CS510: Project3

Chufeng Gao gao513@purdue.edu
Meng-Chieh Lin lin1055@purdue.edu

Part 1.

The buggy rates of training, validation, and test instances are 0.43584, 0.434, and 0.44112 respectively. And the result of ROC curve and AUC is below:

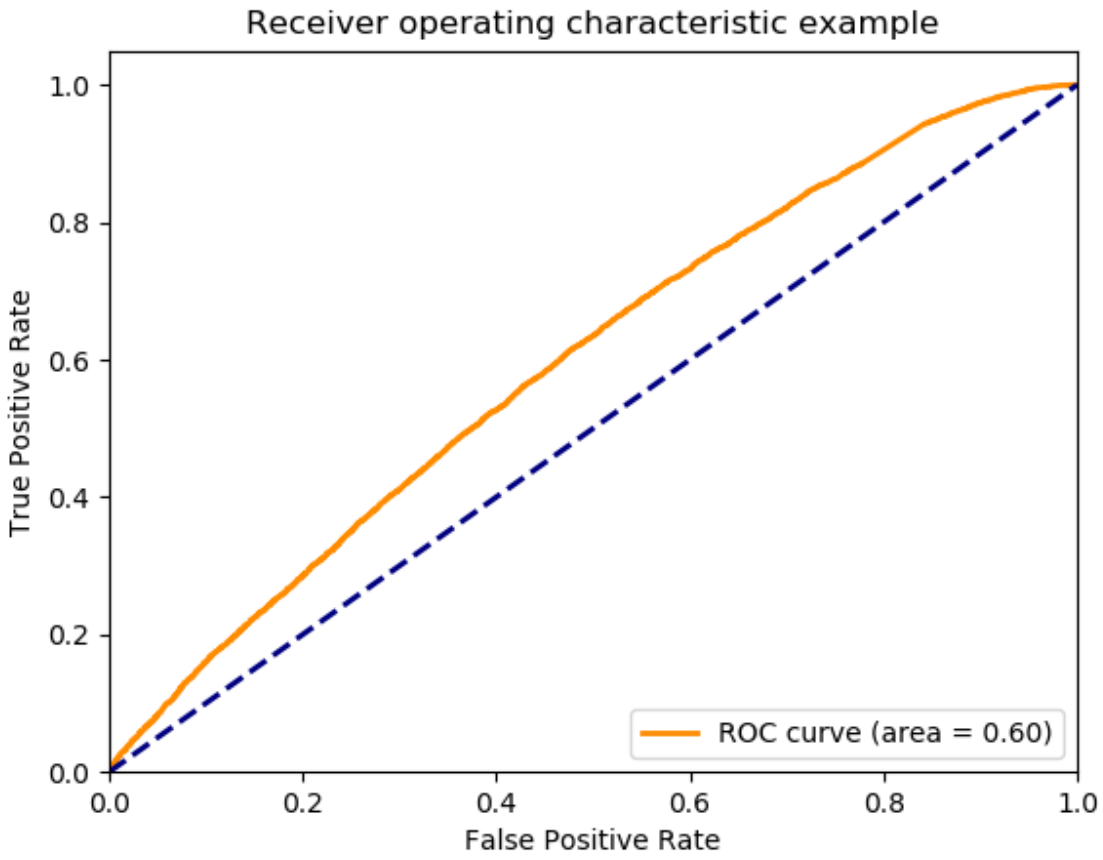

Figure 1: auc_model

Part 2.

We have tried to implement two deep-learning algorithms: one is Independently Recurrent Neural Network (*IndRNN*), and the other one is Hierarchical Attention Network (*HAN*).

Firstly, we applied second model which is *IndRNN* since it is a promised improvement on RNN compared to traditional RNN. We use the same data size as the script sampled

in part 1. It turns out IndRNN takes much longer time to get trained. The accuracy we get from IndRNN is pretty close to the one we get from simple LSTM. IndRNN model is supposed to achieve better result because the model is designed to prevent the gradient exploding and vanishing problems while allowing the network to learn long-term dependencies. I think there are two factors which may have affected our results: one is that since the version of tensorflow on cuda doesn't support the method for generating recurrent mask, we do not apply any kind of masking to our input data, which could cause bad effect during the training process. The other one is that we haven't spent much time tuning the hyper parameters.

Secondly, We chose *HAN* since it has a hierarchical structure that mirrors the hierarchical structure of texts. But we encounter a problem when applying the model to train original data set. Since there is no "sentence" in each sequence of the data set and no weights to apply on different words, we treated each sequence as a "text" and each encoded value as a "sentence". Then we fed them to our *half-HAN* model and applied "uniform" as the weights over sentences. After training, its training accuracy is 0.4288 and validation accuracy is 0.4312 with the same data size and parameters in part 1. Apparently, it performs worse than the algorithm in part 1. We think there are two reasons for that result. For the first reason, to apply the original HAN, we need to define separators for sentences in codes and extract them to build a second level data set during preprocessing. We attempted to do that and found it is hard to separate codes by separators straightly since codes are nested in multipule levels. Thus, we turned to apply half of *HAN*. For the second reason, the application of *HAN* in the GitHub repository used GloVe which is an unsupervised learning algorithm for obtaining vector representations for words to set weights for each word. However, our data are codes but not normal human languages. Therefore we can only use the default setting which is uniform.

