PUBG Finish Placement Prediction

张志远, 郭家兴, 周彦, 刘唐, 崔航, 司文

January 15, 2021

Content

1 Introduction

1.1 Data Description and Visualization

We choose the PUBG Finish Placement Prediction which is a match on kaggle in 2018 as our project. Our goal is to predict the placement of each group based on their performance in a game. The original trainset contains 29 variables, in contrast to 28 variables in testset with one variable called *winPlacePerc* for us to predict. See table 1 in **Appendix** for summary statistics.

To grasp a preliminary understanding of the data, we construct a series of graphics to visually show the dataset. First we show the different match types and their proportion, see figure 1 in **Appendix**. Then we plot some important variables like *kills*, *assists*, *boosts*, *heals*, *weaponsAcquired* and *revives*, to roughly check their distributions. See figure 2 in **Appendix**.

1.2 Existing Methods

Since the match PUBG Finish Placement Prediction has finished two years ago, a variety of different methods have been tried on this dataset, including both traditional machine learning methods and deep learning meth-

ods. We refered to some state of the art methods, and find that they generally contain three parts:

1. Visually show the data, observe correlation and distribution.
2. Feature engineering, extract information as much as possible.
3. Conduct prediction based on some advanced ML methods.

Specifically, most popular features includes mean, maximum and minimum of original variables. Since players in one group get the same $winPlacePerc$, group level variables like $groupKillsTotal$, which is the group's total kills; and $groupKillsMax$, which is the maximum kills by the group's player, are also widely generated. More advanced features includes $headshotrate$, which is the rate of head-shot kills calculated off total kills. Also most popular models they used includes random forests, XGBoost, neural networks etc.

In fact, we find that the selected features and the ML methods they have used work pretty well and there's not much space to improve. But by carefully scanning each variable in the dataset, our team discover that the variable $killPlace$ have a very strong relationship with target variable $winPlacePerc$, which we will cover in the following chapter and we can utilize to genrate a DAG for each game. This DAG will give a significant enhancement to our prediction.

What's more, there's another easily-overseed factor: the missing groups' placement, which we use EM method to find the approximate distribution to help us make better overall prediction.

1.3 Deeper into the data

1.3.1 Interpretation of our mission

To better explain the data and our target, consider a match of N groups, and the dataset collects information of n groups, ranking from 1^{th} up to M^{th}.($n \leq N$ always hold and $n < N$ means that there're $N-n$ groups missing out from this game) And the target variable $winPlacePerc$ is calculated as:

$$winPlacePerc_i = \frac{R_i}{M},$$

where $winPlacePerc_i$ is the $winPlacePerc$ contained in our trainset but not in testset(This is obvious. Because it is the target variable of our prediction!) and R_i is rank of group i not contained in our dataset. M is the variable "$maxPlace$" contained both in trainset and testset. So this can be interpreted as a sort prediction problem, i.e. to predict each group's rank R_i in a game and thus we can calculate $winPlacePerc$.

1.3.2 KillPlace's sorting property

During our analysis of the data, we find that the variable $killPlace$ contains some important information: when two players has the same $kills$, the order of their $killPlace$ is the same as their $winPlacePerc$. For example, if player 1 and player 2 have the same number of kills in a game and $killPlace_1 < killPlace_2$, then we can know that $winPlacePerc_1 < winPlacePerc_2$.

To take advantage of this oracle information, we proceed our prediction in three steps:

For each game, first we can obtain several subsequences based on players' variable $kills$ and $killPlace$(We first choose all players with $kills = 0$ and sort them according to their $killPlace$. The order obtained here should be completely corresponding to the real order of them. Then, we choose all players with $kills = 1$ and apply the totally same operation to it$\cdots\cdots$). And the number of ordered sequences is the same as the unique number of $kills$ in that game.

Then we train prediction models on the target variable $winPlacePerc$ and combine the prediction results with DAG composed of previous suborders. Finally we fit the distribution of the placement of missing groups and

insert from sampling.

2 Prediction

2.1 Feature Engineering and Model Selection

Feature engineering is always a significant step in machine learning missions and if you choose some trivial features that makes little difference to the response variable, you will never make a good prediction no matter how your model advanced.

