Yelp Fake Review Detection

Chuan Chen
cc6580@nyu.edu

Xinmeng Li
xl1575@nyu.edu

Junrong Zha
jz3741@nyu.edu

Dian Zhang
dz584@nyu.edu

Yichen Isabel Zhou
yz6126@nyu.edu

member for submission:
Dian Zhang

August 10, 2020

1 Introduction

Using a crowd-sourced review forum to guide everyday life behavior such as shopping, dining, and entertaining has become common since the internet has enabled everyone to communicate more freely and openly. However, this also gives incentives to post fake reviews. In this project, we seek to create processes with machine learning algorithms to better detect these fake reviews.

2 Approach

2.1 Data cleaning, EDA, and resampling

There are $\sim$ 250k samples in training set and $\sim$ 34k samples in validation set. The training set is imbalanced: only about 10% of the data is fake reviews (Fig 1). We first cleaned the text data by removing HTML tags, punctuation, and stop words. We then extracted root words and converted everything to lower case. Besides, we also added text length and word length as two extra features. The distribution of different features in training set is shown in Fig 2. We also plotted a word cloud to show the reviews after tokenization (Fig 3).

Since our dataset is highly imbalanced, classification results might be largely biased towards the majority class. From the example shown in the imbalanced-learn document [LNA17], we can see that the greater the difference between the number of samples in each class, the poorer the classification results. Since under-sampling the majority class might cause loss of information, we have decided to over-sample the minority class. We used RandomOverSampler(ROS) to over-sample the minority class to have the same number of samples as the majority class. However, since ROS simply replicate samples, it might cause overfitting. Thus, we have also used Synthetic Minority Oversampling Technique (SMOTE) to over-sample. Instead of replication, SMOTE generates synthetic minority samples through interpolations between minority neighbors. We have also implemented Adaptive Synthetic Sampling (ADASYN) which also generates synthetic samples but with a focus on the ones that are harder to classify. While SMOTE and ADAYSYN can alleviate the overfitting problem, they are inefficient for high dimensional data and might add additional noise. Thus, we will implement all three methods on our baseline and choose the best.

2.2 Word Features and Embeddings

Count Vectorizer and TF-IDF Vectorizer are widely used to produce text representation. We tuned hyperparameters including `ngram_range` and `max_feature`. `Max_feature` helps the balance between efficiency and performance, and `ngram_range` helps us capture as many semantic relationships as possible. To build word embeddings, we first applied Phraser on reviews to extract common phrases occurring more than 30 times, then we trained the Gensim's Word2Vec Skip-Gram model on the phrases corpus to generate high dimensional word representation of the nearby words probability distribution. We utilized multiprocessing to train the model distributedly across multiple cpu cores. We use the typical hyper-parameter settings for Skip-Gram such as a small window size 2 and the number of neurons 300. NRC Emotion Lexicon[MT13] is one of the most popular approaches to generate emotion-based feature. We use the affect frequencies of eight emotions, which are anger, fear, anticipation, trust, surprise, sadness, joy, and disgust, to detect the potential pattern of emotion fraud reviews.

2.3 User Behavior Analysis

Besides the review content-based analysis, analyzing the difference between spammers and non-spammers behaviors also provides insight to identify the fake review characteristics. We observe that the average activity window of nonspammers are 16.3 months while that of spammers is only 1.8 months, which indicates that spammers tend to review in short bursts. Similarly, the average of number of reviews of spammers is 2.3 while that of non-spammers is

7.7. Compared with non-spammers, spammers tend to deviate from the average rating of products, and write shorter reviews. From the above analysis, we added user behavior features including activity window, whether the user is only active in month, whether the user writes review for longer than 6 months, the maximum number of reviews a user post in a month, average of the user's ratings, user's review numbers, average rating of a product, the user's deviation from the average product rating, word counts in the review. Table 11 shows the comparison of the AUC and AP scores of each above feature on the Logistic Regression Classifier.

