

Using Apache Spark to predict finish placement in PlayerUnknown's Battlegrounds game

Pathan Faisal Khan, Letterkenny Institute of Technology

Abstract—

Index Terms—

I. INTRODUCTION

PLAYERUNKNOWN'S Battlegrounds (PUBG) [1] is an online shooter multiplayer game of battle royale genre [2] inspired by the Japanese film "Battle Royale" [3]. The game is playable on Personal Computers (PC), Sony's PlayStation, Micorsoft's Xbox and mobile devices (Android and iOS). It was developed and published in late 2017 by South Korean video game company Bluehole's subsidiary PUBG Corporation. Since its launch, PUBG has seen exponential growth. It crossed the 1 million players mark within 48 hours of its release on consoles and a total of 30 million copies were sold for both PC and consoles just within a few days of its release. It has gained quite a good traction in Asian countries especially in China and India on to its mobile version with India standing at 116 million downloads which makes 21% of its total worldwide downloads followed by China with 108 million downloads which accounted for 19% while the USA stood at 8% with 42 million downloads [4]. With these many players on its platform, it generated vast amounts of data.

The game's most popular mode, classic mode, allows up to 100 players to parachute on an area of up to 8 x 8 kilometers island on which players will fight with each other in teams of maximum 4 players, the team to survive at the last is winner. The game world depending on the map selected has different terrains with sea, rivers, deserts, forests, snow, roads and bridges. Buildings and towns are spread across the map which has weapons and other essential items for a player to fight and survive in the game.

This technical report is going to explore a publicly available dataset [5] of the popular game. At the time of writing this report, the dataset is hosted on a popular online community for data scientists and machine learning practitioners, Kaggle [6]. The code for this report has been open-sourced and hosted on Github [7].

The data is taken from Kaggle's competition on "PUBG Finish Placement Prediction" [5]. It contains data of 65,000 matches where each row defines a player's stats after they finish their game. The training set has 4,446,966 rows while the testing set has 1,934,174 rows and 29 variables. The data is complete, but it requires some pre-processing for converting

categorical variables to numerical.

The primary goal of this technical report is predicting percentile of a player's finishing position where 0 denotes last position and 1 stands for 1st position in a match. The report has also answered the following questions with visualizations based on analysis of the data:

- 1) Average time a player spends in a match.
- 2) Total number of kills done by users when playing alone versus when playing in a team.
- 3) How better a player performs (in terms of rankPoints, winPlace, winPoints) when playing alone versus when playing in a team?

All these computations will be performed on distributed computing framework due to large size of the data.

Organization of the report is formatted in the following way. The paper will start with a brief overview of the program paradigm in Section II. Later on, the algorithms used for prediction will be covered in Section III. The implementation plan which describes the data pre-processing stage as well as model creation will be discussed in Section IV. Section V will discuss the observations of the analysis answering the questions mentioned earlier. This section will also compare the used algorithms. The paper will finally conclude with the observations from this report in Section VI.

II. SYSTEM OVERVIEW

To run the algorithms, Apache Spark has been selected as the proposed solution which swiftly handles large datasets and computes parallelly. Apache Sparks is used for performing large-scale data analytics using technologies used in modern data centers on the production level. Apache Spark will be implemented on a cloud solution. Google Cloud Platform (GCP) has been selected as the cloud service provider. Under GCP, the project will be implemented using Cloud Dataproc [8], BigQuery [9] and Apache Spark's Machine Learning Library [10]. With the combination of these services, we will be making out our analytics and prediction. Cloud Dataproc is described on the GCP website as "a fast, easy-to-use, fully managed cloud service for running Apache Spark and Apache Hadoop clusters in a simpler, more cost-efficient way" [8]. They have claimed that using their service, it has proven to reduce time consumption on operations which took hours or days

on a traditional system to seconds or minutes [8]. BigQuery will be used in our project to store our datasets which provides a serverless and highly-scalable data warehousing solution. For implementing our model, we will be using Apache Spark's Machine Learning Library. The library is available in Java, Scala, and Python, we are interested to implement our model in Python.

