Washington University in St. Louis

OLIN BUSINESS SCHOOL

Deep Learning for Prediction of Business Outcomes

Final Project
Toxic Comments Prediction for Social Media Platform

Project Group 53

James Liao (MSBA)

December 21, 2021

Zoom: https://drive.google.com/file/d/1r4byiM_uRztmk9KJsdVrubIXM4kDlDH_/view?usp=sharing

Content

Abstract

Research about toxic comments detection is increasing. Different from negative comments, toxic comments stand for aggressive and hated words that stemmed originally from people's malice. Several social media platforms are actively preventing the prevalence of those inappropriate reply on the website because those comments are lowering the advertisement interest from other companies. In this project, I aim develop a deep learning model to detect the toxic comments for social media platform to deal with the toxic comments problem. I establish three deep learning models for comparison: simple CNN, CNN-RNN, and CNN-LSTM. The final results show that CNN-LSTM model has the highest accuracy and AUC score, with respective of 0.9077 and 0.9684.

Introduction

NLP has been promoted to improve customer service and product service by assisting Know Your Customer (KYC) process. Amazon has applied this technique to filter out potential problematic products by screening product reviews; Financial institution also extract comments and reviews on social media to predict the public trend in stock market. Every day, social media platforms such as Facebook, Twitter consuming over 18 billion text messages to connect people worldwide. However, social media platform is currently facing the problem of ignoring or spreading hated comments. A great number of toxic texts is growing in recent years and keeps making the loss for some social media platforms. The main loss comes from the Non-cooperation movement from advertisers and company due to their unwilling to post their advertisement on the platform that has negative corporate image. To solve this existing business problem, our goal is to build an advanced deep learning-based model to detect the problematic words. I will use the toxic database made from Google Jigsaw Team and establish three models using deep learning approaches. Specifically, I will build simple CNN, CNN-RNN, and CNN-LSTM models for accuracy and AUC score comparison.

Literature Review

CNN application in text classification

Convolutional Neural Network (CNN) is another prominent deep learning technique in processing image and NLP task. The main reason of selecting CNN model is that it can automatically detect the important features efficiently, which allows fewer weights to considered as the parameters. Previous literatures have demonstrated the power of using CNN model as the text classification tool. Sharma, Chaurasia et, al (2020) construct a CNN model with fine-tuned Word2Vec to categorize sentimental short sentences. The proposed model provides 0.99 of accuracy for the text class prediction. Chen, Hasan, et, al (2019) build an effective CNN model to solve text classification at a multi-institutional scale. The generative model results in 0.99 in F1-score, showing the validation of the CNN model in dealing with features engineering and NLP task.

RNN application in text classification

Recurrent neural network (RNN) is a type of Artificial Neural Network (ANN) that connections between nodes are structured to a graph by a sequence. RNN is useful when model is under the time process or the situation such that previous value is considered to the next value. In brief, when the model makes the decision, it considers the current input as the variables and weighted with the previous inputs that has learned from the model. Its application has proved efficient when processing text sequences by several academic research papers. Wang, Hsiao, et, al (2020) developed a sentence generator module using simple RNN structure to solve the problem of difficult and high standard writing requirements in academic research. The three hidden layers RNN model results in the high accuracy in generating sentences from selecting keywords, with average accuracy 0.86, precision rate 0.91, and recall rate 0.78. Banerjee, Ling, et al (2019) establish an advanced information extraction system by applying RNN method. The model can precisely encode the sentenced-level text from medical services and generates diagnostic reports. The final F1 score for the RNN falls between 0.91 ~0.97, stating the strong efficiency of RNN framework in solving text classification/ prediction problem.

LSTM application in text classification

Long Short Term Memory Network is an advanced type of RNN. It has achieved a great success in Natural Language Processing (NLP). One of its advantages is that it can handle the vanishing gradient problem in RNN. In other words, LSTM is designed to avoid the long-term dependency problem that makes model struggle to learn properly. Research has proved that LSTM is a huge step in improving text classification and prediction when compared to simple RNN model. Sari, Rini, et, al (2019) apply LSTM model to classify multi-label text from news reviews. LSTM model effectively solve the problem of high dimensional inputs data that traditional neural network model is facing. The final results show the accuracy of 0.951, and both average precisions, recall rate are 0.95. Li, Wang, et al (2018) use combined LSTM model to fulfill Chinese news sentiment classification. The accuracy also reaches 0.92 for the LSTM model, compared to 0.87 in RNN model. In short, LSTM model potentially performs better than simple RNN model.

Brief Summary in literature review:

1. CNN is a useful method for extracting data features. I consider it as the input layers in this project
2. Applying pre-trained Fast Text embeddings generates better results and save the training time. In this project, I select wiki-news-300d-1M.vec for word embedding matrix source.
3. LSTM and RNN are especially efficient in classify text sequence. I apply mix models, which are CNN-LSTM and CNN-RNN in this project

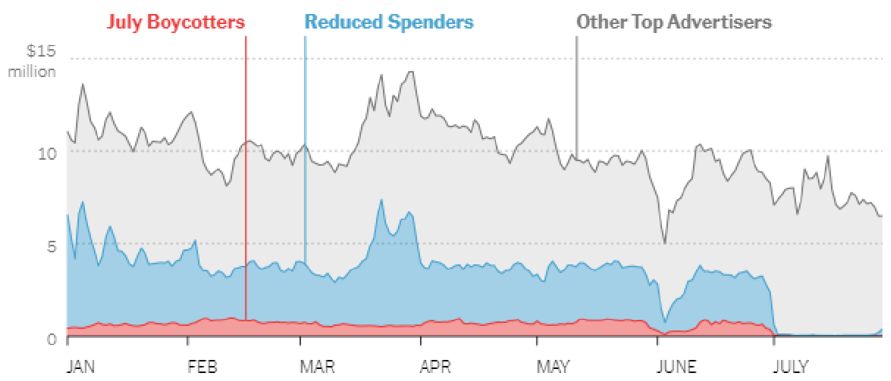
Problem Description

Toxic Text is more harmful than people thought. As google NLP team Jigsaw announce: Toxicity online isn't just an issue of civility and healthy discourse. It poses a threat to publishers, platforms, and society. Social media making money by selling ads. Those ads are the investment activity by advertisers or enterprises. Few months ago, several companies just announced they will stop collaborating with Facebook due to its promotion and ignore of spreading hated comments. The event also called 2021 July Boycott, or #StopHateForProfit. From the Figure (1), it is obvious to find that almost 50% of existing advertisers stop using Facebook to expose or advertise their products.

Case Study1: **Instagram's Rolling Out New Tools to Remove "Toxic Comments".**

Case Study2: **More Than 1,000 Companies Boycotted Facebook. Did It Work?**

Figure (1) Estimated Spending Trend in Facebook's Top 100 Advertisers



Note: "Reduced spenders" are companies that did not officially announce boycotts, but decreased their spending in July by at least 90 percent compared to June. • Source: Pathmatics • By Eleanor Lutz

Social media platforms have responsibility to provide a reliable and healthy communication environment. No matter how people think of speech freedom, social media must prevent supporting language abuse or promoting discrimination. By monitoring those unnecessary aggressive text and filtering those negative language usage, social media platforms can appeal more advertisers and companies to collaborate, and eventually earn more profit and respectful social status at the same time.

Database Background and Data Preprocessing

Dataset Background and Descriptive Statistics

The dataset is about text comments and their severity. The data source is from Kaggle competition held by Google Jigsaw team. The comments are collected from Wikipedia toxic comments randomly. The dataset has 159571 rows, including columns: “text_id”, “comment_text”, “toxic”, “severe_toxic”, “obscene”, “threat”, “insult”, “identity_hate”. The last six columns stand for the severity category of each text.

