Hallo Community Growth Project

Michael Fryer, Cody Kesler, Eric Todd

Brigham Young University, Provo, UT

April 15, 2020

Abstract

Hallo's online platform is beginning to suffer from malicious users who threaten to disrupt the social community through their interactions with others. We experimentally examine the effectiveness of several methods through which Hallo can detect and flag these malicious users for removal. Various combinations of feature reduction techniques, sampling methods, and models are analyzed and compared to find the optimal classifier. We are able to correctly identify banned users 86% of the time, with an overall accuracy of 87%. With these promising results, Hallo will be able to provide its users with a better overall experience - generating community growth.

1 Introduction

Hallo is a startup based in Provo, Utah built around a global community for English language learning through a mobile application. The application allows users to: create a profile; message, chat, call, and friend other users; and view streams and videos of English teaching streamers. The community that naturally forms through interactions on the app is the focus of this analysis and project.

We met the founders of Hallo at the BYU career fair and discovered they had a need for data analysis and modeling. We partnered with their CTO to define the goals, scale, and data for this project. The CTO has provided access and guidance in working with the data under a loose non-disclosure agreement.

One of the primary concerns of Hallo is growing its community which is often stifled by *users* that send spam and other inappropriate messages, also known as *bad actors*. Bad actor detection is a problem faced by many companies, and machine learning is becoming a common way to identify and reduce unwanted content to provide a better experience for users. We explore various approaches and their effectiveness in identifying users participating in these activities to help Hallo flag and remove these unwanted users.

1.1 Background

Some research related to solving this problem has been done to identify bots, malicious users, and hackers on large social media platforms [1]. Other research includes using interactions and patterns of users to detect malicious behavior [2], taking advantage of a signed social network to evaluate users' behavior [3], or even detecting Wikipedia vandals from patterns of behavior [4].

While these methods have proven successful in other situations, given our data, community set up, and project scope, we do not directly apply them to our problem. We instead use their underlying principles to create our own methods for detecting bad actors on Hallo's platform. One area of research we do take full advantage of is email spam detection [5], which is a widely known and used technique for filtering spam emails. It takes advantage of the tf-idf (*term frequency - inverse document frequency*) differences between words in a spam message or normal message. We use this method to assist in identifying bad actors through the content of their messages.

1.2 Motivation

Hallo allows users to send text, audio and video messages to each other as well as participate in group chats. Both of these modes of communication allow for user engagement in the Hallo community and for improving language skills. Recently, there has been a surge of spam filling up these communication channels. Some users have been using messaging for other means such as advertising, pornography, or scamming. This greatly diminishes the learning experience and squanders community engagement. Thus, in order for Hallo to achieve community growth, effective detection and removal of spammers is crucial.

1.3 Problem Statement

Given the above concerns of Hallo, our work focuses on answering the following questions:

- How accurately can we identify bad actors on Hallo's platform to flag for review?
- What features contribute the most to being able to identify users that should be banned?

Focusing on these efforts will allow us to provide Hallo with a successful model to flag potential bad actors for review.

2 Data

2.1 Source

The dataset was obtained from Hallo's database which was stored in NoSQL format on Google's online suite, Firebase. This is the database format Hallo decided to use to collect data from their application. Most of the data obtained from Hallo has been programmatically processed and downloaded and thus has very minimal error. The cleaning primarily dealt with reformatting the data from its exported .json form to a Pandas Dataframe. See Section 2 of [6] for data cleaning details.

2.2 Format

Schema includes: "users", "friends", "messages", "calls", and "banned". These tables include data about users and their interactions with other users and the app. They also contain information about both banned (bad actors), and not banned users. The data relevance is given by the fact that it consists of recorded actions of our users and the usefulness is demonstrated by the success of our models in the following section.

The data used for this work comes primarily from the users, messages, and banned tables, with aggregate data and features being engineered using the information from the friends and messages tables.