3 Experiments

3.1 SVM

We first used SVM to predict our reviews. After oversampling, we have 450110 instances in our training set. Since the `sklearn.svm.SVC` class is not efficient beyond tens of thousands of samples. We have resorted to using the linear SVM classifier with SGD training.

Out of the 3 oversampling techniques, ROS, SMOTE, and ADASYN, the 2 options of including and not including extra features, the 2 transformations of text, Bag-of-Words and TF-IDF, and the 2 ngram options of unigram and unigram + bigram, it seems that a uni-gram TF-IDF representation with the 3 extra features from ROS up-sampled data forms the best feature matrix for SVM training. The results of these single baseline implementations can be found in Table 1.

A grid-search was conducted on the linear SVM using both BoW with extra features and TF-IDF with extra features. The parameters and values searched can be seen in Table 2. From the grid search results of the BoW data, shown in Table 3, we can see that by allowing early stopping for our model, the mean in-sample cross-validated AUC has decreased but the out-of-sample validation AUC has increased, suggesting that early stopping has prevented some model over-fitting. Also, early stopping increased model fitting time efficiency for about 3 times. These results showed that the `early_stopping` criteria needs to be tuned separately to avoid selecting an over-fitted model. From the grid search, the best performing linear model, using the TF-IDF representation, has parameter values and evaluation scores shown in Table 4. The ROC curve of the best linear model is shown in figure 4.

Even after grid searching, the best auROC we got on the validation set was still mediocre. This could suggest that our feature vectors are not linearly separable. Thus, for further optimization, we have decided to implement the kernelized SVMs. Due to the large sample size of our data and the inability to use `sklearn.svm.SVC` class, we did not have the option to explicitly set a kernel during training. The `sklearn.kernel_approximation.Nystroem` class provides a kernel approximation method that takes a subset of training samples, instead of the entire sample space, to construct the kernel matrix [Mue12]. Although we have sacrificed learning of the real feature map, we have gained much more efficiency that makes kernel implementations achievable. A grid search was conducted over kernel parameters shown in Table 5. Out of all kernels, the `poly` kernel with parameter values shown in Table 6 performed best on the baseline SVM model. This kernel transformation was then used in the grid search of for the best SVM model parameters using values shown in Table 2.

Finally, the best performing SVM using the polynomial kernel has parameters and evaluations shown in Table 7. The ROC curve is shown in Figure 5. Although heuristic, this is not an improvement from the best linear SVM model. The best performing SVM model overall is the linear SVM with parameters shown in Table 4 trained with TF-IDF vector with 3 extra features upsampled using ROS.

3.2 Logistic Regression

As the input data in this study is a very sparse matrix with numerous features, the logistic regression classification model with l1 Lasso regularization then become very helpful which can complete feature selection automatically. By

trying both BoW and TF-IDF vectorization methods, `ngram_range` and grid-search, the performace is given in Table 8. Similar to training for SVM, adding ngram also has minimal or even negative effect on model performance.

3.3 Random forest and XGBoost

We also apply random forest and XGBoost algorithms on this problem. As previous experiments in SVM, We still try bag of Words and TF-IDF for text vectorization, and the same features for modeling. After grid search, the random forest predicts a best validation AUC of 0.68 with TF-IDF, and XGBoost predicts a best validation AUC of 0.706 with TF-IDF.

3.4 BERT

We also consider pre-trained BERT models, which can reduce training time by fixing the word embedding. Chosen the "bert-base-uncased", we first build a simple model that only looks at the first 128 tokens in each review. This is very efficient because many reviews are very long. Using Cross Entropy and Adam optimizer with learning rate of $2 \times e^{-5}$, we receive micro AUC = 0.48 and micro AP = 0.09 for validation set.

