STA 208 FINAL PROJECT

Jigsaw Bias in Toxicity Classification

-A Kaggle Competition Project

Hemiao (Amy) Cui

HIGHLIGHTS:

1. In this project, total of >10 GB data was manipulated, which included two pretrained embedding files.
2. Majority of the work was conducted using AWS EC2 Deep Learning AMI (Ubuntu) Version 23.0 GPU instance, which costs 1.5 $/hour.
3. Key findings include identification of simpler algorithm to achieve high prediction outcome.

I. INTRODUCTION

In a previous Kaggle competition, models were built to recognize toxicity in comments. However, these models often incorrectly associate several words to toxicity. The model would tag comment containing specific words as toxic even though the comment is not toxic. The goal of this project is to detect toxic comment and at the same time minimize such unintended bias. For example, if a sentence is "I am a gay woman", our algorithm should detect it as neutral rather than toxic even though it contains the word “gay”.

II. DATA DESCRIPTION

Here are some facts about the data used in this project:

Data	Size	Source
Train	816 MB	https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification/data
Test	7.2 MB	Same as above
GloVe embedding	6 GB	https://nlp.stanford.edu/projects/glove/
Paragram embedding	5 GB	https://cogcomp.org/page/resource_view/106

The training dataset mainly includes following columns:

Variable names	Variable Annotation	Column content description
Id	ID of text	Will not be used
toxicity label	Target	Value between 0 and 1; value >=0.5 is considered as positive (toxic); highly unbalanced with the majority of class being nontoxic
Comment_text	Text content	An example: This is so cool. It's like, 'would you want your mother to read this??' Really great idea, well done!

Male/female/ homo_sexual_gay/ homo_sexual_lesbian/Christian/Jewish/Muslim/ Black/ white/Psychiatric_ /Mental illness	Major identity attributes	Value between 0 and 1 where 1 means annotators believe that the text has mentioned the identity

Here, the response variable is the target variable. In addition to above columns, the training data set also has columns on annotator count as well as civil comments. But, they are not available in test set and thus will not be used for prediction.

The toxicity label, as well as identity attribute labels, are obtained by aggregating ratings from annotators. More specifically, annotators were asked to indicate all identities that were mentioned in the comment. The answers were aggregated into fractional values representing the fraction of raters who said the identity was mentioned in the comment.

III. METHODS

3.1. Preprocessing

3.1.1. Text Manipulation

GloVe (glove.840B.300D) words embedding and Paragram (paragram_300_sl999) embedding were used in this project to obtain vector representations of words. The coverage of these embeddings were first check toward the vocabulary in the dataset. To increase the coverage of these embeddings, text manipulation needs to be done. These include:

- Lowering all capitals (e.g. “That” to “that”)
- Eliminating contractions (e.g. “aren’t” to “are not”)
- Eliminating punctuations (e.g. ‘*#)
- Spelling correction (e.g. “whst” to “what”)
- Changing ambiguous word (e.g. “Drumpf” to “Trump”)
- Removing numbers

3.1.2. Meta-embedding

Meta embedding were constructed with the weight of 0.7 for GloVe and 0.3 for Paragram.

3.1.3. Feature Engineering

Finally, in the preprocessing step, feature engineering was done. This includes adding a Text Blob polarity variable, subjectivity variable, unique word count and total word count for every comment in both train and test sets. Polarity is a variable in a sentiment analysis, ranging from -1 to 1, where -1 indicates a negative statement and 1 indicates a positive statement.

3.2. Initial Model Fitting

Before model fitting, train set was split into training set and validation set by 4:1 ratio. Cross-validation was not considered because the data set is very big.

3.2.1. Bag of word model

Texts were first stemmed and lemmatized since Tf-idf transformation will not conduct lemmatization process. Then, tokens were converted into features using Tf-idf transformation. Light-GBM was used for model fitting as it is high speed and good accuracy proven in various competitions. GridSearchCV was used for hyperparameter tuning. ROC curve and PR curve were plotted using optimal tuning parameters.