Figure (2) Numbers/Comments Length

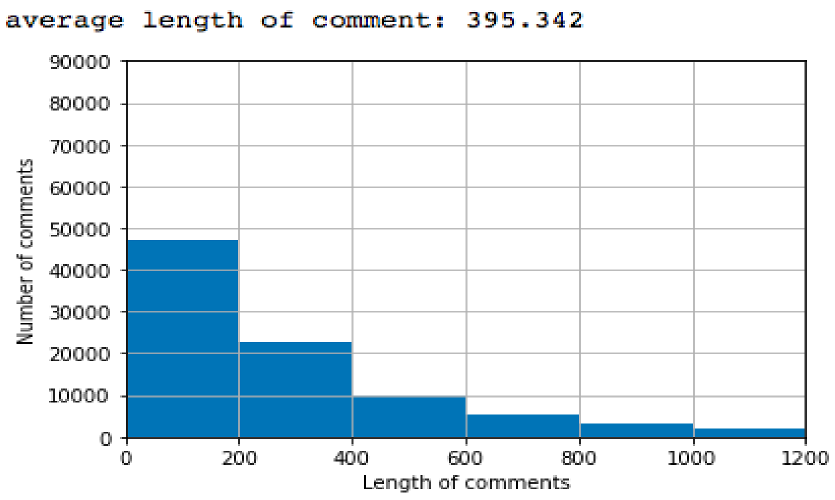


Figure (3) Numbers of Each Class

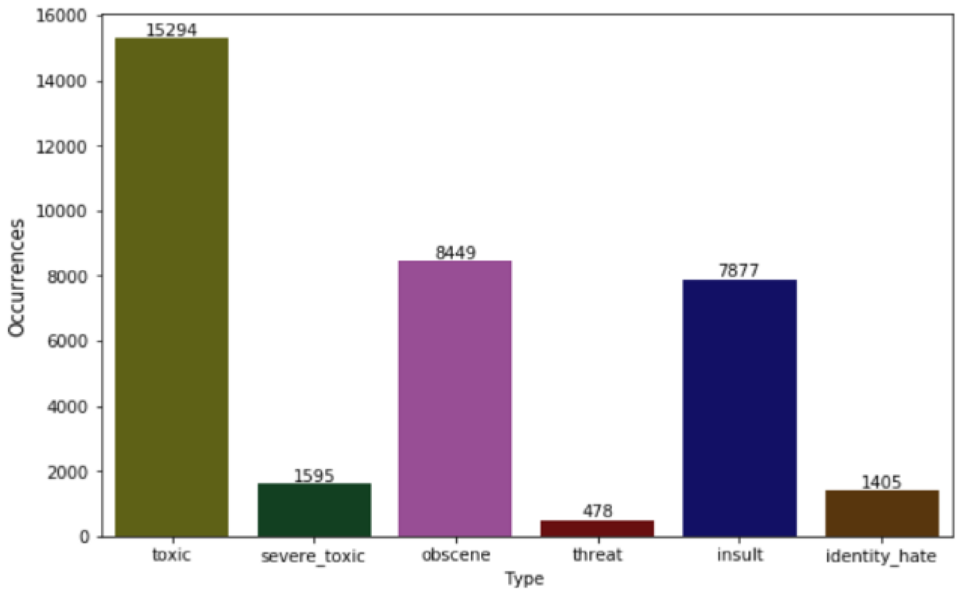
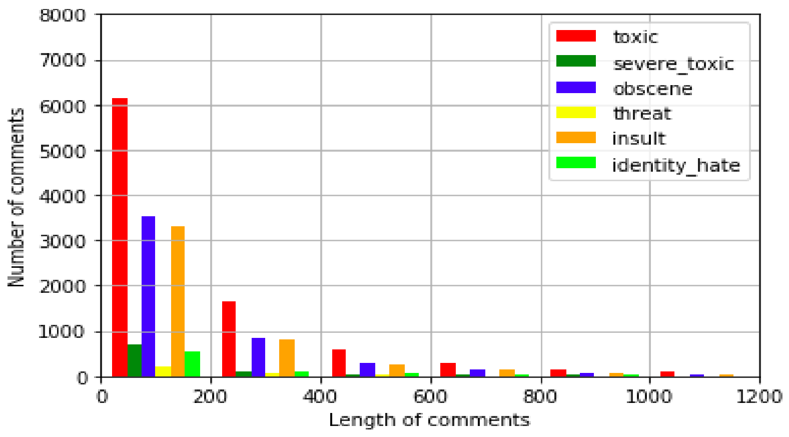


Figure (4) Numbers of Each Class/Length (Training)



From the figures displayed above, I can find that most comments have length below 200 words, and the number of toxic comments outweighs that of other categories. For the sentence below 200 words, top three classes are: toxic, obscene, and insult. For the sentence over 400 words, there is no obvious discrepancy in number of the comments among all categories. As shown in figures, the data has imbalanced data distribution. After viewing the data, it is obvious to find that the comments that has more than 400 words typically are not the aggressive comments. The data is collected from the Wikipedia reviews that includes tons of unnecessary comments such as product descriptions, facts, etc. To deal with this problem, I will have our data preprocessed by under sampling method to further balance the dataset. Also, I will exclude the text that is over 400 words.

Data Preprocessing

I first use stemming and lemmatizing function from NLTK package to normalize the text into lower case and original form. Subsequently, I remove all punctuation in the text. Studies show that removing number and punctuation will increase the prediction accuracy of the text classification. Afterwards, I start word embedding and create embedding matrix.
Here, I follow the steps from Sharma, Chaurasia et, al (2020), using the pre-trained vector and generate embedding matrix. I select wiki-news-300d-1M.vec as the embedding source. The training process will follow this embedding matrix as the weighted matrix and finally optimize the model. Afterwards, I can start design the deep learning models. For the train test split process, I exclude biased data such as comments over 400 words to increase the precision of the model.

Figure (5) Data Pre-processing Steps

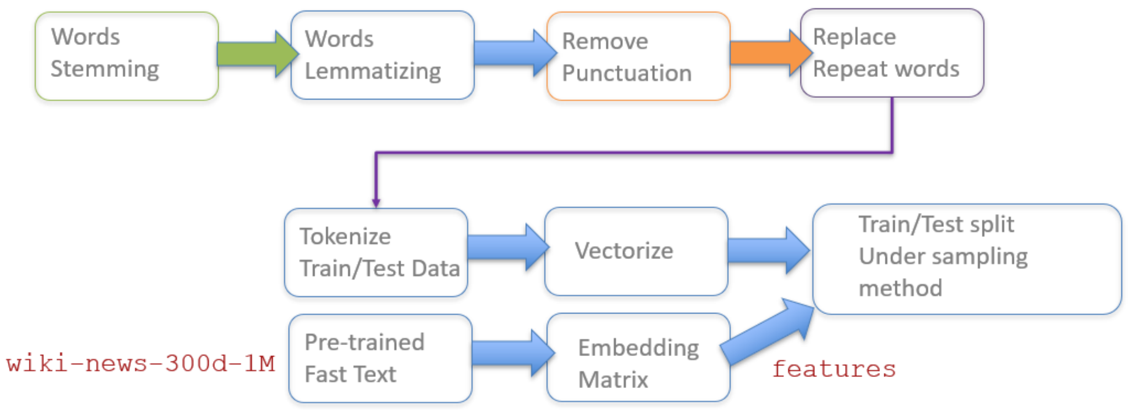
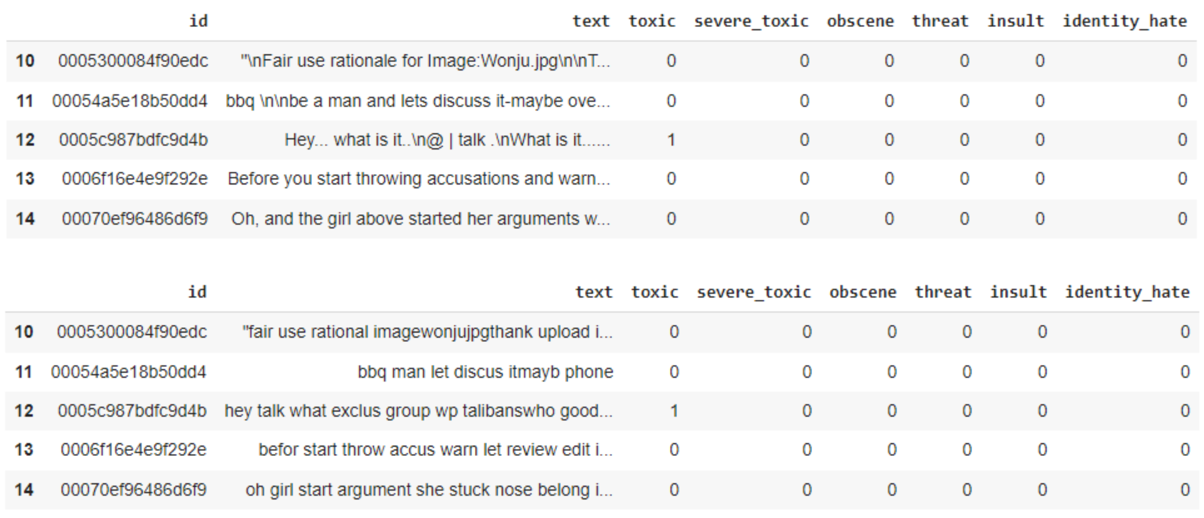


