FAI Final Project

B11705043 資管二 古昭璿

1 Introduction

This report aims to document the process and results of implementing and evaluating various machine learning models for the Texas Hold'em Casino. We focused on using ensemble learning methods to improve performance of our agent to win the poker game. Therefore, we need to try different methods to collect the game feature and train the data until it can perfectly predict its actions to compete with its opponents.

Texas Hold'em is a poker which each of the players will get two private cards, called the "hole cards", and the table will have up to five face-up cards, called the "community cards". Then, the player's goal is to use the community cards combined with their hole cards to build a biggest five-card poker hand. The hands rank from the biggest to the smallest are Royal Flush, Straight Flush, Four of a Kind, Full House, Flush, Straight, Three of a Kind, Two Pair, One Pair, and the Hight Card, respectively. Also, Players can only choose action "call", "raise" or "fold", which means to match the amount of the big blind, increase the bet within the limits of the game, and throw the hand away. Therefore, in this report, what we should do is to develop a machine learning model to help our agent to make the best decision to get the most amount of the stack as they can.

2 Methods

In this Texas Hold'em Casino, we aim to win the game. Hence, it's important for us to make our agent have its own way to determine the actions and the amount of the stack in each round of the poker game.

2.1 If-else Determination

In this if-else determination model, we follow the Texas Hold'em rules to control the agent's next move. First, we evaluate the strength of the combination of hole cards and community cards. We let the rank values match their original numbers, which 2 is 2, J is 11, and A is 14, etc. We also let the suits have their own represent number. Then, we count the occurrences of all ranks and suits, and check for any card combination that may happen in order to count their score of hand strength.

The decision that the agent made for its action is based on the score of hand strength. Hence, we made an if-else model. In this model, if the hand strength is bigger than a specific amount, the agent will choose to raise; and choose to call or fold, otherwise. Also, to prevent from the call amount that our agent is not able to handle or it's too risky, the model will check the raise amount and the hand strength together when the raise amount is over some large number. Lastly, if we can’t fit any of the situations above, the agent chooses to fold the cards.

2.2 Random Forest Classifier

To better improve our decision trees in the If-else Model, we apply the Random Forest Classifier. As we know, Random Forest is an ensemble learning method. It is known for its robustness and ability to handle large datasets with higher dimensionality. It also helps in reducing overfitting by averaging multiple decision trees.

In our implementation, the Random Forest Classifier is trained on a dataset that includes features such as hand strength, the number of community cards, raise and call amounts, opponent behavior, and the agent's current stack size. The evaluation of hand strength is the same as if-else determination mode, and we also create a system to compute the weighting score of opponent behavior to better judge other players’ confidence in their hole cards. In order to calculate the weighting score we first need to check the action isn’t mine, then, the score will add 1 if the action is “raise”, add 0.5 if the action is “call”, and minus 1 if the action is “fold”. After each round, the model is updated with new training data, which includes the features of the hand and the action taken (fold, call, or raise). This continuous learning process allows the agent to improve its strategy over time, making more informed decisions based on the patterns observed in the game.

2.3 Gradient Boosting Classifier (GDBTs)

In the realm of machine learning, both Random Forests and Gradient Boosting Classifiers (GBDTs) are powerful ensemble methods known for their robustness and ability to handle complex datasets. However, there are several compelling reasons for transitioning from Random Forests to GBDTs in our poker-playing agent:

1. Higher Predictive Accuracy：GBDTs generally provide better predictive accuracy by sequentially correcting errors of previous trees.
2. Better Handling of Class Imbalance：GBDTs effectively manage class imbalances, crucial in poker where actions like fold, call, and raise occur at different frequencies.
3. Reduced Overfitting：GBDTs minimize overfitting more effectively through a gradient descent approach.
4. Feature Importance：GBDTs offer clearer insights into feature importance, aiding in refining the model.
5. Adaptive Learning：The sequential learning process of GBDTs adapts better to the complexities of poker.

Therefore, in our implementation, the method of training data remains unchanged, and the score calculation logic also doesn’t change, except that the Random Forest Classifier is replaced by the Gradient Boosting Classifier. At the same time, in order to avoid insufficient data leading to inability to train, we retain the random forest Classifier as a backup plan, so that the model can be trained in every situation.

2.4 Comparison of Methods

We measure the relationship between the agent and the baseline multiple times to calculate the winning rate of the model as a performance evaluation method.

2.4.1 If-else Determination

The model can have almost 100% winning rate on baseline 0 and baseline 1. But when it competes with baseline 2, the winning rate will drop to 50%, and other baselines don’t even have a chance of winning even though I have tried every possible parameter value.

2.4.2 Random Forest Classifier

The model can have almost 100% winning rate from baseline 0 to baseline 2. When it competes with baseline 3, the winning rate will drop to about 70%, and baseline 4 is about 50%. Other baselines barely have a chance of winning even though I have tried every possible parameter value.

2.4.3 Gradient Boosting Classifier (GDBTs)

The model can have almost 100% winning rate from baseline 0 to baseline 4. When it competes with baseline 5 and 6, the winning rate will drop to about 70%, and baseline 7 is about 50%. However, it is still barely able to win baseline 7.

3 Configurations

3.1 Hyperparameters

3.1.1 Random Forest Classifier

- ♦ Number of estimators = 80
- ♦ Max depth of decision trees = 5
- ♦ Random state = 0

3.1.2 Gradient Boosting Classifier

- ♦ Number of estimators = 80
- ♦ Max depth of decision trees = 5
- ♦ Random state = 0

3.2 Data Preparation

The training data was collected during the gameplay, storing features and labels for each hand. The data was then split into training and testing sets with a ratio of 80:20. Features included hand strength, community cards count, raise amount, call amount, opponent behavior, and player's remaining money. The calculation logic of each feature item is as described in the previous chapter and will not be explained again.

4 Discussion and Conclusion

4.1 Observations

Throughout the development and evaluation of our poker-playing agent, we observed several key points:

- ♦ If-else Determination: Simple but limited, struggling against stronger opponents.
- ♦ Random Forest Classifier: Improved performance against moderate opponents but struggled with complex scenarios.
- ♦ Gradient Boosting Classifier (GBDTs): Outperformed other models with higher predictive accuracy and adaptability, maintaining high winning rates across all baselines.

4.2 Challenges and Limitations

- ♦ Data Sufficiency: Need for large and diverse datasets.
- ♦ Computational Resources: Intensive training requirements.
- ♦ Opponent Behavior: Complexity in accurately capturing and reacting to varied strategies.

4.3 Selected Method

The Gradient Boosting Classifier (GBDTs) was the best method due to its superior accuracy, handling of class imbalances, reduced overfitting, and the highest winning rate of each baseline.

4.4 Future Work: XGBoosting

Exploring XGBoost for:

- ♦ Efficiency and Speed: Faster training without compromising accuracy.
- ♦ Enhanced Regularization: Better overfitting control.
- ♦ Scalability: Handling larger datasets and complex models more effectively.

Implementing XGBoost aims to further enhance the agent's decision-making and competitiveness in Texas Hold’em.