Next, to let the model work efficiently with different review lengths, we come up with a new idea that first split long reviews into chunks and then modify the algorithm of calculating `y_pred` and `y_score`. We assign each review an index number. For all reviews that are longer than 200 tokens, we split them into chunk of 200 tokens each, with 50 tokens overlapped. In this case, all splitted reviews have the same, short length and since chunks are overlapping, information can be carried to the next chunk. Now, each long review is corresponded to multiple rows in the new data frame and we still use the pre-trained "bert-base-uncased". However, to calculate the `y_pred`, some additional steps are needed. First, we need to group short review chunks based on review indices so that one group is corresponded to one piece of review. Then we look at all the predictions in this group. If pred = 1 occurs for any prediction in this group, meaning at least one part of this review is predicted as fake, this long review should be predicted as fake. We only mark this review as not fake to those with only pred = 0. Then if a review is predicted as fake (or not fake), we let `y_score` to be the maximum (or minimum) of all probabilities associated with this review. With these modifications, the validation AUC becomes 0.53 and AP becomes 0.12.

To tune this modified BERT model, we try different oversampling and undersampling ratios and various learning rate. Some results are shown in Fig 6. In conclusion, for BERT, the best model is using Adam optimizer with learning rate of $2 \times e^{-5}$ and undersampling 100%. We are able to get Micro AUC = 0.71 and Micro AP = 0.15. AUC and confusion matrix are shown in Fig 7 and Fig 8.

3.5 Hybrid Model of MutiLayer Perceptron and Logistic Regression

Due to the complex composition of our features, which includes word representation, word embeddings, emotion representation, and user behaviors, we first train subclassifiers on the feature subsets respectively and then linearly combine their predictions. First, we compare the performance and efficiency of LR, xgboosting and MLP on the TF-IDF features. Among them, logistic regression has a relatively high AUC score and fast speed. We then compare the performance of LR and MLP on the rest of the features. It turns out that Skip-Gram sentence-average embedding performs better with LR and NRC and user behaviors perform better with MLP. We combine all of the predictions of sub-classifiers by taking the average. We compare the AUC and AP score with or without each subfeature and remove the Skip-Gram feature which lowers the total score. To alleviate the fluctuations caused by the randomness in oversampling, we oversample and run the hybrid model three times and take the average of these three predictions. Please refer to tables from Table 12 to Table 16 for the results of hyperparameter tuning.

3.6 Dummy fake reviewer classifier

Besides `rating` and `review`, the feature `user_id` also provides vital information. Upon inspection, 16027 unique users are in both the train and validation set, meaning the user behavior present in the training set can largely contribute to the prediction of their reviews in the validation set. Intuitively, if a user has been known to post fake reviews, then it is more likely s/he will continue to do so. We have tried to create dummy variables but the shear volume of users are too large. Then we have decided to create a dummy classifier to reinforce our predicted scores. From the train set, we would record the user_id's of users who have posted only fake reviews, and users who have posted only genuine reviews. Then, when we encounter a sample in the validation set posted by a recorded user, we would manually overwrite the `y_score` to be 0.01 for known genuine users and 0.99 for known fake users. This overwrite value is set to the minimum train score and maximum train score respectively for `y_scores` calculated by the `decision_function`. Before implementing this reinforcement, the 5 best models and their corresponding auROC and AP can be seen in table 9. After implementing this dummy reinforcement on the `y_scores`, the 5 best models and their corresponding auROC and AP can be seen in table 10. As shown, the dummy reinforcement significantly increased both auROC and AP. Besides this extreme reinforcement using extreme known samples, we have also tried to set a softer threshold, such as overwriting samples that belongs to users who have posted more than 80% fake reviews or 80% genuine reviews. However, this did not really change the resulting auROC and AP, so we have chosen to not include these results in the report.

4 Conclusion

The best 5 models with dummy reinforcement was each fitted on the train + validation data, and the predicted score on the test data was submitted to the auto-grader. The model that resulted in the best test auROC and AP is the MLP+LR model.

5 Discussion

The kernelized SVM implementation did not improve, but rather worsened model performance, which is highly unexpected. One reason of this might be that, due to time and memory limitations, we were unable to produce any kernel matrix larger than 800×800. Since 800 is way smaller than the number of actual samples we have, our approximation may have been very poor. Also, we were unable to try the `additive chi2` and `chi2` kernels because our sparse matrix could not be converted to a dense matrix due to memory issues. It might also be that none of the provided kernel methods were suitable for our data. However, given the high dimensionality of features, it would be difficult to construct a self-defined kernel function.

