International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 3, Issue 12, ISSN No. 2455-2143, Pages 107-118
Published Online April 2019 in IJEAST (http://www.ijeast.com)

PUBG WINNER PLACEMENT PREDICTION USING ARTIFICIAL NEURAL NETWORK

Madhurya Manjunath Mamulpet
Faculty of Engineering, Environment and Computing, Coventry University
MSc. Data Science and Computational Intelligence (ECT104) Stage 1
Coventry, United Kingdom

Abstract: **The main objective is to predict the placement ranking of the player in PUBG (Players unknown battlegrounds) using the player's position and stats of data. PUBG is a game that has turned out to be famous in recent years. For this diversion, the last position is the most essential marker of the player's capacity. This task centres on anticipating the last position and finding ideal techniques of the diversion. With information from PUBG, we apply a few Artificial Neural Network techniques including Deep Learning, light GBM model, and MLP regression model to make the winner predictions.**

Keyword**: Deep learning, LGBM, MLP.**

I. INTRODUCTION

Player Unknown Battle Grounds is a multiplayer online battle game i.e. developed by the PUBG Corporation. This is a game where 100 players fight or play in battle with each other. Players can choose to fight alone, with a group of 2 or 4. Although, the size of the team can be modifiable through featured engineering. The winner of the game will be the player that survived or stays alive till the end of the game. This can be through various strategies such as killing an opponent using a weapon or surviving by healing or hiding. There are many tools in a game that helps the player to kill his enemies like armour, weapons, healing kits, vehicles and many other resources. The dataset contains five million records out of which there are 28 features(1). The main objective is to predict the player winning percentage based on various aspects such as the number of kills obtained, survival rates, number of players alive and many more variables. This is done using an algorithm the players win percentage will be ranged between 0-1. These problems are solved through analysing data and feature extraction technique upon which the winning percentage depends on.

The techniques used on the player records to analyse and extract are neural network MLP Regression, Light GBM, Deep Learning, Tensor flow etc. The accuracy of the model is validated and the result obtained reached the desired requirement.

This paper is presented in the following order.

- Related work that is researched on the same data as well as a similar game prediction.
- The methodology used for placement prediction
- Data description with the explanation of their features
- EDA data analysis
- Experimental Setup the process explanation to perform validation on data
- Results obtained by the applied methods
- The conclusion of the overall research.
- Ethical, legal and Social Issues
- Bibliography

II. RELATED WORK

The researchers have used the same data set to predict the final rank(2). They trained the model on various techniques such as regression and tree models which also provided optimal strategy. The model trained used the linear regression, ridge regression and LGBM technique the models were tuned and the 5-fold cross was applied to validate the data. LGBM aggregates the GBDT with GOSS algorithm and EFB. Mathematically represented as

$$\tilde{V}_j(d) = \frac{1}{n}\left(\frac{\left(\sum_{x_i \in A_l} g_i + \frac{1-a}{b}\sum_{x_i \in B_l} g_i\right)^2}{n_l^j(d)} + \frac{\left(\sum_{x_i \in A_r} g_i + \frac{1-a}{b}\sum_{x_i \in B_r} g_i\right)^2}{n_r^j(d)}\right),$$

Equation 1: LGBM Computation formula

In particular, there is also a semi-supervised combination of Gaussian method used with generalized k-mean clustering to aggregate informal information.

The results Obtained with an MAE of 0.0204 which was ranked 57 in the leader board of the competition. That also concludes that the LGBM using a learning rate of 0.05 was faster, lighter and best fit model.

MAE on 20% validation set	Linear Regression	Ridge Regression	Light GBM
Raw features	0.09000	0.08989	0.05654
Raw + mean	0.05736	0.05736	0.04158
Raw + mean + sum + max - min	0.04845	0.04845	0.02896
Everything above + match_mean + size	0.04825	0.04825	0.02755
Everything above + cross validation	0.04812	0.04810	0.0204