Figure (6) Data Pre-Process- Before vs After



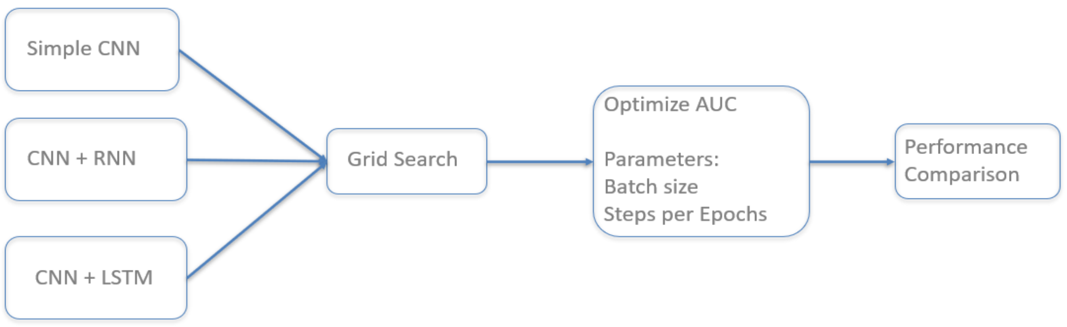
	id	text	toxic	severe_toxic	obscene	threat	insult	identity_hate
10	0005300084f90edc	"\nFair use rationale for Image:Wonju.jpg\n\nT...	0	0	0	0	0	0
11	00054a5e18b50dd4	bbq \n\nbe a man and lets discuss it-maybe ove...	0	0	0	0	0	0
12	0005c987bdfc9d4b	Hey... what is it..\n@ | talk .\nWhat is it......	1	0	0	0	0	0
13	0006f16e4e9f292e	Before you start throwing accusations and warn...	0	0	0	0	0	0
14	00070ef96486d6f9	Oh, and the girl above started her arguments w...	0	0	0	0	0	0

	id	text	toxic	severe_toxic	obscene	threat	insult	identity_hate
10	0005300084f90edc	"fair use rational imagewonjujpgthank upload i...	0	0	0	0	0	0
11	00054a5e18b50dd4	bbq man let discus itmayb phone	0	0	0	0	0	0
12	0005c987bdfc9d4b	hey talk what exclus group wp talibanswho good...	1	0	0	0	0	0
13	0006f16e4e9f292e	befor start throw accus warn let review edit i...	0	0	0	0	0	0
14	00070ef96486d6f9	oh girl start argument she stuck nose belong i...	0	0	0	0	0	0

Model Building

In this project, I use 70% for training and 30% for testing on modified dataset. I select three prominent deep learning algorithms based on the performance in research literature. I design three neural network related models, including RNN, LSTM, and CNN. I apply grid search for hyperparameters tuning in batch sizes [30,35,40,45,50,55] and steps per epochs [20,30,50,100,150] in order to find out the optimized hyperparameters in each model. I also include the call backs function to save the best model in testing AUC score.

Figure (7) General Modeling and Evaluation Steps

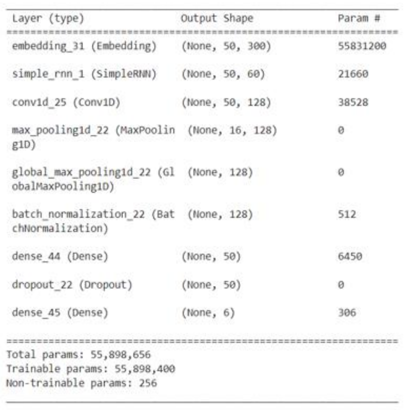


CNN Based Model – CNN/ CNN+RNN/ CNN+LSTM

According to the literature reviews, adding maximum pooling filter is a good operations to filter features while training, because it calculates the largest value in each patch of each feature map. It is especially useful when processing text sequences. The batch normalization is a good method to standardize inputs into mini–batch to avoid explosion problem when calculating neural network. I also include the dropout method to prevent overfitting problem. Finally, I consist the connected layers with 50 nodes and the output layer with 6 nodes. The final the activation function for output is 'sigmoid'. For the CNN-RNN and CNN-LSTM models, I add second hidden layer based on simple CNN model. For the CNN-RNN and CNN-LSTM, I select the nodes of 60 based on the literature reviews. Our models include multiple hidden layers and convolution layers; thus, it is better to select limited nodes rather than huge size of input number. The structure of the model is displayed in following:

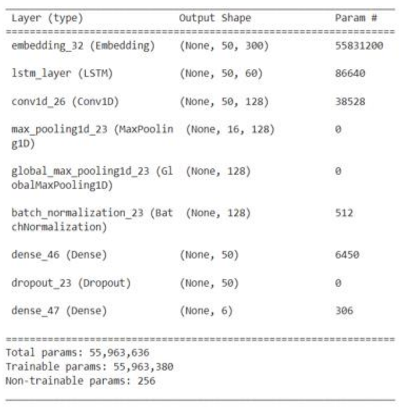
Figure (8) Final Models Comparison

CNN + RNN



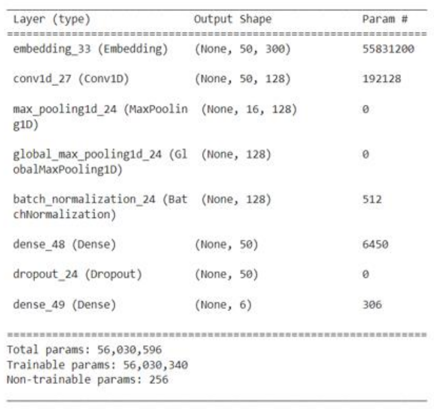
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_31 (Embedding)     (None, 50, 300)           55831200

simple_rnn_1 (SimpleRNN)     (None, 50, 60)            21660

conv1d_25 (Conv1D)           (None, 50, 128)           38528

max_pooling1d_22 (MaxPoolin  (None, 16, 128)           0
g1D)

global_max_pooling1d_22 (Gl  (None, 128)               0
obalMaxPooling1D)

batch_normalization_22 (Bat  (None, 128)               512
chNormalization)

dense_44 (Dense)             (None, 50)                6450

dropout_22 (Dropout)         (None, 50)                0

dense_45 (Dense)             (None, 6)                 306

=================================================================
Total params: 55,898,656
Trainable params: 55,898,400
Non-trainable params: 256
```

CNN + LSTM



```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_32 (Embedding)     (None, 50, 300)           55831200

lstm_layer (LSTM)            (None, 50, 60)            86640

conv1d_26 (Conv1D)           (None, 50, 128)           38528

max_pooling1d_23 (MaxPoolin  (None, 16, 128)           0
g1D)

global_max_pooling1d_23 (Gl  (None, 128)               0
obalMaxPooling1D)

batch_normalization_23 (Bat  (None, 128)               512
chNormalization)

dense_46 (Dense)             (None, 50)                6450

dropout_23 (Dropout)         (None, 50)                0

dense_47 (Dense)             (None, 6)                 306

=================================================================
Total params: 55,963,636
Trainable params: 55,963,380
Non-trainable params: 256
```

CNN



```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_33 (Embedding)     (None, 50, 300)           55831200

conv1d_27 (Conv1D)           (None, 50, 128)           192128

max_pooling1d_24 (MaxPoolin  (None, 16, 128)           0
g1D)

global_max_pooling1d_24 (Gl  (None, 128)               0
obalMaxPooling1D)

batch_normalization_24 (Bat  (None, 128)               512
chNormalization)