3.2.2. CNN (Convolutional Neural Network)

Before deep learning, data was first converted to embedding matrix, which serves as a look-up table of position of a word in hyperdimensional space. using pre-trained embedding, specifically GloVe for CNN. Besides, texts were padded to have equal length. A classical vanilla CNN model was fitted. The layers include:

Input layer
Embedding layer/GloVe
Convolution 1 D
Max Pooling l D
Convolution 1 D
Max Pooling l D
Convolution 1 D
Max Pooling l D
Flatten
Fully connected layer
Fully connected layer

3.2.3. RNN (Recurrent Neural Network)

A vanilla RNN model (id: rmodel_r1) was first fitted as shown below. Model fitting was conducted with 5 epochs.

Input layer
Embedding layer/GloVe
LSTM layer
Dropout layer
Dense layer

3.3. Initial Model Comparison

In this project, models are compared using the following self-defined metric[1]:

$$w_0 * AUC_{overall} + \sum_{a=1}^{a=3} w_a * M_p(m_{s,a})$$

$$M_p(m_s) = (1/N * \sum_{s=1}^{N} m_s^p)^{1/p}$$

where $AUC_{overall}$ refers the ROC-AUC for the full evaluation set; p = -5;
$m_{s,a}$ refers to bias for identity subgroup *s* using sub metric *a*; N is number of identity subgroups which corresponds to identity attributes as mentioned in Data description part;
$w_0 = w_1 = w_2 = w_3 = 0.25$; They are the weights for individual sub metric.

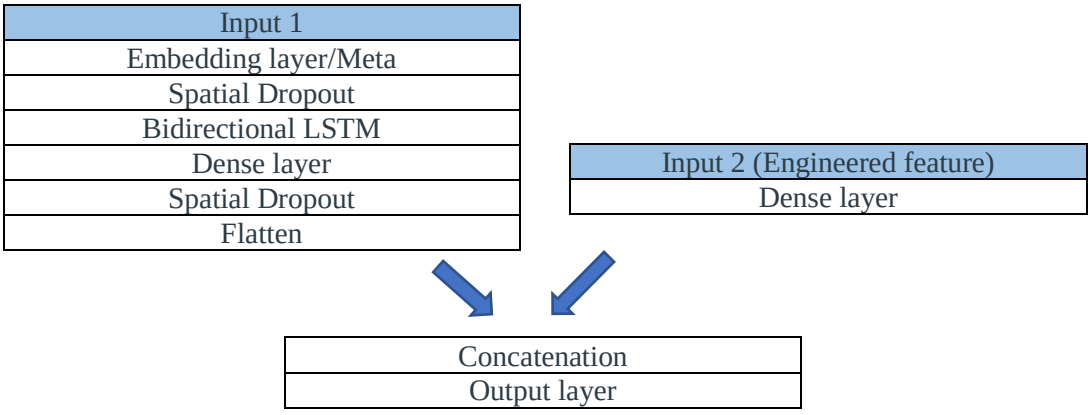
There are 3 sub-metrics, which include:

- Subgroup AUC: AUC for the dataset restricted to specific identity subgroup;
- BPSN (Background Positive, Subgroup Negative) AUC: AUC for the test set restricted to the non-toxic examples that mention the identity and the toxic examples that do not.
- BNSP (Background Negative, Subgroup Positive): AUC for the test set restricted to the toxic examples that mention the identity and the non-toxic examples that do not.

3.4. RNN Model tuning

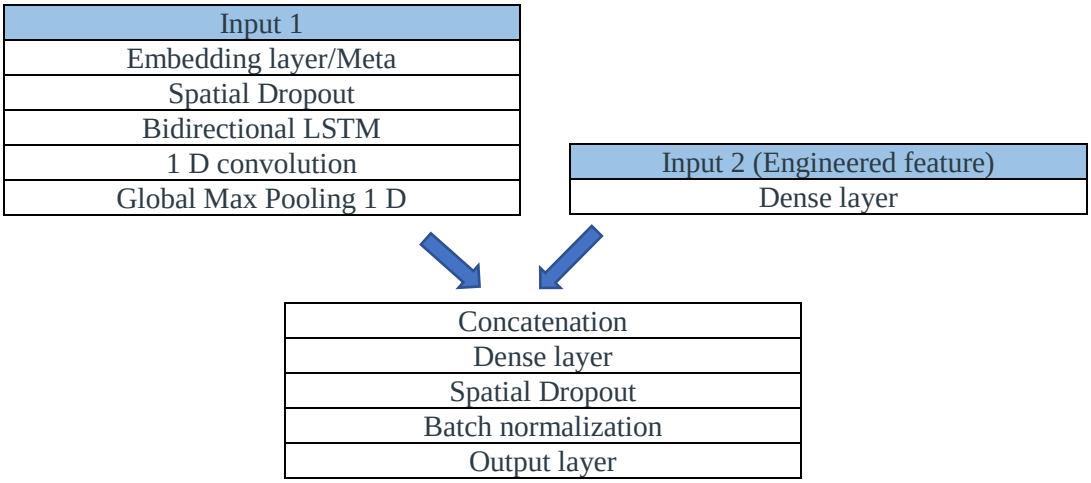
Since for Light-GBM, optimal hyperparameters were already identified and applied, model tuning was conducted only for RNN. The following three models were considered:

3.4.1. Advanced RNN (model id: model_r2)
The major changes from vanilla RNN is the use of meta embedding, bidirectional RNN as well as the addition of engineered features. Besides this, gradient clipping was also used, which can prevent exploding gradient. Specifically, gradient norm scaling was used, which adjusted the derivatives of the loss function when the norm exceeds a threshold value of 0.1.



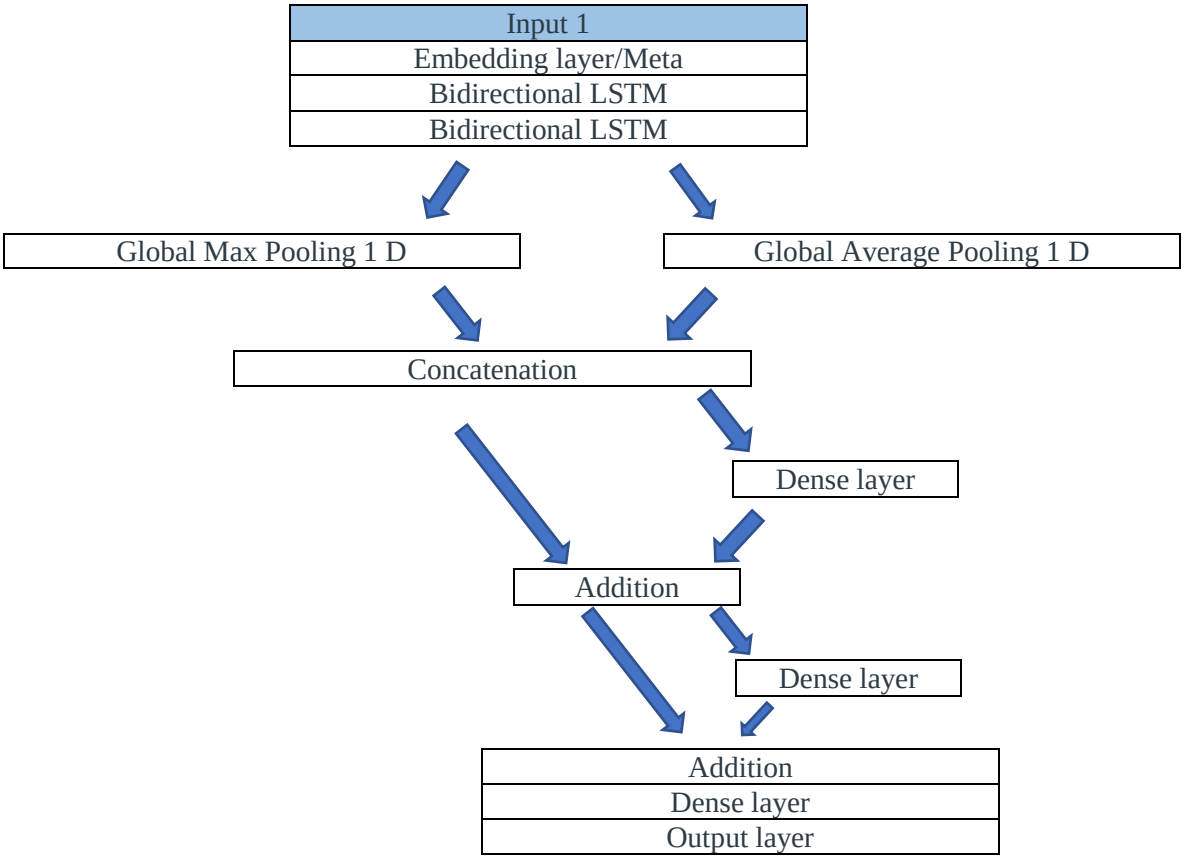
3.4.2. Complex RNN (model id: model_r3)

It has suggested from earlier competition that adding convolution after LSTM (Long Short-Term Memory) may improve model performance[2]. Inspired by it, a more complex model architecture was evaluated.



