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 it's 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 parallely. 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

Under the supervision of Dr. Shagufta Henna, Letterkenny Institute of Technology, Letterkenny, CO. Donegal 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:

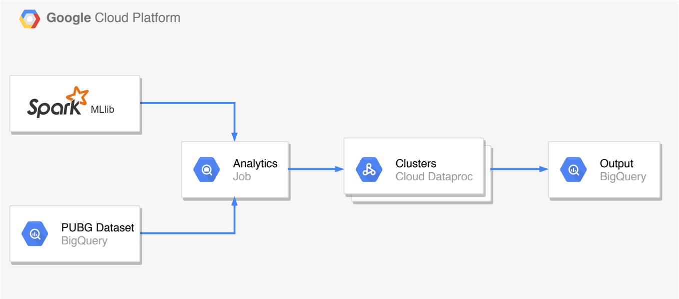


Fig. 1. Workflow of the analytics operation which is described in the Implementation section IV.

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 independetly 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

The code of this report has been implemented in python using APIs of Apache Spark's machine learning library, PySpark. This library specifically for python to use Apache Spark's resources on a cluster. The code uses python 3.7.0. The program workflow as shown in figure 1, relies on GCP for data storage as well as Apache Spark clusters.

The data is stored in BigQuery which is used for storing large amounts of data both streaming as well as non-streaming. BigQuery provides prallel processing of SQL queries on a large scale and outputs result in fraction of seconds. This is why BigQuery has been selected as an ideal data hosting service. The data from BigQuery is fed to our program where it is stored as RDD (Resilient Distributed Dataset). Since we had to do some pre-processing of the data, we converted the data to Pandas Dataframe so that we can utilise its APIs. This pre-processed data is then stored in BigQuery so that we won't have to perform pre-processing everytime our cluster runs the program thus preventing huge bills.

This newly stored pre-processed data is then used by our main program where multiple supervised learning algorithms are applied on it to get predictions. This program is then deployed on a cluster running on Dataproc.

In the following sections, each of the steps defined above is described briefly.

A. Data Pre-processing

The original dataset had 29 variables. A few of them were categorical variables rest were numerical continuous variables. The following categorical variables were dropped from the dataset which seemed to be not much useful:

TABLE I
CATEGORICAL VARIABLES WHICH WERE DROPPED

Variable	Description	Sample Data
Id	Player's unique identifier	7f96b2f878858a
$matchId$	Unique identifier for matches	a10357fd1a4a91
$groupId$	Unique identifier for groups in a match	4d4b580de459be

We then converted the column $matchType$ from categorical variable to numerical variable, resulting in 15 categories. Since some matches did not had 100 players, we cannot compare data of different matches as the number of players is not same in both, so we normalized the $kills$ and $damageDealt$ variables. To normalize them, we used the formulaes given below:

$$killsNorm = \frac{kills}{totalPlayers} * 100$$

$$damageDealtNorm = \frac{damageDealt}{totalPlayers} * 100$$

We then created another normalized variable, $normMatchType$ with $matchType$ values. Due to huge differences in the size of the solo game or of a team game, we set this 'other' and to the normal team size: 1 for solo; 2 for a double; ¡ 4 for the team; and other for the rest. The data are different from the standard team sizes. Data from the custom games are also made up of different values of team sizes like 10, 15, 92, etc. that are added to a different value.

We then did feature engineering where we created 13 additional variables:

1) totalDistance: $totalDistance$ is calculated by summing up $rideDistance$, $walkDistance$ and $swimDistance$.

2) maxPossibleKills: $maxPossibleKills$ is the maximum kills a player can do in a match which is calculated using $totalPlayers - teamSize$.

3) itemsUsed: $itemsUsed$ is all the items the player has picked up or used. Items which we have data of are $boosts$, $heals$ and $weaponsAcquired$.

4) itemsPerDistance: $itemsPerDistance$ is the normalized value of $\frac{itemsUsed}{totalDistance}$.

5) killsPerDistance: $killsPerDistance$ is the normalized value of $\frac{kills}{totalDistance}$.

6) damageDealtPerDistance: $damageDealtPerDistance$ is the normalized value of $\frac{damageDealt}{totalDistance}$.

