Survival in PUBG Demystified

Team 2 - Apoorv Pandey, Saptarshi Chatterjee and Prinila Ponnayya

{apoorv.pandey, saptarshi.chatterjee, prinilairene.ponnayya}@colostate.edu

Problem Formulation

PUBG (Player Unknown's BattleGround) is a battle royale style shooter game in which upto a hundred players or teams engage in combat until the last remaining player, who is the victor. If players partake as a team instead, the team to which the last remaining player belongs is considered the winner.

Since the objective of the game is to be the last player standing, with our project, we've attempted to provide a prediction of the player's survival time with respect to factors such as game size, team size, etc. We've also performed various Big Data analyses which provide keen insights into the game, for example the likely rank of a player based on their survival time. Another example is listing the most popular weapons in the game, an insight which can then be leveraged to develop apropos counter-strategies.

Another use case we envision suitable for application of our work is determining the best resources to obtain within a budget. With PUBG being a freemium game, knowing the best resources to invest in gives players an upper hand. Doing so allows players to optimize their in-app spending and get the greatest return on their purchases. For instance, pre-pubescents with limited discretionary income will be able to make better use of their money and hopefully develop financial discipline while being catapulted into the spotlight of their social group. We would like to clarify that our work does **not** generate strategies from scratch, it instead helps in choosing between different strategies. In other words, for a budget X, our project will not come up with strategy A or B. However, when the player comes up with strategies A and B, the insights provided by our work and our survival time prediction will help them choose either A or B.

The outcome of our project will serve as a guide for players regarding strategies they might select in order to survive the longest they can, consequently achieving their best possible rank.

Methodology

From a Big Data perspective, an enormous dataset would take forever to be analyzed on a single commodity machine. Hence, performing machine learning on large datasets necessitates a distributed set-up to achieve manageable load balancing across machines and reasonable model training times. In addition to machine learning, we also performed various analyses on the dataset, revealing characteristics about survival metrics which could be leveraged as heuristics for good game strategy. Achieving that for a 19 GB dataset required a lot of processing and aggregation, beyond the capabilities of any single machine we had access to.

The machine learning component was implemented using distributed PyTorch, while the Big Data analytics were performed using PySpark submitted using Apache Spark. The dataset was stored in an HDFS cluster with over 30 worker nodes and a min replication of 3 for the data files. All machines were from the CSB 120 lab of CSU. For the Big Data analyses, we set up our Spark cluster on top of our HDFS cluster, making every worker in the HDFS cluster a slave in our Spark cluster. Hence, all machines in the Spark cluster were also from the CSB 120 lab. We configured each slave in the Spark cluster to spawn 2 executors having 2 GB memory per executor.

Dataset

We found a dataset on Kaggle [1] that contained aggregated data of about 720,000 matches in PUBG and records of over 65 million deaths in said matches. The dataset consists of two parts, data from the matches and details of various deaths split into multiple CSV files of two types, aggregate and deaths. The total uncompressed size of the data is approximately 19 GB. In the "deaths" subset of the data, the files recorded every player death that occurred in the 720,000 matches. Each row captured information about an event where a player died within the match. In the "aggregate" files, each match's meta information and player statistics were summarized (as provided by PUBG).

While working with the data, we saw that there were a few outliers in the dataset, wherein some players have survival times in the range of millions when it should ideally be below 2000. For some analysis jobs, we omitted these data points by filtering any time record that had the *player_survive_time* attribute above 2500. We tested our averages on the dataset by omitting records that had survival times below 2000 as well. The results for omitting records with survival time below 2000 and 2500 were the same so we went for a bigger threshold of 2500 in case we later realized that we missed some other data points.

