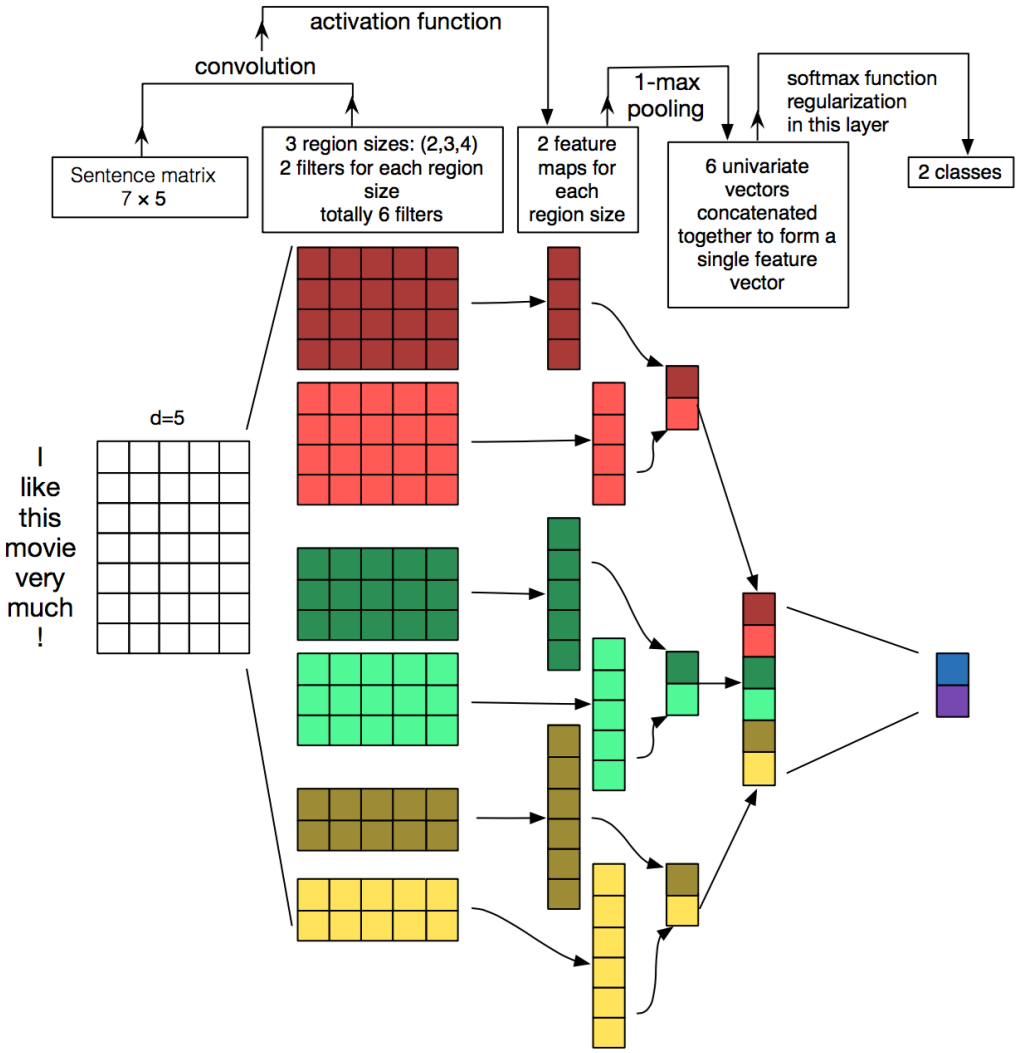
Convolutional Neural Networks (CNNs) were responsible for major breakthroughs in image classification and are the core of most computer vision systems today. CNNs have also achieved remarkably strong performance on sentence classification. For this project, I intend to use CNN model described in this article[8].

Fig. 4 and the following description of the model are from this article.

Illustration of a CNN architecture for sentence classification. We depict three filter region sizes: 2, 3 and 4, each of which has 2 filters. Filters perform convolutions on the sentence matrix and generate (variable-length) feature maps; 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded. Thus, a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer. The final softmax layer then receives this feature vector as input and uses it to classify the sentence; here we assume binary classification and hence depict two possible output states.

[8] Y. Zhang, B. Wallace. A sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification 2016

Fig. 4 CNN Architecture for Sentence Classification



To classify comments, we need a representation for words that capture their meanings, semantic relationships and the different types of contexts they are used in. Word representations are implemented using word embeddings as a word vector. Prediction based vectors such as Word2vec are built by CBOW (Continuous bag of words) and Skip-gram models. These models are shallow neural networks. The models map word(s) to the target variable which is also a word(s). The techniques learn weights which act as word vector representations.[9] FastText is an

[9] https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/

extension to Word2Vec. It breaks words into several n-grams before training in neural network. The benefit is that rare words can now be represented.[10]

The pre-trained word vectors we used here are based on fastText model trained on Crawl (600B tokens) with an embedding size of 300. There are 2 million vectors. If we choose a maximum length of a comment to be 200, we cover over 97% of the comments without cut off. We pad each comment with less than 200 words to a length of 200. Thus, for a 200-word comment, we would have a 200 x 300 matrix as the 'image' for our input into CNN.

Filters with different region size (h) are used to scan the comment 'image' vertically since we have a fixed dimensionality of the word vector (300). The number of parameters to be estimated will be h x 300. We use stride size 1, which is typical in NLP. The feature map with this region size from this convolution layer will be a matrix of (200 – h + 1) by 1, which is a function of h. Pooling is used to allow use of variable size of region and reduce the output dimensionality. 1-max pooling will be applied to the feature map to yield just a single number as is typically done in NLP.