In terms of feature generation, besides NRC, the emotion-intensity-based lexicon, there are various types of emotion indicators such as polarity-based. For example, with Bing Liu's opinion lexicon [HL04], we can summarize the opinion based on aspects and determine whether the review is positive, neutral, or negative.

For the analysis of user behavior, we failed on the content similarity task due to the computation power limit. Intuitively, spammers are prone to write highly similar reviews while non-spammers write true experience with low similarity.

As for our self-implemented dummy classifier, it has proven to be a success. Despite our various data processing and parameter tunning efforts, the auROC and AP have more or less stayed within the same range. However, the dummy classifier is the only approach that drastically improved our predictions. Future work might include fixing some of the issues mentioned above along with implementations of this dummy classifier to achieve even better results.

References

[HL04] Minqing Hu and Bing Liu. "Mining and summarizing customer reviews". In: *KDD '04*. 2004.

[LNA17] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning". In: *Journal of Machine Learning Research* 18.17 (2017), pp. 1–5. URL: `http://jmlr.org/papers/v18/16-365.html`.

[MT13] Saif M. Mohammad and Peter D. Turney. "Crowdsourcing a Word-Emotion Association Lexicon". In: 29.3 (2013), pp. 436–465.

[Mue12] Andreas Mueller. *Kernel Approximations for Efficient SVMs*. 2012. URL: `https://peekaboo-vision.blogspot.com/2012/12/kernel-approximations-for-efficient.html`.

Appendix A Tables

featurization	auROC	ap
BoW from ADASYN	0.674	0.187
BoW from SMOTE	0.678	0.187
BoW from ROS	0.711	0.195
BoW with extra features from ROS	0.713	0.198
TF-IDF with extra features from ROS	0.727	0.217
TF-IDF(1,2) with extra features from ROS	0.718	0.215

Table 1: auROC for single linear SVM implementation on different feature matrices

loss	penalty	alpha	tolerance	early_stopping
log	l2	10^{-5}	10^{-4}	True
hinge	l1	10^{-4}	10^{-3}	False
	elasticnet	10^{-3}	10^{-2}	
		10^{-2}		
		10^{-1}		
		10^{0}		

Table 2: parameter values for Linear SVM grid search

early stopping	True	False
time	1295.57	4741.60
best mean cross-validated in-sample auROC	0.691	0.717
best validation auROC	0.718	0.693
best validation ap	0.205	0.181

Table 3: linear SVM grid search results for two options of early stopping

alpha	early_stopping	loss	penalty	tol	auROC	ap
10^{-5}	True	hinge	l2	0.01	0.721	0.210

Table 4: best linear SVM parameters and evaluation results

kernel	n_components	gamma	coef0	degree
rbf	100	10^{-2}	10^{-1}	1
laplacian	300	10^{-1}	10^{0}	2
cosine	600	10^{0}	10^{1}	3
polynomial		10^{1}	10^{2}	
sigmoid				

Table 5: parameter values for kernel approximation grid search

kernel	n_components	gamma	coef0	degree
poly	600	1	1	2

Table 6: best performing kernel on baseline SVM

alpha	early_stopping	loss	penalty	tol	auROC	ap
10^{-3}	True	hinge	l2	0.01	0.713	0.206

Table 7: best SVM parameters using the polynomial kernel and evaluation results

featurization	auROC	ap
BoW with extra features from ROS	0.745	0.453
BoW (1,2) with extra features from ROS	0.743	0.453
TF-IDF with extra features from ROS	0.741	0.459
TF-IDF(1,2) with extra features from ROS	0.737	0.457

Table 8: auROC for L1 LogReg implementation on different feature matrices

model	auROC	ap
TF-IDF + Linear SVM	0.721	0.210
TF-IDF + Logistic Regression	0.725	0.217
TF-IDF + XGBoost	0.706	0.201
BERT	0.710	0.151
TF-IDF+UserBehavior+Emotion+MLP+LR	0.769	0.250

Table 9: The 5 best model performances on the validation set before dummy reinforcement.

model	auROC	ap
TF-IDF + Linear SVM	0.912	0.609
TF-IDF + Logistic Regression	0.741	0.459
TF-IDF + XGBoost	0.810	0.527
BERT	0.922	0.643
TF-IDF+UserBehavior+Emotion+MLP+LR	0.910	0.608

Table 10: The 5 best model performances on the validation set after dummy reinforcement.