Distributed Machine Learning

Taking a closer look at the data set schema, only the "aggregate" subset has data relevant to machine learning. The fields relevant to player survival are: *game_size*, *match_mode*, *party_size* and *player_dmg*. Game size affects survival duration since games with more number of players last longer. Match mode refers to whether the player was playing in first person perspective or third person perspective, which affects the player's view, controls, etc. and thereby influences survival duration. Party size was selected as an input since one snippet of our Big Data analysis revealed that teams with larger parties tended to have longer survival durations. Finally, player damage is a measure of a player's offensive capacity, thereby affecting their likelihood of survival in a shoot-out with another player. The other parameters in the "aggregate" subset unfortunately do not have a correlation with the player survival duration.

The aforementioned four fields are considered inputs for the neural network, and thereby extracted from the "aggregate" subset along with *player_survive_time*, which serves as the target field for the neural network.

Since our data was not image-based or time-series based, we did not use CNNs or LSTMs. Instead, we used a classical fully-connected neural network which used backpropagation for training, and implemented it using distributed PyTorch.

The first design issue we ran into was the nature of the *match_mode* field, since it was categorical compared to all the other fields which were numerical. We looked into some of the proposed solutions [2] and found that solution 1 resulted in unnecessary complexity of implementing an ensemble algorithm. Solution 3 was a good fit, and we planned to implement it by one-hot encoding the match mode and feeding it as two separate inputs to the neural network. However, upon exploring the dataset, we found out that despite over 67 million records in the "aggregate" subset, none of them pertained to first-person perspective (fpp) game mode. Hence, the need for accommodating fpp mode became unnecessary due to lack of data, and we proceeded with the other aforementioned parameters as inputs.

The next issue we encountered was finding the structure of an optimal network. We started off by using a network with only one hidden layer and 20 hidden units. When the result was a high test error

(Root Mean Square Error), we believed our network had not learned enough, and subsequently tried with a larger network structure, a network with two hidden layers, 10 units each. When the test error increased, we arrived at the conclusion that our model was overfitting the training data massively instead. For a network with a small number of inputs (4) and outputs (1), a smaller network was more effective. Consequently, we found that a network with one hidden layer and 5 hidden units worked best, and we continued to use that structure.

Through trial and error, we found that the learning rate which performed decently was 10^{-5}, for around 500 iterations. Reducing the learning rate by an order resulted in far more iterations to converge to a similar result, and increasing the learning rate caused the model to overfit when trained for 500 iterations. Using hyperbolic tangents (TanH) and rectified linear unit (ReLU) activation functions resulted in similar performances, however, ReLU seemed to achieve lower error rates with lesser number of iterations. Hence, we made use of the ReLU activation function.

The data was split 80:20 into training and testing subsets. Due to the massive size of the data, we thought of using a higher split of training data, but it resulted in the model overfitting, hence we continued with the 80:20 split.

The neural network code was written using PyTorch. The *torch.dist* library was used for distributing the execution of the neural network across multiple machines. *InsecureClient* within the *hdfs* library was used to connect with HDFS. The network was trained on 12 CUDA-enabled machines from the CSB 325 lab of CSU.

Big Data Analytics

A few things fell into place in order to facilitate the analytics on this dataset. The dataset was rich, voluminous and we had access to an hdfs cluster to potentially store and compute results on these datasets. Since all these resources were made available to us by the department and courtesy of the dataset curator referenced in [1], we decided that we would do some analyses on this data in order to gain more insight about the actual metrics that are calculated in the game and how that might affect gameplay results and advise strategies. The different aggregation analyses we performed on the data are:

- **Average Kill Distances by Weapon:** According to the dataset and the data curator, the map in the game PUBG is divided into latitudes and longitudes. All the kills are recorded in the deaths dataset along with the weapon used for the kill as well as the latitudinal and longitudinal coordinates for both the killer and the victim. We used MapReduce in order to process the entire dataset and calculate the kill distance in each record and aggregate it by weapon, emitting the aggregated (key, value) pair from the mapper to be collected in the reducer.

- **Number of Kills per Weapon:** For this analysis, we aggregated the number of instances a particular weapon was used to kill someone. The weapons listed in the dataset also included some of the game mechanics. Dataframes and aggregation modules in the pyspark sql module were used to perform this computation.