Multiple filters with the same region size are used to learn complementary features from the same regions. Filters with different region size can be used to learn the 'image'. Each filter generates a feature map, which after pooling is just a single number. A feature vector is created by concatenating the single number pooled from all feature maps. The feature vector can be fed into a dense layer with sigmoid activation function for 6 toxicity classification. The number of filters and the region size can to tuned.

We use dropout after both embedding and feature concatenating layers to reduce overfitting. The dropout rate can be tuned.

We use binary_crossentropy loss function to minimize the loss.

In summary, we will try to tune the following:

1. region size – size of the filter
2. number of filters – number of filters in the convolution layer
3. type of activation functions – relu, tanh, elu, etc.
4. layers – layers of the neural network (convolution, pooling, dense, etc.)
5. dropout rate – rate at which neurons are randomly ignored to avoid overfitting
6. batch size – number of comments in each training step
7. epochs – number of times the training dataset passed forward and backward through the neural network
8. optimizers – optimization algorithms used in optimizing a neural network (adam, rmsprop, adadelta, adagrad, etc.)

We used Logistic Regression for our benchmark. Logistic Regression uses a logistic function to model a binary dependent variable. It is the building block of all that constitutes Deep Learning.

[10] https://fasttext.cc/docs/en/unsupervised-tutorial.html

It is simple and fast and has been used as baseline for sentence classification. For word representation with Logistic Regression, we use term frequency-inverse term frequency (TFIDF) which can be fed into the classifier properly. The TFIDF vectorization of a text document assigns each word in a document a number that is proportional to its frequency in the document and inversely proportional to the number of documents in which it occurs. Thus, common words such as "a" or "is" receives lower weights compare to words that are specific to the document in question. Specifically, we use Sklearn TfidfVectorizer. We limit vocabulary to 100000 due to memory capacity. The result from TfidfVectorizer is a sparse matrix of number of comments x 100000. This will then be fed into Logistic Regression classifier.

The specific implementation of the models including Hyperparameters tuning is discussed in methodology.

Benchmark

We use Logistic Regression with TF-IDF to generate our benchmark score.

The following parameters are used for TfidfVectorizer:

sublinear_tf=True: Apply sublinear tf scaling – replace tf with 1 + log(tf)

strip_accents='unicode': Remove accents and perform other character normalization with 'unicode' method

analyzer='word': feature is made of word

ngram_range=(1,1): set the lower and upper boundary of the range of n-values for different n-grams to be extracted

max_features=100000: consider on the top max_features ordered by term frequency across the train corpus

The default parameters are used for LogisticRegression classifier:

(*penalty='l2', dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None*)

The ROC-AUC scores from our benchmark are the followings:

stop word removing	data cleaning	score
n	y	0.9759
n	n	0.9758
y	y	0.9763
y	n	0.9763

The Kaggle challenge does provide leadership scores. For example, the top score is at 0.9885. There is an overview on the first-place solution[11]. However, for most of the top scores, we do not know the models used to achieve those scores and the hardware setup. We use the top scores from Kaggle to see how far we are away from them and try our best to tune our model to be as close to them as possible.

III. Methodology

Data Preprocessing

1. The dataset provided do not have missing values or other abnormalities.
2. Filter out entries in test dataset where the target values are -1 since they are not used for scoring purpose.
3. In exploratory data analysis, we see no obvious impact of stop words, IP addresses, usernames and URL links on a comment being clean or toxic. So, we tried to remove them to avoid overfitting. However, we found we get better score without this data cleaning. We might lose some insight by removing stop words, IP addresses, usernames and URL links. So, in our final model, we do not do data cleaning in this area.
4. For benchmark, use Sklearn TfidfVectorizer to generate word vectors as features for both train and test comments
5. For CNN model, we generate embedding matrix as follows. This embedding matrix will be needed in embedding layer in the model.
    a. We use Keras Tokenizer to convert comment texts to sequences of word indices. A "word index" is an integer ID for the word. We only consider the top 100000 most commonly occurring words in the dataset.
    b. Loop through each word vector in pre-trained word vectors and create an embedding dictionary for each word and corresponding vector.
    c. Loop through each word index created in step a, generate a vocab (100000) by maxlen (200) embedding (weight) matrix.

Implementation

1. We implement the proposed benchmark solution with TFIDF and Logistic Regression classifier from Sklearn
    - Use Sklearn LogisticRegression classifier with the parameters documented in benchmark section
    - Use cross-validation with 3 folds to train the dataset
    - Refit with all training data
    - Predict with test data and calculate the roc_auc score
    - Plot ROC and AUC graph
    - Code reference: benchmark-LogisticRegression-*.ipynb