dense_48 (Dense)             (None, 50)                6450

dropout_24 (Dropout)         (None, 50)                0

dense_49 (Dense)             (None, 6)                 306

=================================================================
Total params: 56,030,596
Trainable params: 56,030,340
Non-trainable params: 256
```

Figure (9) Grid Search AUC – CNN + LSTM

Steps/ Batch	30	35	40	45	50
20	0.93	0.92	0.92	0.91	0.93
30	0.92	0.94	0.94	0.92	0.89
50	0.94	0.968	0.965	0.93	0.88
100	0.85	0.91	0.88	0.89	0.9
150	0.82	0.82	0.83	0.88	0.87

Results Evaluation and Discussion

Model Evaluation Indicators

AUC

AUC (Area Under Curve) represents the area under the ROC curve and can represent a commonly used statistical value of the predictive ability of the classifier. The closer the ROC curve is to the upper right, the better. Therefore, the larger the area under the ROC, the higher the efficiency and predictive power of the model. It can be interpreted as the follow:

- AUC = 1: The representative classifier is perfect, but this is an ideal situation that is almost never achieved in real world.
- AUC > 0.5: It means that the precision of the classifier is better than random guessing, thus the model has predictive value.
- AUC = 0.5: The precision of classifier is the same as random guessing, thus the model has no predictive value.
- AUC < 0.5: It means that the precision of the classifier is worse than random guessing, but if applying reverse prediction, it will achieve the result of AUC > 0.5.

Accuracy

Accuracy is equal to the number of correct predictions (including whether the correct prediction is positive or negative) divided by the total number of predictions (including whether the correct prediction is positive or negative, and the wrong prediction is positive or negative). From the formula, it is (TP+TN)/(TP+TN+FP+FN). Basically, the higher the accuracy, the better the model is.

Comparison and Discussion

Based on the figures below, the results show that all three models have explicit improvement in both AUC score and accuracy as the number of epochs increases. For the simple CNN model, the marginal effect of AUC decreases when the epoch reaches to 8. However, its accuracy increases slowly with the epochs and only reach almost 0.5 in the figure (11). On the contrary, the AUC score of CNN+RNN model has precipitous slope in accuracy trend at the beginning, after a small drop then quickly bounce back to 0.8 level of accuracy. Its AUC score keeps relatively constant as the epochs reach to 30. For the CNN+LSTM model, it sharply increases both in AUC score and accuracy as the model run through epochs. To be noticed, both AUC and accuracy in CNN+LSTIM model are above other two models in most of the time, demonstrating its strong predictive power and model efficiency. Figure (12) shows the maximum AUC score and accuracy in all three models as the epochs reach to 100.

Figure (10) AUC in all models

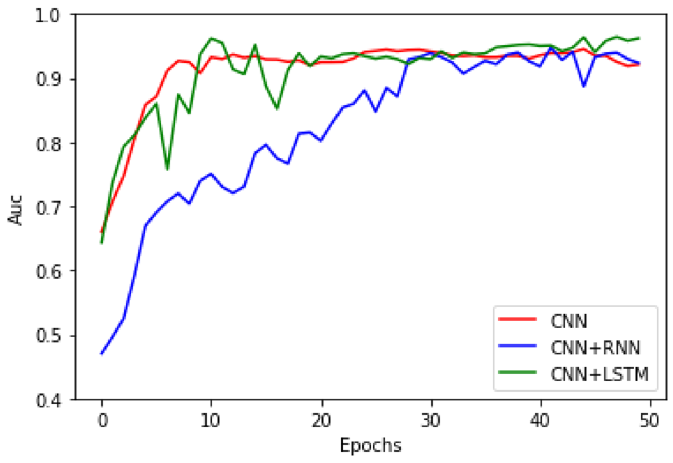


Figure (11) Accuracy in all models

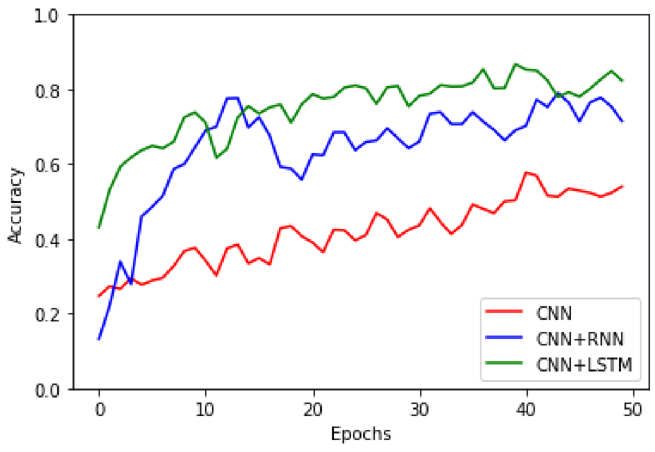


Figure (12) Model Evaluation-AUC, Accuracy

	CNN	**CNN + RNN**	**CNN + LSTM**
AUC	0.9596	0.961	0.9684
Accuracy	0.8657	0.8840	0.9077

Conclusion

Summary

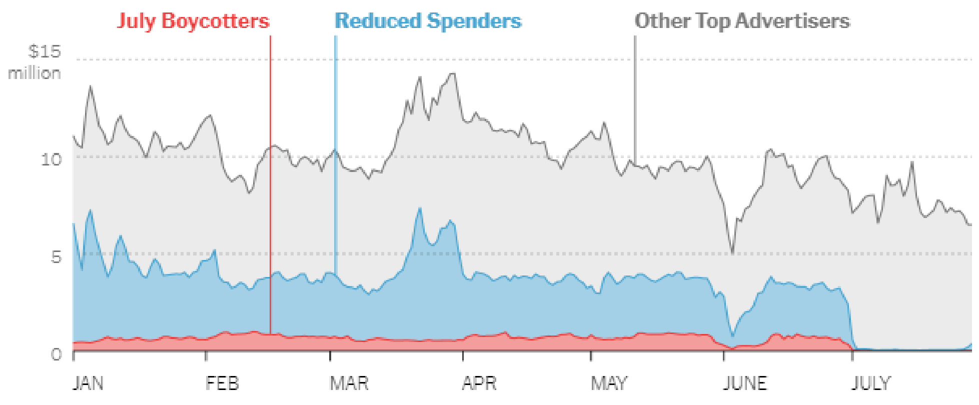
In this project, I establish the three neural networks-based classification model to predict the toxic text messages. I first cleaned the data using stemming and lemmatizing process to normalize the text, then I apply the pre-trained fast text as the embedding matrix to train the data. Afterwards, I validate the model using the 30% rest of data. The result is satisfying and significant. Both three models have high accuracy and AUC score, stating their string efficiency in predicting toxic text. Among all three models, CNN+LSTM model performs the best. It has nearly 0.97 AUC score and 0.91 accuracy. The featured model could be implemented in social media platforms such as Facebook, Twitter, or Reddit to monitor and detect the toxic text, in order to guard against the language abuse problem. By doing so, social media platforms can not only receive full respect from advertisers and companies and raise their interest in investing more money in advertising, but also build up a positive corporate image to broad customers and online users.

Future Works

Although the models perform strongly and efficiently from the results, there exists some improvement that could possibly be fulfilled in the future works. First, I only consider the CNN-based model in this project due to the limitation of the GPU and calculation power. There is no a completely perfect model for given business problem, hence it will be more persuasive if the selected model is compared with more different approaches. If available, the model could compare with more GPU-needed model such as Bi-GRU model, Bi-LSTM model…etc. Also, studies show that adding Attention Layers could improve accuracy of the text classification problem. For more precise validation accuracy purpose, future works can also add K-fold cross validation process to further proof the effectiveness of the model prediction.

Appendix

Figure (1) Estimated Spending Trend in Facebook's Top 100 Advertisers



Note: "Reduced spenders" are companies that did not officially announce boycotts, but decreased their spending in July by at least 90 percent compared to June. • Source: Pathmatics • By Eleanor Lutz

Figure (2) Numbers/Comments Length