- **Survivors Beyond a Given Time:** The objective in PUBG for any player is to survive as long as they possibly can and the winner is the last man standing or the team who has the last man standing. Thus we realized that in order to advise anyone on what percentile of players they belong to we need to figure out how many players actually survive beyond that time. Our total dataset had details of 720,000 matches, which we used to extrapolate significant insights. These

survival percentiles should be relatively true across all games, bar the developers changing game mechanics significantly. Dataframes and aggregation modules in the pyspark sql module were used to perform this computation.

- **Survival Times by Team Positioning / Ranking (Mean and Median analysis):** The ranking of one's team post-game completion is a clear indicator of how the team performed. A team is assigned its rank based on the players that comprise the team. If the last standing player belongs to Team A, then Team A placed first. This is how rankings are assigned. Thus we wanted to do some aggregation analyses what kind of correlation a metric like team positioning has with the survival time, the maximization of which is the goal of the game, and there are multiple members in a team usually and how do those averages stack up when we consider all players. For this analysis we removed outlier data points. Dataframes and aggregation modules in the pyspark sql module were used to perform this computation for the averages. We also performed the same analysis with the median of survival times to see if the result differed significantly compared to the mean analysis. The median calculation was done using MapReduce because MapReduce provides greater granularity with data passing between map and reduce tasks, which dataframes do not allow.

- **Survival Times by Team Ranking and Game Size (Mean and Median analysis):** For this analysis, we built upon the idea mentioned above. We also took into account the fact that game size was a metric affecting survival times on an absolute scale. For instance, a game with 15 players would not last as long as a 100 player game. Therefore we group our data by team positioning and then pivot it by game size in order to have data for both these configurational metrics. For this analysis we got rid of outlier data points. Dataframes and aggregation modules in the pyspark sql module were used to perform the computation for the averages. The median calculation was done using MapReduce due to MapReduce providing greater granularity with data passing between map and reduce tasks, which dataframes do not allow.

- **Average Survival Times by Party Size:** Party Size refers to the number of players in a team, playing as one unit and the members of a team affect the results of the entire team. We wanted to see if there was any observable trend in survival time based on the size of the teams (possible team sizes are 1, 2 and 4). For this analysis we removed outlier data points. Dataframes and aggregation modules in the pyspark sql module were used to perform this computation.

- **Average Damage by Team Positioning / Ranking:** As aforementioned, team positioning / ranking is a very strong identifier of the quality of someone's abilities in the game environment. Thus, we wanted to get a brief idea of the relation between player damage and positioning of their team. We theorized seeing a direct proportionality; furthermore, one can extrapolate someone's ranking by their damage or vice versa. Dataframes and aggregation modules in the pyspark sql module were used to perform this computation.

Results and Evaluation

The data was split into training and test data, with a randomized 80-20 split. We used the test data to compute the root mean square error of model predictions compared to actual survival durations in order to determine the performance of our networks, and compile a comprehensive summary of all results

across our explored models. The test split also served as the evaluation metric for our model, which was able to predict survival times accurately.