[11] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/52557

2. We use Keras for CNN solution. We also use Sklearn GridSearchCV for some initial model tuning. Sklearn GridSearchCV is used to narrow down the potential best models. However, due to the limitation of the integration of Keras wrapper with Sklearn API, we could not get cross validation loss metrics. So, we refit without Sklearn GridSearchCV for potential best models to see the improvement of cross validation loss and roc-auc score.
    - We define a function called make_model which takes tuning parameters and returns a CNN model for the problem. This function executes the following steps:
        1. Define an input layer with an output shape of maxlen for receiving vector representation of the comments.
        2. Load the embedding matrix into embedding layer.
        3. Add a spatial 1d dropout layer for variational dropout[12] to avoid overfitting.
        4. Add a reshape layer to add one dimension to connect to 2d convolution layer.
        5. Loop through tunable list of region size and create pairs of convolution and max pooling layers.
        6. Concatenate all max pooling layers.
        7. Add a Flatten layer to flatten the concatenated matrix.
        8. Add a dropout to regulate the input to the dense layer.
        9. Add a dense layer with sigmoid activation function to classify the 6 labels.
    - With GridSearchCV,
        i. To use Sklearn GridSearchCV for model tuning, we need to wrap Keras CNN model with KerasClassifier
        ii. We use ShuffleSplit from Sklearn to generate training and validation set
        iii. We then create GridSearchCV with model hyperparameters to be tuned including epochs and batch_size
        iv. We perform the grid search to find the best model
        v. We perform prediction with the best model and calculate ROC-AUC score
        vi. We finally plot the score vs tuning parameters
        vii. Example code in Phase 1 code folder: cnn2-grid-tune-spatialDR-5epochs-plot-0.9817.ipynb
        viii. Code Reference: FinalModelGridSearchCode.ipynb
    - With Keras fitting
        i. ModelCheckpoint - we use this Keras callback to monitor validation ROC-AUC score improvement at each epoch and save the best model
        ii. EarlyStopping – we use this Keras callback to monitor validation ROC-AUC score and stop training if there is no improvement after 2 epochs.
        iii. RocAucMetricCallback – we create this custom function to log and output both training and validation score with roc_auc_score API from Sklearn at the end of each epoch. We then use Keras callback functions to execute this custom function. Note that Keras custom metric API does not support Sklearn metric. We implemented Tensorflow auc and Tensorflow binary auc. However, we can see some difference in the score outputs. We finally choose to use Sklearn roc_auc since it is consistent with the use of it in non-Keras environment and in Kaggle for comparison reason.

[12] https://arxiv.org/pdf/1512.05287.pdf

iv. We also use train_test_split from Sklearn to generate training (90%) and validation (10%) dataset. We need this to be able to generate ROC-AUC for train and validation score.
v. We perform fitting with training and validation dataset
vi. We perform prediction with unseen test dataset and calculate ROC-AUC score
vii. We plot ROC curves and calculate AUC for 6 toxicity labels and both micro and macro average
viii. We plot model ROC-AUC vs epoch and loss vs epoch for both train and validation score
ix. Code reference: FinalModelCodeAndResults.ipynb

Refinement

There are many hyperparameters that can be tuned. Y. Zhang's paper provide some guidance on where to start[13]. We start with a simple model with 100 filters and one region size of [7]. We use adam optimizer with default parameters. We use 1-max pooling. We choose 3 epochs with batch size 128. We set Spatial dropout rate at 0.5 and dropout rate before the output layer at 0.0. We use activation function relu for the conv layers. With this model, we get a ROC-AUC score at 0.9814. (Code reference: cnn-starting-0.9814.ipynb) That is a big improvement from benchmark of 0.9758.

We then tried multiple regions suggested in Y. Zhang's paper just to get a feeling how long each epoch will take with my laptop without GPU. The elapsed time really depends on number of trainable parameters and thus depends on hyperparameters such as filter size, number of regions. We see from 200 seconds to 800 seconds per epoch with many of our experiments.

In general, we search for the best model utilizing the following techniques:

1. Start with some suggestions from Y. Zhang's paper
2. Utilize Grid Search to narrow down best performance hyperparameters
    a. We use it to automate the search overnight
    b. However, we do not have details of training loss and score with each epoch
    c. We use unseen test datasets each time to ensure we pick up the best model
3. Utilize Keras Callback API to monitor best performance model
    a. RocAucMetricCallback – to log Sklearn ROC-AUC scores
    b. Checkpoint – to save best performance model based on ROC-AUC score improvement
    c. EarlyStopping – to stop after no improvement in ROC-AUC score
    d. we use unseen test dataset each time to ensure we pick up the best model
4. Manually adjust one parameter to fine tune the model

[13] Y. Zhang, B. Wallace. A sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification 2016

We conducted many experiments. The following variables are considered:

- regions: [7], [2], [2,3], [2,3,4], [2,3,4,5], [3,6,7,8], [2,3,5,7], [5,6,7,8], [2,3,4,5,6], [5,6,7,8,9]
- number of filters: 16, 32, 40, 48, 100
- dropout rate: 0.0 – 0.6
- activation functions: relu, elu, tanh
- epochs: 3, 5, 10
- batch size: 128, 256, 512, 1024
- with or without removing stop words
- with or without removing IP's, usernames and URL links
- validation dataset: 20%, 10%
- tested Glove pre-trained word vector
- optimizers: adam(`(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)`

The results and the parameters used from the first phase experiments are tabled in appendix A. There, we highlighted the ROC-AUC scores higher than 0.9820. We found that validation size of 10%, batch size of 256, activation function tanh and filter size of 32 perform better. We also found all these top performers are without removing stop words, IPs, URL links and usernames. So, in our second phase fine tuning, we narrow down to the following to tune

- regions: [2,3,4,5], [2,3,4], [3,4,5], [2,3,5,7]
- number of filters: 32
- spatial dropout rate: 0.3 – 0.5
- dropout rate: 0.0 – 0.3
- activation function: tanh, relu, elu
- epochs: 5, 10, 20, 25, 30
- batch size: 256
- no stop words removing
- no removing IPs, usernames and URL links
- validation dataset: 10%
- optimizers:
    - adam (lr=0.001, 0.0005, 0.0003),
    - adadelta (lr=1.0, rho=0.95, epsilon=None, decay=0.0)
    - adagrad `(lr=0.01, epsilon=None, decay=0.0)`
    - rmsprop `(lr=0.001, rho=0.9, epsilon=None, decay=0.0)`

The results and the parameters used from second phase tuning are tabled in appendix B.