In this project, our team first choose 15 features including maximum kill, maximum treatment, maximum walking distance and maximum damage in the group etc because by data analysis in the very first step, we found strong correlation between these variables and the target variable $winPlacePerc$. And we choose three ML methods XGBoost, LightGBM and Multi-Layer Perceptron to make prediction. The goodnesses of our predictions measured by mae are shown as follows:

model	consumption time(s)	validation mae
XGBoost	102	0.0792
LightGBM	5.2	0.0716
MLP	62.31	0.138

We find that LightGBM consumes the least time and reaches the lowest validation mae, i.e. LightGBM is the best model under these features. However, we have to say that all three models' validation mae are still a little bit high and it's necessary for us to do the feature engineering the second time.

This time we choose to enlarge the feature set vastly. We choose 247 features in total by combining current features to obtain second order features and some statistical features among groups, such as max, min, mean, etc. We still use the former three models to make prediction and get the following result:

model	consumption time(s)	validation mae
XGBoost	802.8	0.0299
LightGBM	326	0.0273
MLP	678.83	0.0376

We could see that this time the LightGBM method still has the best performance and most importantly, the validation mae is much lower than

it when there are only 15 features. Hence, we choose the LightGBM as our ultimate model and these 247 features as our ultimate features.

2.2 Combination of Topological Sort and Prediction

To obtain a complete groups order of a match, we treat each group as nodes and each priority relation in subsequences obtained from $killPlace$ as edges and construct DAG. It strikes us that we can use toppological sort to generate an order of the groups. However, topological sort is often not unique which may give us a lot of possible order of the same groups. But luckily, we can utilize our LightGBM prediction results as tie breaker to solve all these uncertainties. The algorithm are shown as following pseudo code:

To enable fast sorting, we use a matrix to store a match's information, each column storing a suborder list, and a model-predicted complete order to realize our algorithm.

This sorting algorithm take advantage of the subsequence orders information and combine it with model prediction while avoiding any contradiction. Also it works extremely fast using distributed computations, i.e., R's *apply* function family, sorting nearly 1 million groups in about 2 mimutes.

Algorithm 1 Combination of topological sort and prediction

Input: this game's DAG;

Output: The sorted group order list L;

Initialize a group order list L to store sorted groups;
while There're nodes in the DAG **do**
    if There're more than one node with no prior nodes **then**
        Use our prediction result to give an order of these nodes;
        Attach an edge from the prior node to the latter node, after which there's only one node in this DAG;
    end if
    Rut the only node without prior nodes into the ordered list L and delete the node and all the edges reached this node from the DAG;
end while
return L;

2.3 Distribution Of Missing Groups' Ranks

After we obtain the fully sorted group order list, we are still one-step short from getting our target variable $winPlacePrec$. Recall previously that

$$winPlacePerc_i = \frac{R_i}{M},$$

where $winPlacePerc_i$ and R_i is the $winPlacePerc$ and rank of group i. And our available information is not complete, with the presence of missing groups. See figure 3 in **Appendix** for clear view. Since we do not have any information about those missing ranks, it is impossible to construct any predictive models. So instead we focus on their empirical distribution and try to fit it. Since different match types will have differnet number of groups, so we fit several distributions. Here we take the match type Duo as example to illustrate our work.

From the figure 4 in **Appendix** we notice that the empirical distribution of the missing ranks have two peaks and scales very much similar to a mixture Gussian distribution. So we try fit the two-dimensional mixture Gaussian distribution model, which refers to the probability distribution model with the following form:

$$P(Y|\theta) = \sum_{k=1}^{2} \alpha_k \phi(y|\theta_k),$$

where $\alpha_k \geq 0, k = 1, 2$; and $\sum_{k=1}^{2} \alpha_k = 1$; and $\phi(y|\theta_k)$ is a Gaussian density function, with $\theta_k = (\mu_k, \sigma_k^2)$. Here the $\phi(y|\theta_k) = \frac{1}{\sqrt{2\pi}\sigma_k} exp(-\frac{(y-\mu_k)^2}{2\sigma_k^2})$ is denoted as the k^{th} submodel.

It is hard to estimate the parameters of this model directly, so we apply the EM algorithm instead to do our estimation iteratively. In particular, after setting the initial values $\theta_k^{(0)}, k = 1, 2$, we iterate the following steps. Specifically, in the m^{th} iteration, we conduct the E-STEP and M-STEP as follows.