Distributed Machine Learning

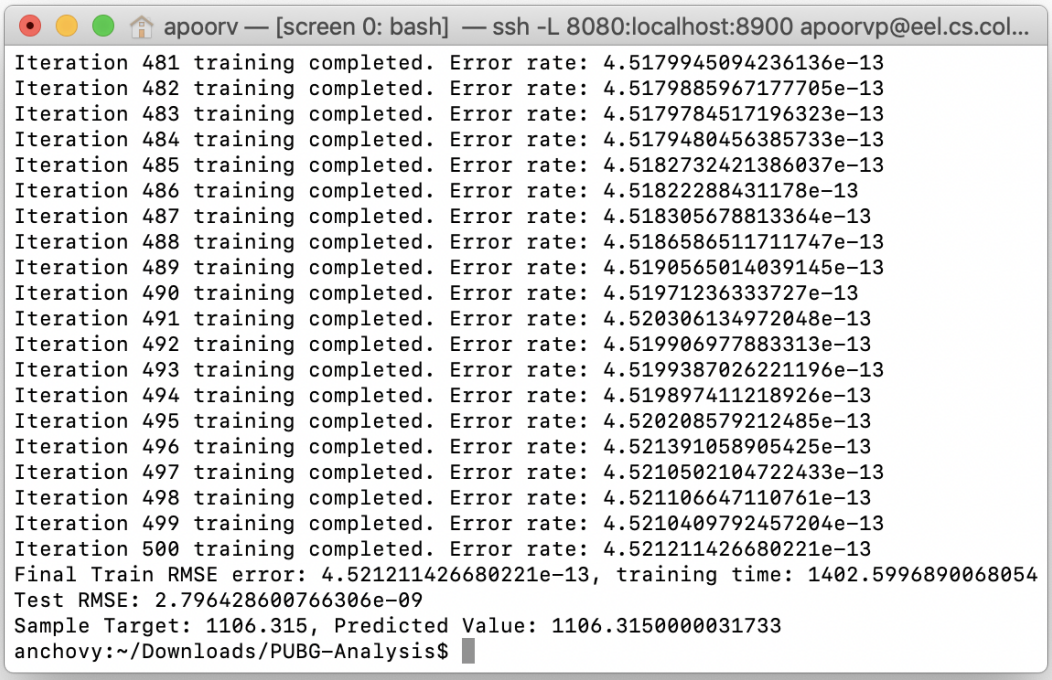
As we mentioned earlier, we trained the model over 12 CUDA-enabled machines from the CSB 325 lab at CSU. Initially while attempting to train, the models across different machines varied wildly. This was due to network gradients being computed individually on each machine, leading to different gradients across different machines and different explorations of the result space. Hence, the iterations and error rates were completely unsynchronized, and so were the results. To remedy this, we modified the training function to average gradients across all machines before backpropagating. This led to immediate synchronization of iterations. The error rates were not the same, however, they were quite similar across machines. More importantly, instead of completely different explorations leading to very different training RMSEs for the same iteration number, the gradient averaging led to exploring the same direction of result space, and relatively same rate of decrease of training RMSE for each iteration. At the end of training, RMSEs across various machines had a negligible difference in error.

The results of our various network training configurations (in chronological order) have been captured in the table below. Note that all results are for training for 500 iterations, and with ReLU activation function.