Table 1: Results Obtained

Similarly, Researchers have been trying to predict the winner of the game of football with many data science and mathematical techniques. This paper (3) predicts the winning team in the NFL American football team using neural network deep learning techniques. According to the game, the team of two will have a ball that must be put to either side of the net which is assigned to each of the time the goal is to predict which team can do it first. They also used probability technique to see the ranking of each player with their strength. In this case, they were able to correctly classify the data with 98% using many to one method. Also used an LSTM technique architecture. The data consists of 13 datasets with 130 features in the game. Using the following methods the researcher has trained and validated the dataset. The ANN output predicts the instance of training there can be a classification wrongly predicted to minimize that a cross entropy is used by doing this we can apply the back propagation method. For this model RNN i.e. recurrent Neural Network with a t-1 output this is helpful for predicting the NFL-game that helps in knowing which team often wins. Many to one classification are used to predict a binary classification. LSTM is used for dependencies of the long term that is better than RNN they include batch size, time, features as components for input format for example time of steps is determined by the data passed RNN uses k-folds of 10 to train the data set both RNN and LSTM use a 4 layer model. All the testing for deep learning was trained for a 4 layer architecture in deep learning with a Relu activation function parameter and if optimizer which has a better performance. The results obtained from all the models do not have much difference, However, they have concluded that LSTM is the best fit compared to any other model with an accuracy of 63.31% this model is robust and can outperform for predicting.

Model	Optimal parameter	Accuracy (%)	95% confidence interval
ANN	Structure: (2,2,2) Activation: tanh	61.73	[0.56, 0.68]
LSTM	Structure: (5,5,5) Batch size: 1790 Activation: tanh	63.31	[0.61, 0.66]
RNN	Structure: (25,25,25) Batch size: 1790 Activation: tanh	62.05	[0.57, 0.67]

Figure 1: Results obtained

III. METHODOLOGY

i) Light GBM (Gradient Boost Machine)

It’s fast and high-speed Gradient framework with a new algorithm which is subsided from decision tree algorithms that divides the data depth or widthwise instead of leaf wise(4). The LGBM can decrease the level order by using the same leaf wise algorithm and results are more effective and improvised that can boost the current existing algorithm.

Learning Rate	0.1
N_Estimators	50,250
Number of leaf	200
Boosting Type	Gbdt, dart, goss, rf

Table 2: LGBM Parameters

When the model is trained through the deep learning data is optimised using the Adams learning rate which is 0.01

The benefits of using this model are

- It is fast in training data and uses less memory
- Provides better compatibility with large data
- It has improvised the accuracy by using the gradient boosting algorithm

According to our model, the main objective is to reduce the function loss which means the MSE i.e. The mean squared error. Gradient boosting model helps in minimizing the loss and to find the predicted value it takes alpha as the learning rate. The best parameters used for this model is learning rate at 0.3 N estimator with a best of 250 iterations and there is 200 leaf node.

ii) Deep Learning

The neural system is made out of three layers, to be specific information layer, concealed layer and the yielding layer. The actuation work is utilized to locate the weighted contribution of each unit in the layers. Hyperparameter tuning is done to accomplish demonstrate enhancement. Utilised Keras library to enable us to prepare the best model for our information(5). Deep learning is a neural network comprising of progressive layers in which each layer later changes the data into more unique representation. In deep learning adapting more layers, means higher learning levels of the model. The output layer consolidates all the highlights and makes an assumption. Thus it contrasts from Neural Network. While straightforward Neural Network utilizes just a single concealed layer which isn't reasonable for learning complex highlights, deep

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 3, Issue 12, ISSN No. 2455-2143, Pages 107-118
Published Online April 2019 in IJEAST (http://www.ijeast.com)

learning utilizes various concealed layers to become familiar with these intricate highlights which may hazard overfitting. To keep away from this, we use batch normalization which standardizes and scales the yield of the enactment at each layer to keep away from esteems heading off to the limits. This enables each layer to gain uniquely in contrast to different layers, and thus abstains from overfitting. It also has the impact of diminishing the preparation time. We moreover use dropout at each concealed layer to abstain from overfitting by overlooking a set measure of neurons yield at each layer. Thus, deep learning can be extravagant and requires huge dataset to prepare itself on.