We will be creating jobs using Apache Spark ML library and running them on our Dataproc clusters. The Figure 1 shown below describes our expected workflow:

III. ALGORITHMS

Algorithms used in this report are for prediction of the win placement percentile of a given player. We have used regression algorithms, AdaBoost Regressor, Gradient Boosting Regressor, Random Forest Regressor and Decision Tree Regressor.

A. AdaBoost Regressor

AdaBoost, also known as Adaptive Boosting, is one of the first boosting algorithms proposed by Gödel Prize winners Robert E. Schapire and Yoav Freund in 2012 in their published book "Boosting: Foundations and Algorithms" [11].

A boosting algorithm works by combining multiple weak algorithms which have no correlations amongst each other and voting for the best solution. In this way, a couple of weak algorithms come together to make one strong algorithm. Boosting algorithms are based on ensemble learning, which literally means "a group producing a single effect".

AdaBoost uses decision tree with a single split as its weak algorithm often called as decision stumps and applies boosting over it. Since AdaBoost is based on ensemble learning, it tries to fit all data points, it fails with noisy data. The most astonishing thing about this algorithm is that it does not overfits even though it is trying to fit all the data points. AdaBoost Regressor can be used both for classification and regression problems.

B. Gradient Boosting Regressor

Similarly to AdaBoost discussed earlier, gradient boost is also an ensemble learning based boosting algorithm. It also works on decision stumps. This algorithm was proposed by Friedman in 1999 in his paper titled "Greedy Function Approximation: A Gradient Boosting Machine" [12]. The only difference between AdaBoost and gradient boost is that depth level of the tree varies from 8 to 32 in the latter while the depth level is 1 in the former. This algorithm can also be applied for solving classification and regression problems.

C. Random Forest Regressor

Ho discovered this supervised learning algorithm in 1995 [13]. Random forest is also based on ensemble learning similarly to AdaBoost and gradient boost. There are two types of ensemble learnings, boosting and bagging. Unlike AdaBoost and gradient boost, random forest is a bagging

based technique. Bagging also referred to as bootstrap aggregating which samples data randomly with a possibility of replacement. This technique enables multiple models to operate without intervening with each other.

In the random forest algorithm based on bagging, all its trees run independently together. These trees are decision trees generated during training. The benefit of this algorithm is that it can swiftly handle large amount of data but it overfits with noisy data.

D. Decision Tree Regressor

Decision tree is a predictive modelling algorithm which with the help of a collection of binary rules predict an output. These rules are formed by a QnA by the algorithm on the data to narrow down to the problem until it reaches a final conclusion. It can be used for both classification and regression problems. But this algorithm is not suitable for continuous variables and is also prone to over fitting.

In this report, we will be using the above mentioned algorithms to perform our primary goal of predicting win placement percentile of players. These algorithms will be compared on the basis of their accuracy.

IV. IMPLEMENTATION

V. EXPERIMENTATION RESULTS AND DISCUSSIONS

TABLE I
PERFORMANCE OF DIFFERENT ALGORITHMS RUN ON THE SAME DATASET

Algorithm	Category	Accuracy
AdaBoost Regressor	Ensemble Learning Boosting	91.71%
Gradient Boosting Regressor	Ensemble Learning Boosting	94.86%
Random Forest Regressor	Ensemble Learning Boosting	93.97%
Decision Tree Regressor	-	81.58%