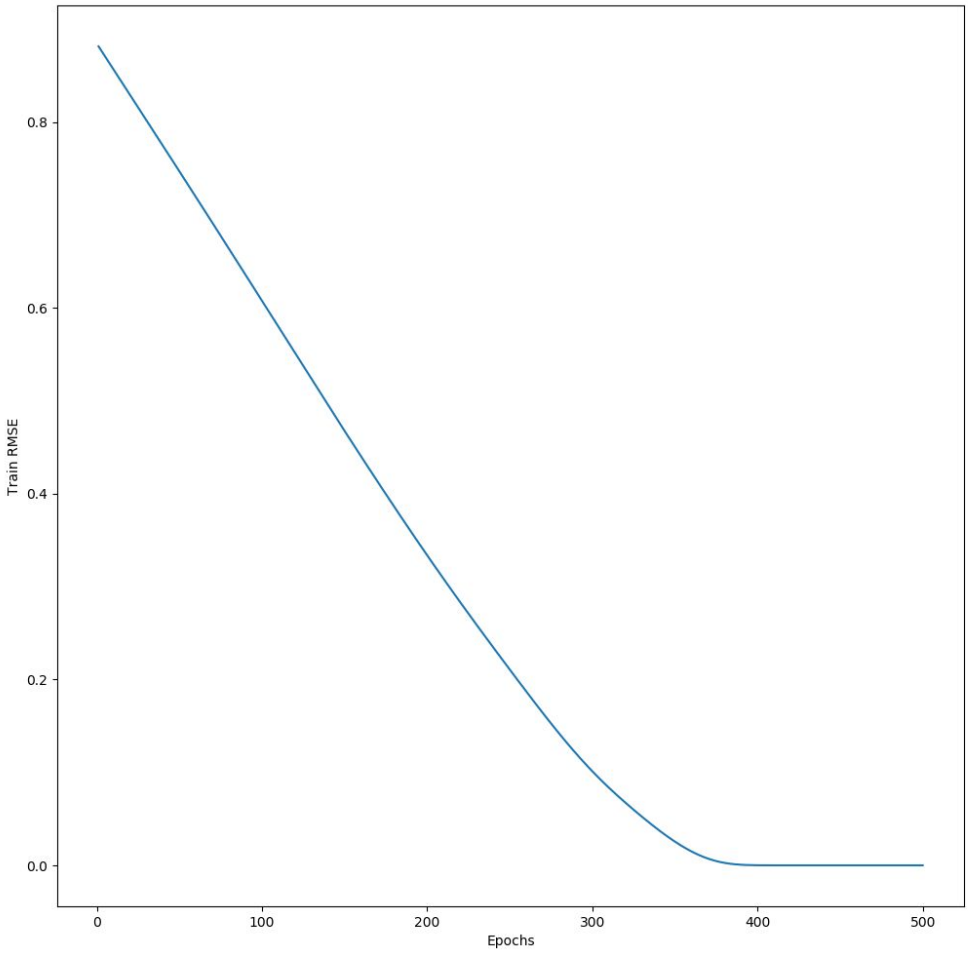
Input data	**network structure**	**learning rate**	**batch size**	**train RMSE**	**training time**	**test RMSE**	**Comments**
~ 67 million	[20]	10^{-5}	10000	0.064	6418 s	303.45	Overfitting
~ 67 million	[10, 10]	10^{-5}	10000	4.32e-07	7052 s	1408.759	Massive overfitting
~ 67 million	[5]	10^{-5}	10000	3.061e-06	6936 s	12.859	Little overfitting
~ 67 million	[5]	10^{-5}	100000	7.634e-05	654 s	76.133	Less overfitting
~ 67 million	[5]	10^{-5}	10000	1.255e-07	6930 s	1.086	Almost accurate
~ 67 million	[5]	10^{-6}	67000	0.736	995 s	609.994	Results not converged
~ 67 million	[5]	10^{-5}	67000	4.521e-13	1403 s	2.796e-09	Outliers removed

As can be observed, increasing network complexity resulted in massive overfitting. Increasing batch size led to faster training, due to less weight updations. Due to the gradient averaging, weight updation results in network communication, leading to performance slow down. We achieved a test RMSE of ~1, which we deemed almost accurate. This is because survival times in the dataset were of the order 10^2 or 10^3, for which a prediction with a margin of error of 1 is fairly accurate. We switched to a batch size of 67000 as an intermediate between 10000 and 100000, and since the data had ~ 67 million rows. Also at the same

