Automatically detecting player roles in Among Us

1st Harro Tuin
Serpentine AI
Eindhoven University of Technology
Eindhoven, the Netherlands
among-us-2021@serpentine.ai

2nd Martin Rooijackers
Serpentine AI
Eindhoven University of Technology
Eindhoven, the Netherlands
among-us-2021@serpentine.ai

***Abstract*—Player role identification in the online social deduction game Werewolf has been a common research topic in the past decade. Among Us, a fairly new online social deduction game, gained a lot of popularity recently. Given the popularity of the game, an opportunity arises to extract information on social deduction.**

This research focuses on extracting information from emergency meetings in Among Us and subsequently using this information to automatically detect player roles. First, a framework is presented that can be used to extract information from videos with gameplay of Among Us. This framework can extract the chat messages from the emergency meetings, detect imposters at the end of the game, and extract voting data.

Secondly, the framework is used to process videos with 59 games of Among Us gameplay. The data produced by the framework will be normalized and transformed into numerical data using term frequency-inverse document frequency (tf-idf). Finally, two types of Support Vector Machines (SVM) and a Naive Bayes classifier are used to automatically detect player roles. We show that detecting player roles is a difficult yet learnable challenge.

***Index Terms*—Data extraction, NLP, Machine Learning, SVM, tf-idf, social deduction, Among Us**

I. Introduction

In the past decade, there has been a lot of attention in automatically classifying player roles in the Werewolf game in which only imperfect information is available [1]–[5]. A relatively new online video game with imperfect information is the social deduction game Among Us.

The game Among Us is played online with up to ten players. Among these players, there is one (or more) imposter who tries to eliminate the other players, called crewmates. The identity of the imposter(s) is unknown to the crewmates. The crewmates can win the game by voting out all impostors in the game, or by completing all tasks. If the imposter achieves to eliminate all crewmates before all tasks have been completed, the imposter wins the game. However, players (including the imposter) can call emergency meetings in which they can discuss (via chat or voice) and vote on who they think is the imposter. Each emergency meeting the player with the most votes is eliminated from the game (the vote can also be skipped).

The goal of this research is twofold: (1) provide a framework that can be used to extract data from Among Us gameplay and (2) provide a baseline for classifying player roles using machine learning, based on the chatlogs of the emergency meetings.

While communicating through voice is possible in Among Us, the framework focuses on extracting information from games where only chatting enabled. The main reason to only use chat is because chat messages are easy to associate to the respective player, whereas with voice this is not necessarily the case.

The data extracted by the framework has to be preprocessed in order to use machine learning methods. The variation of the words used in the messages found by the framework will be reduced using various techniques. Subsequently, the words will be transformed into numerical vectors using term frequency-inverse document frequency (tf-idf) weighting.

For the classification of player roles, three different methods will be compared. The first two models (with linear, and RBF kernel) are a type of Support Vector Machine (SVM). Previous research showed that SVMs can achieve high accuracy on classifying short phrases with simple syntax in the Werewolf game [2]. The performance of the two SVM models will be compared to a probabilistic classifier called Naive Bayes. The performance of the machine learning methods is validated by using k-fold cross validation.

II. Framework

A framework is built to extract information out of videos of Among Us gameplay and is open source[1]. The framework uses the OpenCV library to go through a video frame by frame [6]. For each frame, a check is done to see if this frame is a chat window, voting screen, or end screen. Based on which type of information is displayed in the frame, the frame is sent to the appropriate subroutine. An example of the framework locating the messages and player names during an emergency meeting in Among Us is displayed in Fig. 1.

A. Features

The framework extracts several features from a video with Among Us gameplay. It can detect and read the chat messages that are sent during emergency meetings. Furthermore, it can extract voter data like finding who started the discussion, who voted for whom, and the color of the body that was reported (if the discussion was started this way). Next to that, it can

[1]Our code can be found on:
https://github.com/TeamSerpentine/among-us-2021

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

Fig. 1. Example of text detection.

extract which of the players are an imposter at the end of each game.