The *users* table has information such as native country, native language, if the user is a streamer, if the user is an administrator, whether the user has been banned or suspended, English proficiency, etc. The *messages* table has information about the type of message, the message timestamp, user sender, and the users involved in the conversation. In the message data, a *conversation* is treated as the collection of all interactions between two users.

Feature	Type	Description
calls	Numeric	The user's average number of calls per conversation
link_sent	Numeric	The user's average number of links sent per conversation
missed_calls	Numeric	The user's average number of missed calls per conversation
audio_sent	Numeric	The user's average number of audio messages sent per conversation
image_sent	Numeric	The user's average number of images sent per conversation
text_count	Numeric	The user's average number of text messages per conversation
avg_msg_chars	Numeric	The user's average number of characters per message
avg_msg_words	Numeric	The user's average number of words per message
avg_msg_word_length	Numeric	The user's average word length per message
english_proficiency	Categorical	A self-reported score (0-4) of the user's English Proficiency
friend_cnt	Nominal	The number of friends that a user has on the app
followed_cnt	Nominal	The number of streamers the user is following

Table 1: User Features Description

2.3 Feature Engineering & Dimension Reduction

In addition to the original features included in the users table, we felt that aggregating friends and message data on a per-user basis would be helpful in being able to more accurately describe a user. We first created a set of features on a per-conversation basis; such as statistics about the number of text, image, and audio messages sent between two users, and about the text itself such as average message

length and average word length. With these features, we hope to capture the rate and level of interaction for a given conversation. We then aggregated these features on a per-user level to be used in prediction. The final list of per-user features can be seen in Table 1.

3 Ethical Considerations

Our problem does involve ethical issues, however as our algorithm will not be implemented as a judge, jury, nor executioner, our ethical concerns do not hold extreme weight. Our algorithm's purpose is to flag accounts for Hallo to manually review. With that said, we consider the following ethical considerations for our experiments:

- Using native country and language as features might increase our models' ability to predict who would and would not be banned. However, using those features could be considered discrimination since there might be a few users that could be flagged for spamming solely because of where they are from or what language they speak. Thus, they were not used.
- We have actual user data that does need to be anonymized, so we need to be careful about conclusions and what sorts of data we share.

4 Methods

The problem we set out to answer was whether we could predict bad actors that were using the app, and if so, what features help most in doing so. To accomplish this we decided to combine different machine learning classification models with different data types and techniques to assess the probability a user is a bad actor. Through this exhaustive and experimental approach, we are able to empirically examine the effectiveness of each method and take the best from each.

The first models we applied to accomplish this task are classifier models to hard-classify if a user should be banned or not using the user interaction and profile data. One of the challenges with the user feature data is that the number of banned users we have data for is 1437, while the number of unbanned users that we have is $250,098$. So, there is a definite class imbalance between banned and unbanned users. To tackle this issue, we tried both over-sampling and under-sampling our data to have a more balanced representation of banned and unbanned users.

The second method used is a Naive Bayes model used on a Bag of Words of spam and inappropriate messages. By analyzing the content from the messages of banned users, we can determine the probability of certain words appearing in a banned user's message and thus determine if a user's message is considered spam. This information provides another classification method to identify bad actors.

4.1 User Attributes as Features

Baseline Because our dataset consists of $250,098$ non-banned users, labeled 0, and 1437 banned users, labeled 1, the accuracy baseline of guessing all users should not be banned is **99.40%**. This class imbalance suggests that naive accuracy is not the best metric. We are primarily interested in correctly identifying users that should be banned, so we focus on the true positive rate (TPR) in all of our algorithms to make sure we are predicting the more sparse class appropriately. While TPR is important, we also want to make sure our models do not unnecessarily flag users as bad actors because this will create more work and manpower for the manual reviewer. Thus, we also report and take into account the true negative rate (TNR), precision (PRC), and accuracy (ACC) for each model to ensure our models and results are not biased.