```
average length of comment: 395.342
```

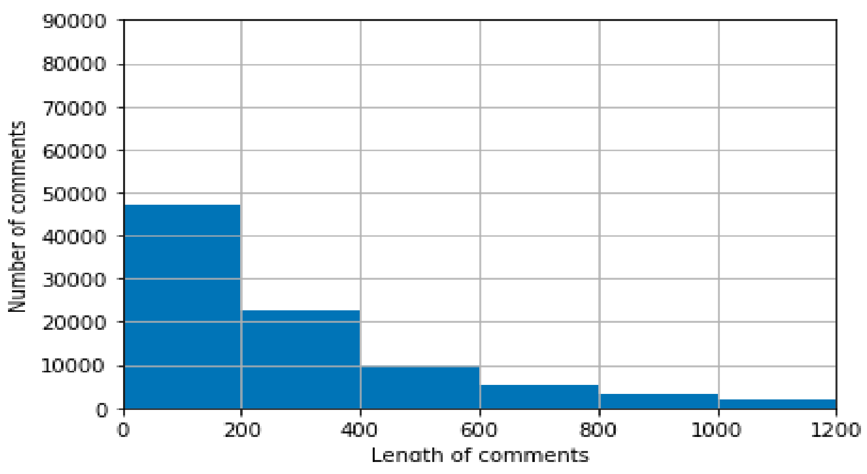


Figure (3) Numbers of Each Class

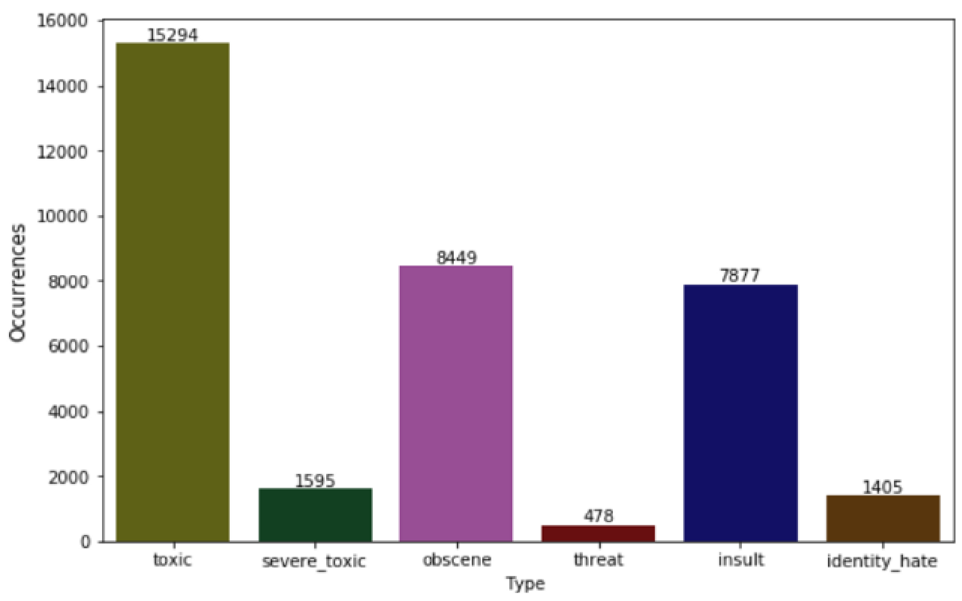


Figure (4) Numbers of Each Class/Length (Training)

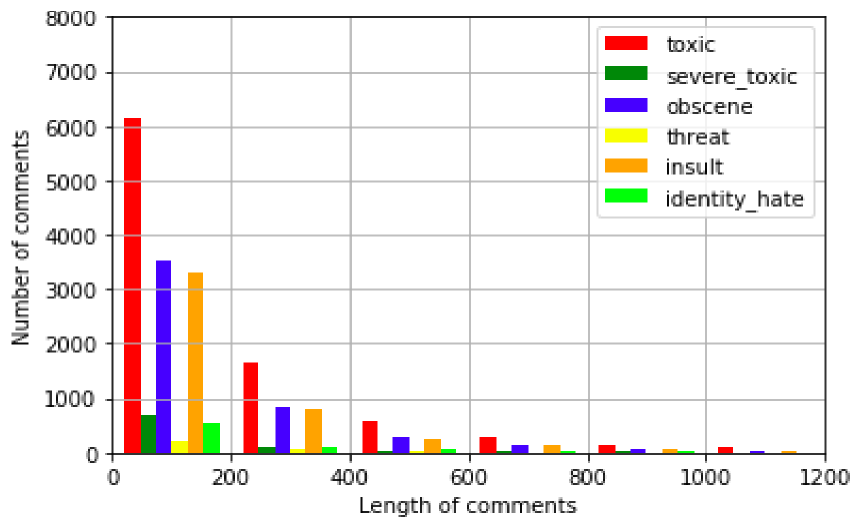


Figure (5) Data Pre-processing Steps

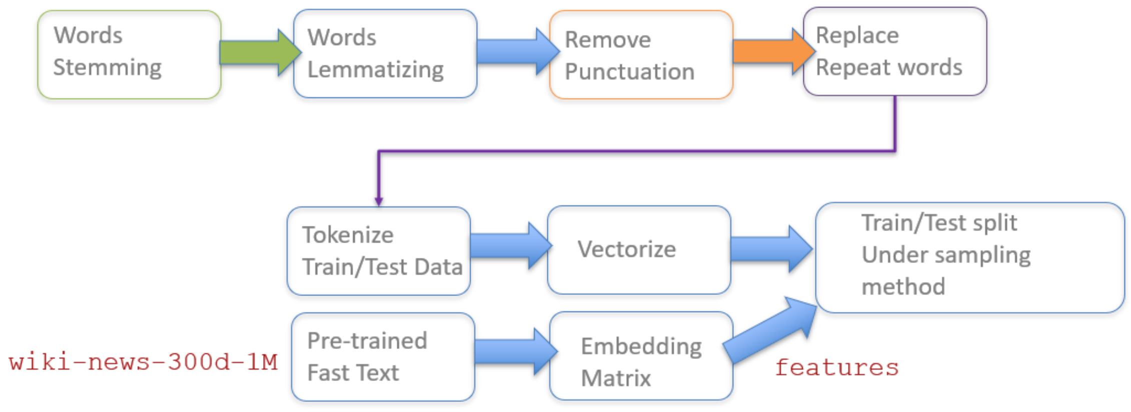
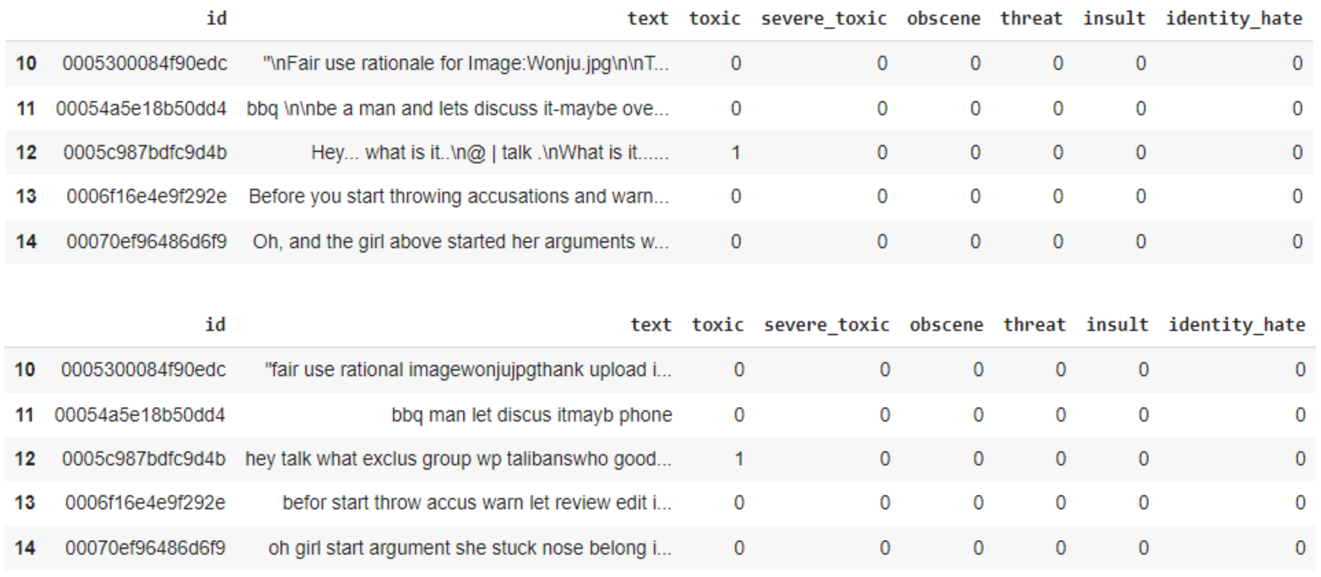


Figure (6) Data Pre-Process- Before vs After



	id	text	toxic	severe_toxic	obscene	threat	insult	identity_hate
10	0005300084f90edc	"\nFair use rationale for Image:Wonju.jpg\n\nT...	0	0	0	0	0	0
11	00054a5e18b50dd4	bbq \n\nbe a man and lets discuss it-maybe ove...	0	0	0	0	0	0
12	0005c987bdfc9d4b	Hey... what is it..\n@ | talk .\nWhat is it......	1	0	0	0	0	0
13	0006f16e4e9f292e	Before you start throwing accusations and warn...	0	0	0	0	0	0
14	00070ef96486d6f9	Oh, and the girl above started her arguments w...	0	0	0	0	0	0

	id	text	toxic	severe_toxic	obscene	threat	insult	identity_hate
10	0005300084f90edc	"fair use rational imagewonjujpgthank upload i...	0	0	0	0	0	0
11	00054a5e18b50dd4	bbq man let discus itmayb phone	0	0	0	0	0	0
12	0005c987bdfc9d4b	hey talk what exclus group wp talibanswho good...	1	0	0	0	0	0
13	0006f16e4e9f292e	befor start throw accus warn let review edit i...	0	0	0	0	0	0
14	00070ef96486d6f9	oh girl start argument she stuck nose belong i...	0	0	0	0	0	0

Figure (7) General Modeling and Evaluation Steps

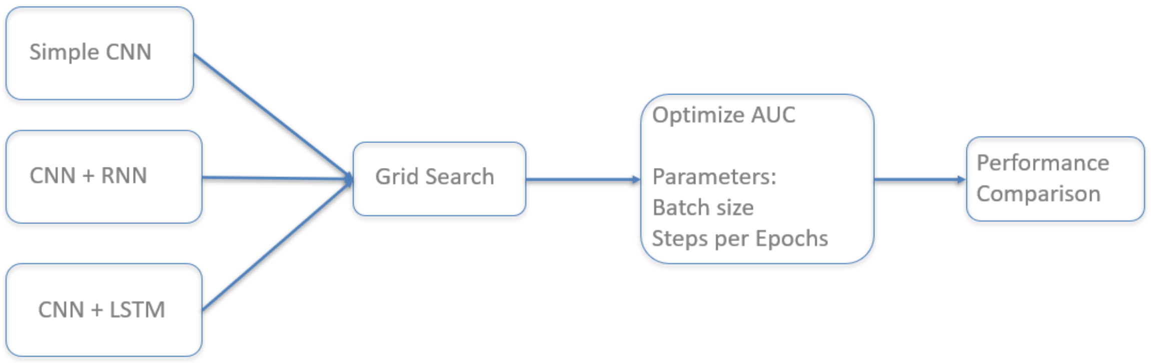
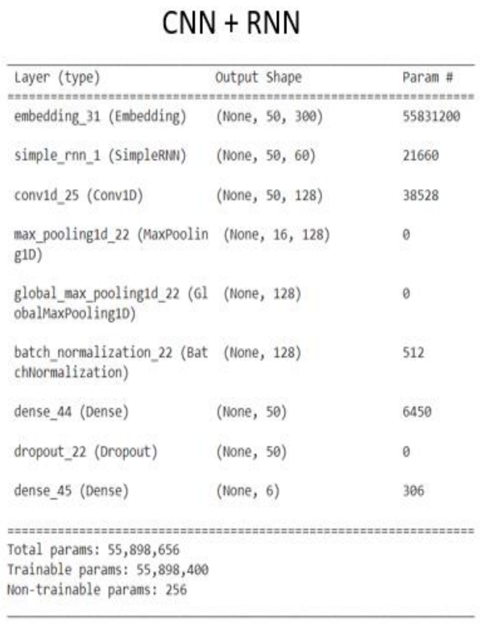
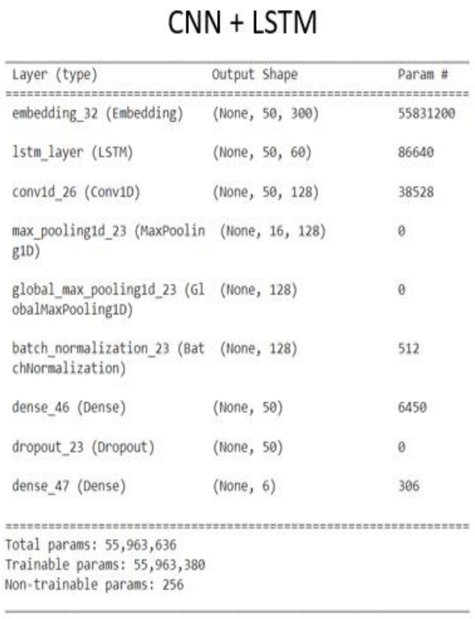


Figure (8) Final Models Comparison



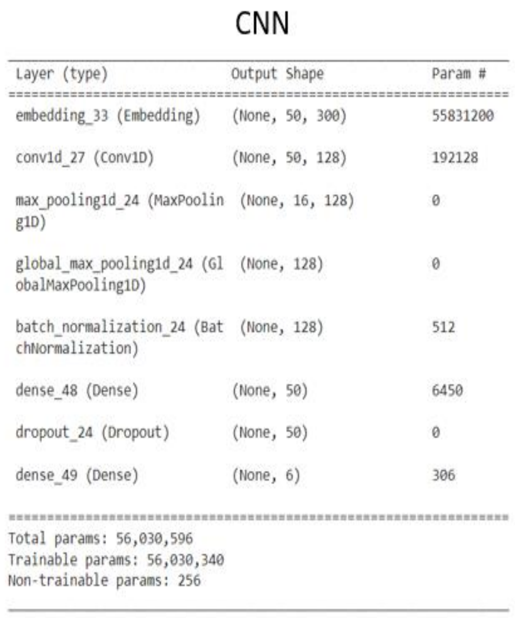
CNN + RNN

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_31 (Embedding)     (None, 50, 300)           55831200

simple_rnn_1 (SimpleRNN)     (None, 50, 60)            21660

conv1d_25 (Conv1D)           (None, 50, 128)           38528

max_pooling1d_22 (MaxPoolin  (None, 16, 128)           0
g1D)

global_max_pooling1d_22 (Gl  (None, 128)               0
obalMaxPooling1D)

batch_normalization_22 (Bat  (None, 128)               512
chNormalization)

dense_44 (Dense)             (None, 50)                6450

dropout_22 (Dropout)         (None, 50)                0

dense_45 (Dense)             (None, 6)                 306

=================================================================
Total params: 55,898,656
Trainable params: 55,898,400
Non-trainable params: 256
```