B. OCR for chat messages

The main part of the framework that is used in this paper is the optical character recognition (OCR) of chat messages. These messages can be extracted any time the chat screen is open in the video that is being analyzed. The following process is used to extract the text from a chat screen:

1) Re-scale the image to a 1920 by 1080 pixel resolution, this way the feature used to detect a chat screen (see Fig. 1) is always at the same location.
2) The image is checked to see if the chat screen is properly opened. To do so, the algorithm checks for a pattern "$x/100$" (with x being any natural number) at the bottom right part of the screen (again, see Fig. 1 for an example). If it exists, then we know that the chat screen is fully opened and can process the frame with OCR.
3) Turn the image to grayscale, then threshold with a value of 245 (with a max of 255) to filter out messages from players already eliminated from the game. This will create separate white rectangular boxes with usernames followed by their chat message. We can locate these texts with the EAST text detection [7] and process them with Tesseract OCR engine [8] to turn the images into JSON files containing the messages.
4) The left side of these boxes (or the right side, if the message is sent by the player itself) contains an image of the player who sent the message. These locations are cross-referenced with the original image to find the color of the player who sent that message.

C. Detecting impostors in training data

In the videos with the imposter ejection confirm functionality turned on, the framework can parse the video and (potentially) extract who the impostors are. There are two aspects to this, (1) detecting who gets voted out, and (2) the end screen which shows the impostors if they win.

The voting data is acquired by checking the voting screen when the votes are processed. If a player has the most votes, this player will be eliminated, resulting in a message several frames later which will indicate if this player was an impostor or not. Without confirm eject, the framework can still find all impostors if the number of players eliminated is equal to the number of impostors.

If the impostors win, the end screen will show all impostors. If this is the case, the framework can detect all player models and extract the colors from them.

D. Voting data

There are three types of data extracted from the voting screen. First, who started the meeting. Second, the color of the body that was reported to start the meeting. Third, the framework can extract who voted for whom (in case voting is not anonymous), and when the vote was cast.

The first two types of data can be determined from the announcement of a meeting. This frame will have the person who started the meeting, and the body (with color) if the meeting was called due to a body being reported.

The third piece of data can be extracted by looking at the icons that appear under someone's username when voting ends.

III. Data preparation

The framework is used to process 59 unique videos with Among Us gameplay, with each video containing one full game. While the framework extracts additional features this research only focuses on using the chatlogs.

A. Preprocessing

After the framework has processed the gameplay, some initial checks are done. The first check is to see if the number of imposters matches the number of colors found as imposters by the framework. During this check, 4 files are found in which the numbers do not match. For example, this could be due to an imposter leaving the game before the game has actually ended. These 4 files are therefore excluded from further processing. After completing the checks, a label is added to each message indicating whether it was sent by an imposter (label = 1), or by a crewmate (label = 0). The messages and the labels are the only data that is used for further processing.

B. Natural language processing

In order for a machine learning algorithm to learn a function that can map the data to the corresponding labels, the data has to be transformed from natural language to numerical values.

1) Tokenization: First the messages found by the framework are *tokenized*. Tokenization is a technique that splits up strings of text into tokens ("words") based on spaces and punctuation. This implementation uses word level tokenization based on spaces.

2) Text normalization: The variation across tokens will be minimized to reduce randomness, and increase the machine learning performance. Non-alphanumerical characters are removed from the tokens, and all characters will be transformed into lowercase. Furthermore, tokens will be lemmatized. Lemmatization is the process of grouping words together in their inflected forms. Effectively, this means that all morphological affixes from words are removed. This paper uses the WordNet lemmatizer implementation of the Python Natural Language Toolkit (NLTK) to lemmatize [9].

3) Stop-words: Stop-words are words that occur frequently but do not contain useful information. The collection of stopwords as listed by NLTK is used to remove the stopwords.

4) Word Vectorization: Term frequency-inverse document frequency (tf-idf) is used to transform text documents into vectors. Tf-idf gives more weight to words that are frequent in a specific document but infrequent over all documents. Equation 1 specifies how weight is given to a text document. The weight is found by multiplying the term-frequency of term i in document j by the log of the total number of documents N scaled by the number of documents containing i:

$$w_{i,j} = tf_{i,j} \cdot log(\frac{N}{df_i}) \tag{1}$$

In order to transform the tokens into numerical features a zero matrix of size $m \times n$ is initialized, with the messages as rows and unique tokens as columns. Subsequently, if a word is in a message (row) the zero corresponding to that word will be replaced by the tf-idf value.

IV. Experiment

In this experiment, the messages from Among Us gameplay are analyzed and automatically classified in terms of player role. Effectively, this is a binary classification problem with the labels: imposter (label = 1) or a crewmate (label = 0).

Three models will be trained and evaluated. The first model is a Naive Bayes classifier, this model is used because the model is fast, and hyperparameter tuning is relatively simple.

Secondly, SVM is used with both a linear and RBF kernel. Both models use the LIBSVM implementation [10].