Method To tackle this problem of classification given a set of user features, we employ and compare multiple feature reduction techniques and model types. We first approach this problem with naive classification using the user attribute features from Table 1. As the results in Table 2 show, the performance of these models is sorely lacking, though expected given the imbalanced nature of our data. We thus turn to sampling methods, hyperparameter optimizations, and dimension reduction.

We used under-sampling to create an evenly balanced training set by taking a random subset of the majority class, unbanned users. We then used a grid search of model parameters to find the parameters that gave the best TPR. While under-sampling was efficient for training and creating balanced classes, one potential drawback is that there were only 2874 data points and may not be representative of Hallo's

Model	TPR ↑	Precision ↑	TNR ↑	Accuracy ↑
Logistic Regression	0.46%	8.00%	99.43%	99.40%
Random Forest	2.55%	52.38%	99.44%	99.43%
XGBoost	5.32%	**57.50 %**	99.46%	**99.44**%
Gradient Boosted Trees	5.32%	29.11%	99.46%	99.38%
K Neighbors Classifier (KNN)	2.31%	27.78%	99.44%	99.41%
Linear Discriminant Analysis (LDA)	**15.05 %**	11.95%	**99.51 %**	98.88%
Support Vector Machine (SVM)	0%	0%	99.42%	99.42%

Table 2: User Features Results: Naive Classification

user base. However, in Table 3 we can see that under-sampling and hyperparameter tuning using a grid search did have a positive effect on the TPR, Precision, and TNR for each model.

Model	TPR ↑	PRC ↑	TNR ↑	ACC ↑
Logistic Regression	76.50%	82.43%	79.41%	80.76%
Random Forest	80.34%	80.92%	81.74%	81.34%
XGBoost	**80.58%**	81.95%	**82.12%**	**82.04%**
Gradient Boosted Trees	79.38%	80.54%	80.97%	80.76%
K Neighbors Classifier	76.74%	77.29%	78.40%	77.87%
LDA	74.34%	80.10%	77.52%	78.68%
SVM	76.01%	**83.42**%	79.29%	81.11%

Table 3: User Features Results: Hyperparameter Tuning & Under-Sampling

Given that under-sampling had such a positive effect and since it is known to lose information, using over-sampling methods combined with under-sampling methods could potentially yield higher performance using our hyperparameter-tuned models.

Model	TPR ↑	PRC ↑	TNR ↑	ACC ↑
Logistic Regression	79.27%	61.09%	92.88%	83.07%
Random Forest	**86.83%**	61.93%	**95.21%**	84.64%
XGBoost	80.24%	67.56%	93.46%	**86.14%**
Gradient Boosted Trees	82.44%	64.38%	94.00%	84.99%
K Neighbors Classifier	64.63%	60.09%	88.71%	81.39%
LDA	76.10%	55.91%	91.60%	80.06%
SVM	77.45%	**80.15%**	79.56%	79.84%

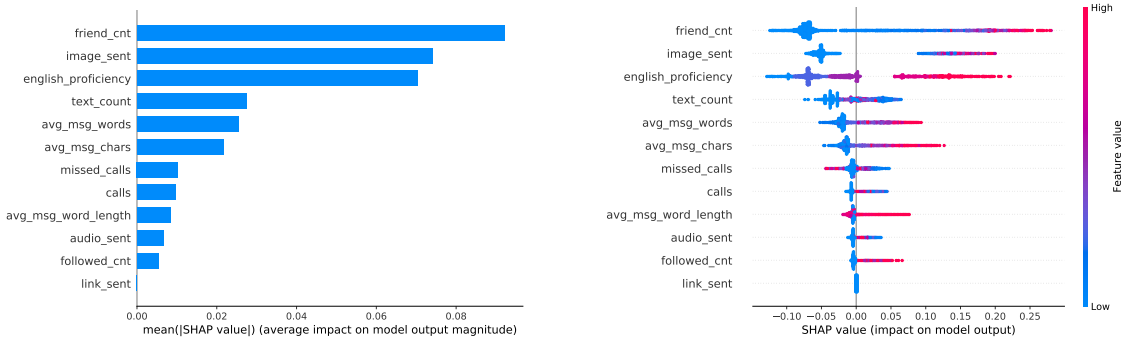
Table 4: User Features Results: Over-Sampling with SMOTE & Tomek Links

