Machine Learning for Pinnacle Matchmaking in *Destiny 2*

Garrick Fernandez
Department of Computer Science
Stanford University
Stanford, CA
garrick@cs.stanford.edu

***Abstract*—Video games often contain online components that allow players across the world to interact and play with one another. The systems which perform the matching of players into teams are often collectively referred to as "matchmaking." We focus on analyzing matchmaking in the video game *Destiny 2*, an online first-person shooter developed by Bungie. In this paper, we develop and apply machine learning techniques to Destiny 2 game data exposed by the Bungie.net Platform API, with the goal of predicting the "success" of the matching of a group of player, as measured by end of game standing and time to activity completion. The main contributions of this paper are (1) a structured data pipeline for retrieving, processing and aggregating data scraped from the Platform API, (2) data analysis and application of machine learning for supervised and unsupervised tasks, and (3) explorations of and proposals for future work in feature engineering.**

***Index Terms*—Category: Finance and Commerce, video games, matchmaking, regression, data collection, API**

I. Introduction and Problem Motivation

In the realm of online gaming, *Destiny 2* is unique in that it facilitates player interaction in many ways. Players can organically encounter each other online while exploring the game's many worlds, as well as prompt to get matched with other players in order to participate in cooperative and competitive activities. This latter process, a more explicit form of matchmaking, depends on the fact that there are other players currently online who are looking to play the same activity. However, there are some activities that have matchmaking disabled, due to the difficulty, coordination, and prerequisites required for the activity (these activities are referred to as "pinnacle" or "aspirational" activities—the things you do to demonstrate mastery over the game).

This design choice has created an ecosystem around finding people to play with, and there are already several established networks that exist for finding teammates, such as LFG ("looking for group") websites, and clans, organized groups of players that regularly play together. Bungie themselves have an in-game matchmaking solution, called Guided Games, which attempts to pair solo players (Seekers) and organized teams (Guides). Unfortunately, Guided Games has been in beta since the game's original release in 2017 [1], and it suffers from low concurrent player populations and, by extension, long queue times. The advantage of clan membership (and, to some extent, using LFG apps) are that players can schedule activities in advance, but these routes can be intimidating to new or introverted players. One potential approach that could improve the experience of looking for a team would be to use a system to recommend other players they could play an activity with right now, or in the near future: a "team-activity-time" recommendation. Instead of putting the onus on a player to find a group, the system can take advantage of the fact that many people are in similar situations, wanting to play the same activity—and proactively match them.

As a step towards that goal, we employ a number of machine learning techniques on data publicly accessible via the Bungie.net Platform API [2], with the goal of evaluating the success or fitness of a matched group of players. This system could be employed in a matchmaking pipeline to make recommendations on which players could play an activity together.

More formally, we define our task as a supervised learning task, where the examples are historical games whose information we can query from the API, and the labels are measures of "success" reported by the game—for example, the time it takes to complete an activity, or the team's standing, victory or defeat, at the end of a game.

The paper is structured as follows: In section II, we outline a process for scraping data from the *Destiny 2* API. In section III, we outline the methods we tried for learning on our task. In section IV, we evaluate the results of our exploration In section V, we situate our work and results in the existing literature In section VI, we summarize our results and discuss avenues for future work.

II. Data Collection

Collecting data was a significant component of the project due to the nature of its source. Traditional datasets with well-formed features are not publicly available for *Destiny 2*; however, some data is exposed via the Bungie.net Platform API, opening the door for scraping together a dataset. The API contains upwards of 100 endpoints that allow both first-party and third-party apps to access and manipulate various in-game and player data.

This project was done for CS229: Machine Learning. The code is available on GitHub at garrickf/d2-ml. Thanks to Prof. Moses Charikar, Prof. Chris Ré, and the rest of the teaching team for their feedback on the project and a great class!