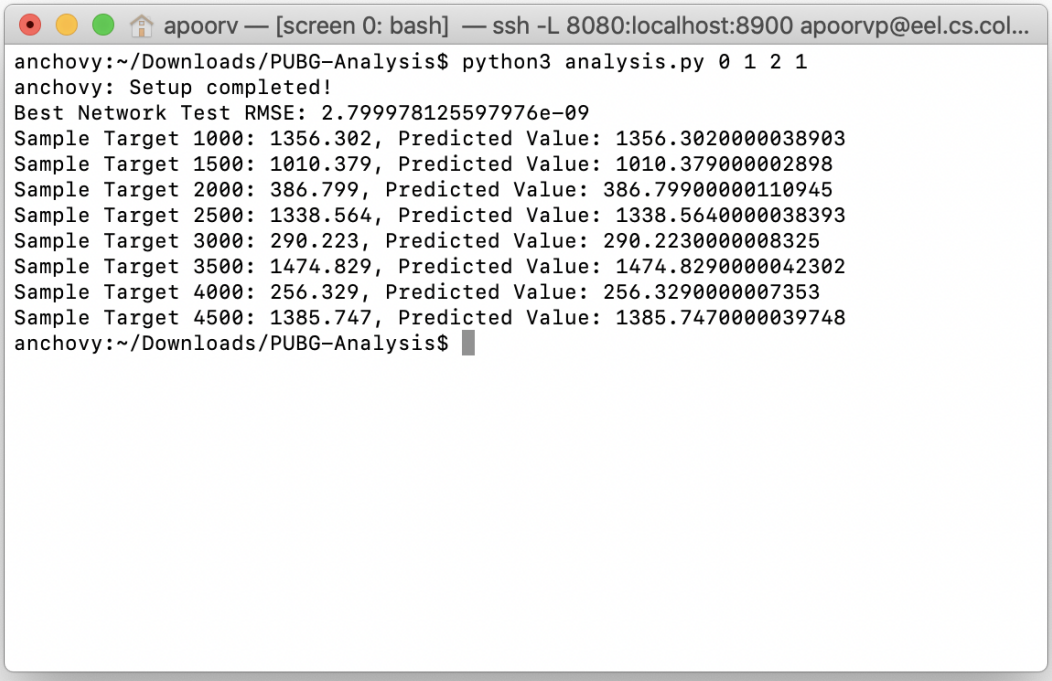
time, we reduced learning rate to 10^{-6}, which led to training being unable to converge within 500 iterations. Around this time, our Big Data analytics revealed that few survival duration outlier data points were probably affecting both our training and evaluation. We suspected that the outlier data points were a result of players using cheats or hacks, since other survival durations were less than 2500, while the outlier durations were over 6 million. Hence, we performed data processing to remove the outliers, and performed our training again. Removing the outliers reduced the number of rows from 67,369,231 to 67,369,186 (45 rows). The preprocessing led to a massive improvement in training, with a test RMSE of only 2.796e-09, which can be construed as accurate, given the typical order of survival durations. A snapshot of the end of training outcome can be seen below.



The error rate plot for the last training looked as follows:



To test how well the network performed, we tried feeding it some input samples, and these were the results:



As can be seen from the figure, the predictions made by our model are extremely precise, upto the 6th decimal point. Given that the survival duration samples were 10^2 or 10^3, our model is, practically speaking, perfectly accurate.

Big Data Analytics

For some analyses, the results had a large number of essential data points, thereby rendering graphs an improper visualization. For instance, there were 54 weapons in the dataset, hence plotting results for only some would be incomplete and lacking a holistic view. Therefore, we are presenting a portion of our results, with comprehensive results submitted alongside the source code. The results for the analyses (presented in the order followed in methodology) are as follows:

- **Average Kill Distances by Weapon:**

Weapon	DP-28	Hit by Car	Mk 14	Pickup Truck	M24	M416	Micro UZI	M16A4
Avg. Kill Distance (Scaled)	0.1405	0.6371	0.1837	0.3720	0.1985	0.1275	0.1771	0.1555

Looking at the results of this analysis, one can make inferences about the most effective weapons by range, thus allowing players to make an appropriate weapon selection based on their preferred play range. For instance, based on the sample presented, a player preferring to play from further away should choose an M24, whereas a player who likes to get close should choose an M416.

- **Survivors Beyond a Given Time:**

Time Slot	0.0	2050.0	1150.0	250.0	2300.0	1400.0
Survivors	126	34882	1222534	3671268	16	1197996

From this result sample it is evident that fewer people survive for longer and that is what we expect and using this we can advise beginners on what chances they have of survival given the average amount of time that they survive. This average can be approximate and does not need to be exact because we perform the same kind of bucketing for the analysis. We round every survival time attribute per record up or down to the nearest 50 and that is how we bucket the players into time slots. This helps us gain an estimate on how skilled a player is based on the average times they have survived.