Referenced links are below:
HAN: https://github.com/richliao/textClassifier
IndRNN: https://github.com/batzner/indrnn, https://github.com/titu1994/Keras-IndRNN

Part 3. We have tried two methods to improve the performance of our model in part 2:

1. Use more training data:
   We first compare two different training data sizes, 50000 and 100000, under following parameters: epochs are 1 and hidden units are 64. The data size of testing and validation are 25000. The accuracy and AUC-ROC plots are shown below:

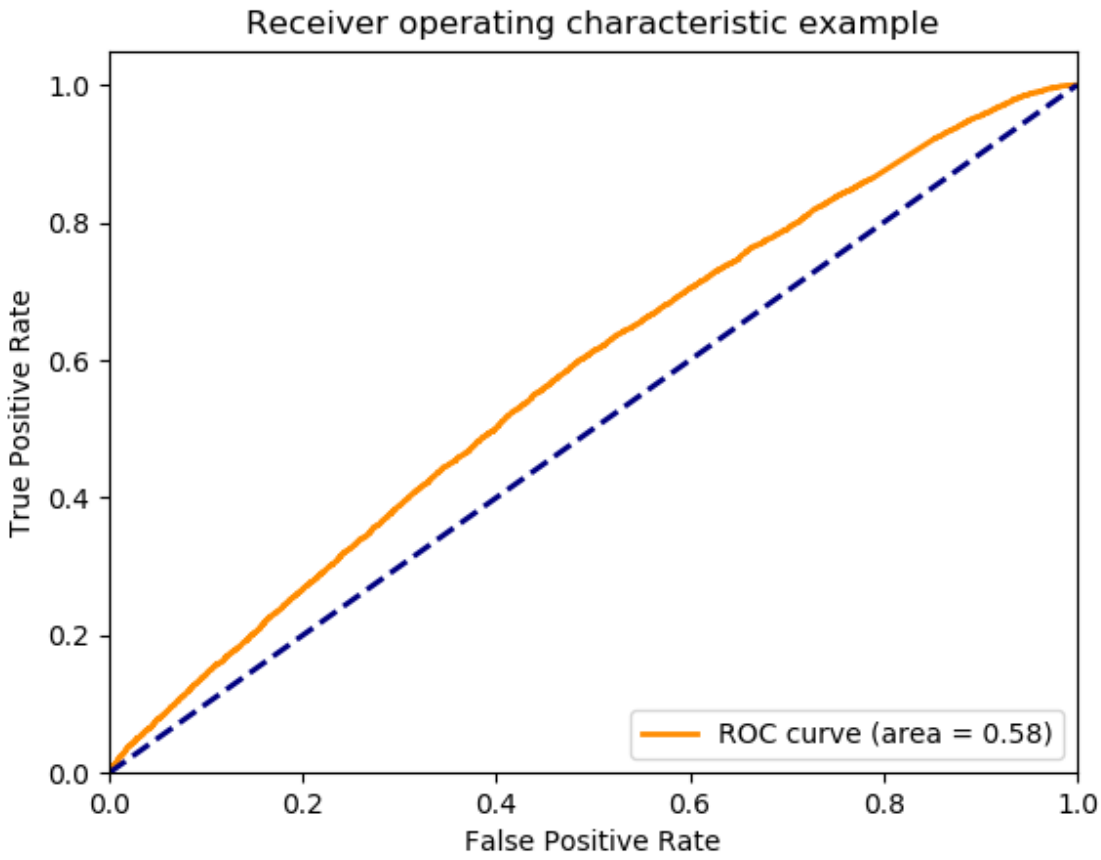

