Destiny 2 Trials of Osiris Match Predictor

Philip Hicks
University of Tennessee Knoxville
Phicks6@vols.utk.edu

Introduction

Destiny 2 is an online MMO/First Person Shooter that has a mix of player vs environment and player vs player. The highest level of player vs player content is Trials of Osiris, where two teams of three guardians face off. Each guardian can be one of three main classes: hunter, warlock, and titan. The goal of this project was to create a match predictor, that when given six players and their team affiliation will predict who will win the match.

Motivation

Esports has been growing fast and becoming a large industry with big tournaments with large cash prizes (1). An accurate predictor could have huge implications on betting and gambling on matches. While Trials of Osiris isn't really an official Esport and doesn't have official cash prices, it is a game that is close to heart and one that I really enjoy playing. The developer, Bungie, has made an incredibly detailed API that can be used to control things in game to gathering detailed stats. A lot of work on match predictors has been done on games like Dota 2 with many different types of heroes/classes (2). These tend to only focus on the presence of heroes or classes and not the skills of the players playing those classes. Other games like Counter-Strike: Global Offensive don't have unique classes and only focus on the skills of players (3). The finished model might be able to inform certain recommendations like what class given your skill level and help players improve their chances of winning games.

Approach

Data Collection

The first step was to gather the data necessary for the models. Unfortunately, Bungie's API doesn't have a call for returning large amounts of Trials of Osiris match results. I created a script that looked at my past Trials of Osiris and gathered the results of all those matches. I then went through those gathered matches and began a list of other players I encountered. I would then gather the matches of the first player and add them to the running player list. I continued this process until I had enough matches.

Then for each player, I scraped their Elo from destinytracker.com. Elo is a stat that combines win rates and player performance into a number that roughly describes a player's skill. While this number isn't as indicative of skill as something like TrueSkill, it has been computed for me without having to have the entire Trials match dataset. This is unfeasible as millions of trials games are played every weekend.

Data Preprocessing

To turn the match data into something that a model can ingest, I took each player's class and Elo and laid it out into a row with the match result on the end. The first three players represented the first team and the next three players were the second team. The winning team would be randomly placed as either the first team or the second team. Elos were standardized. A players' class was encoded as the following: Hunter=1,0,0, Warlock=0,1,0 Titan=0,0,1. An example match would be [1,0,0,Elo,1,0,0,Elo, 1,0,0,Elo, 0,1,0,Elo, 0,1,0,Elo, 0,0,1,Elo,1] (3 Hunters winning vs 2 Warlock and a Titan). Another dataset was also made that just contained Elos to see if class info was actually helping the models predict the match results.

Next to help the models learn to predict the winning team regardless of player placement on a team. Every match was turned into 36 matches with player placement being enumerated. With 7650 matches after throwing out bad games due to missing player data, the end result after this operation was 275386 matches.

Model Selection

I choose three different model types: a K-nearest Neighbor Classifier, a two hidden layer neural network, and a Random Forest Classifier. To find the optimal parameters, each model was put through some grid search. The Knn classifier

was tested with manhattan distance with k=[1,2,5,10,15,20,25,30,35,40,45,50,100,150,200,50,500]. The neural network searched over activation functions linear, sigmoid, relu, and tanh and found that the first hidden layer being linear and the second layer being sigmoidal had the best results. It was then further tested the first layer having [6,10,20,30,35,40,45,50,60] neurons and the second having [20,30,35,40,45,50,60] neurons. A small amount of testing was done with 3-layer neural networks. The random forest classifier was tested with no limit to the features, log2 max features, and sqrt max features. With several different numbers of estimators equaling [1,5,10,20,25,50,75,100, 150,200,250,500,750,1000,1250,1500,2000].

Evaluation

Evaluation was done over 5 folds in the data with a 20-80 train-test split.

K-Nearest Neighbors Classifier

KNN ending up performing better without the player's class information and just their elo. The best number of neighbors was found to be 10 and had the following results over 5 folds:

Average Accuracy	Average Precision	Average Recall
0.8086816323269	0.8077690875550	0.81183904801

Neural Network

The 2-layer neural networks performed better with the player's class information included in the data. The best performer used a first hidden layer of 30 neurons using a linear activation and 30 neurons using a sigmoidal activation in the second layer. It had the following results:

Average Accuracy	Average Precision	Average Recall
0.798178556644	0.8013374163023	0.79466191655

I also fed this network a premade match with 1 team having slightly higher Elo than the other. I used the 36 different enumerations of this match to view how resistant the model is to player position. The network's lowest prediction for the better team was 56% and the highest was 90%. The wide range of this was rather disappointing because it meant that player position is having a significate effect on the predicted outcome. Another batch of fake matches was also fed to this neural network to see how the teams' class composition affected their chances to win. An interesting note from this was that an average team of hunters had a 71% predicted chance to win vs a team of 3 average warlocks but only a 14% chance to win against a team of 3 average titans. This high disparity of win rates of a given class is not something that is seen in the real world and leads me to believe the network is putting too much emphasis on the class of the player.

3-layer Neural Network

Not much testing of parameters was done with 3 layers but one with a first hidden layer of 6 neurons using a linear activation, 30 neurons using a rectified linear activation in the second layer, and 30 neurons using a rectified linear activation in the third layer. Managed to beat the best 2-layer network with the following scores:

Average Accuracy	Average Precision	Average Recall
0.8292440429070	0.8245794577184	0.83864411196

It had a similar but not quite as extreme range of prediction for the fabricated matches.

Random Forest

The random forest classifiers performed better with the player's class information included in the data. The best performer had a log base 2 limit on features and used 750 estimators. It had the following scores:

Average Accuracy	Average Precision	Average Recall
0.8405227571481	0.8426460804190	0.83873097218

Not only did the random forest classifier have the best scores but it also performed the most consistently over the 36 different enumerations of the same match with the predicted outcome varying by less than 5%. It also had a much more mild prediction of who would win based on a team's class composition.

Summary

The random forest classifier performed the best of the 3 types of models I tested with. However, not much testing was done with 3-layer networks and a proper search of the parameter space would probably result in a network with similar accuracies to the random forest classifier. However, the random forest classifier has the neural networks beat when it comes to variation of the same match and players' class's impact. Knn, while having the lowest performance,

showed that even a simple, yet computationally heavy, approach can get decent results.

Future Work

The most obvious work to be done is more testing with different depths of neural networks. I didn't expect increasing from 2 to 3 layers would have such an impact on performance and that led me to only test it out towards the end where there wasn't enough time to properly search the parameter space. Neural networks could definitely match or beat the accuracy of random forest classifiers if the right parameters can be found.

Another idea for future work would be to create some sort of hybrid machine. The neural network's tendency to overestimate the impact of a players' class could be used to help further separate results if a more accurate classifier like random forest classifiers are uncertain about the outcomes of a match.

Lastly, I would like to create a more friendly UI to the model after it has been trained to more easily test fabricated matches and to release it to the Destiny 2 community to play around with.

Contributions

All the work was done by Philip Hicks.

References

1. Taylor, T.: Raising the Stakes: E-sports and the Professionalization of Computer Gaming. MIT Press, Cambridge (2012)
2. Pobiedina, N., Neidhardt, J.: On successful team formation. Technical report (2013)
3. Predicting Winning Team and Probabilistic Ratings in "Dota 2" and "Counter-Strike: Global Offensive" Video Games