A. Input data

The input dataset is a sparse matrix with tf-idf values. The dataset contains 1115 messages after going through the data preparation steps, however, following these steps some messages become empty (e.g. only stopwords). Therefore, these messages are excluded from the dataset. Following this exclusion, 811 messages are left.

In this research mainly games with a single imposter were used as input data. Consequently, if the imposter sends as many messages as a single crewmate, this will lead to an imbalance in the labels of the data. To decrease the effect of imbalance in the data, the weights of the classes are adjusted inversely proportional to the class frequencies in the input data. Since SVM models are distance-based all features need to be on the same scale. Since the features in the data contain the tf-idf values, no further scaling is needed.

The dataset is split into two stratified sets to make sure the machine learning methods generalize well. The trainset will contain 80% of the preprocessed dataset and the testset the remaining 20%. The trainset will be used to train and validate the model hyperparameter settings, this is done by using k-fold cross validation with 5 folds. The testset is used to evaluate the model after tuning the hyperparameters.

B. Hyperparameter tuning

Gridsearch is used to find the optimal hyperparameters for all three models. For the SVM, 15 different parameters *C* and γ will be tested on a logarithmic scale between 0.01 and 64. For the Naive Bayes classifier, the fit_prior parameter will be true or false. The fit_prior indicates whether the class prior probabilities should be learned or not [11]. When set to false, a uniform prior will be used. The smoothing parameter α will be varied on a linear scale between 0.1 and 1 with a step size of 0.1. ROC AUC score is used as an evaluation metric since this yields the best overall models [12].

The optimal parameters found by the gridsearch for the SVM with the RBF kernel are $C = 0.1$ and $\gamma = 0.03$. For the SVM with a linear kernel multiple parameter settings give the same results, we choose the median of the optimal parameters: $C = 0.3$, $\gamma = 1$. The best results for the Naive Bayes classifier are without setting priors and with smoothing parameter $\alpha = 1e^{-7}$.

C. Results

The results of the models with tuned hyperparameters evaluated on the testset are displayed in Fig. 2, Fig. 3, and Fig. 4.

V. Discussion

The results show that the SVM with the RBF kernel has the highest average AUC score (0.62) over all folds. The figures show a red dashed line which indicates predicting the labels correctly by chance. In terms of average score, the performance of the SVM with the linear kernel is slightly worse (AUC score 0.57) compared to the RBF model. Similarly, the Naive Bayes classifier also achieves an AUC score of 0.57. Furthermore, Fig. 2 shows a very smooth curve compared to Fig. 3 and 4. This could be due to the independence assumptions between the features, which reduces the number of parameters exponentially. For all models, the scores are above the line that indicates chance. From the figures, we can see there is a lot of variation between the folds with AUC scores ranging from 0.58 to 0.68 for the SVM with the RBF kernel. It is expected that this variation is quite strong due to the small size, and imbalance, of the dataset.

VI. Limitations

The main limitations of the research in this paper are the following: the OCR and color detection of the framework, the amount of available data, preprocessing of the data, and the imbalance in the labels.