Of these many endpoints, only a subset provide any useful information for our learning task. For example, the `Destiny2.GetPostGameCarnageReport` endpoint returns a summary of information related with a given `activityId`, internally referred to as a post-game carnage report (PGCR). The `activityId` is an unsigned 64-bit integer, and it is assigned to activities in ascending order. See Fig. 1 for an example PGCR response.


```
{
    "Response": {
        "period": "2021-05-07T10:06:12Z",
        "startingPhaseIndex": 0,
        "activityDetails": {
            "referenceId": 1575864965,
            "directorActivityHash": ...,
            "instanceId": "8400554258",
            "mode": 63,
            "modes": [
                64,
                63
            ],
            "isPrivate": false,
            "membershipType": 3
        },
        ...
    }
}
```


Fig. 1. Truncated JSON PGCR response (full length is 2844 lines) for `activityId = 8400554258`. This is a game of Gambit (a competitive player vs. player mode, with additional non-player combatants (the "environment"), often called PvPvE. There are additional fields for player statistics and game metadata.

Assuming we rate-limit our requests to 25 per-second (the limit imposed by the API), querying activity information for every `activityId` from September 2017 to the present would take upwards of 10 years! To make things feasible, we limit our range to a period of 100K/1M activities performed starting at an arbitrary `activityId` (we picked one for a game played around May). We implemented a threadpool to parallelize outgoing requests, rate-limiting to avoid throttling; this sped up data collection by up to fourteen fold.

An additional layer of complication in the data collection is the presence of hashes in the response rather than localized English strings (see See Fig. 1, `directorActivityHash`). The API is internationalized, and so the API responses are decoupled from any language, and an additional hydration step is needed to transform hashes into localized English strings. In particular, a manifest (a file containing metadata for a group of accompanying files—in our case, all the data we can query from the platform API) can be downloaded as a compressed SQLite database. The tables therein map hashes to localized strings, representing static definitions of objects found within *Destiny*.

We wrote an additional script script to download the manifest and process it into an index used in the data collection routine.

Once collected and processed, our data had

III. Methods

There are multiple factors of "success" in a single *Destiny 2* game or match. If the game mode does not contain opposing teams (player vs. environment, or PvE), the activity completion time can be taken as a metric of team cohesion and success. The longer the time to completion, the worse the team did. In competitive game modes (PvP or PvPvE), there are separate competing teams, and the standing, or end-of-game result (victory or defeat) can be taken as a metric of team success.

In order to learn and predict completion time, we consider several methods. One method is linear regression, whose cost function is given by:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2 \tag{1}$$

where $h_\theta(x) = \sum_{i=0}^{d}\theta_i x_i = \theta^\top x$ (we use the convention of letting $x_0 = 1$, the intercept). This function is convex. We can derive this cost function from a probabilistic/MLE perspective. The vectorized update rule for batch gradient descent is

$$\theta := \theta + \alpha\sum_{i=1}^{n}(y^{(i)} - h_\theta(x^{(i)}))x^{(i)} \tag{2}$$

Notice we don't normalize over the number of examples (as is convention with neural networks). Also note that the negative sign typical in gradient descent (we move against the direction of the gradient to minimize the cost function) has been pushed inside the sum in (2).

As there are a lot of features, and some of them may be collinear (for example, the number of kills versus the number of combatants defeated, i.e., kills and assists), regularization may result in better, simpler models and less overfitting to the training data. We consider ridge regression and lasso regression, whose cost functions are least squares with L2 and L1 regularization, respectively. The cost function for Ridge regression is:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\|\theta\|_2 \tag{3}$$

Where λ is the regularization strength. The cost function for Lasso regression is:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2 + \gamma\|\theta\|_1 \tag{4}$$

Where γ is the regularization strength. By taking a Bayesian interpretation of regularization, we can view ridge and lasso regression as having a Gaussian and Laplace prior over the parameters θ, respectively. Both priors encourage the parameter values to be closer to their mean (i.e., zero), resulting

in a shrinkage effect. In particular, lasso regression is known to result in sparse parameters, where most parameter values are zero, and only some are non-zero. This could be useful for identifying which features are the most useful for our learning task.

A. Tools

We used an external library, scikit-learn [3], to perform the methods described above.

IV. Experiments, Results and Interpretation

V. Related Work

VI. Conclusion and Future Work

References

[1] "Guided games: bungie help," https://help.bungie.net/hc/en-us/articles/360049198951-Guided-Games (Accessed June 2021).

[2] "Bungie.net api," https://bungie-net.github.io/multi/index.html (Accessed June 2021).

[3] "Scikit-learn: machine learning in python," https://scikit-learn.org/stable/ (accessed May 2021).