3.4.3. Complex RNN (model id: model_r4)

This model architecture was inspired by a population kernel[3]. It was used here as a positive control to compare with the vanilla RNN models to see if it is really needed to use such a complex architecture.



IV. RESULTS AND DISCUSSION

4.1. Preprocessing

The table below shown coverage improvement after text manipulation. 'vocab' is the percentage of words that are existing in the pre-trained embedding. Meanwhile 'all text' is the percentage of comments (row of data) that is covered by the embedding. Here, not all vocabulary in a comment is covered by the embedding, which is indicated by the lower percentage of 'vocab' than 'all text'.

A.

Methods	Vocab	all text
No treatment	15.519%	89.608%
Lowering capitals	15.638%	89.637%
Eliminating contractions	13.506%	90.394%
Eliminating punctuations	54.207%	99.723%
Misspells and other ambiguous texts	54.223%	99.739%

B.

Methods	vocab	all text
No treatment	8.796%	78.632%
Lowering capitals	8.796%	78.632%
Eliminating contractions	13.699%	90.399%
Eliminating punctuations	57.633%	99.739%
Misspells and other ambiguous texts	57.651%	99.754%

Table 1. Coverage check for GloVe (A) and Paragram (B)

From the table, the coverage significantly improved after punctuation removal. Paragram embedding was also improved significantly after contraction elimination. After cleaning, the final coverage obtained were close between the two embeddings. It should also be noted that capital lowering did not affect the Paragram embedding, since it already contained both words starting with and without capital letter.

Besides feature engineering, data augmentation was also tried by mapping English to French and then back to English using Text Blob and Googletrans. But, neither methods worked out for the entire train and test sets, which may due to the limit of API request.

4.2. Initial Model Fitting

4.2.1. Light GBM

Figure 1 shows the ROC curve and PR curve for Light-GBM. A good ROC curve should be very curved towards the left-top corner, as Figure 1 A shows. A good PR curve should be curved towards the right-top corner. In this case, the ROC is close to optimal, but the PR is sub-optimal. It has been suggested that the PR curve is more sensitive to unbalanced data[4], which is the case in this data set. The overall AUC achieved is 0.92.