We then further narrow down to two best models – the only difference between these 2 models are the regions: [2,3,4] and [2,3,4,5] with other variables as following:

- number of filters: 32
- spatial dropout rate: 0.4

- dropout rate: 0.2
- activation function: tanh
- epochs: 25
- batch size: 256
- no stop words removing
- not removing IPs, usernames and URL links
- validation dataset: 10%
- optimizer: adadelta (lr=1.0, rho=0.95, epsilon=None, decay=0.0)

Comparison of the performance from these two models are summarized in Model Evaluation and Validation section.

Here is a summary of observations:

- No cleaning performs better. It appears we lost critical insight when we filtered out IPs, usernames and URL links.
- No stop words removal looks also better. Again, we might lose some insight into the toxic comments by removing all stop words.
- GridSearch does not support Keras Callback API. So, we may get a model in Keras that is not optimized due to overfitting or underfitting because we always go through a fixed number of epochs without saving the best model via Keras Callback API. This would lead to suspicious best model when comparing results that are overfitting or underfitting.
- We implement "save best" Callback for Keras fitting to ensure we capture the best model based on ROC-AUC score.
- The ROC-AUC scores from these experiments range from low at 0.9735 to the best at 0.9830

IV. Results

Model Evaluation and Validation

1. The test dataset is unseen with 63978 samples. The comments in these samples has various number of words up to 2142 as discussed in exploratory visualization and shown in fig. 3. The following table summarized the performance with unseen test dataset of the best 2 models with 4 runs. The only difference between the 2 models are the regions. One has multiple regions of [2,3,4] and the other has multiple regions of [2,3,4,5]:

model	runs	regions	Sklearn auc score
A	1	[2,3,4]	0.9830
A	2	[2,3,4]	0.9828
A	3	[2,3,4]	0.9830
A	4	[2,3,4]	0.9830
B	1	[2,3,4,5]	0.9830

B	2	[2,3,4,5]	0.9829
B	3	[2,3,4,5]	0.9829
B	4	[2,3,4,5]	0.9829

Both models perform about the same. Model A has 87078 trainable parameters and takes about 430 seconds each epoch while model B has 135302 trainable parameters and takes about 530 seconds each epoch.

2. We also run the test with changes in test datasets (removing stop words and/or removing IP addresses, usernames and URL links) to generate changes in input to validate the robustness of the models. The following are the results:

model A, run4

stop words removing	data cleaning	score
n	n	0.982969
n	y	0.982943
y	y	0.981452
y	n	0.981485

model B, run4

stop words removing	data cleaning	score
n	n	0.982941
n	y	0.982915
y	y	0.981261
y	n	0.981297

3. We calculate both label ranking average precision (LRAP)[14] and label ranking loss(LRL)[15] for predictions (run4) from test datasets. Both models show excellent performance.

model	metric	score
A	LRAP	0.997903
B	LRAP	0.997687
A	LRL	0.001520
B	LRL	0.001634

4. Overall, model A seems perform slightly better with less training time. With the above comparison, we choose model A to be the final model with the following parameters/variables:
    - regions: [2,3,4]
    - number of filters: 32

[14] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.label_ranking_average_precision_score.html

[15] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.label_ranking_loss.html

- spatial dropout rate: 0.4
- dropout rate: 0.2
- activation function: tanh
- epochs: 25
- batch size: 256
- no stop words removing
- not removing IPs, usernames and URL links
- validation dataset: 10%
- optimizer: adadelta (lr=1.0, rho=0.95, epsilon=None, decay=0.0)

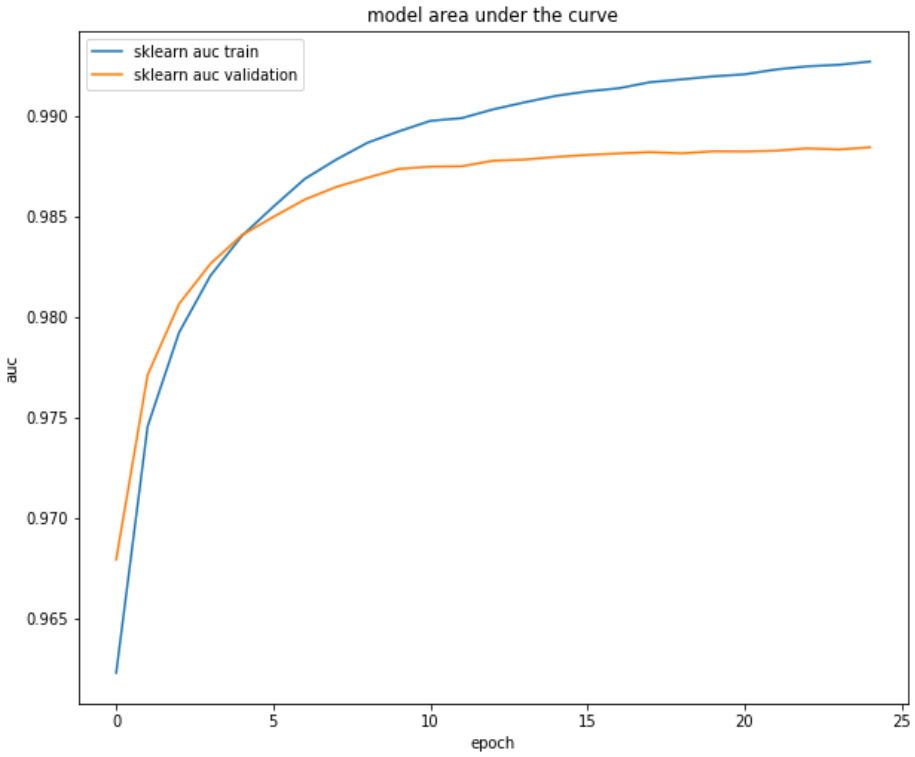
Model summary:

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 200)	0	
embedding_2 (Embedding)	(None, 200, 300)	30000000	input_2[0][0]
spatial_dropout1d_2 (SpatialDro	(None, 200, 300)	0	embedding_2[0][0]
reshape_2 (Reshape)	(None, 200, 300, 1)	0	spatial_dropout1d_2[0][0]
conv2d_4 (Conv2D)	(None, 199, 1, 32)	19232	reshape_2[0][0]
conv2d_5 (Conv2D)	(None, 198, 1, 32)	28832	reshape_2[0][0]
conv2d_6 (Conv2D)	(None, 197, 1, 32)	38432	reshape_2[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 1, 1, 32)	0	conv2d_4[0][0]
max_pooling2d_5 (MaxPooling2D)	(None, 1, 1, 32)	0	conv2d_5[0][0]
max_pooling2d_6 (MaxPooling2D)	(None, 1, 1, 32)	0	conv2d_6[0][0]
concatenate_2 (Concatenate)	(None, 3, 1, 32)	0	max_pooling2d_4[0][0] max_pooling2d_5[0][0] max_pooling2d_6[0][0]
flatten_2 (Flatten)	(None, 96)	0	concatenate_2[0][0]
dropout_2 (Dropout)	(None, 96)	0	flatten_2[0][0]
dense_2 (Dense)	(None, 6)	582	dropout_2[0][0]

Total params: 30,087,078
Trainable params: 87,078
Non-trainable params: 30,000,000

5. In the second phase of model tuning, we use 10% the training dataset as validation set. The following graph shows ROC-AUC train and validation score vs number of epochs for the Model A run 4. As number of epochs increase, ROC-AUC train score keeps going up. However, the score for validation will increase and then hit the best and flatten out as expected.

Fig. 5 Model train and validation ROC-AUC score vs epoch



6. The final model has been tested with various inputs as described and generalizes well to unseen data. It is robust enough for separating clean from toxic comments. The small perturbations in the input space do not greatly affect the results. The final parameters of the model seem appropriate. The final model is reasonable and aligning with solution expectations. We should be able to trust it in separating clean from toxic comments. (Result reference: FinalModelCodeAndResults.ipynb)

Justification

- The final model has consistent best scores with unseen test dataset

model	runs	Sklearn ROC-AUC score
A	1	0.9830
A	2	0.9828
A	3	0.9830
A	4	0.9830

- With the ROC-AUC score at around 0.9830, it is much better than the benchmark score of 0.9758.
- Kaggle challenge is judged only by macro ROC-AUC score. With macro ROC-AUC score at 0.9830, the model successfully separates clean from toxic comments. I believe we have solve the problem posed by Kaggle challenge.
- Both label ranking average precision (LRAP) and label ranking loss (LRL) for predictions (run4) from unseen test dataset show excellent performance.

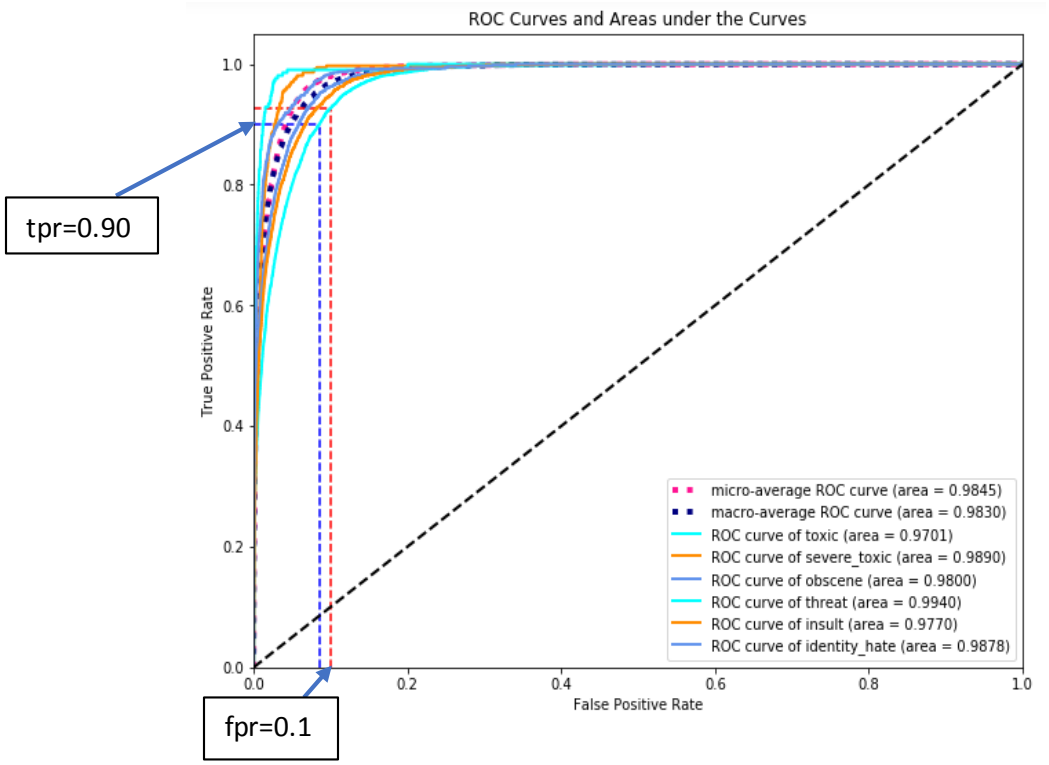
metric	score	reference best score
LRAP	0.997903	1.0
LRL	0.001520	0.0