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 512)               22528
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_6 (Batch (None, 128)               512
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 129
=================================================================
Total params: 190,465
Trainable params: 188,673
Non-trainable params: 1,792
_________________________________________________________________
```

Figure 2: Deep learning Model Architecture.

To specify the parameters used in deep learning model architecture depicted in the above figure. We set the epoch to 20 i.e. the number of time the data cycle runs. The higher the number of times the better improvement in the model and there are 4 hidden layers on the normalisation of which the optimizer uses a learning rate of 0.01 and epsilon is used to reduce the error that prevents from diving by zero, and decay is the weight used for the optimisation

$$f(x) = max\,(0, x)$$

Equation 2: Deep Learning

Relu Rectified linear unit function that permits the activation function used for network design in the hidden layer to input is derived from its domain and the sigmoidal function for its output.

iii) MLP Regression

Multilayer Perceptron Regression model is similar to Logistic regression in which the inputs are changed by a non-linear function(6). MLP is a supervised learning method which is known as back-propagation for dataset training through multiple layers and utilises the nonlinear transformation which differs from linear perceptron. The data cannot be linearly separable. The primary objective of non-direct relapse is to give an estimation of the genuine parameter on account of perceptions ((x1, y1), ·· ·, (in, in)). This model is used for predicting the inputs when real-value quantity is predicted this should be possible by limiting the MSE work. The file type is csv text data for input. According to math in MLP they are able to approximate an XOR operator with other non-linear function. Here there is also a boundary error rate where the data execution continues no longer than this. This stat is called convergence.

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) = \varphi(\mathbf{w}^T\mathbf{x} + b)$$

W: weights of vector
X: vector inputs
B: Bias; phi non –linear activation function

Equation 3: MLP Function

IV. DATA DESCRIPTION

	Attribute	Definition
1	Player ID	Identification number given to each player
2	Group ID	Identification number given to each group
3	Boosts	Total boost items used
4	Headshot Kills	Number of enemies killed by a direct headshot
5	Kill Assist/Assist	Enemy players knocked out that were killed by teammates
6	Damage Dealt	Total damage dealt by the player
7	DBNO	Total enemies knocked out
8	Heals no	Total healing items used
9	Kill place ranking	Player ranking based on total enemies killed
10	Kill points	Kill based overall ranking of player
11	Kill streak	Total enemies killed by player in a short amount of time
12	Kills	Number of enemy players killed in a game
13	Longest kill	Longest distance between player and enemy killed
14	Num groups	Total groups in a match
15	Match duration	Time duration of a match in seconds
16	Match Id	Identification number given to a match
17	Match type	Type of match (solo,duo,squad,arcade, etc..)
18	Max place	The maximum rank a team player got in the match
19	Rank points	Elo like ranking of player
20	Revives	Number of times a player revives his team mates
21	Ride distance	Total distance covered by riding/driving vehicles
22	Road-kills	Number of enemies killed while driving
23	Swim distance	Total distance covered by swimming
24	Team kills	Total number of teammates killed
25	Vehicle destroys	Total number of vehicles destroyed
26	Walk distance	Total distance covered by walking or running
27	Weapons acquired	Total number of weapons acquired
28	Win points	Score based on overall ranking of player
29	Win place perc	The winning percentage/probability of the player

Table 3: Data Description

V. DATA ANALYSIS & FEATURE ENGINEERING

The data analysis pattern can be determined by knowing the feature usages, this leads us to apply

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 3, Issue 12, ISSN No. 2455-2143, Pages 107-118
Published Online April 2019 in IJEAST (http://www.ijeast.com)