7) maxTeamKills: $maxTeamKills$ shows the maximum number of team kills a player has done.

8) totalTeamKills: $totalTeamKills$ is the total kills a team has performed together.

9) headshotKillRate: $headshotKillRate$ is made for identifying the skill of the player by dividing total number of headshot kills he did, $headshotKills$ by the total $kills$.

10) itemsUsedPerTeam: $itemsUsedPerTeam$ is the total number of items used by a team combined.

11) percKill: $percKill$ is the percentage of kills a player has performed in a match based on the maximum kills a player can do in a match.

12) percTeamKills: $percTeamKills$ is the percentage of kills a team has performed in a match based on the maximum kills a team can do in a match.

13) meanTeamKillPlace: $meanTeamKillPlace$ is the mean of $killPlace$ for all players in the team.

This data is now ready for applying our machine learning algorithms and is stored in BigQuery.

B. Applying Machine Learning Algorithms

The above processed data is now fetched in our main PySpark program which will run on cluster for predictions. While fetching this data, the library somehow interprets the variables as $object$ type rather than numeric. So we have to convert these values to numeric value with the below code:

```
for col in cols:
   pubg = pubg.withColumn(col,
       pubg[col].cast(FloatType()))
```

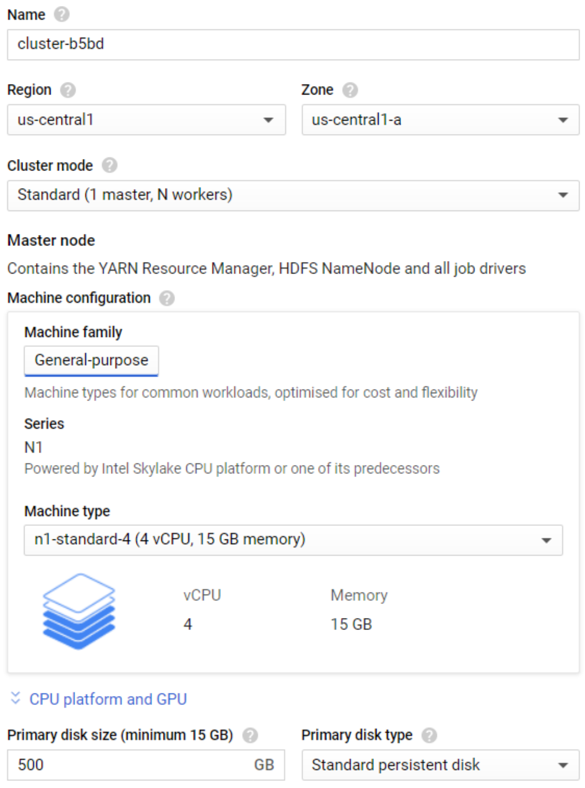
The algorithms applied on the data are AdaBoost, Gradient Boost, Random Forest and Decision Tree. $learning_rate$ has been choosen 80% for AdaBoost and Gradient Boost while $n\_estimators$ is 10 for Random Forest. The code is shown below for the algorithms applied:

```
model_1 = AdaBoostRegressor(learning_rate=0.8)
model_1.fit(X_train,y_train)

model_2 =
    GradientBoostingRegressor(learning_rate=0.8)
model_2.fit(X_train,y_train)

model_3 =
    RandomForestRegressor(n_estimators=10)
model_3.fit(X_train,y_train)

model_4 = DecisionTreeRegressor()
model_4.fit(X_train,y_train)
```

C. Deploying the program to Dataproc cluster

Now this program is ready for deploying it on the Apache Spark cluster. Before deploying it to the cluster, the cluster needs to be setup from Google Cloud Console or using $gcloud$ cli command.

The following figure 2 shows a screen capture of the Dataproc where it is showing inputs for clusters formation.