User Behavior Feature	auROC	ap
activity window	0.635	0.135
single month user + long time user	0.634	0.134
maximum number of review in a month	0.532	0.108
user average rating+user review numbers+rating deviation from product average	0.652	0.161
word count	0.634	0.164

Table 11: auROC and AP for user behavior feature on LR.

Feature+Hyperparameter	auROC	ap
Behavior+Default	0.718	0.201
Behavior+Adaptive Learning Rate	0.724	0.205
Behavior+Alpha=0.01	0.724	0.201
Behavior+Learning Rate=0.1	0.637	0.135
Behavior+Learning Rate=0.01	0.710	0.195
Behavior+Learning Rate=0.0001	0.726	0.203
Behavior+Batch Size=50+Adaptive Learning Rate	0.725	0.204
Normalized Behavior+Default	0.659	0.144
NRC without Upsample+Default	0.560	0.137
NRC+Adaptive Learning Rate	0.611	0.151
Behavior+NRC+Adaptive Learning Rate	0.724	0.204

Table 12: auROC and AP for MLP Hyperparameter Tuning(Oversampled features by default)

Feature	auROC	ap
Behavior	0.713	0.190
NRC	0.561	0.137
Behavior+NRC	0.714	0.192

Table 13: Compare the performance of Behavior and Emotion Feature on LR

Iteration	auROC	ap
1	0.7671	0.2431
2	0.7663	0.2431
3	0.7665	0.2431
Average of 3 iterations	0.7668	0.2432

Table 14: Compare the performance of the Average of Three Iterations with each of Hybrid Model on LR

Feature	auROC	ap
Skip-Gram	0.714	0.202
TF-IDF	0.730	0.217
Behavior	0.712	0.188
Behavior(Normalized)	0.703	0.183
Average Prediction of TD-IDF and Behavior	0.735	0.215
Average Prediction of TD-IDF and Behavior(Normalized)	0.703	0.199

Table 15: Compare the performance of the features on LR

Model	auROC
XG-Boosting	0.545
Logstic Regression	0.687
Multi-layer Perceptron	0.652

Table 16: Compare the performance of TF-IDF(Not Oversampled) with various models

Appendix B Figures

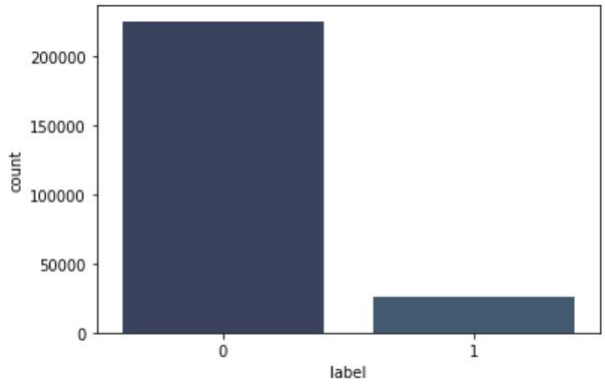

Figure 1: Count of genuine reviews (0) and fake reviews (1)