EDA Exploratory Data Analysis alludes to the basic procedure of performing introductory examinations on data to find patterns, to spot anomalies, to test speculation and to check presumptions with the assistance of synopsis measurements and graphical presentation(7). By analyzing the data we could generate a new set of a feature from the existing one that could help in improvising our model. So we find the total number of players by their match id and group id, we can find the total distance by combining the ride distance, swim distance, walking distance and many more(8). Shown below that contribute to evaluating the prediction efficiently.

Features	Explanation
totalPlayers	Total Players in the match
teamSize	Total team member in a team
normMatchType	Match type like solo, duo,quad etc.
totalDistance	Swim+Ride+Walk Distance
maxPossibleKills	Total kills by team
itemsUsed	boost+heals+weapon

Figure 3: Feature Engineering

When we look into the data distribution pattern there are some players who score exactly 0 or 1, the rest are in between the range. We shall assume that on an average the winning percentage per match is 0.5 (mean) but this cannot be same for all the matches which depict on a lower average this happens due to players quitting the game before it ends. The correlation that we found between these attributes is as follows.

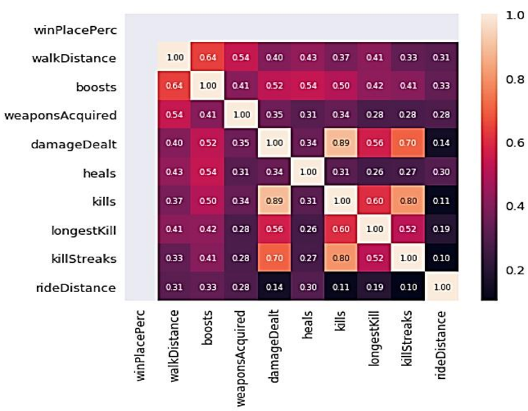

Figure 4: Correlation Matrix

VI. EXPERIMENTAL SETUP

The data is taken from kaggle PUBG prediction placement competition with nearly five million datasets out of which 1.3 million datasets are trained and tested with 29 attributes present the training and testing validation is split in the ratio of 7:3(9). Jupyter Notebook is used for coding with libraries such as sci-kit learn, numpy and Keras Tensor flow Backend is used for hierarchical deep learning models. Performed on a windows 8 computer. Tensor flow is used to control structure and it is an open source by Google. The data frame is iterated through all the columns of a data frame and the datatype is altered to reduce the memory usage. The dataset is loaded from a CSV file. Once this is done we try to find the Data Distribution pattern and correlation by plotting appropriate graphs.

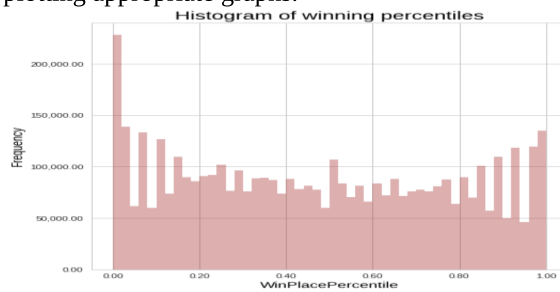

Figure 5: Winning percentile chart

The data set must be filtered i.e. done by dropping the columns or records which are 0 or no value which helps in preventing the error and reduces memory usage. Now feature engineering is done by assessing the numeric value for generated correlated new features which was previously discussed in the Feature engineering section. This new match consists of features like solo, duo and squad which depicts the size of the team.

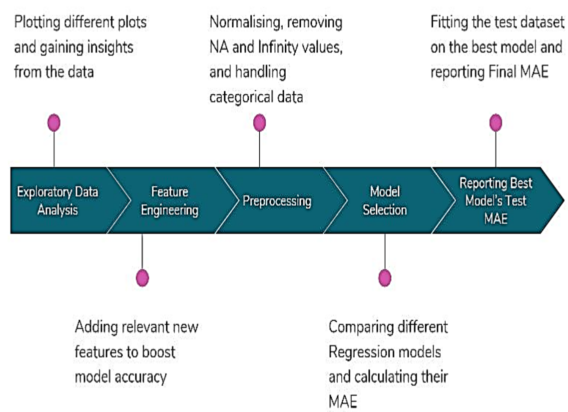

Figure 6: Process flow

We construct the model LGBM Regression. A regression model is used than classification as there are independent variables that are correlated to dependent data this method is used to find the relation between the data. Deep reinforcement learning methods were not easy to use as that requires high computational power and consumed a lot of time. Hence using these methods to compensate and reduce memory consumption and compilation time.

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 3, Issue 12, ISSN No. 2455-2143, Pages 107-118
Published Online April 2019 in IJEAST (http://www.ijeast.com)

The initialisation of the simple MLP takes places by imparting training and validating the data with appropriate parameters than by facing a loss the next model i.e. the deep learning technique uses batch normalisation to handle the loss of data by using more layers and an activation function which then iterates at its best of 200 with a 20 epoch and early stop set to 10. Lastly, the LGBM model is initialized first experimented with a grid search that took endless run time then switched to this technique that took 222 seconds of run time with best parameters and a learning rate of 0.05

VII. RESULTS

The accuracy of all the models trained and tested are shown as below:

Model	LGBM	Deep Learning	MLP
MAE	0.0539	0.05947	0.08525
R^2	93.6%	90.61%	85.4%

Table 4: Result