We used an over-sampling method called SMOTE (Synthetic Minority Oversampling TEchnique), which attempts to generate new observations of the minority class by randomly sampling along the line segments connecting a minority point to some of its k nearest minority neighbors. We combined our application of SMOTE with an under-sampling method called Tomek Links, which identifies pairs of observations of opposite classes that are each other's nearest neighbor and removes them if they are too indistinguishable. This serves to remove cases from the training set that are the most difficult to separate in order to give the classifier the best representative sample of our data. Applying these methods allowed us to almost triple the number of observations we used for our models to 7224. Table 4 shows substantially increased TNR, and a moderate improvement in TPR from applying this combination of over- and under-sampling methods for most classifiers we tested, though the precision was reduced.

Part of our goal was to identify a set of features that would be the most helpful to use in our predictions. Since there are only 1437 banned users, it is critical that we pick the most relevant features so that our models don't suffer from the curse of dimensionality. Our first approach to do this was to use a method called SHAP (SHapley Additive exPlanations) to test the classification contribution of each feature and to remove those that have little or no contribution.

SHAP attempts to explain the output of machine learning models using an approach from Cooperative Game Theory. It treats an observation as a coalition and computes the average marginal contribution of

each feature towards a model's output, across all permutations of the features [7]. This contribution is called a SHAP value. Since this is computed per observation, this allows SHAP to discover the average impact of a feature on a model's output across all observations.



(a) Average Impact of Features on Model Output

(b) SHAP Feature Contributions on Model Output

Figure 1: Features and Model Output

Applying this to our Random Forest model, we are able to gain insight into the different user features and their average impact on model output, seen in Figure 1(a). This tells us friend_cnt, image_sent, and english_proficiency were the three most impactful features. However, to understand how changing the value of a feature for a given observation impacts the model output, we examine the SHAP values for each observation and feature. This is plotted in Figure 1(b). The original feature's relative value is noted by the color bar, and the line thickness indicates multiple observations with that SHAP value.

The relative impact of each feature is the same as in Figure 1(a), but this plot allows more insight into how different values for a feature drive our model's decision for any given user. SHAP values greater than 0 suggest a feature contributes towards the model predicting "banned", while negative SHAP values indicate a feature contributes towards the model predicting "not banned". For example, we see that when observations have high values of English Proficiency (pink), this tended to suggest to the model that those users should be classified as "banned". Similarly, observations with lower values of English Proficiency (blue) tended to suggest to the model that these users were less likely to be banned.

We also see that having a lower number of missed calls influenced the model to classify someone as "banned" whereas having more missed calls contributed to the model classify someone as "not banned".

As can be seen in Figures 1(a) and 1(b), the feature link_sent has almost no impact on the model's choice and as a result, we can drop the feature. Since the other features have a significant effect on the models, we do not wish to lose the information they contain by dropping them. Hence, we turn to other dimension reduction techniques (DRT's) to continue reducing our feature space.

Since models do not benefit in the same ways from various DRT's, we ran multiple grid-searches that found the combination of hyper-tuned DRT and model that provides the best performance. As can be seen in Table 5, the models performed similarly to those we trained on under-sampled data (see Table 3). Hence, for most of the models, the benefits of a lower-dimensional feature space could not overcome the corresponding information loss.