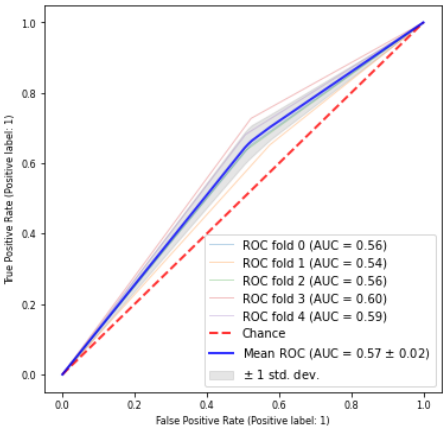

Fig. 2. ROC AUC curve Naive Bayes

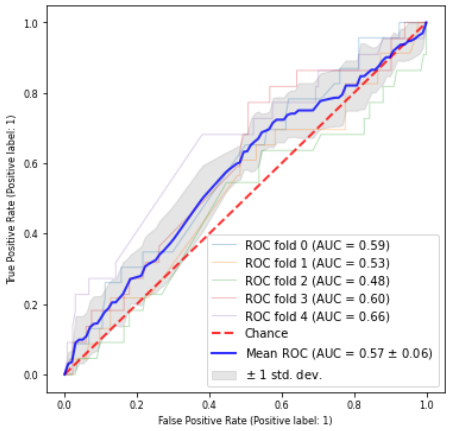

Fig. 3. ROC AUC curve linear SVM

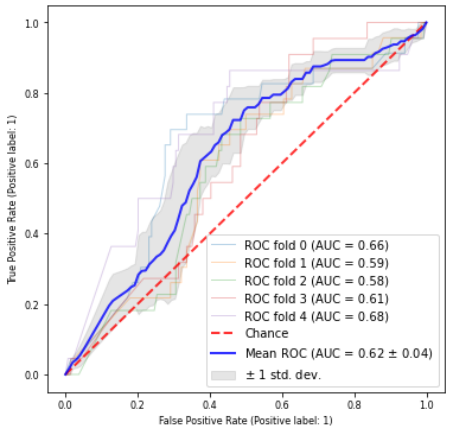

Fig. 4. ROC AUC curve SVM RBF kernel

First, the OCR of the framework that is used to extract information is not always correct. It has difficulty distinguishing a *t* and an *i* in particular. Furthermore, it can misidentify the color of a player due to the player using cosmetics (items to customize characters).

Secondly, only 59 games could be analyzed. It is expected that the performance of the classifiers will improve once more data becomes available.

Furthermore, the preprocessing steps impact the performance of the classifiers. Since the messages sent in Among Us are informal and sent under time constraints they often contain uncommon languages and typographical errors. This increases the challenge of preprocessing the data.

Finally, the training data is imbalanced. Most of the games that were used for the analysis only contained 1 imposter and only roughly 15% of the messages analyzed were sent by an imposter.

VII. Conclusions and Future Research

The experiments described in this paper showed that chatlogs from the game Among Us can be used to identify player roles. The results showed that all three different classifiers successfully learned something from the input data, this shows that identifying player roles is a learnable problem.

The contribution of this paper is twofold. First, the paper addressed a framework that allows extracting information from videos of Among Us gameplay. This framework is open source and can be used by other researchers. Secondly, an experiment was conducted to demonstrate that identifying player roles automatically based on brief and casual chat messages is a learnable challenge.

We invite other researchers to use the framework developed in this paper to generate more training data. It is expected that increasing the size of the dataset can significantly improve the performance of classification models. Additionally, the features extracted through NLP could be combined with other features generated by the framework. Next to that, the preprocessing steps could be improved and tailored to the context in which the messages are written. Finally, it could be interesting to see word embeddings work well with the informal and short messages.

Acknowledgments

This project was completed at Serpentine AI. The authors thank Serpentine AI for creating an amazing environment for AI enthusiasts. Furthermore, the authors thank Long Nguyen for contributing a module to the framework. Finally, this project would not have been possible without Wolf van de Hert and the TAB who took time to review the paper.

References

[1] G. Chittaranjan and H. Hung, "Are you a werewolf? detecting deceptive roles and outcomes in a conversational role-playing game," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 5334–5337, IEEE, 2010.

[2] T. Fukui, K. Ando, T. Murakami, N. Ito, and K. Iwata, "Automatic classification of remarks in werewolf bbs," in *2017 5th Intl Conf on Applied Computing and Information Technology/4th Intl Conf on Computational Science/Intelligence and Applied Informatics/2nd Intl Conf on Big Data, Cloud Computing, Data Science (ACIT-CSII-BCD)*, pp. 210–215, IEEE, 2017.

[3] Y. Hatori, S. Wu, Y. Lin, and T. Utsuro, "Mining preferences on identifying werewolf players from werewolf game logs," in *International Conference on Entertainment Computing*, pp. 353–356, Springer, 2017.

[4] M. Hagiwara, A. Moustafa, and T. Ito, "Using q-learning and estimation of role in werewolf game," in *Proceedings of the Annual Conference of JSAI 33rd Annual Conference, 2019*, pp. 2O5E303–2O5E303, The Japanese Society for Artificial Intelligence, 2019.

[5] Y. Lin, M. Kasamatsu, T. Chen, T. Fujita, H. Deng, and T. Utsuro, "Automatic annotation of werewolf game corpus with players revealing oneselves as seer/medium and divination/medium results," in *Workshop on Games and Natural Language Processing*, pp. 85–93, 2020.

[6] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[7] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang, "East: An efficient and accurate scene text detector," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[8] R. Smith, "An overview of the tesseract ocr engine," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2, pp. 629–633, 2007.

[9] S. Bird, "Nltk: The natural language toolkit," in *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[10] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, pp. 1–27, 2011.
[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.
[12] J. Huang and C. X. Ling, "Using auc and accuracy in evaluating learning algorithms," *IEEE Transactions on knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.