VI. CONCLUSION

REFERENCES

- [1] PlayerUnknown's battleground- this is a battle. [Online]. Available: <https://www.pubg.com/>
- [2] "Battle royale game," page Version ID: 930356754. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Battle_royale_game&oldid=930356754
- [3] K. Fukasaku, "Battle royale," page Version ID: 931632941. [Online]. Available: [https://en.wikipedia.org/w/index.php?title=Battle_royale_\(film\)&oldid=931632941](https://en.wikipedia.org/w/index.php?title=Battle_royale_(film)&oldid=931632941)
- [4] A. McAloon. Now out of early access, battlegrounds crosses 30m players. [Online]. Available: <https://bit.ly/2PbIlps>
- [5] PUBG finish placement prediction (kernels only). [Online]. Available: <https://kaggle.com/c/pubg-finish-placement-prediction>
- [6] Kaggle: Your home for data science. [Online]. Available: <https://www.kaggle.com/>
- [7] P. F. Khan, "faisal3325/pubg-prediction." [Online]. Available: <https://github.com/faisal3325/pubg-prediction>
- [8] Dataproc - cloud-native apache hadoop & apache spark | cloud dataproc. [Online]. Available: <https://cloud.google.com/dataproc/>
- [9] BigQuery: Cloud data warehouse. [Online]. Available: <https://cloud.google.com/bigquery/>
- [10] MLlib: Main guide - spark 2.0.0 documentation. [Online]. Available: <https://spark.apache.org/docs/2.0.0/ml-guide.html>

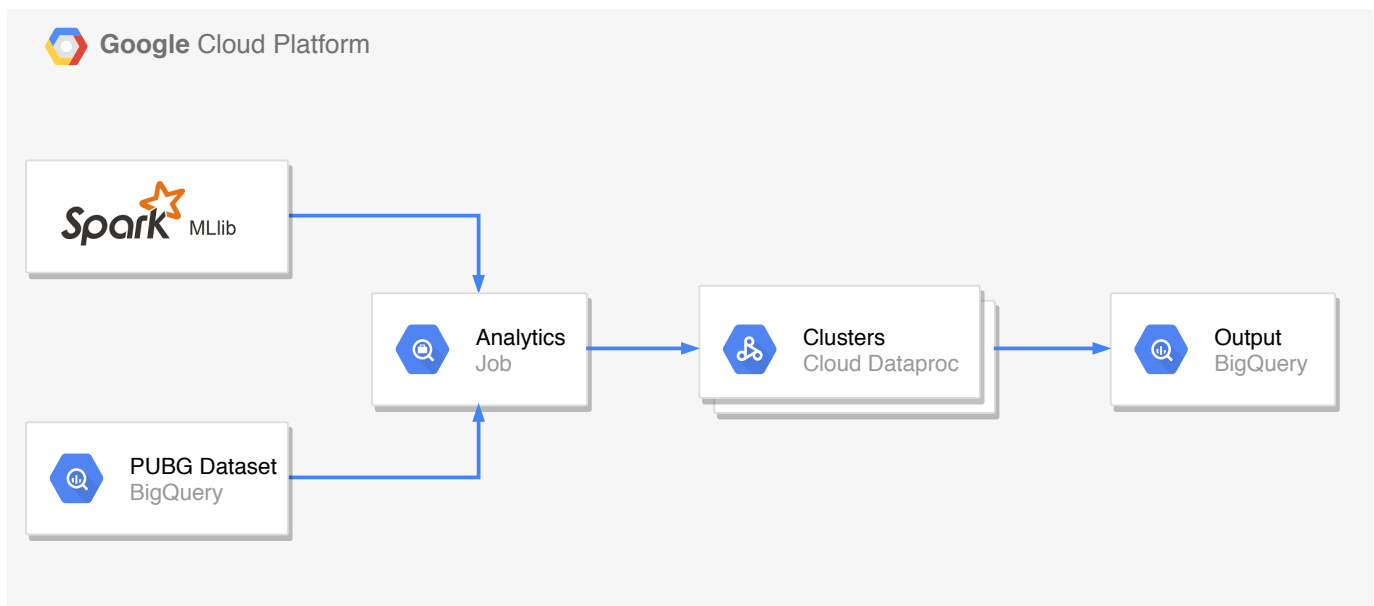


Fig. 1. Workflow of the analytics operation.

- [11] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*. MIT Press.
- [12] J. H. Friedman, "Greedy function approximation: A gradient boosting machine." vol. 29, no. 5, pp. 1189–1232. [Online]. Available: <https://projecteuclid.org/euclid.aos/1013203451>
- [13] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, pp. 278–282 vol.1.