Name

cluster-b5bd

Region

us-central1

Zone

us-central1-a

Cluster mode

Standard (1 master, N workers)

Master node

Contains the YARN Resource Manager, HDFS NameNode and all job drivers

Machine configuration

Machine family

General-purpose

Machine types for common workloads, optimised for cost and flexibility

Series

N1

Powered by Intel Skylake CPU platform or one of its predecessors

Machine type

n1-standard-4 (4 vCPU, 15 GB memory)

vCPU

4

Memory

15 GB

Primary disk size (minimum 15 GB)

500 GB

Primary disk type

Standard persistent disk

Fig. 2. Google Cloud Console screen of cluster creation

Alternatively, we can create a cluster using *gcloud* cli.

```
$ gcloud dataproc clusters create
    cluster-b5bd --region us-central1
    --subnet default --zone us-central1-a
    --master-machine-type n1-standard-4
    --master-boot-disk-size 500
    --num-workers 2 --worker-machine-type
    n1-standard-4 --worker-boot-disk-size
    500 --image-version 1.3-deb9 --project
    lyit
```

The parameters we used for creating clusters are:

TABLE II

GOOGLE CLOUD DATAPROC PARAMETERS FOR CLUSTER

Parameter	Value
Region	us-central1 (default)
Subnet	default
Subnet zone	us-central1-a
Master machine type	n1-standard-4
Master boot disk size	500 GB
Workers	2
Worker machine type	n1-standard-4
Worker boot disk size	500 GB
Image version	1.3-deb9

D. Visualizing the output

Some of the visualizations have been directly done on Jupyter notebook but most of the visualizations have been done on Google Data Studio which gives a connector to BigQuery and gives flexibility for visualizing large volumes of data.

V. EXPERIMENTATION RESULTS AND DISCUSSIONS

In this section we will be looking at the results of our program and answer the questions which we set out with earlier in the introduction section I.

A. Average time a player spends in a match

Using $matchDuration$ variable, we have calculated the average time a player spends in a match.

The average time is 1579.51 seconds which translates to 26.32 minutes.

B. Total number of kills done by users when playing alone versus when playing in a team

The following figure 3 shows the share of users performing kills in different $matchType$ scenarios.

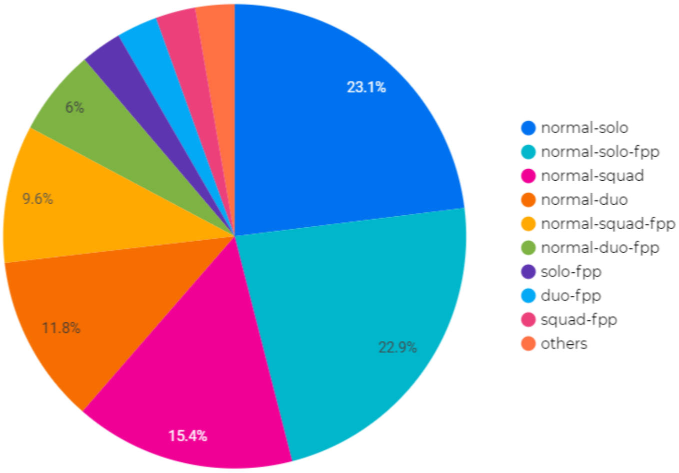


Fig. 3. Kills done by users when playing alone versus when playing in team

From the figure, it can be noted that solo players perform most kills rather than team players. Around 50% of kills are performed by solo players.

C. How better a player performs (in terms of rankPoints, winPlace, winPoints) when playing alone versus when playing in a team?

In the following table III we can see that a player playing solo performs more better in terms of winning place while rank points is achievable when playing in team. Winning points is also achievable more when playing solo as it is correlated with winning place.

TABLE III

PLAYER PERFORMANCE IN TERMS OF $rankPoints$, $winPlace$, $winPoints$ WHEN PLAYING ALONE VERSUS WHEN PLAYING IN A TEAM

Game Type	rankPoints	winPlacePerc	winPoints
Team	772.25585284	0.592398494983	736.991638795
Solo	759.890547263	0.691486069651	779.452736318

D. Predicting win placement (winPlacePerc)

The algorithms we discussed earlier in section III is compared in this section.

TABLE IV

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%

We can see that gradient boosting regressor has the highest performance that is 94.86%. It can also be noted that decision tree regressor has the lowest accuracy that is 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

[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

[11] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*. MIT Press, 2012.

[12] J. H. Friedman, "Greedy function approximation: A gradient boosting machine." *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. [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, 1995, pp. 278–282 vol.1.