V. Conclusion

Free-Form Visualization

- The roc curves and auc plot for model A in run 4 below shows excellent in the performance of the model separating clean from all levels of toxic comments. From the roc curves, we see the trade-offs between tpr and fpr. We can set the cutoff threshold for prediction probability to ensure either tpr no less than some value or fpr no more than some value. For example, we calculate the cutoff probability value in prediction for label "toxic". To ensure tpr > 0.90, we need cutoff threshold value at 0.3633 and to ensure fpr < 0.1, we need cutoff threshold at 0.2805. Thus, the cutoff threshold value is purely a decision of business. (Code reference: FinalModelResultAnalysis1.ipynb)

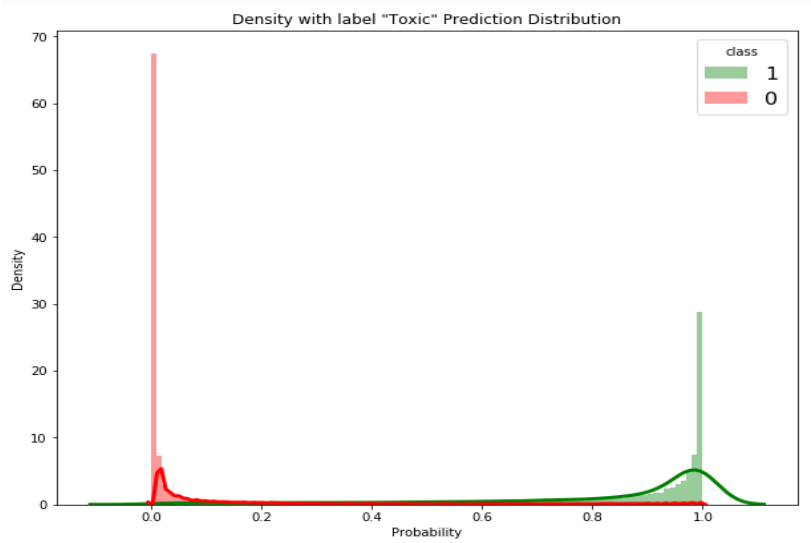
  Fig. 6 ROC curves and calculated AUC



- We also plot prediction probability distribution for test datasets. The green one is for positive label (class=1) and the red one is for negative label (class=0).

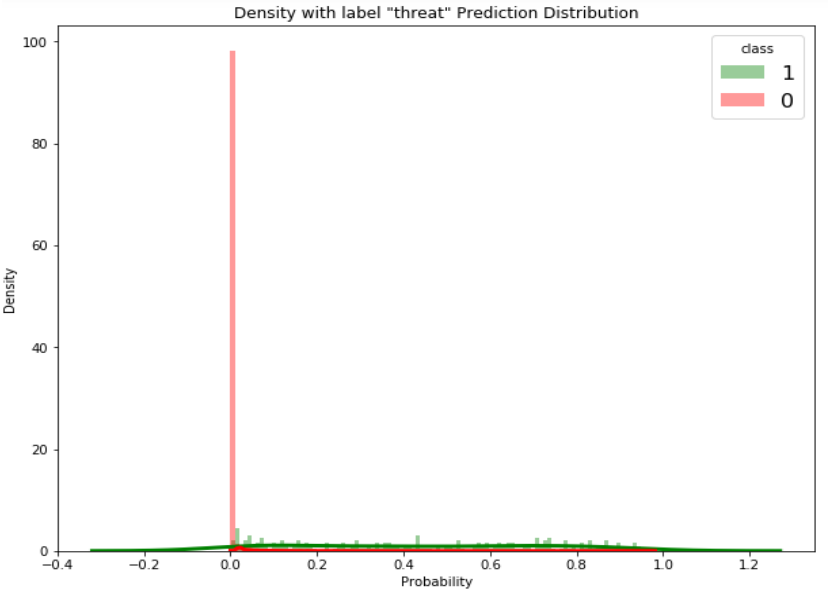
For label "toxic" in fig 7, we see the model can separate clean from toxic comments very good. In fact, if we set cutoff threshold value at 0.3633, we ensure tpr no less than 0.9 and also fpr no more than 0.0852. (Code reference: FinalModelResultAnalysis1.ipynb)

Fig. 7 Prediction distribution for label "toxic"



For label "threat", however, the model does not do well for class 1 as can be seen in fig. 8. In fact, to ensure tpr no less than 0.5, we need to set cutoff threshold value at 0.4282. At this cutoff threshold, fpr value is no more than 0.0013. Because we have a small fpr value, we still have very good ROC-AUC score. The reason that the model does not do good is due to class imbalance for label "threat". In Exploratory Visualization section, we find that positive label for "threat" only accounts for about 0.3% train comments. (Code reference: FinalModelResultAnalysis2.ipynb)

Fig. 8 Prediction distribution for label "threat"



Reflection

The process used for this project can be summarized as follows:

1. Download dataset provided by Kaggle. Download pre-trained word vectors.
2. Analyze the dataset including the type of the data, basic statistics, abnormalities, etc. and estimate vocabulary and maximum length of the comments.
3. Preprocess the data.
4. Create benchmark with Logistic Regression classifier.
5. Train CNN model with different variables such as layers, hyperparameters, batch size, epochs, etc.
6. Utilize different fitting techniques including Keras model fitting and Sklearn GridSearch.
7. Develop a custom class to log Sklearn ROC-AUC score. Utilize Keras Callback features such as checkpoint and early stopping to save the best model based on improvement of the Sklearn ROC-AUC score
8. Create function to plot roc curves and auc based on Sklearn examples.
9. The validation ROC-AUC score and loss were also plotted against epoch to help understand the training process
10. The prediction probability distributions were plotted for label "toxic" and "threat" to understand the separation of clean from toxic comments and the impact of class imbalance.

The most challenging part of this project is to figure out how to monitor Sklearn ROC-AUC score for saving the best model and stop the training when the score is not improving. Keras model does not support ROC-AUC scoring and Tensorflow ROC-AUC or binary ROC-AUC score shows some difference comparing to Sklearn ROC-AUC scores. We were able to implement a function to log both train and validation Sklearn ROC-AUC via Keras callback API.

Also, since Sklearn GridSearch does not support Keras Callback, we do not have training and validation auc score while utilizing Sklearn GridSearch to find the best model. For the same reason, the result from Sklearn GridSearch might not be reliable.

The class imbalance impact on prediction can be seen for labels such as "threat" which accounts for only 0.3% of the train comments. Although Kaggle challenge only look at ROC-AUC score, in practice, we may need to address class imbalance issue.

Improvement

1. With model A, each epoch takes around 430 seconds. For more extensive fine tuning of the hyperparameter, especially fine tuning with multiple hyperparameters at the same time, we may need a more powerful machine with GPU.
2. Based on Kaggle top score, there are certainly other models and techniques which will generate better ROC-AUC score. In article "Comparative Study of CNN and RNN for Natural Language Processing"[16] , three most widely used deep neural networks (DNN) –

[16] https://arxiv.org/pdf/1702.01923.pdf

convolutional neural network (CNN), gated recurrent unit (GRU) and long short-term memory (LSTM) – were compared. GRU and LSTM or the combined model may perform better for this project

3. Extending beyond Kaggle challenge, for practical use cases, we need to address class imbalance issue. We may investigate the use of techniques such as under sampling or oversampling.

Appendix A

1. with Sklearn GridSearchCV, optimizer: adam

Table a: regions: [7], validation set: 10%

filters	Sdroprate	droprate	activation	epochs	batch	stop words removing	data cleaning	score
100	0.5	0.0	relu	3	128	y	n	0.9814
100	0.3	0.0	relu	3	128	y	y	0.9784
100	0.1	0.0	relu	3	128	y	y	0.9769
32	0.4	0.1	elu	3	256	n	n	0.9735
32	0.4	0.1	tanh	3	256	n	n	0.9754
100	0.5	0.0	Relu*	3	128	y	y	0.9790

*with GridSearch on activation: [relu, elu, tanh]

Table b: regions: [2], number of filters: 32, Spatial dropout rate: 0.4, dropout rate: 0.1, batch size 256, not remove stop words, no data cleaning, validation set: 10%

activation	epochs	score
elu*	3	0.9754
elu	3	0.9762
elu	3	0.9762
elu	5*	0.9788

* with GridSearch on epochs [3,4,5,6,7]

Table c: regions: [2,3,4,5], number of filters: 32, epochs: 5,

Sdroprate	droprate	activation	batch	stop words removing	data cleaning	validation	score
0.2	0.1	tanh	256	n	n	0.1	0.9817
0.0	0.1	tanh	256	n	n	0.1	0.9811
0.4	0.1	tanh	256	n	n	0.1	0.9821
0.4	0.1	tanh	256	y	n	0.1	0.9822
0.1	0.1	tanh	256	y	y	0.1	0.9804
0.4	0.1	tanh	256	n	n	0.1	0.9827
0.3*	0.0	tanh	256	y	y	0.2	0.9814
0.4	0.1	tanh	128	y	n	0.1	0.9819
0.3	0.1	tanh	256	y	n	0.1	0.9820
0.3	0.1	tanh	256	n	n	0.1	0.9822
0.4	0.1	tanh	256	y	n	0.1	0.9820
0.4	0.1	elu	256	n	n	0.1	0.9817
0.3**	0.1**	tanh	256	n	n	0.1	0.9821
0.4	0.1	Tanh***	256	n	n	0.1	0.9821

*with GridSearch on Sdroprate: [0.0,0.1,0.2,0.3,0.4,0.5,0.6]
**with GridSearch on Sdroprate and droprate – Sdroprate: [0.1,0.2,0.3,0.4,0.5], droprate: [0.1,0.2,0.3,0.4,0.5]
***with GridSearch on activation: [relu, elu, tanh]

Table d: regions: [3,6,7,8], number of filters: 32, activation: tanh, epochs: 5, batch size: 256, validation set: 10%

Sdroprate	droprate	stop words removing	data cleaning	score
0.4	0.05	y	y-train	0.9819
0.4	0.1	y	y-train	0.9819
0.4*	0.05*	n	n	0.9819

* with GridSearch on Sdroprate and droprate – Sdroprate: [0.2,0.3,0.4,0.5], droprate: [0.05,0.1,0.2]