The obtained result shows a 5% improvisation between the MLP and deep learning model. Here MLPs learning rate is Adaptive. The first time the improvement of the model was stagnant as the model stopped running after 2 continuous epochs which mean the loss of the data which stopped at best 14 iterations. The Mae was better obtained.

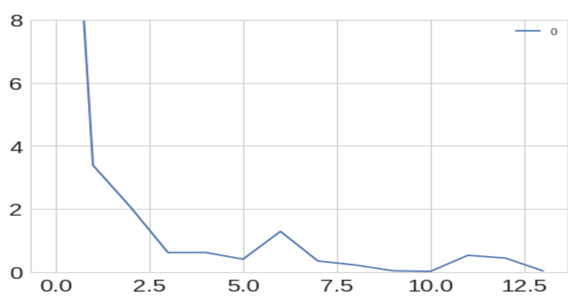

Figure 7: Loss Curve MLP

When the model was trained on deep learning we used the previous loss of data results to compare with the new value the kernel initialization was normal and used relu activation and sigmoid activation function after batch normalization of data to remove the unwanted data. Here epsilon is 1e-4 used to prevent from 0 division errors. The model tested on deep learning gave a satisfying result.

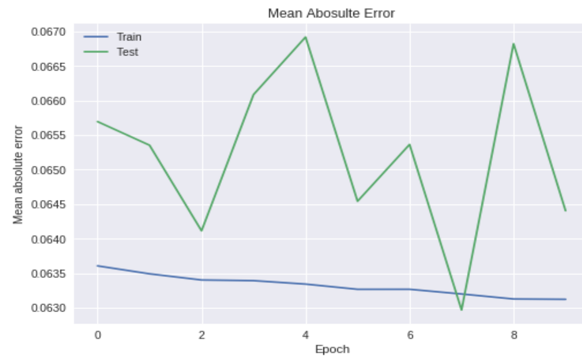

Figure 8: MAE v/s Epoch

This above graph compares the data error rate and iteration rate the green line is the tested data and blue for trained data.

LBGM model that satisfied all the conditions with the best fit for data, among all the other models this model used less memory, it was fast for a large dataset and the best iterations done was 250 with a random state that no other model could do with the training of data. With a speed of 222 seconds which is instantaneous, it predicts the winning percentage of a player in pubg game.

The graph states the parameters used with n estimators of 250 between the mean absolute error which is set to early stop rounds of 10 which means that before it reaches the max value it's self its stops with a leaf node of 200. The overall conclusion states the validation applied to the dataset this model gave the highest MAE with 0.0539.

Figure 9: LGBM MAE estimators

VIII. CONCLUSION

The results obtained from the research are significantly strong to support the strategies in which a player can play in the battlefield with various attributes that contribute towards obtaining the winning percentile. The various comparison attributes to consider for the winning players is depicted in the graph as follows

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 3, Issue 12, ISSN No. 2455-2143, Pages 107-118
Published Online April 2019 in IJEAST (http://www.ijeast.com)

Figure 10: Winning percentile based on No of Kills

Figure 11: Winning percentile based on No of Vehicles destroyed

We have also discovered that the size of the team matters for predicting the winning percentile and in fewer cases the accuracy of the prediction goes high in which the case is where the player would have ended the game before it would have reached the end that gives us a wrong assumption. Further when we test the data records of the players or users playing the game worldwide its more than millions of people we used their records to predict the winner by using few methodologies out if which we have succeeded with all the three models that were successfully implemented with a better run time. LGBM model outstands with better improvement in the model, reduction in memory usage and faster run time with a large amount of dataset with an accuracy of 93% best fit model. Next, with a very less difference deep learning method in neural networks was trained and tested on the data using batch normalization helped in recovering the data loss which had occurred in MLP with a Lower accuracy of 85% and fewer iterations. Further, we can also conclude that a player can also win by hiding in the safe zones in the game(10).

IX. ETHICAL, LEGAL AND SOCIAL ISSUES

The technology information for this particular area consists various ethical(11), legal and social issues many user playing on PC or Xbox have been reduced drastically as they have identified software developed bugs in the PUBG Road map and quality of life issue with the game for example player holding a weapon while moving close to rock under water that must be fixed. The game is poorly optimized on a powerful computation. As far as the legal issue concerns (12) there was a complain file against a pubg corporation that specifies about the copyrights of a similar game known as fortunate.

X. REFERENCE

1. Hodge V, Devlin S, Sephton N, Block F, Drachen A, Cowling P. Win Prediction in Esports: Mixed-Rank Match Prediction in Multi-player Online Battle Arena Games. 2017 Nov 17 [cited 2019 Mar 30]; Available from: http://arxiv.org/abs/1711.06498