1. E-STEP

In the E-STEP, we estimate $\alpha_k^{(m)}$ by its conditional expectation given $\theta_k^{(m-1)}$. Here,

$$\alpha_k^{(m)} = E(\alpha_k|\theta_k^{(m-1)}) = \frac{\sum_{j=1}^{N} \gamma_{jk}^{(m-1)}}{N},$$

where $\gamma_{jk}^{(m-1)} = \frac{\alpha_k^{(m-1)}\phi(y_j|\theta_k^{(m-1)})}{\sum_{k=1}^{K} \alpha_k^{(m-1)}\phi(y_j|\theta_k^{(m-1)})}$ is the response of the k^{th} submodel to the observed data y_j.

2. M-STEP

In the M-STEP, we estimate $\theta_k^{(m)}$ by its Maximum likelihood estimator.

Here,

$$\hat{\mu}_k^{(m)} = \frac{\sum_{j=1}^{N} \hat{\gamma}_{jk}^{(m-1)} y_j}{\sum_{j=1}^{N} \hat{\gamma}_{jk}^{(m-1)}}, \quad \hat{\sigma}_k^{2(m)} = \frac{\sum_{j=1}^{N} \hat{\gamma}_{jk}^{(m-1)} (y_j - \mu_k^{(m)})^2}{\sum_{j=1}^{N} \hat{\gamma}_{jk}^{(m-1)}}.$$

Repeat the above two step until convergence and we get our final estimation for parameters.

There are 16 match types in total, based on match types and their group numbers, we fit 7 different two-dimensional mixture Gaussian distribution models. See figure 4 in **Appendix** for the empirical and model-fitted distribution for missing ranks of match type *Duo*. The detailed parameters for corresponding match types are listed in table 2 in **Appendix**.

Notice here for match type $flarefpp$ and $crashtpp$ we do not fit two-dimensional mixture Gaussian distribution because their plots look like uniform distribution. Also the number of matches of these two match type is very small, so we try the uniform distribution for these two match types.

After we get the distribution of missing ranks for each match type, we conduct random sample from corresponding distribution and instert sampled ranks into our sorted group order list. This way we can compute the target $winPlacePerc$.

3 Results

We proceed our project as dexcribed above. Since it comes from a kaggle match, we are able to evaluate our result and compare with other competitors. As a supervised learning match with predicting target $winPlacePerc$, the evaluation criterion is mean absolute error which takes the form:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|.$$

The MAE result for our prediction is **0.01528**, ranking **third** among all **1528** participants. Click here to see our results on Kaggle as shown in figure 5 in **Appendix**.

References

[1] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: a highly efficient gradient boosting decision tree. In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 3149-3157.

[2] Chen, T. , & Guestrin, C. . (2016). XGBoost: A Scalable Tree Boosting System. $the22ndACMSIGKDDInternationalConference.$ ACM.

[3] Conan-Guez, R. B. . (2005). Functional multi-layer perceptron: a non-linear tool for functional data analysis. $NeuralNetworks$.

4 Appendix

Table 1: Summary Statistics

variable	explanation	count	min	max	std
Id	Player Id	4446966	none	none	none
MatchId	ID to identify match	4446966	none	none	none
groupId	ID to identify a group	4446966	none	none	none
assists	Number of enemy this player assist in killing	4446966	0	22	0.59
boosts	Number of boost items used	4446966	0	33	1.72
damagedealt	Total damage dealt	4446966	0	3407	169
DBNOs	Number of enemy knocked	4446966	0	33	1.144
headshotkills	Number of enemy killed with headshots	4446966	0	19	5932
heals	Number of healing items used	4446966	0	39	2.66
Killplace	Ranking in match of number of enemy	4446966	1	100	27
kills	Number of enemy killed	4446966	0	30	1.54

killstreaks	Max number of enemy killed in short time	4446966	0	14	0.710
longestkill	Longest distance between players killed and killer	4446966	0	999.9	50.8
matchduration	Duration of match in seconds	4446966	258	258	2237
matchtype	String identifying the game mode that the data comes from	4446966	none	none	none
maxplace	Worst placement we have data for in the match	4446966	2	100	23.82
numgroups	Number of groups we have data for in the match	4446966	1	100	23.28
revives	Number of times this player revived teammates	4446966	0	19	0.47
ridedistance	Total distance traveled in vehicles measured in meters	4446966	0	19890	1493
roadkills	Number of kills in a vehicle	4446966	0	9	0.0738
swimdistance	Total distance traveled by swimming measured in meters	4446966	0	1974	30.215