Table e: other regions, number of filters: 32, dropout rate: 0.1, activation: tanh, epochs: 5, batch size: 256, not removing stop words, no data cleaning, validation set: 10%

regions	Sdroprate	score
[2,3,5,7]*	0.3	0.9823
[2,3,4]	0.3	0.9823
[5,6,7,8]	0.4	0.9817
[5,6,7,8,9]	0.2	0.9811

[2,3]	0.3	0.9817

* with GridSearch on regions: [2,3,5,7], [3,5,7]

2. **Table f**: With Glove word vector (glove.840B.300d.txt), optimizer: adam, regions: [2,3,4,5], number of filters: 32, Spatial dropout rate: 0.4, dropout rate: 0.1, activation: tanh, epochs: 5, batch size: 256, validation set: 10%

stop words removing	data cleaning	score
y	n	0.9803
n	n	0.9802
y	n	0.9806

3. With Keras model fitting, optimizer: adam

Table g: regions: [2,3,4,5], number of filters: 32

Sdroprate	droprate	activation	epochs	batch	stop words removing	data cleaning	validation	score
0.5	0.1	relu	3	256	n	n	0.2	0.9780
0.4	0.1	elu	5	512	n	n	0.1	0.9803
0.4	0.1	tanh	10	256	y	y	0.1	0.9814
0.3	0.0	tanh	20	512	y	y	0.2	0.9803
0.3	0.0	tanh	10	512	y	y	0.2	0.9812
0.3	0.0	tanh	10	1024	y	y	0.2	0.9794
0.3	0.0	tanh	10	512	y	y	0.1	0.9807
0.4	0.1	tanh	10	256	y	y	0.1	0.9767
0.4	0.1	tanh	5	256	n	n	0.1	0.9816
0.3	0.0	tanh	10	512	y	y	0.2	0.9812
0.3	0.0	tanh	10	256	y	y	0.2	0.9805
0.4	0.1	tanh	5	256	n	n	0.1	0.9819

Table h: regions: [2,3,4,5,6], number of filters: 32, dropout rate: 0.1, epochs: 5, not removing stop words, no data cleaning:

Sdroprate	activation	batch	validation	score
0.3	relu	256	0.2	0.9798
0.4	relu	256	0.1	0.9807
0.35	relu	256	0.1	0.9804
0.45	relu	512	0.1	0.9802
0.4	tanh	512	0.1	0.9811
0.5	relu	256	0.2	0.9805
0.4	elu	512	0.1	0.9807

Table i: regions: [2,3,5,7], number of filters: 32, spatial dropout rate: 0.4, dropout rate: 0.1, activation: tanh, validation set: 10%

epochs	batch	stop words removing	data cleaning	score
5	256	n	n	0.9823
10	128	y	y	0.9802
10	128	y	y	0.9801
10	256	n	n	0.9821
5	256	n	n	0.9822
5	256	n	n	0.9822

Table j: other regions, remove stop words, clean data, validation set: 10%:

regions	filters	Sdroprate	droprate	activation	epochs	batch	score
[2,3,4,5,6,7]	16	0.4	0.1	tanh	10	256	0.9809
[7]	100	0.5	0.0	relu	10	128	0.9789
[7]	100	0.5	0.0	relu	5	128	0.9787

Appendix B

Table k: regions: [2,3,4,5], number of filters: 32, Spatial dropout rate: 0.4, droprate:0.1, optimizer: adam

activation	epochs	score
tanh	10	0.9821
elu	10	0.9813
relu	10	0.9807
tanh	5	0.9816

Table l: regions: [2,3,4], activation: tanh

Sdroprate	epochs	droprate	filters	optimizer	score
0.4	10	0.1	32	adam	0.9823
0.5	10	0.1	32	adam	0.9822
0.3	10	0.1	32	adam	0.9822
0.4	5	0.1	32	adam	0.9816
0.4	10	0.0	32	adam	0.9820
0.4	5	0.0	32	adam	0.9817
0.4	10	0.2	32	adam	0.9824
0.4	10	0.2	24	adam	0.9820

0.4	10	0.3	32	adam	0.9817
0.4	20	0.2	32	adagrad	0.9820
0.4	20	0.2	32	rmsprop	0.9827
0.4	10	0.2	32	adam	0.9826*
0.4	20	0.2	32	Adam (lr=0.0005)	0.9825
0.4	20	0.2	32	Adam (lr =0.0003)	0.9827

*with GridSearch (splits = 1, validation set: 10%)

Table m: optimizer: adadelta, number of filters: 32, activation: tanh

runs	Sdroprate	epochs	droprate	regions	score
1	0.4	25	0.2	[3,4,5]	0.9824
1	0.4	10	0.2	[2,3,4]	0.9819
1	0.4	20	0.2	[2,3,4]	0.9829
1	0.4	25	0.2	[2,3,4]	0.9830
1	0.4	25	0.1	[2,3,4]	0.9828
1	0.5	25	0.2	[2,3,4]	0.9829
1	0.4	25	0.2	[2,3,5,7]	0.9828
1	0.4	25	0.2	[2,3,4,5]	0.9830
1	0.4	30	0.2	[2,3,4,5]	0.9828
1	0.4	30	0.2	[2,3,4]	0.9829
2	0.4	25	0.2	[2,3,4]	0.9828
2	0.4	25	0.2	[2,3,4,5]	0.9829
3	0.4	25	0.2	[2,3,4]	0.9830
3	0.4	25	0.2	[2,3,4,5]	0.9829
4	0.4	25	0.2	[2,3,4]	0.9830
4	0.4	25	0.2	[2,3,4,5]	0.9829