CNN + LSTM

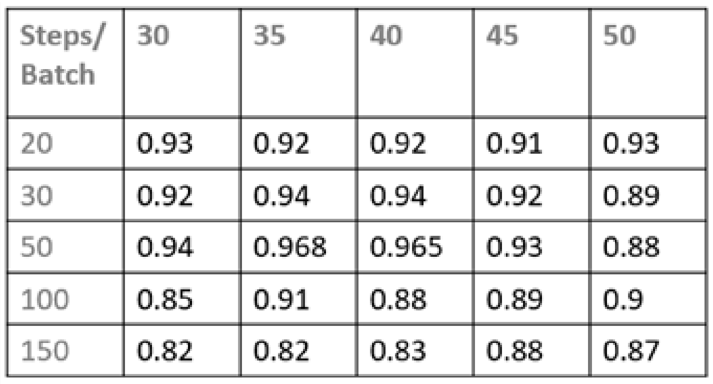
```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_32 (Embedding)     (None, 50, 300)           55831200

lstm_layer (LSTM)            (None, 50, 60)            86640

conv1d_26 (Conv1D)           (None, 50, 128)           38528

max_pooling1d_23 (MaxPoolin  (None, 16, 128)           0
g1D)

global_max_pooling1d_23 (Gl  (None, 128)               0
obalMaxPooling1D)

batch_normalization_23 (Bat  (None, 128)               512
chNormalization)

dense_46 (Dense)             (None, 50)                6450

dropout_23 (Dropout)         (None, 50)                0

dense_47 (Dense)             (None, 6)                 306

=================================================================
Total params: 55,963,636
Trainable params: 55,963,380
Non-trainable params: 256
```



CNN

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_33 (Embedding)     (None, 50, 300)           55831200

conv1d_27 (Conv1D)           (None, 50, 128)           192128

max_pooling1d_24 (MaxPoolin  (None, 16, 128)           0
g1D)

global_max_pooling1d_24 (Gl  (None, 128)               0
obalMaxPooling1D)

batch_normalization_24 (Bat  (None, 128)               512
chNormalization)