Figure 2: epochs = 1, training size= 50000, units = 64

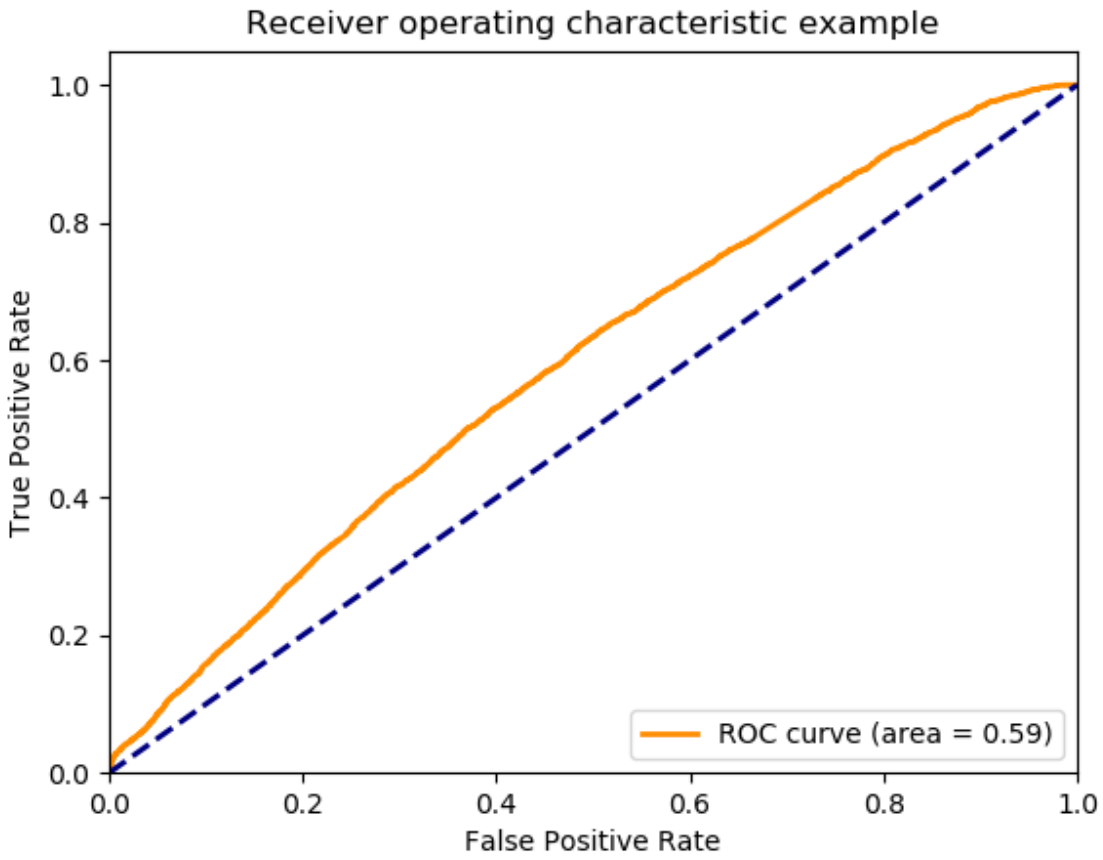

Figure 3: epochs = 1, training size = 100000, units = 64

Training Data	Train Loss	Train Accuracy	Valid Loss	Valid Accuracy	ROC Area
50000	0.6805	0.5651	0.6748	0.5728	0.58
100000	0.6751	0.5730	0.6690	0.5786	0.59

Figure 4: epochs = 1, units = 64

We continue compare the same two training data sizes with same testing and validation data size under following parameters: epochs are 1 and hidden units are 128. The accuracy and AUC-ROC plots are shown below:

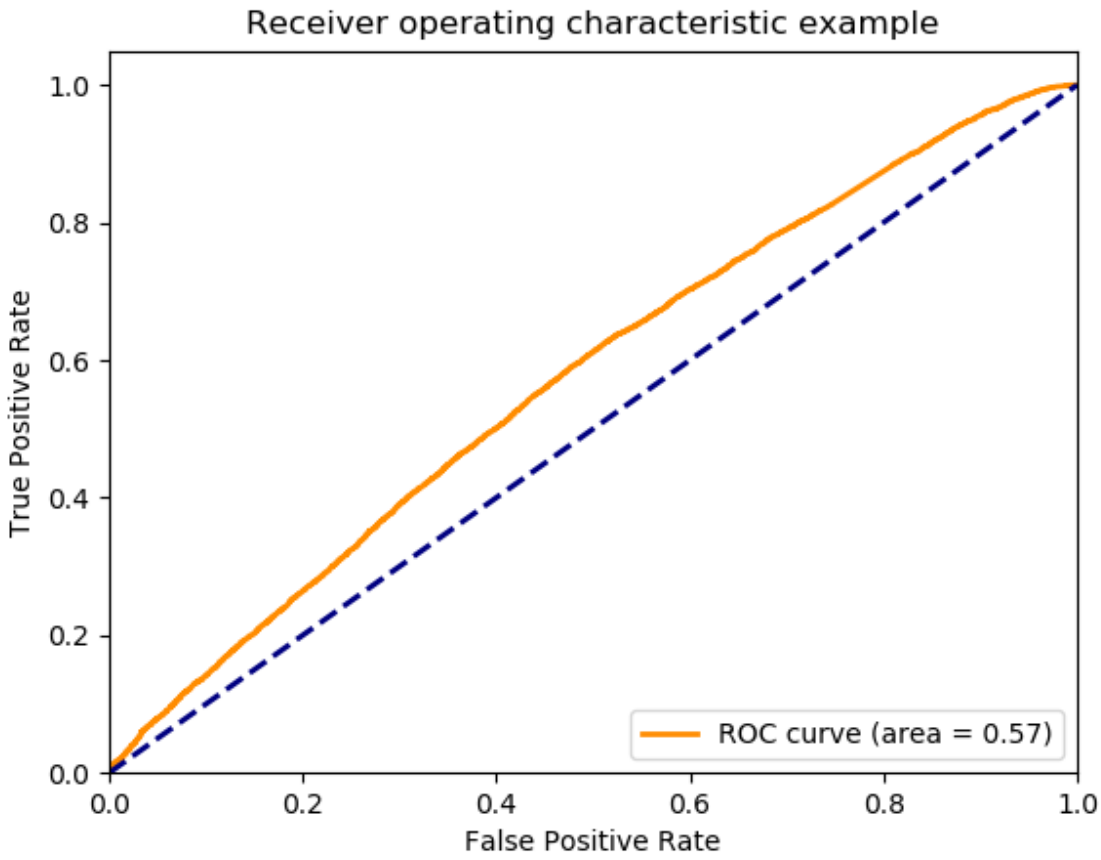

Figure 5: epochs = 1, training size = 50000, units = 128

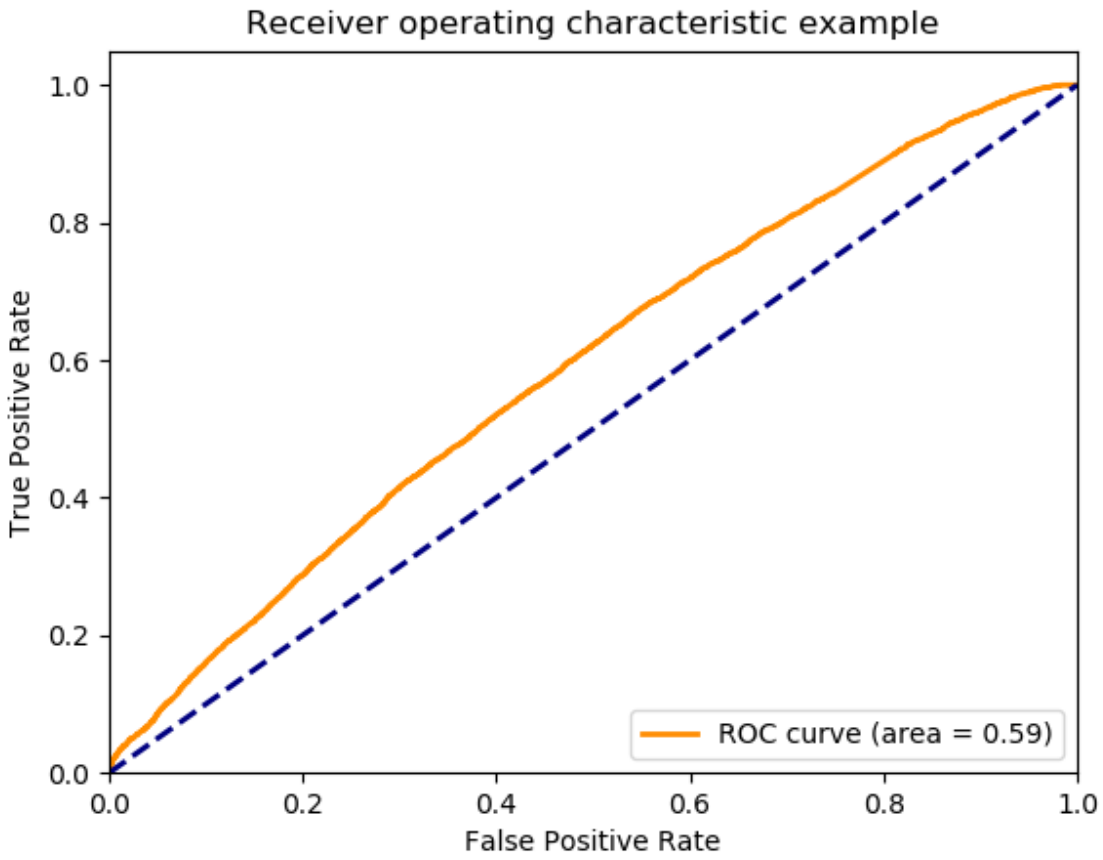

Figure 6: epochs = 1, training size = 100000, units = 128

Training Data	Train Loss	Train Accuracy	Valid Loss	Valid Accuracy	ROC Area
50000	0.6811	0.5663	0.6751	0.5692	0.57
100000	0.6759	0.5725	0.6694	0.5747	0.59

Figure 7: epochs = 1, units = 128

By observation, increasing training data set can slightly improve the performance in both conditions (hidden units). By the way, we also discover that increasing hidden units does not make any improvements.

2. Tuning and Building Deeper models: There are several parameters we experiment, including epochs, hidden units, and learning rate. Firstly, we try to improve the performance of our model in part 2 by tuning the ***learning rate***. We achieve the

best result when taking learning rate around 1e-3, considering the training time and accuracy trade-off. The accuracy and AUC-ROC plots are shown as follows:

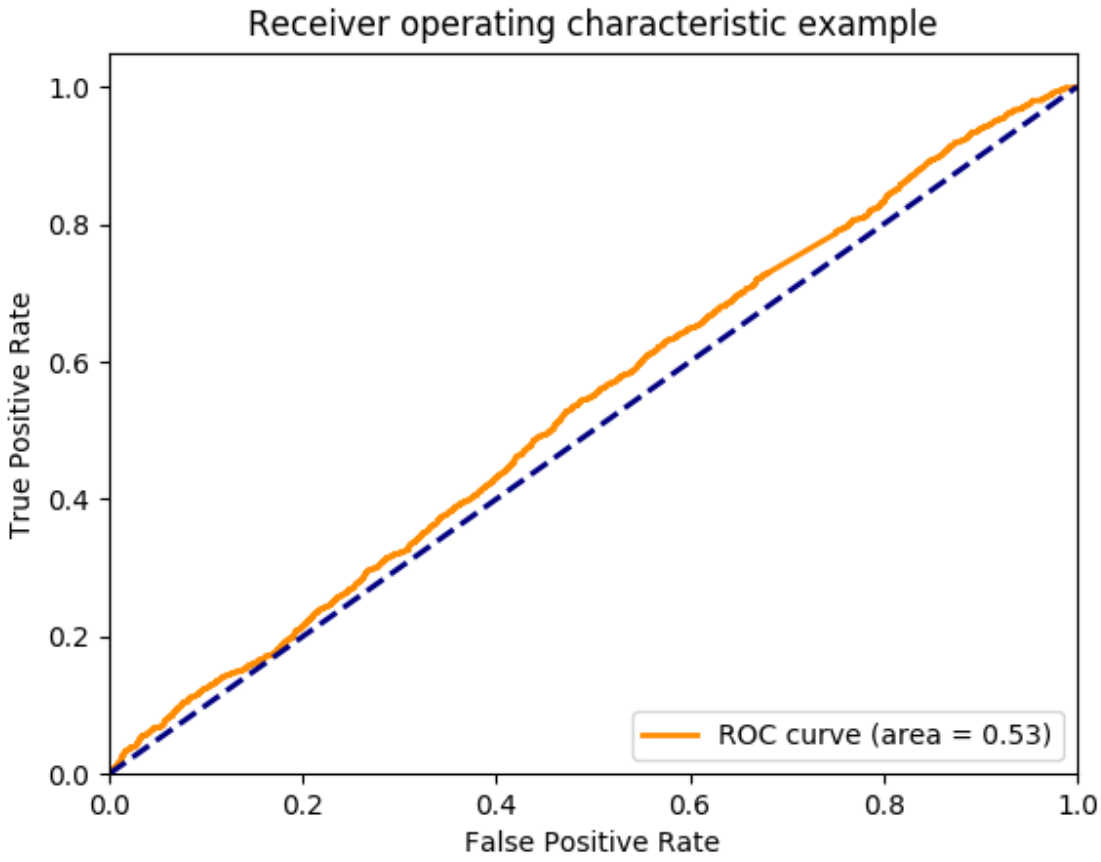

Figure 8: epochs = 3, training size= 10000, units = 128, learning rate = 5e-4

```
_________________________________________________________________
Epoch 1/3
100/100 [==============================] - 467s 5s/step - loss: 0.6847 - accuracy: 0.5699 - val_loss: 0.6836 - val_accuracy: 0.5688
Epoch 2/3
100/100 [==============================] - 417s 4s/step - loss: 0.6767 - accuracy: 0.5712 - val_loss: 0.6841 - val_accuracy: 0.5688
Epoch 3/3
100/100 [==============================] - 466s 5s/step - loss: 0.6407 - accuracy: 0.6183 - val_loss: 0.7110 - val_accuracy: 0.5464
50/50 [==============================] - 79s 2s/step
```