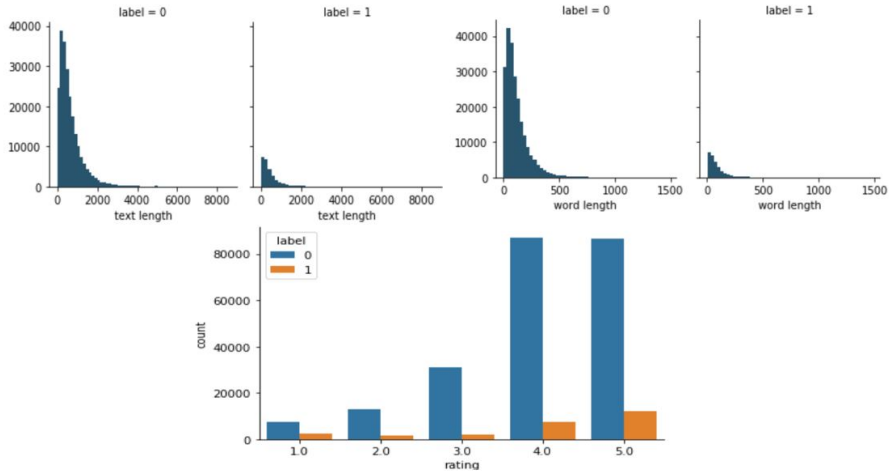

Figure 2: Distribution of different features: text length, word length and rating