dense_48 (Dense)             (None, 50)                6450

dropout_24 (Dropout)         (None, 50)                0

dense_49 (Dense)             (None, 6)                 306

=================================================================
Total params: 56,030,596
Trainable params: 56,030,340
Non-trainable params: 256
```

Figure (9) Grid Search AUC – CNN + LSTM



Steps/ Batch	30	35	40	45	50
20	0.93	0.92	0.92	0.91	0.93
30	0.92	0.94	0.94	0.92	0.89
50	0.94	0.968	0.965	0.93	0.88
100	0.85	0.91	0.88	0.89	0.9
150	0.82	0.82	0.83	0.88	0.87

Figure (10) AUC in all models

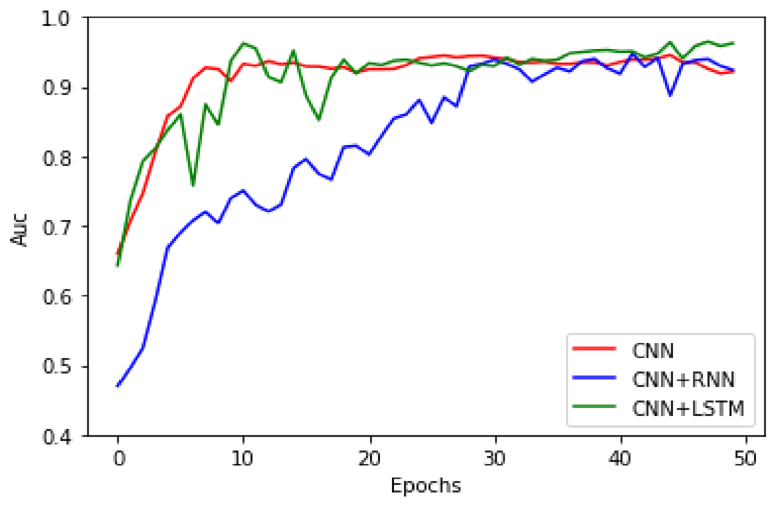


Figure (11) Accuracy in all models

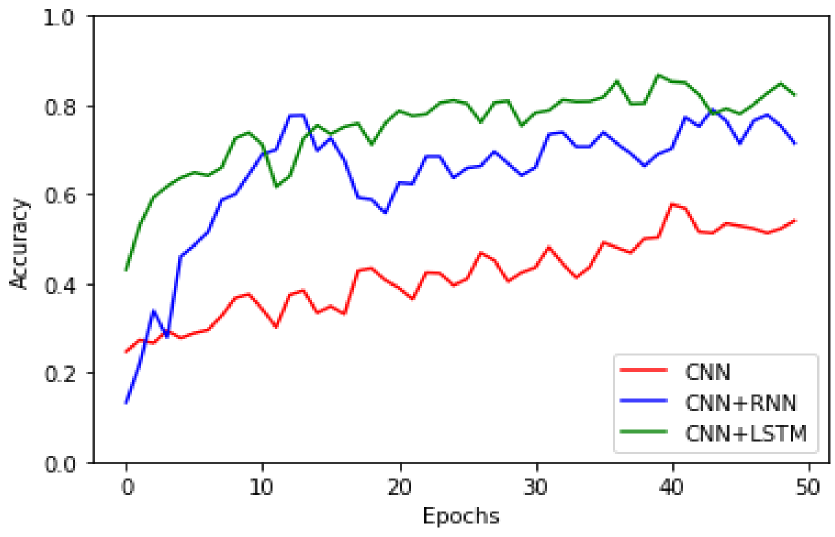
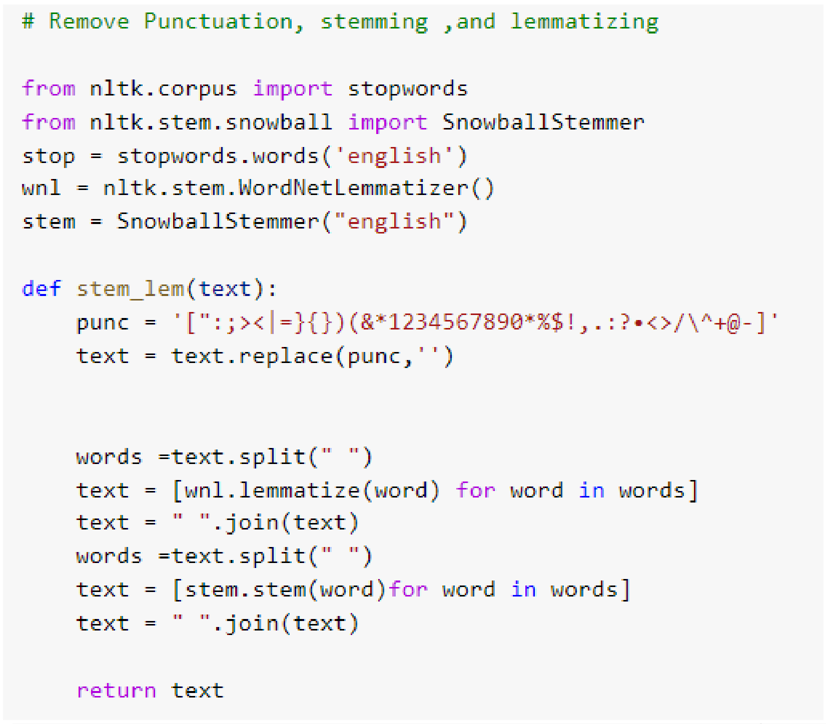


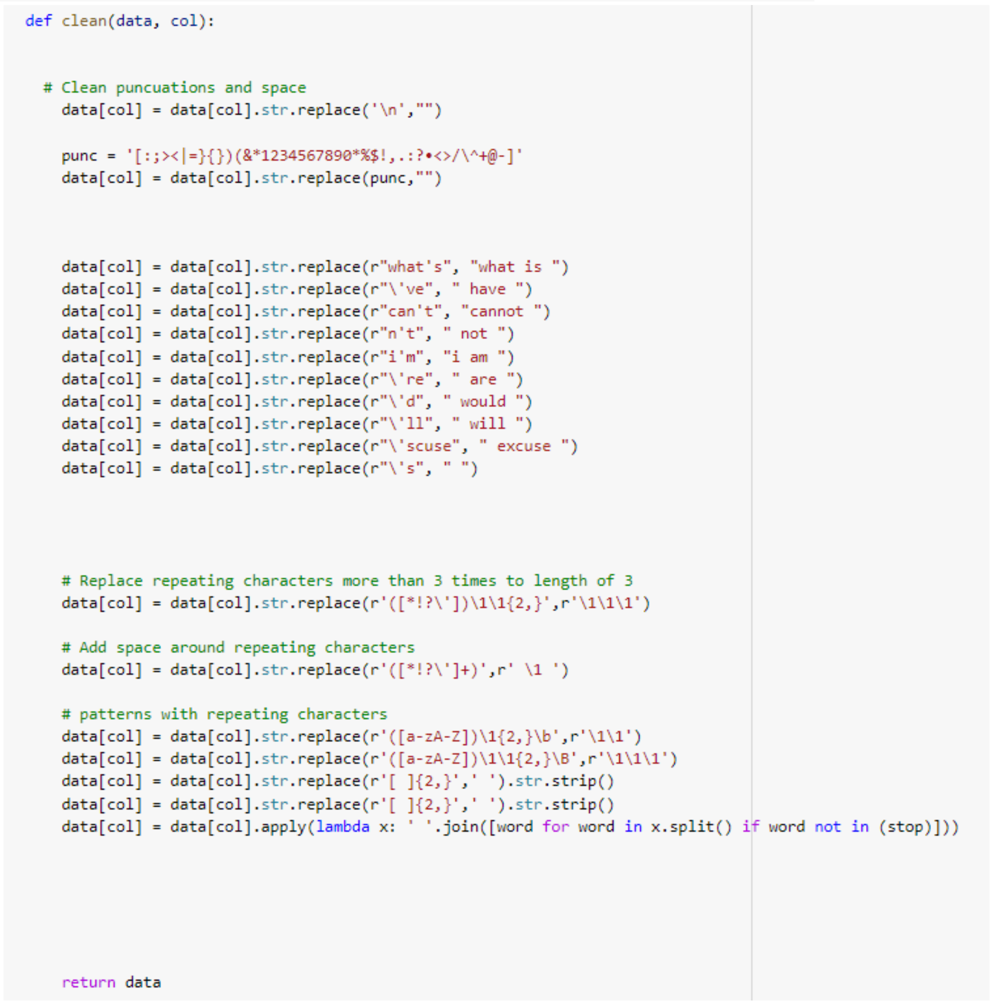
Figure (12) Model Evaluation-AUC, Accuracy

	CNN	**CNN + RNN**	**CNN + LSTM**
AUC	0.9596	0.961	0.9684
Accuracy	0.8657	0.8840	0.9077

Main Python Code



```python
# Remove Punctuation, stemming ,and lemmatizing

from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
stop = stopwords.words('english')
wnl = nltk.stem.WordNetLemmatizer()
stem = SnowballStemmer("english")

def stem_lem(text):
    punc = '[":;><|=}{})(&*1234567890*%$!,.:?•<>/\^+@-]'
    text = text.replace(punc,'')


    words =text.split(" ")
    text = [wnl.lemmatize(word) for word in words]
    text = " ".join(text)
    words =text.split(" ")
    text = [stem.stem(word)for word in words]
    text = " ".join(text)

    return text
```



```python
def clean(data, col):


  # Clean puncuations and space
    data[col] = data[col].str.replace('\n',"")

    punc = '[:;><|=}{})(&*1234567890*%$!,.:?•<>/\^+@-]'
    data[col] = data[col].str.replace(punc,"")



    data[col] = data[col].str.replace(r"what's", "what is ")
    data[col] = data[col].str.replace(r"\'ve", " have ")
    data[col] = data[col].str.replace(r"can't", "cannot ")
    data[col] = data[col].str.replace(r"n't", " not ")
    data[col] = data[col].str.replace(r"i'm", "i am ")
    data[col] = data[col].str.replace(r"\'re", " are ")
    data[col] = data[col].str.replace(r"\'d", " would ")
    data[col] = data[col].str.replace(r"\'ll", " will ")
    data[col] = data[col].str.replace(r"\'scuse", " excuse ")
    data[col] = data[col].str.replace(r"\'s", " ")




    # Replace repeating characters more than 3 times to length of 3
    data[col] = data[col].str.replace(r'([*!?\'])\1\1{2,}',r'\1\1\1')

    # Add space around repeating characters
    data[col] = data[col].str.replace(r'([*!?\']+)',r' \1 ')

    # patterns with repeating characters
    data[col] = data[col].str.replace(r'([a-zA-Z])\1{2,}\b',r'\1\1')
    data[col] = data[col].str.replace(r'([a-zA-Z])\1\1{2,}\B',r'\1\1\1')
    data[col] = data[col].str.replace(r'[ ]{2,}',' ').str.strip()
    data[col] = data[col].str.replace(r'[ ]{2,}',' ').str.strip()
    data[col] = data[col].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))




    return data
```

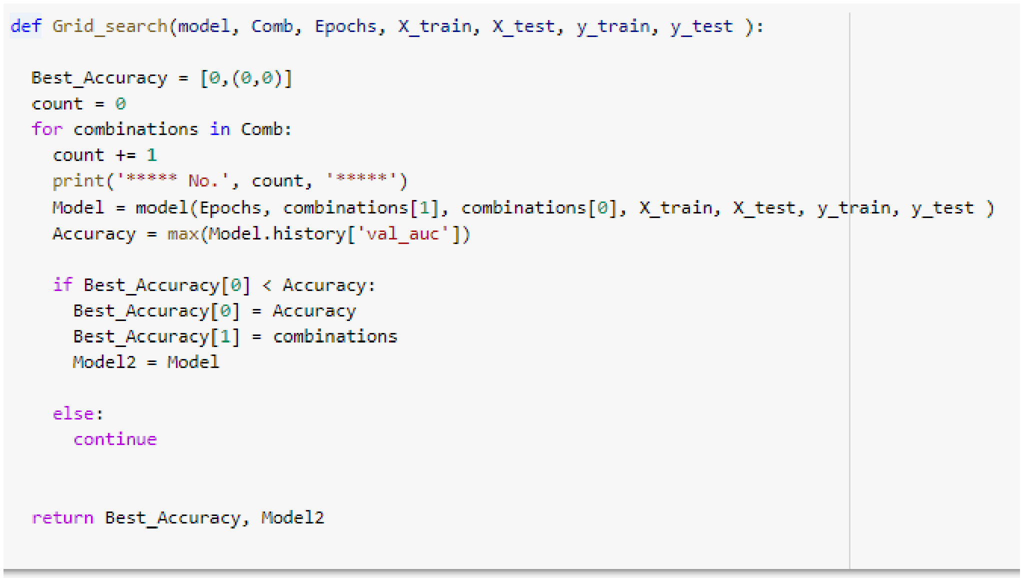
```python
# Tokenize and Pad