Figure 9: epochs = 3, training size= 10000, units = 128, learning rate = 5e-4

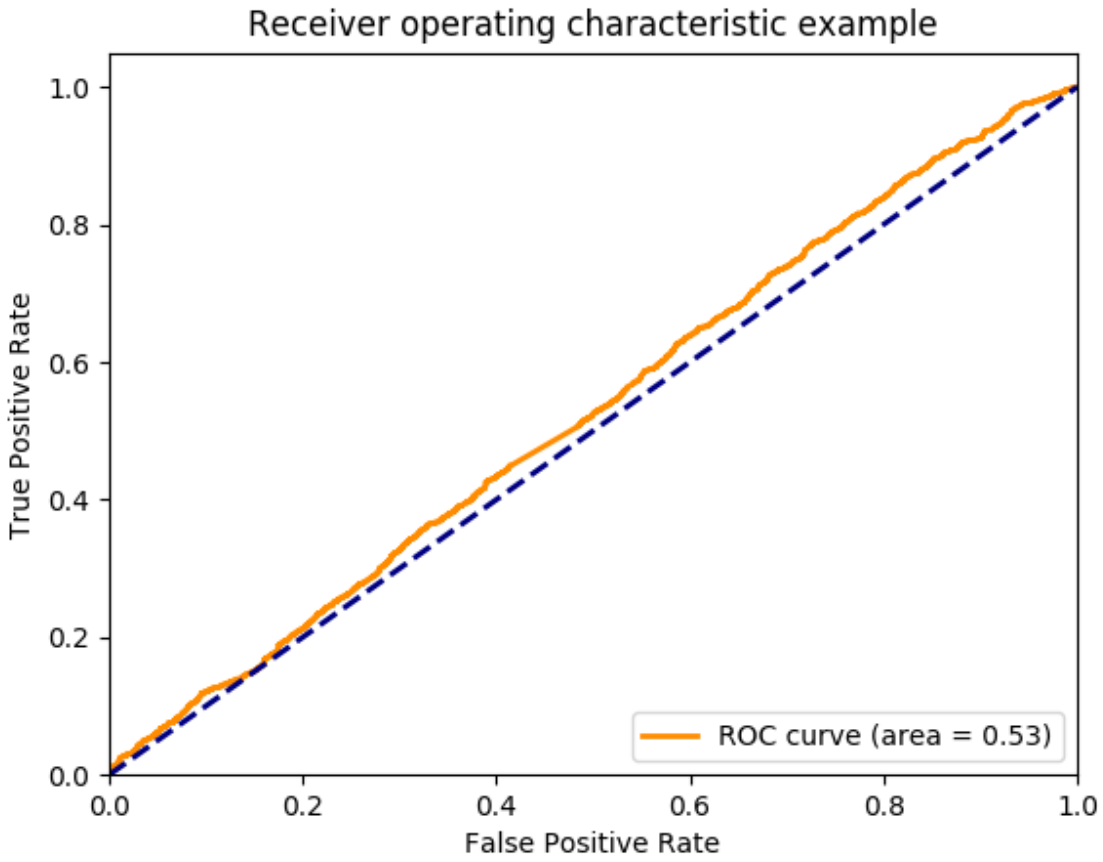

Figure 10: epochs = 3, training size= 10000, units = 128, learning rate = 1e-2

```
_________________________________________________________________
Epoch 1/3
100/100 [==============================] - 437s 4s/step - loss: 0.6849 - accuracy: 0.5699 - val_loss: 0.6835 - val_accuracy: 0.5688
Epoch 2/3
100/100 [==============================] - 458s 5s/step - loss: 0.6408 - accuracy: 0.6230 - val_loss: 0.6987 - val_accuracy: 0.5642
Epoch 3/3
100/100 [==============================] - 467s 5s/step - loss: 0.5403 - accuracy: 0.7189 - val_loss: 0.7958 - val_accuracy: 0.5538
50/50 [==============================] - 79s 2s/step
```

Figure 11: epochs = 3, training size= 10000, units = 128, learning rate = 1e-2

Secondly, we try to improve the performance by increasing training epochs. We compare two epoch numbers, 1 and 3, under following conditions: training data size is 50000 and hidden units are 64. The data size of testing and validation are 25000. The accuracy and AUC-ROC plots are shown below:

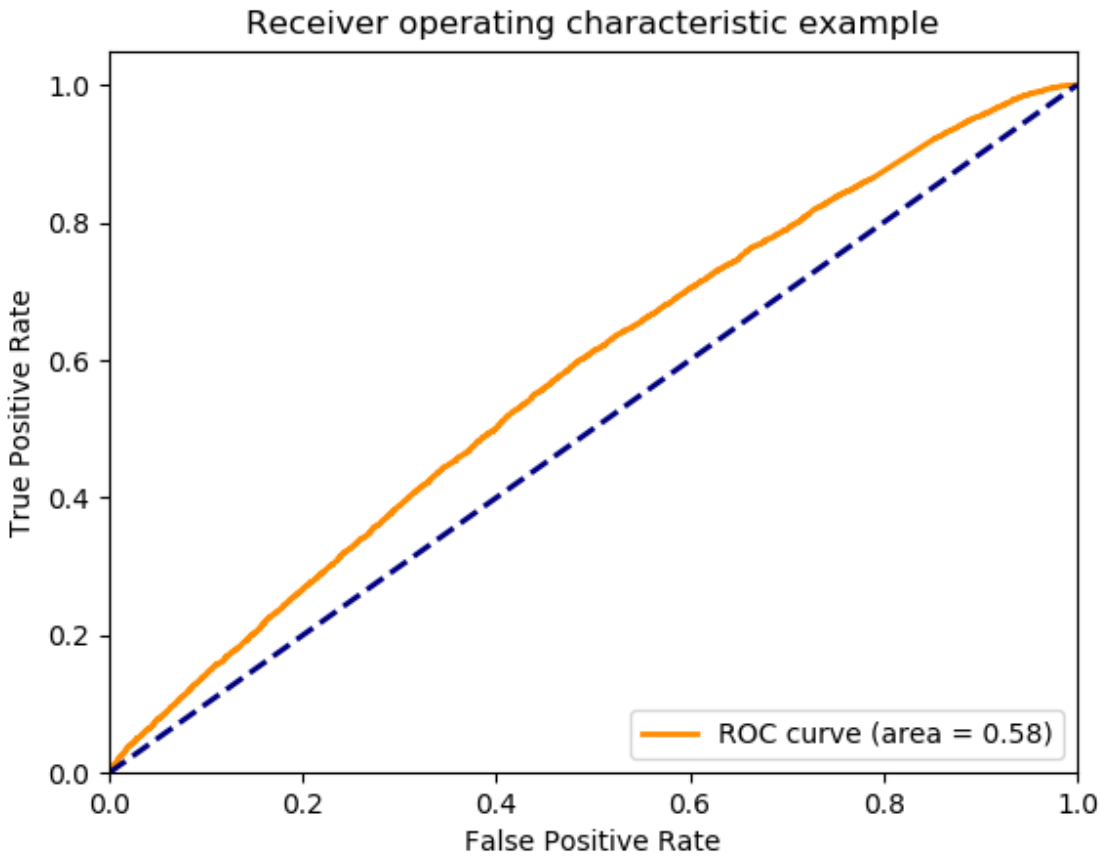

Figure 12: epochs = 1, training size= 50000, units = 64

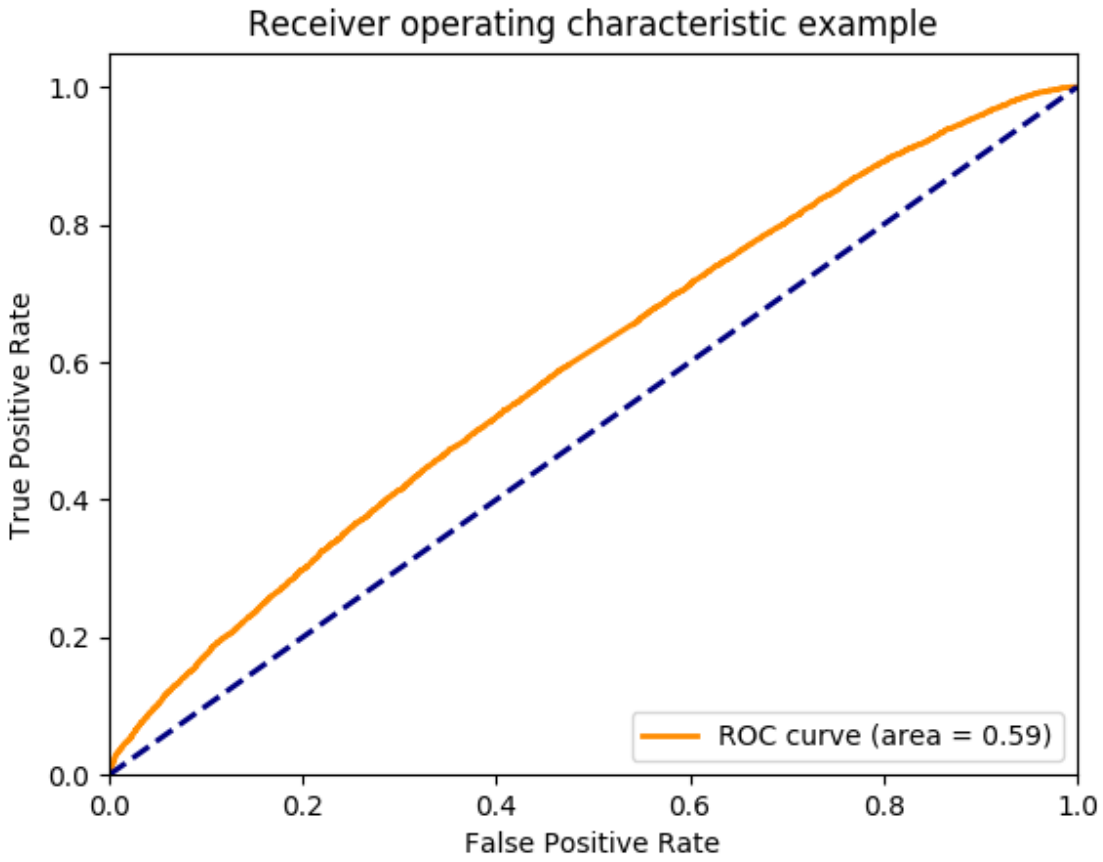

Figure 13: epochs = 3, training size= 50000, units = 64

Epochs	Train Loss	Train Accuracy	Valid Loss	Valid Accuracy	ROC Area
1	0.6805	0.5651	0.6748	0.5728	0.58
3	0.5336	0.7105	0.7448	0.5741	0.59

Figure 14: training size = 50000, units = 64

We continue comparing the same two epochs with the same conditions except hidden units. The following results trained with 128 hidden units.