Model	**Dim. Reduc.**	**TPR ↑**	**PRC ↑**	**TNR ↑**	**ACC ↑**
Logistic Regression	KPCA (Linear)	75.06%	81.94%	78.38%	79.95%
Random Forest	KPCA (Linear)	**79.61%**	80.98%	**81.23%**	**81.11%**
XGBoost	KPCA (Linear)	77.71%	81.00%	79.91%	80.41%
Gradient Boosted Trees	NMF	77.94%	79.85%	79.82%	79.84%
K Neighbors Classifier	Random Projection	76.25%	78.13%	78.30%	78.22%
LDA	NMF	73.86%	77.38%	76.55%	76.94%
SVM	KPCA (Linear)	75.78%	**83.16%**	79.09%	80.88%

Table 5: User Features Results: Over-Sampling with SMOTE & Tomek Links and Dimension Reduction

4.2 Message Content as Features

Baseline To set up and run this model with context we first establish a baseline calculation. Given the data for messages, we first need to see how well our Naive Bayes model classifies spam messages. For banned users, the average number of messages classified as spam is 47.40%, but the first quartile is 26.60%. Thus, if the percent of messages classified as spam for a user is above 26.60%, and we classify them as a banned user, we capture 75% accuracy for banned users. Not to be biased, we also look at the baseline for percent of messages classified as spam for users who were not labeled as banned. The mean value is 19.60%, but the 3rd quartile is 27.70%. Thus we can safely classify users who are not banned if we check for less than 27.70% of messages classified as spam.

Method Given a classifier which predicts spam messages, we are able to predict whether a user is a bad actor or not by building another simple classifier on top of this message classifier. We calculate the number of messages considered spam by the Naive Bayes algorithm for a given user. If this percent of the messages classified as spam is above 25%, then the classifier predicts that the user is a bad actor and that they should be banned. The results of this method are shown below in Table 6.

Model	TPR ↑	PRC ↑	TNR ↑	ACC ↑
Naive Bayes Classifier	73.93%	15.11%	97.63%	72.12%

Table 6: Bad Actor Detection via Message Content

4.3 Ensemble

In order to best take advantage of the classification models we have created, we combine the models to create an ensemble that maximizes the use of each model. To do this, we use a Neural Network to learn the weights that each model should have in an ensemble-like prediction. Learning the weights that each classifier should carry in an ensemble is a novel method we use which was not previously covered in class. The details of our implementation are discussed below.

Baseline We first establish a baseline accuracy of just a straight average among the models with the best TPR. The baseline achieves **85.11%** accuracy with the rest of the results shown in Table 7.

Method After establishing a baseline, we first, create a weighted ensemble based on how accurate each model is, second, based on the TPR of each model, and third, we use a Neural Network to take the predictions from each model and learn a weighting to achieve the best possible accuracy. We also note that because each of our models are far from being weak, the 7 models we do use should be enough to take advantage of the benefit that ensembles provide.

Model	TPR ↑	PRC ↑	TNR ↑	ACC ↑
Baseline Average	**79.16**%	54.28%	**95.40**%	85.11%
Accuracy Weighting	78.47%	53.93%	95.25%	85.24%
TPR Weighting	78.81%	53.91%	95.32%	85.46%
Network Weighting	72.57%	**60.00**%	94.24%	**87.00**%

Table 7: Ensemble Methods Results

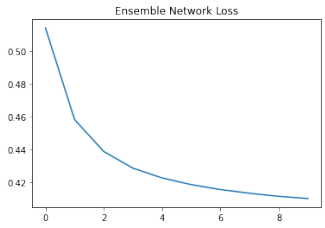


Figure 2: Ensemble Network Loss

To establish weightings for the models using accuracy and TPR, we took the given metric for the models and performed a simple normalization of dividing each percentage by the sum of the total percentage. This gave a normalized score for each model weighted by how well it did.

Each model's prediction of whether a user should be banned is fed into the network. The input then passes through 4 linear layers, 7x100, 100x50, 50x20, and 20x2, each with a ReLU layer in between. After the last linear layer, a Softmax layer produces probabilities for the two classes. A cross-entropy loss is then used to calculate the loss for the output and target label. The network's job is to combine the prediction from each model in the best way possible to predict the status of the user. Results achieved are shown in Table 7. Because of the nature of deep learning, and sensitivity to hyperparameters, training, and data, a simpler network is preferred. Figure 2 shows the loss curve for training the network indicating learning and convergence of the network.