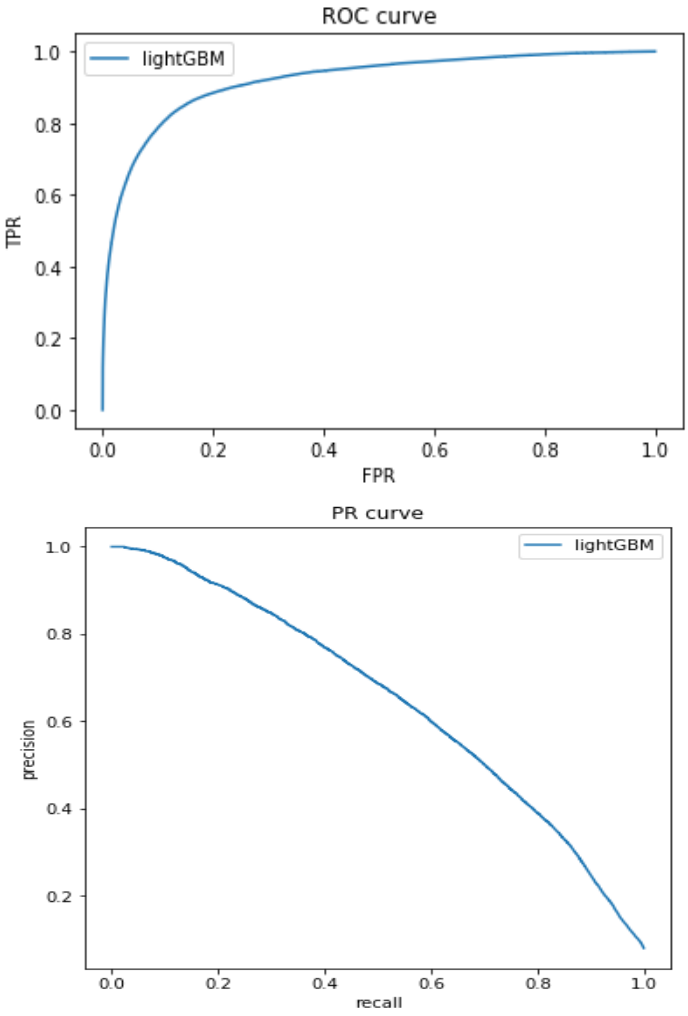


Figure 1. ROC and PR curve for Light GBM

4.2.2. CNN

Only around 0.08 accuracy for train and validation sets were obtained after 1 epoch and thus CNN will not be further considered.

4.2.3. RNN:

The vanilla RNN model (id: rmodel_r1) was first tried. Figure 2 shows that the training loss is decreasing over epochs and it has kind of reached plateau at the end of 5th epoch. The validation loss, on the other hand, did not change over epochs.

Since the training loss is very close to validation loss, we consider this as just right fitting, meaning not overfitting or underfitting. It is weird that the validation loss is not decreasing over epochs. Also, the accuracy in validation set is also slightly higher than the training set, which may due to the fact that much less amounts of errors are averaged together since the validation set is much smaller.

Figure 2. Loss and accuracy for RNN rmodel_r1 over 5 epochs

4.3. Initial Model Comparison

The Light-GBM and RNN vanilla model (model_r1) are compared based on the five metrics: overall AUC (labelled as auc), Subgroup AUC (mp-sub), BPSN AUC (mp-bpsn), BNSP AUC (mp-bnsp), and Average AUC for the previous four AUC outcomes (avg-auc). The average AUC is the most important metric here as it is metric for the Kaggle competition. Figure 3 shows that the Light-GBM obtained much better result than the vanilla RNN model. It has achieved 0.91 overall AUC and 0.85 average AUC. It can be suggested that traditional model like the Light-GBM should always tried as the baseline for deep learning model. Also, a deep learning model may depend on far more factors to tune.