Figure 3: Word Cloud for cleaned reviews

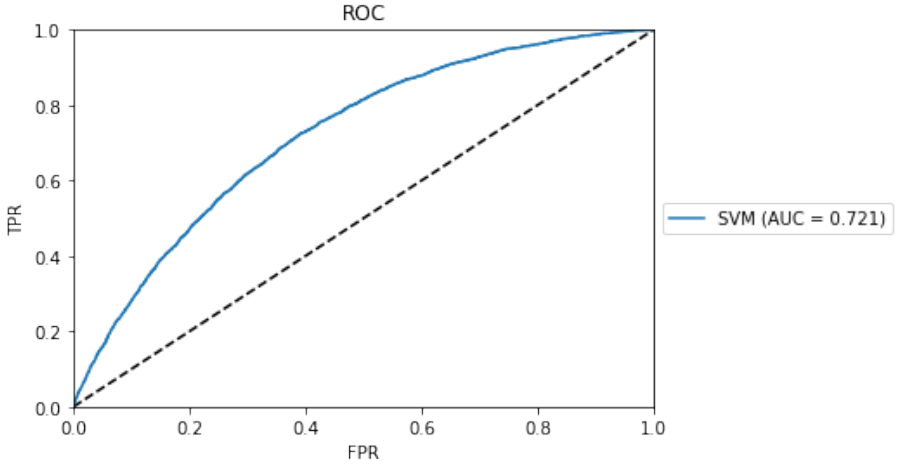

Figure 4: best linear SVM

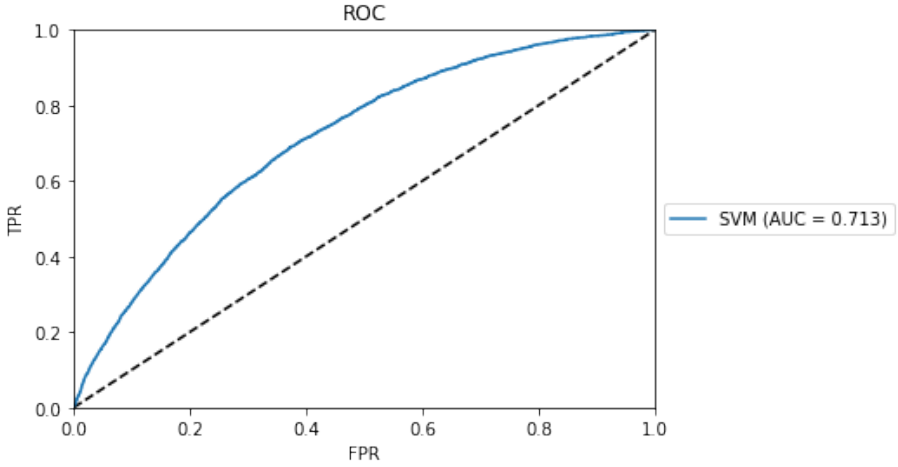

Figure 5: best SVM with polynomial kernel

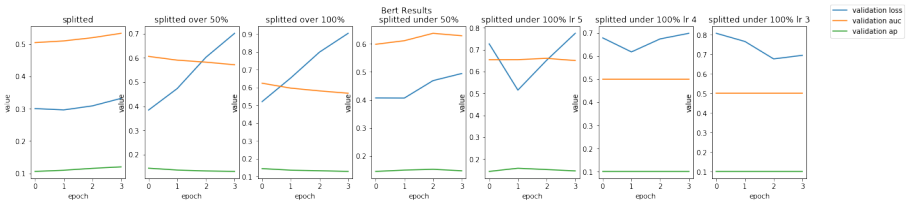

Figure 6: Results for different BERT models

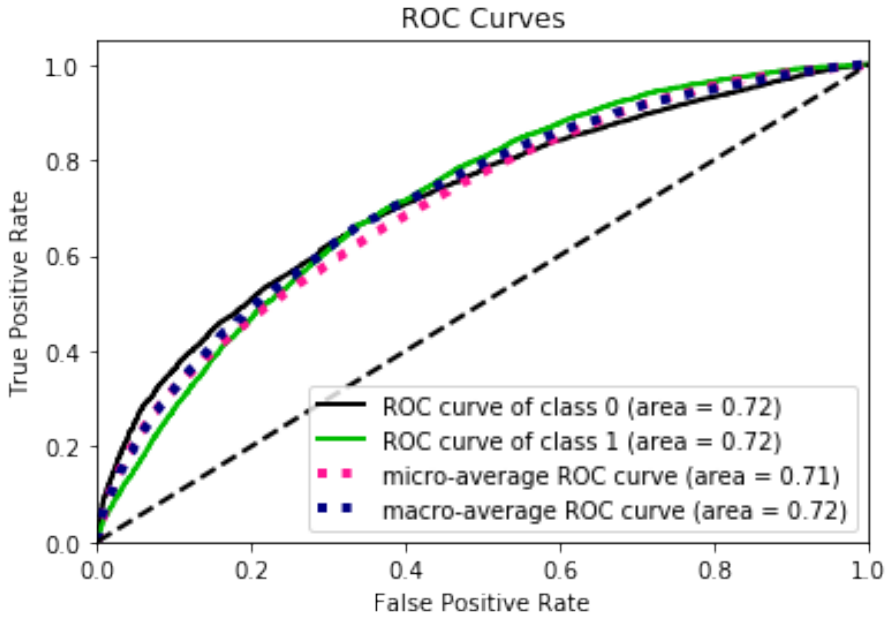

Figure 7: Best BERT AUC

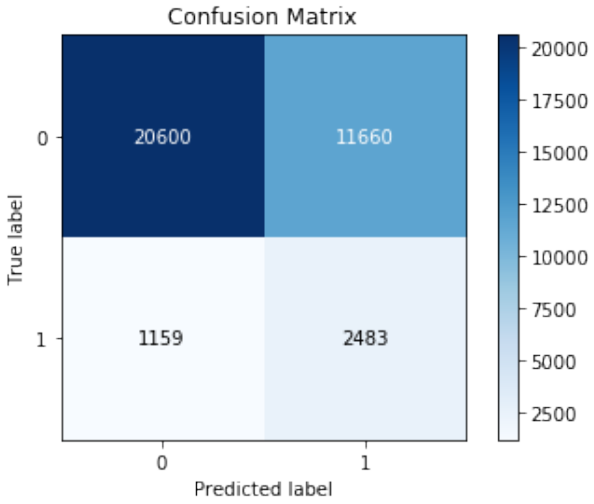

Figure 8: Best BERT Confusion Matrix

Appendix C Contributions

Chuan Chen: Data preprocessing, Resampling, SVM, dummy classifier reinforcement
Xinmeng Li: User Behavior, Skip-Gram, NRC, Hybrid Model, dummy classifier reinforcement
Junrong Zha: EDA, Feature Exploration, random forest, XGBoost
Dian Zhang: Data preprocessing, Resampling, Naive Bayes, LogReg
Yichen Isabel Zhou: BERT, Feature Exploration, dummy classifier reinforcement

Appendix D Source Code

All codes that implement methods described in this report can be found at our Github repository.