teamkills	Number of times this player killed a teammate	4446966	0	10	0.168
vehicledestroys	Number of vehicles destroyed	4446966	0	4	0.092176
walkdistance	Total distance traveled on foot measured in meters	4446966	0	9980	1182
weaponsacquired	Number of weapons picked up	4446966	0	72	2.425
winplaceperc	The target of prediction	4446966	0	1	0.307
killpoints	Kills-based external ranking	1791319	134	2170	197.27
winpoints	Win-based external ranking	1791319	321	2013	68.66
rankpoints	Elo-like ranking of player	2745156	0	5910	279.64

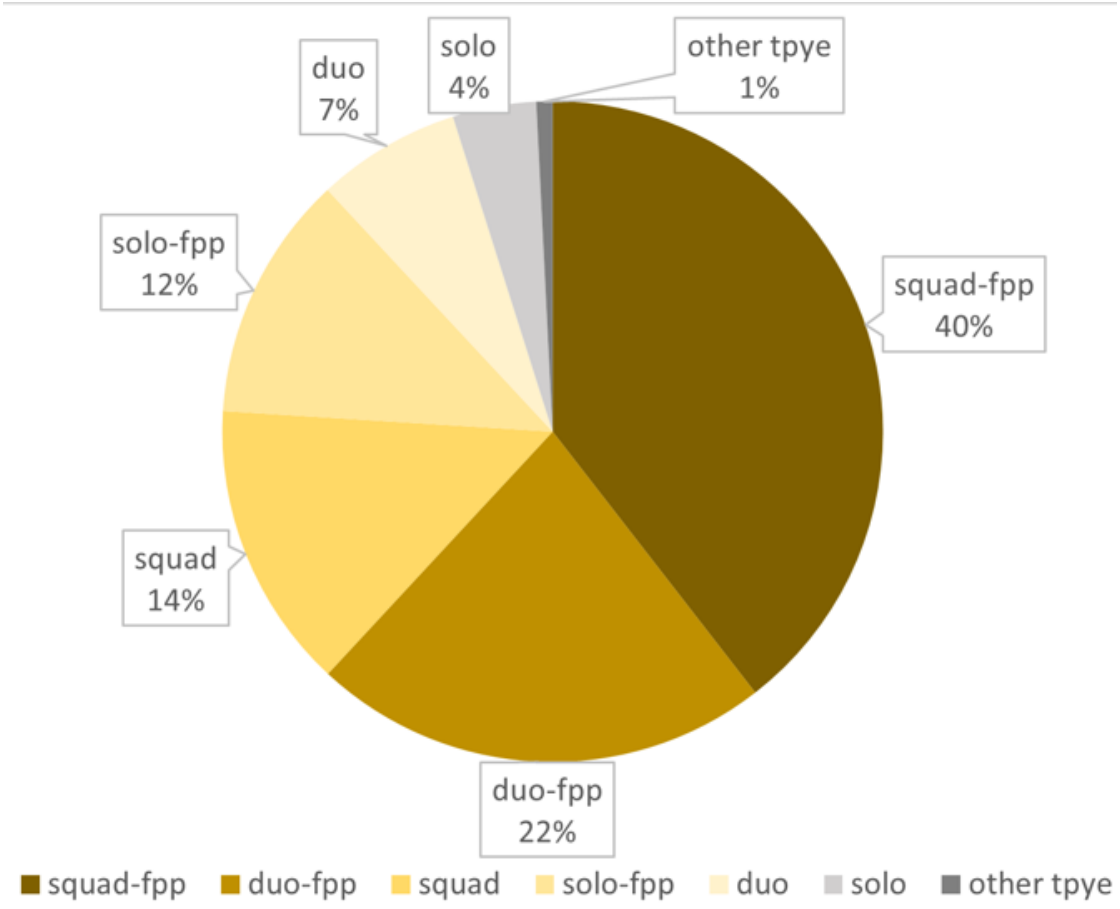

Figure 1: Match Type Proportion

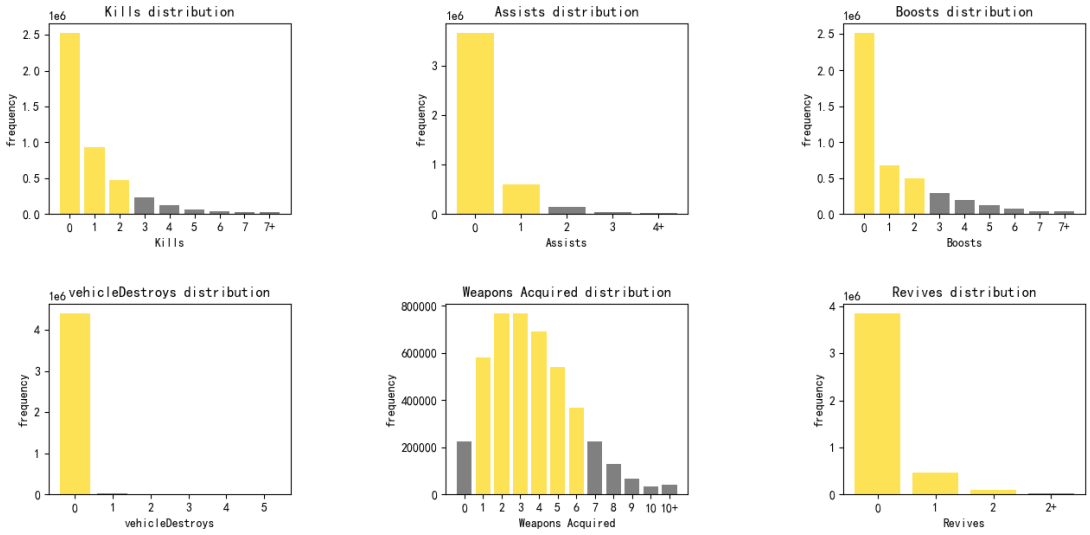

Figure 2: Distribution of 6 Major Variables

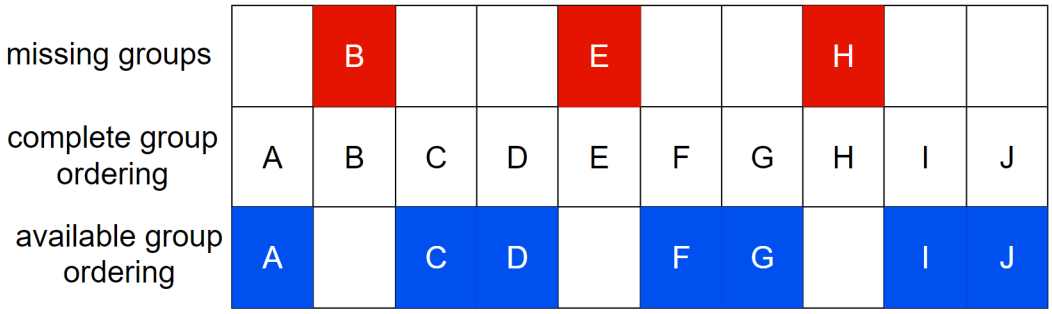

Figure 3: Illustrating Group Ordering

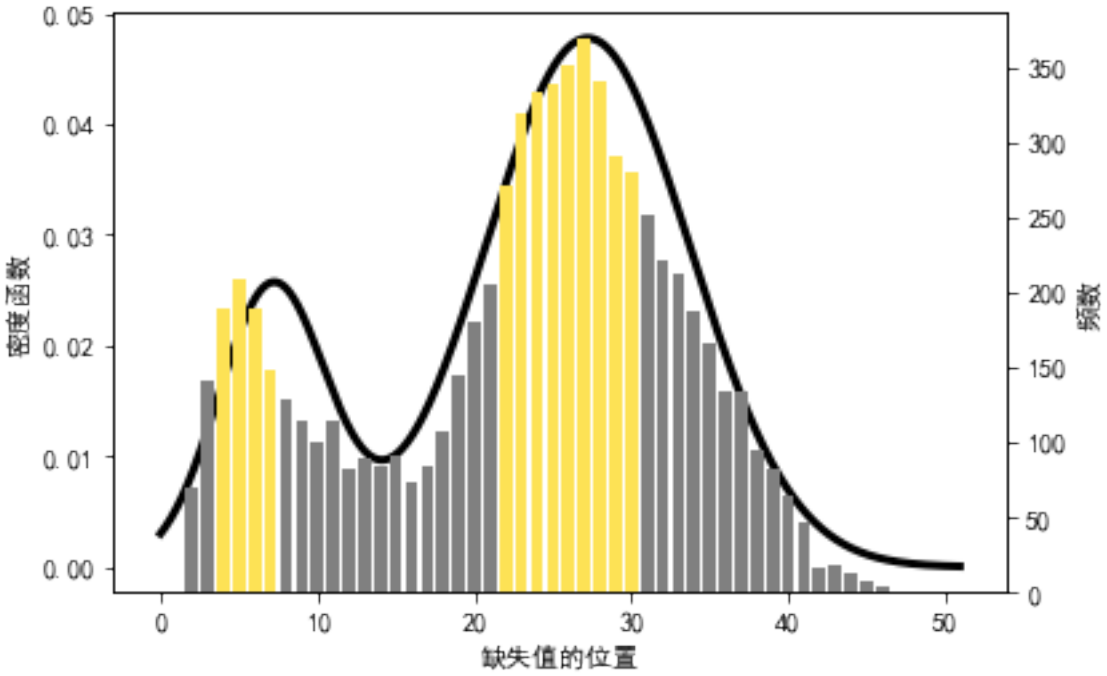

Figure 4: Missing Ranks and Fitted Distribution for Match Type*Duo*

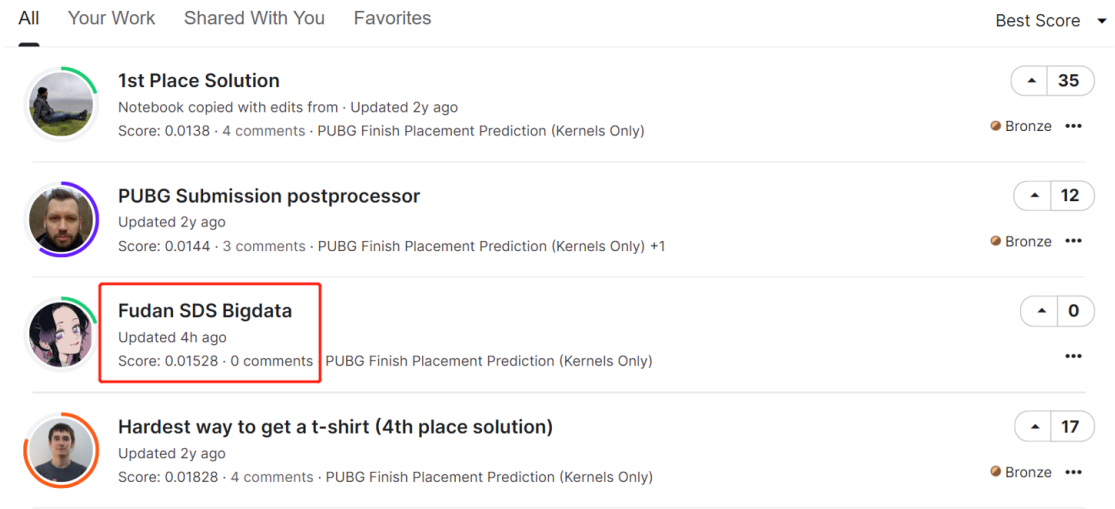

Figure 5: Our Results on Kaggle

Table 2: Two-Dimensional Mixture Gaussian Parameters

match types	(α_1, α_2)	(μ_1, μ_2)	(σ_1, σ_2)
(squad-fpp, normal-squad-fpp)	(0.03, 0.97)	(2.63, 14.75)	(0.69, 4.80)
(solo-fpp, normal-solo-fpp)	(0.22, 0.78)	(21.39, 63.69)	(8.49, 12.41)
(normal-duo-fpp, duo-fpp)	(0.1, 0.9)	(6.66, 28.77)	(2.86, 6.78)
(squad, normal-squad, flaretpp)	(0.07, 0.93)	(3.15, 13.08)	(1.21, 4.91)
(solo, normal-solo)	(0.20, 0.80)	(9.02, 32.19)	(1.21, 4.91)
(normal-duo, duo)	(0.22, 0.78)	(7.15, 27.22)	(3.47, 6.52)
crashfpp	(0.12, 0.88)	(5.46, 21.77)	(2.23, 5.51)