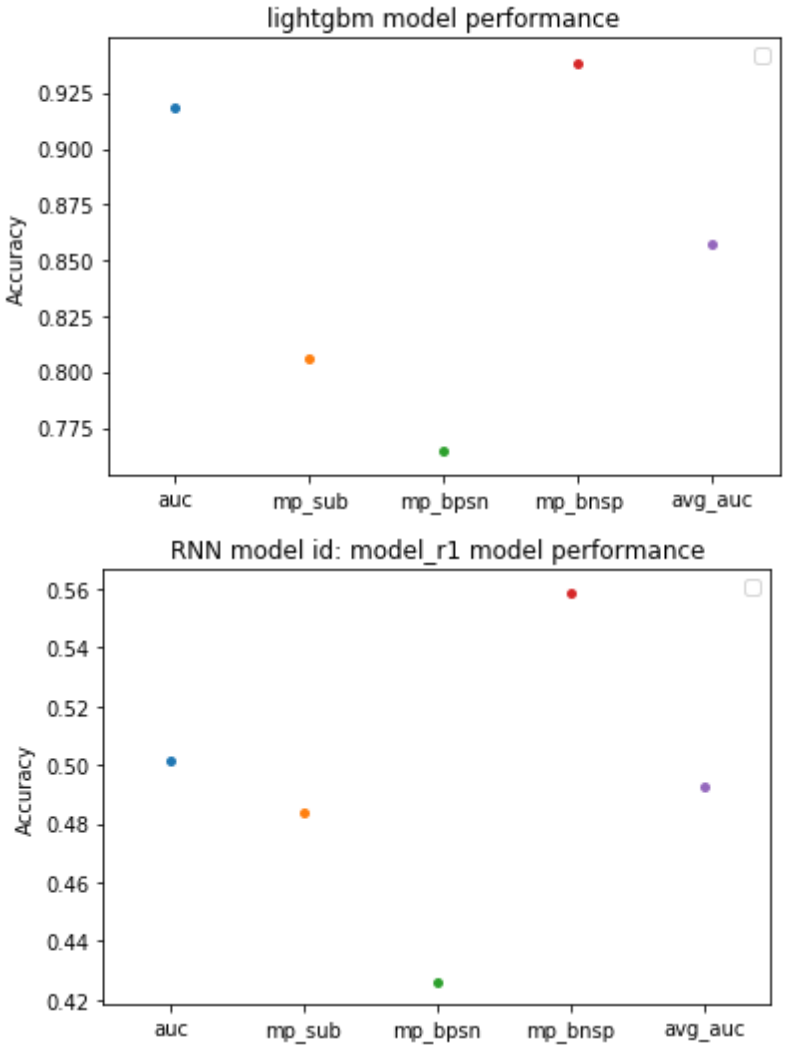
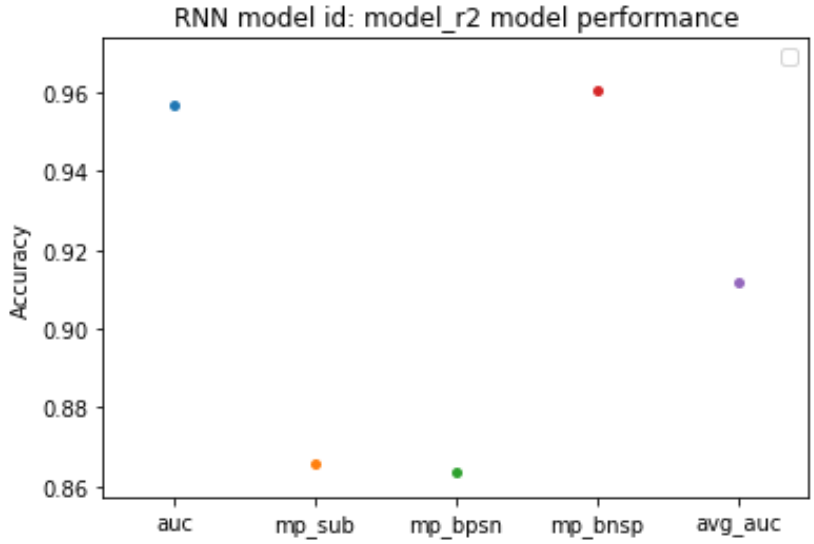
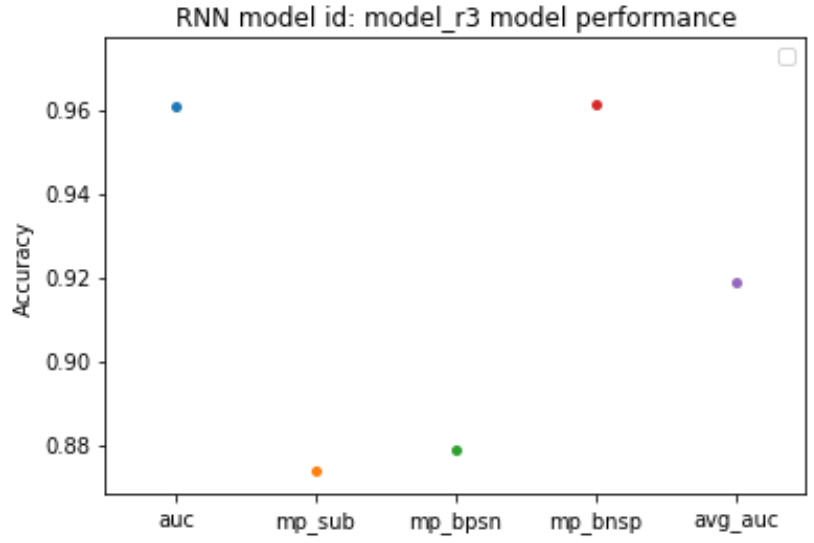


Figure 3. Accuracy comparisons for Light-GBM vs RNN model_r1

4.4. RNN Model Tuning

Here, RNN tuning was focused on the architecture, input, embedding system and the addition of gradient clip. Figure 4 showed the performances for model_r2, model_r3 and model_r4. In model_r2, changes include: adding engineered features; using meta embedding, using bidirectional RNN and applying gradient clipping. There is not much increase in terms of number of layers. It is surprising, though, that a significant performance has been obtained without going into sophisticated architecture. The average AUC has achieved around 0.91. Model_r3 used the architecture from the past Kaggle 1st place winner. It has slightly increased average AUC as compared to model_r2. The major improvement of Model_r3 as compared to Model_r2 was with respect to the BPSN (Background Positive, Subgroup Negative) AUC. As for Model_r4, it improved a little bit on sub group AUC as compared to Model r3. But, model_r4 was based on a very complicated architecture with significantly longer run time (26 mins).