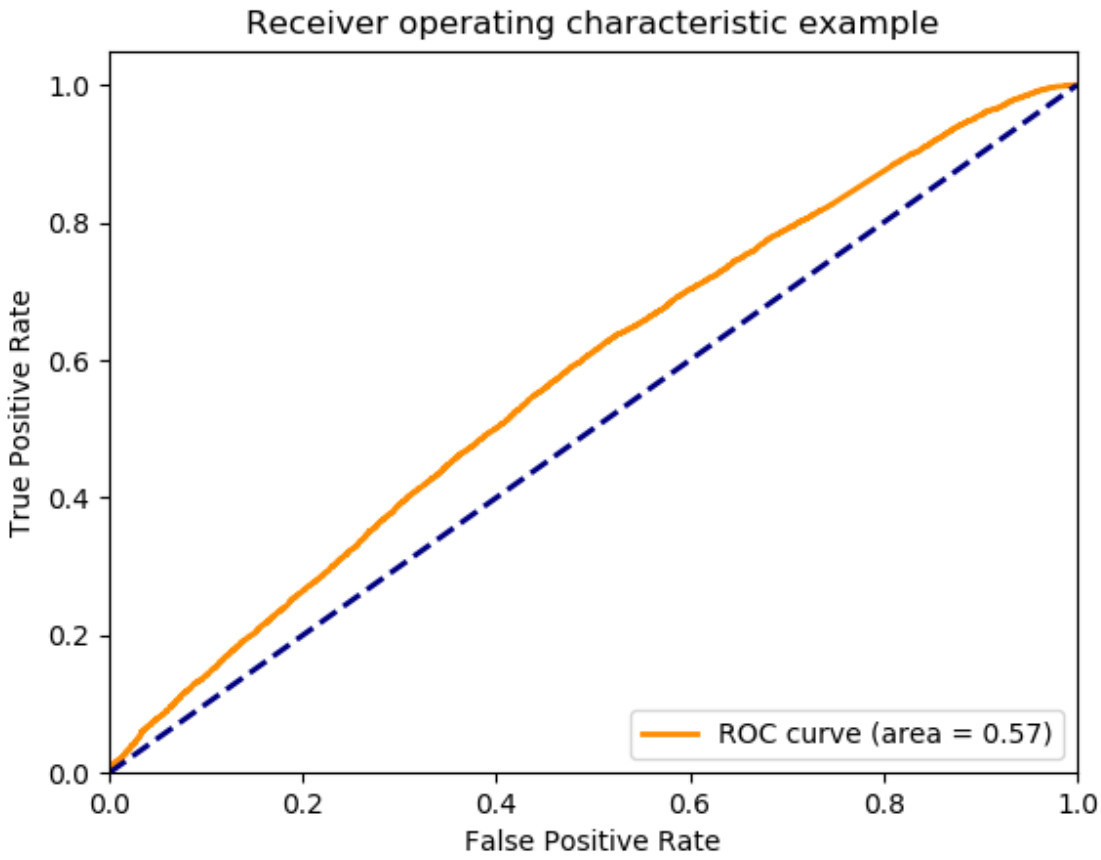

Figure 15: epochs = 1, data size= 50000, units = 128

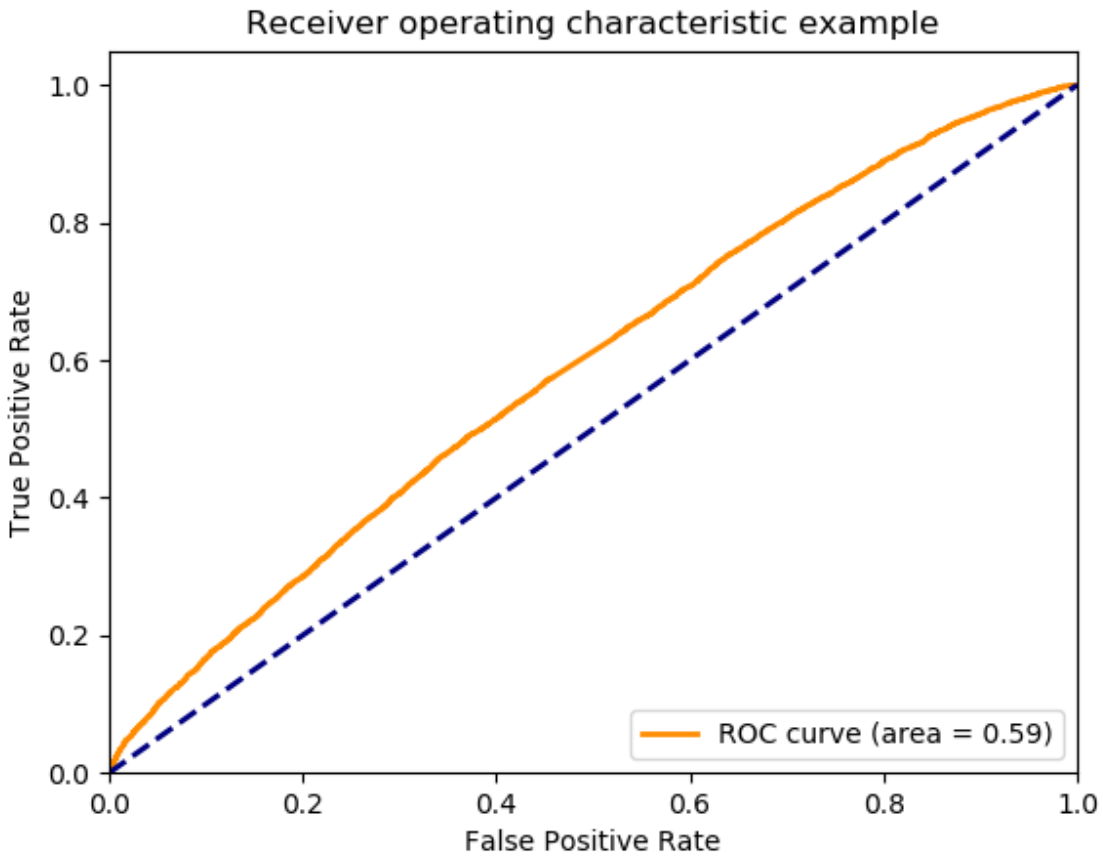

Figure 16: epochs = 3, data size= 50000, units = 128

Epochs	Train Loss	Train Accuracy	Valid Loss	Valid Accuracy	ROC Area
1	0.6811	0.5663	0.6751	0.5692	0.57
3	0.5413	0.7060	0.7621	0.5705	0.59

Figure 17: training data = 50000, units = 128

By observation, the performance is considerably improved by increasing epochs in both 2 kinds of conditions (hidden units). In conclusion, although increasing training data can help the performance, increasing epochs is still the best way to get high accuracy and low loss.

Besides the 2 methods metioned above, we also tried to use different learning methods. One potential solution is to use ***xgboost*** together with ***tf-idf*** encoding. Although

xgboost is not a deep learning model, it is widely used in text classification. However, we haven't got much achievement in accuracy. We assume it is because the poor preprocessing and cleaning of data. Due to limit of time, we give up trying more with ***xgboost***.