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
# Create tokenizer
tokenizer = Tokenizer(num_words=None,
                      filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
                      lower=True,
                      split=" ",
                      char_level=False)

# Fit and run tokenizer
max_len = 50
tokenizer.fit_on_texts(list(x_train))
tokenized_train = tokenizer.texts_to_sequences(x_train)
tokenized_test = tokenizer.texts_to_sequences(x_test)
X_train = pad_sequences(tokenized_train, maxlen=max_len)
X_test = pad_sequences(tokenized_test, maxlen=max_len)
word_index = tokenizer.word_index
```

```python
import numpy as np
embedding_dim = 300
# Get embeddings
embeddings_index = {}
f = open('/content/wiki-news-300d-1M.vec', encoding="utf8")
for line in f:
    values = line.rstrip().rsplit(' ', embedding_dim)
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
# Build embedding matrix
embedding_matrix = np.zeros((len(word_index) + 1, embedding_dim))
for word, i in word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector
```

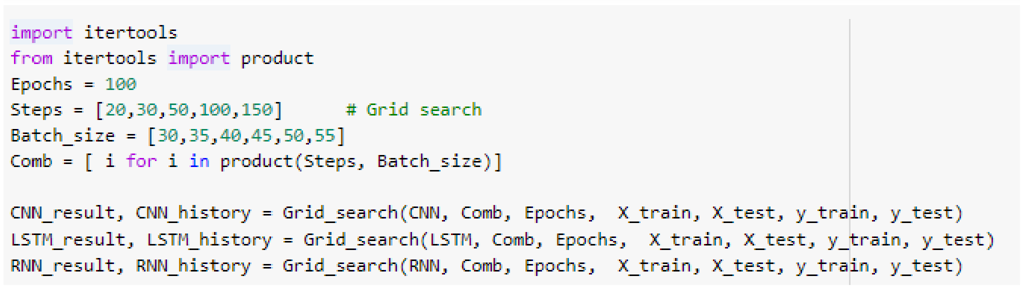
```python
def RNN(Epochs,Batch_size, steps, X_train, X_test, y_train, y_test):
  max_features = len(embedding_matrix)
  model = Sequential()
  model.add(Embedding(max_features, embedding_dim, input_length=max_len))
  model.add(layers.SimpleRNN(60, return_sequences=True))
  model.add(Conv1D(filters=128, kernel_size=5, padding='same', activation='relu'))
  model.add(MaxPooling1D(3))
  model.add(GlobalMaxPooling1D())
  model.add(BatchNormalization())
  model.add(Dense(50, activation='relu'))
  model.add(Dropout(0.3))
  model.add(Dense(6, activation='sigmoid'))

  model.compile(loss = 'mean_squared_error', optimizer='adam', metrics=['accuracy','AUC'])
  checkpoint = ModelCheckpoint(url + 'best_model_RNN.hdf5', monitor='val_loss', verbose=1,
    save_best_only=True, mode='auto', period=1)

  history = model.fit(X_train, y_train,
                    epochs=Epochs,
                    batch_size = Batch_size,
                    steps_per_epoch = steps,
                   validation_data=(X_test, y_test),
                    callbacks=[checkpoint],    # delete monitor
                  verbose = 1)

  return history
```

```python
def CNN(Epochs,Batch_size, steps, X_train, X_test, y_train, y_test):
  max_features = len(embedding_matrix)

  model = Sequential()

  model.add(Embedding(max_features, embedding_dim, weights=[embedding_matrix], input_length=max_len, trainable=True))

  model.add(Conv1D(filters=128, kernel_size=5, padding='same', activation='relu'))
  model.add(MaxPooling1D(3))
  model.add(GlobalMaxPooling1D())
  model.add(BatchNormalization())
  model.add(Dense(50, activation='relu'))
  model.add(Dropout(0.3))
  model.add(Dense(6, activation='sigmoid'))
  model.compile(loss = 'mean_squared_error', optimizer='adam', metrics=['accuracy','AUC'])
  checkpoint = ModelCheckpoint(url + 'best_model_CNN.hdf5', monitor='val_loss', verbose=1,
                               save_best_only=True, mode='auto', period=1)

  history = model.fit(X_train, y_train, epochs=Epochs,
                      batch_size = Batch_size,
                      steps_per_epoch = steps,
                     validation_data=(X_test, y_test),
                      callbacks=[checkpoint],    # delete monitor
                    verbose = 1)

  return history
```

```python
def LSTM(Epochs,Batch_size, steps, X_train, X_test, y_train, y_test):
  max_features = len(embedding_matrix)
  model = Sequential()
  model.add(Embedding(max_features, embedding_dim, weights=[embedding_matrix], input_length=max_len, trainable=True)
  model.add(layers.LSTM(60, return_sequences=True, name='lstm_layer'))
  model.add(Conv1D(filters=128, kernel_size=5, padding='same', activation='relu'))
  model.add(MaxPooling1D(3))
  model.add(GlobalMaxPooling1D())
  model.add(BatchNormalization())
  model.add(Dense(50, activation='relu'))
  model.add(Dropout(0.3))
  model.add(Dense(6, activation='sigmoid'))
  model.compile(loss = 'mean_squared_error', optimizer='adam', metrics=['accuracy','AUC'])
  checkpoint = ModelCheckpoint(url + 'best_model_LSTM.hdf5', monitor='val_loss', verbose=1,
                               save_best_only=True, mode='auto', period=1)

  history = model.fit(X_train, y_train,
                      epochs=Epochs,
                      batch_size = Batch_size,
                      steps_per_epoch = steps,
                     validation_data=(X_test, y_test),
                      callbacks=[checkpoint],    # delete monitor
                    verbose = 1)

  return history
```



```python
def Grid_search(model, Comb, Epochs, X_train, X_test, y_train, y_test ):

  Best_Accuracy = [0,(0,0)]
  count = 0
  for combinations in Comb:
    count += 1
    print('***** No.', count, '*****')
    Model = model(Epochs, combinations[1], combinations[0], X_train, X_test, y_train, y_test )
    Accuracy = max(Model.history['val_auc'])

    if Best_Accuracy[0] < Accuracy:
      Best_Accuracy[0] = Accuracy
      Best_Accuracy[1] = combinations
      Model2 = Model

    else:
      continue


  return Best_Accuracy, Model2
```



```python
import itertools
from itertools import product
Epochs = 100
Steps = [20,30,50,100,150]       # Grid search
Batch_size = [30,35,40,45,50,55]
Comb = [ i for i in product(Steps, Batch_size)]

CNN_result, CNN_history = Grid_search(CNN, Comb, Epochs,  X_train, X_test, y_train, y_test)
LSTM_result, LSTM_history = Grid_search(LSTM, Comb, Epochs,  X_train, X_test, y_train, y_test)
RNN_result, RNN_history = Grid_search(RNN, Comb, Epochs,  X_train, X_test, y_train, y_test)
```

References

Banerjee, Ling, et al, Comparative effectiveness of convolutional neural network (CNN) and recurrent neural network (RNN) architectures for radiology text report classification, Artificial Intelligence In Medicine, 2019

Sharma, Chaurasia, et al, Sentimental Short Sentences Classification by Using CNN Deep Learning Model with Fine Tuned Word2Vec, Procedia Computer Science, 2020

Kohli, Kuehler, & Palowith, and Paying attention to toxic comments online, Havard Business Review

Li, Wang, & Xu, Chinese Text Classification Model Based on Deep Learning, Future Internet, 2018

Sari, Rini,& Malik, Text Classification Using Long Short-Term Memory with GloVe Features, Jurnal Ilmiah Teknik Elektro Komputer dan Informatika, 2020

Wang, Hsiao, & Chang, Automatic paper writing based on a RNN and the TextRank algorithm, Applied Soft Computing Journal, 2020