- **Number of Kills per Weapon:**

Weapon	Pickup Truck	S12K	Micro UZI	Mk14	Grenade	M416
# of Kills	22361	614964	552024	34778	497896	3165145

An aggregation of this particular analytic allows players to pick and choose the more popular weapons in order to advise strategies where one can strategize based on the usage of said weapons, i.e., more frequently used weapons must also be more frequently available and stocking up on ammunition for these weapons is likely to be optimal for victory in a game. Alternatively, players can come up with the best counter weapons for the most popular weapons being used, giving them an inherent advantage over other players.

- **Survival Times by Team Positioning / Ranking (Average and Median Analysis):**

Rank	6	86	16	3	40	20
Average	1389.72	150.66	790.15	1579.99	355.03	575.06
Median	1541.99	121.82	798.35	1749.64	236.17	478.14

From a glance at these results we can see that as the team rank goes higher, the average survival time also goes up with it. Another interesting trend we noticed was that the median survival time was higher than the average for higher ranked teams, and vice versa for lower ranked teams. This could be because lower ranked teams perhaps have only a few members who are good players thus skewing the average to be higher than it is supposed to while the median remains lower. For higher ranked teams, the average is high and so is the median, since while one untimely death could skew the average, the median is

preserved and is safe from outliers such as bad network connections and mistakes. The benefit of this analysis is that based on the predicted survival time via our machine learning model, players can estimate their final in the game.

- **Survival Times by Team Ranking and Game Size (Average and Median Analysis):**

This analysis showed us the same results from before but it was pivoted with the number of total players per game as well. A small example of that would be :

Rank	6	16	86	3	40	20
Game Size	3	50	100	3	50	50
Average	“”	1063.82	158.64	346.16	257.41	885.10
Median	“”	1118.55	141.77	443.27	236.28	922.27

This table, when viewed in full, has multiple implications. We can have an estimate of how long we have to survive in order to get the best rank, or a desired rank depending on the total number of players in the game. It intuitively follows that if your rank is 3, then there must have been at least 3 players in the game. Thus, rank 6 has no entry for game size of 6. This table is quite vast and aggregates the survival times on the most important metrics that influence this time. In a game size of 50, the 20th ranked team, survived almost 3 times longer than the team that placed 40th. We also see the relation we found in our previous analysis where the median of better teams is better than their averages but the inverse is true for lower ranked teams.

- **Average Survival Times by Party Size:**

Party Size	**Average Survival Time**
1	723.59
2	765.38
4	828.90

This is the full result and it clearly provides an indication that on average one will survive more if they play in a team versus playing alone. Since these are average measures, there are certainly exceptions to the aggregate-based insight we have generated. For beginner players however, this result will serve as a useful heuristic/reaffirmation that there is strength in numbers.

- **Average Damage by Team Positioning / Ranking:**

Rank	6	16	86	3	40	20
Average Damage	183.13	116.67	30.17	241.99	71.95	100.66

These results are in line with our expectations. The survival times for these ranks are also presented above giving us a correlation between rank, damage and survival times, wherein we see a kind of linear relation amongst all three (lower numerical value for rank is better).

Contributions

Apoorv Pandey: Distributed Machine Learning, report creation
Saptarshi Chatterjee: Big Data Analytics, HDFS cluster, report creation
Prinila Ponnayya: Dataset exploration and preprocessing, report creation, presentation

Bibliography

[1] PUBG Deaths Dataset URL - https://www.kaggle.com/skihikingkevin/pubg-match-deaths
Author/ Aggregator - Kevin Pei
Date Published - Friday, January 12th 2018

[2] https://datascience.stackexchange.com/questions/29634/how-to-combine-categorical-and-continuous-input-features-for-neural-network-training - Last accessed 5th May 2020

[3] https://spark.apache.org/docs/latest/api/python/pyspark.sql.html - Last accessed 5th May 2020

[4] https://pytorch.org/tutorials/intermediate/dist_tuto.html - Last accessed 5th May 2020