4.4 Methods Not Used

Because the nature of this problem is classification, the valid models to run are all classification models. We examine each type of machine learning and why it did not fit within our project scope.

Regression The problem for this work is classification and thus no regression methods are needed to achieve results toward our end goal.

Time Analysis While our message data did have aspects of time and frequency, the most analysis needed to take advantage of this data is message counts. Thus no time analysis algorithms are needed to classify bad actors. It is, however, a possible area of exploration, given more data, to analyze the frequency of communications of users in relation to identifying bad actors.

Clustering While we did run clustering algorithms, our data did not fit their underlying assumptions (Gaussian distributed, etc.) and as a result, the performance for these models was far below sub-par. Thus, we did not see it fit to include this in our analysis.

5 Results

User Attributes as Features As stated above, we decided to use the TPR as the primary measure of the success of our banned user classifier. The TPR is a better measure than straight accuracy because we are interested in positively identifying users who really should be banned, and takes into account the class imbalance. Success in other metrics (TNR, PRC, & ACC) show our models are not just predicting all users should be "banned".

Under-sampling to balance our training data to a 50/50 split between classes greatly improved the identification of banned users for all models. Our best results after applying under-sampling were from the trained XGBoost Classifier, which achieved a **80.58%** TPR, and **82.12%** TNR.

We were able to improve our TPR even further when we applied a combination of over- and under-sampling, (SMOTE and Tomek Links), to our data. Our best results were from a trained Random Forest Classifier which was able to positively identify banned users **86.83%** of the time, and had a TNR of **95.21%**. Even though this model improved significantly (6.35%) in regards to the TPR of the previous best, the average improvement per model was only 0.25%. This is, in part, because the K Neighbors Classifier (KNN) trained on the over-sampled data did 12.21% worse compared to under-sampled data. Our models were able to correctly identify the increased number of non-banned users, and we saw the TNR improved on average by 12.6% per model.

Our best results after over-sampling and using DRT's were from the Random Forest Classifier combined with Kernelized-PCA which achieved a TPR of **79.61%** and **81.23%** TNR. We also note the precision increased back to the ranges of the purely under-sampled data, while the TPR and TNR did not improve, indicating the over-sampling may be introducing some undo complexity that is counteracted by the dimension reduction.

One reason the KNN struggled on our up-sampled data is that it uses distance to vote with the nearest observations. The corresponding drop in TPR (compare Tables 3 & 4) and improvement after dimension reduction (see Table 5) suggests that the larger sample of users is not as separable as KNN assumes and that other methods would do better.

Message Content as Features With this Naive-Bayes type method, we are able to achieve a TPR of **72.15%** and a TNR of **74.0%**. This method under-performs the best performance we were able to obtain from the user features, but it still is a vital part of identifying a bad actor. This method uses different features from the other models and thus could be useful for balancing the TPR or TNR of the models derived from the user features.

An unfortunate limitation of this method is that creating the tf-idf table is memory intensive since there are many unique words in messages sent between English learners. Thus, this model was only able to be trained on 150000 messages at a time (25GB - Colab's max), but was still able to achieve relatively decent performance.

Ensemble Combining the different prediction models based off of the user features and the message features provided improved results for certain metrics. Because each model uses different methods and features for prediction, we get better results by combining the predictive power of each model. It was able to obtain a highest accuracy of **87.00%**, TPR of **72.57%** and TNR of **94.24%**.

6 Analysis

Our first experiment dealt with under-sampling. Since this balanced out the number of banned users, all the models had a significant increase in TPR and PRC. Since under-sampling reduces each model's exposure to non-banned users, it was expected that TNR and ACC would decrease. XGBoost was the most effective, because the limited amount of data allowed us to search through more parameters to find the best ones. However, both ensembles did perform well for imbalanced classification.

In the next experiment, we tested a mix of over- and under-sampling. There was a general trend of increases in TPR, TNR, and ACC. However, PRC dropped significantly which means the predictions were saturated with False Positives. Since SMOTE upsamples the minority-class, overfitting is likely, which explains the increase of False Positives. The best performing model was Random Forest partly because Random Forests are less prone to overfitting than other models, including XGBoost.