RNN model id: model_r2 model performance
Accuracy
0.96
0.94
0.92
0.90
0.88
0.86
auc
mp_sub
mp_bpsn
mp_bnsp
avg_auc


RNN model id: model_r3 model performance
Accuracy
0.96
0.94
0.92
0.90
0.88
auc
mp_sub
mp_bpsn
mp_bnsp
avg_auc

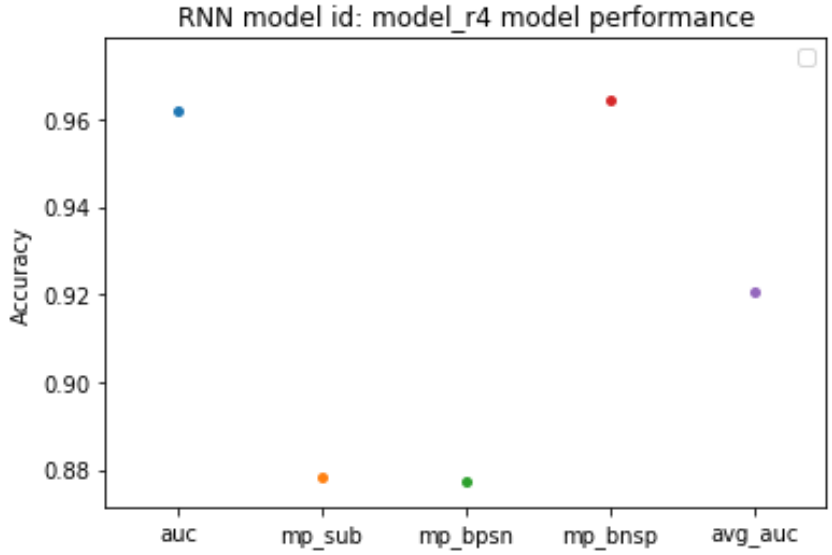

RNN model id: model_r4 model performance
Accuracy
0.96
0.94
0.92
0.90
0.88
auc
mp_sub
mp_bpsn
mp_bnsp
avg_auc

Figure 4. Accuracy comparisons for RNN model_r2, model_r3 and model_r4

In all models, MP-sub and MP-BPSN were the bottle-neck metrics, which reflected the fact that it was challenging to avoid biases. But, the algorithm still reached AUC for Avg-AUC when cross validation was conducted.

V. CONCLUSIONS

Challenges encountered in this project includes, but not limited to:

- Setting up Jupyter notebook for EC2 GPU instance;
- Re-sizing EC2 volume limit due to large data sets;
- Dealing with memory error

Meanwhile, the major findings of this work are:

a. In the preprocessing stage, punctuation removal and eliminating contractions are especially important for improving coverage. In the future, more preprocessing techniques may be considered, such as text normalization and noise removal.
b. Adding engineered features; using meta embedding, using bidirectional RNN and applying gradient clipping significantly improved model performance.
c. Traditional model like light GBM should always be used as baseline for deep learning.
d. Model_r3 (complex RNN) is identified to the optimal model with improvement on various metrics without significant sacrificing on the runtime.
e. Identified Model_r2 and Model_r3 as NOVEL architectures proven to have good performance on this particular data sets to classify the comment with low bias when words contain sensitive tokens as seen in the sub metric outcomes.

REFERENCES

1. Jigsaw. (2019). *Jigsaw Unintended Bias in Toxicity Classification*. Retrieved from: https://www.kaggle.com/c/jigsaw-unintended-bias-in-toxicity-classification/
2. Agarwal, R. (2019). *What my first Silver Medal taught me about Text Classification and Kaggle in general?* Retrieved from: https://mlwhiz.com/blog/2019/02/19/siver_medal_kaggle_learnings/
3. Thousandvoices. (2019). *Simple LSTM*. Retrieved from: https://www.kaggle.com/thousandvoices/simple-lstm
4. Saito, T. & Rehmsmeier, M. (2015). The Precision-Recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLoS One* 10(3): e0118432.