By examining these models, we found the features that contributed the most in predicting a user should be banned were: friend count, images sent, and English proficiency. We found having more friends led to a more likely prediction that a user should be banned. Naturally, this makes sense since spammers will try to increase their reach. English proficiency had a similar correlation, with higher English proficiency being linked to more likely predictions a user should be banned. Since English proficiency is self-assigned, perhaps bad actors are using a high rating to gain trust in the community to then target and take advantage of newer English learners on the app. We dropped link_sent due to its low impact, in an effort to reduce our feature set.

This led to testing various dimension reduction techniques, where we combined SMOTE with the best DRT for each model. This lead to an increase in PRC and a decrease in TPR which implies that False Positives became less frequent at a cost of more False Negatives. However, for each model, TPR was still within roughly 5% of the models without DRT's. The model that performed the best was Random Forest with KPCA using a linear kernel (equivalent to regular PCA). Again, our ensemble methods seemed to be outperforming the other models in this case.

Our naive classifier for focusing on messages provides a different perspective on identifying bad actors. The method used was able to perform at 74% TPR and 97% TNR. These results are in part due to a message content overlap of bad actors and regular users of 25% as noted above. The Naive Bayes approach also assumes independence of the messages which, in a real-world problem, is not feasible but works okay in practice. Because this model uses features which our other models have not used, it will be valuable in an ensemble method with the previous models.

The use of different methods inspired the creation of our network ensemble which attempted to consolidate the unique strengths of our other models. However, the lack of balanced data forced us to train each voting model on the same data, reducing their independence. This weakened the strength of our network ensemble since one underlying assumption providing ensembles their strength requires that each voter be independent. Given more data, bagging methods could be used to increase voter independence. In such a case, we hypothesize the network ensemble would perform the best overall since it could find more complex relationships using the combined power of the voters.

We also note the ensemble results at first glance seem to not have improved much and in fact achieves worse results than some of our other methods, the caveat though is that the network has the highest accuracy of all the methods. We believe these results are caused by two factors. First, our ensemble method is trying to learn to predict whether a user should be banned using the predictions of other imperfect models as inputs. This limits the confidence it can have for the scores it receives for a given user and compounds the error from the 7 different models. Second, our network was able to achieve high accuracy with low scores for the other metrics because it is trained with a cross-entropy loss. This objective only cares to maximize the overall accuracy without regard for the other metrics.

7 Conclusion

Our results address the problem of identifying bad actors on Hallo's platform. In order to prevent bias and achieve the best results in our algorithms, we made sure to explore dimension reduction, understand features' importance, grid search over different models, and combine models using ensemble learning, all while keeping in mind true positive rate, true negative rate, precision, and accuracy.

After our experimentation, a highest accuracy of 87%, TPR of 86%, and TNR of 95% were achieved. Our final models are able to predict bad actors confidently while not falsely identifying too many non-offensive users. These results are extremely promising and will allow Hallo to efficiently flag and remove bad actors to maintain and grow a credible platform for English language learning.

References

[1] Srijan Kumar, Francesca Spezzano, and V.S. Subrahmanian. Identifying malicious actors on social media. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*. IEEE, 2016.

[2] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Inferring strange behavior from connectivity pattern in social networks. volume 8443, 05 2014.

[3] Evelien Otte and Ronald Rousseau. Social network analysis: a powerful strategy, also for the information sciences. *Journal of Information Science*, 28(6):441–453, 2002.

[4] S. Kumar, F. Spezzano, and V. S. Subrahmanian. Accurately detecting trolls in slashdot zoo via decluttering. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 188–195, 2014.

[5] Lauren Steimle Lydia Song. Email spam detection using machine learning.

[6] Eric Todd Michael Fryer, Cody Kesler. Hallo community growth data project. BYU, 2019.

[7] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. Explainable ai for trees: From local explanations to global understanding, 2019.