2. Wei W, Lu X, Li Y. PUBG: A Guide to Free Chicken Dinner [Internet]. 2018 [cited 2019 Mar 30]. Available from: http://cs229.stanford.edu/proj2018/report/127.pdf
3. Bosch P, Bhulai S. Predicting the winner of NFL-games using Machine and Deep Learning [Internet]. 2018 [cited 2019 Mar 30]. Available from: https://beta.vu.nl/nl/Images/werkstuk-bosch_tcm235-888637.pdf
4. What is LightGBM, How to implement it? How to fine-tune the parameters? [Internet]. [cited 2019 Mar 30]. Available from: https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc
5. Lets Play—A “PUBG” Dataset step by step tutorial – Towards Data Science [Internet]. [cited 2019 Mar 30]. Available from: https://towardsdatascience.com/lets-play-a-pubg-step-by-step-tutorial-1c0b38b322e8
6. Adler J. On using Artificial Neural Network models to predict game outcomes in Dota 2 VIKTOR WIDIN [Internet]. DEGREE PROJECT TECHNOLOGY. 2017 [cited 2019 Mar 30]. Available from: https://pdfs.semanticscholar.org/796c/8a7a655f5bdeab7084baca2fa0b9a4f6a01e.pdf
7. Data Analysis of PlayerUnknown’s Battlegrounds (PUBG)—Introduction & Data Preparation [Internet]. [cited 2019 Mar 30]. Available from: https://medium.com/@aliciali_7397/data-analysis-of-playerunknowns-battlegrounds-pubg-introduction-data-preparation-94da7b64614e
8. RPubs - PUBG Placement Prediction EDA [Internet]. [cited 2019 Mar 30]. Available from: https://rpubs.com/tianyi/pubgeda
9. PUBG Finish Placement Prediction (Kernels Only) | Kaggle [Internet]. [cited 2019 Mar

30]. Available from: https://www.kaggle.com/c/pubg-finish-placement-prediction/data

10. PUBG Data Analysis – Exploring PLAYERUNKNOWN'S BATTLEGROUNDS Statistics through Data Science [Internet]. [cited 2019 Mar 30]. Available from: https://pubganalysis.wordpress.com/
11. The first rule of Playerunknown's Battlegrounds says a lot: stop the racial harassment - The Verge [Internet]. [cited 2019 Mar 31]. Available from: https://www.theverge.com/2017/8/3/16093190/playerunkown-battlegrounds-harassment
12. What We Know And What We Don't About PUBG's Legal Fight With 'Fortnite' In Korea [Internet]. [cited 2019 Mar 31]. Available from: https://www.forbes.com/sites/thomasbaker/2018/06/05/what-we-know-and-what-we-dont-about-pubgs-legal-fight-with-fortnite-in-korea/#31c9d7b23901

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 3, Issue 12, ISSN No. 2455-2143, Pages 107-118
Published Online April 2019 in IJEAST (http://www.ijeast.com)

Appendix

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import gc
sns.set(style = "whitegrid")


from google.colab import files

from sklearn.decomposition import PCA

from sklearn.model_selection import train_test_split
!pip install lightgbm
from sklearn.metrics import mean_absolute_error, r2_score
from lightgbm import LGBMRegressor
import datetime

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor
from sklearn.neural_network import MLPRegressor
from keras import models
from keras import layers
from keras import Sequential
from keras import optimizers
from keras.layers import Dense, Dropout, Input, BatchNormalization
!wget https://raw.githubusercontent.com/Zahlii/colab-tf-utils/master/utils.py
import utils
import os
import keras
from keras.callbacks import ModelCheckpoint
from utils import GDriveSync
from keras.models import load_model
!pip install wordcloud
from wordcloud import WordCloud
```

Figure 12:Import Libraries

```
from sklearn.model_selection import GridSearchCV
import lightgbm as lgb
```

```
Requirement already satisfied: lightgbm in /usr/local/lib/python3.6/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from lightgbm) (1.14.6)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from lightgbm) (1.1.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.6/dist-packages (from lightgbm) (0.1
9.2)
Using TensorFlow backend.
```

Figure 13: use of tensor flow

```
In [0]: train.drop(['match_mean','match_median'], axis=1)
        # data.drop('match_median')
        train.columns

Out[0]: Index(['Id', 'groupId', 'matchId', 'assists', 'boosts', 'damageDealt', 'DBNOs',
               'headshotKills', 'heals', 'killPlace', 'killPoints', 'kills',
               'killStreaks', 'longestKill', 'matchDuration', 'matchType', 'maxPlace',
               'numGroups', 'rankPoints', 'revives', 'rideDistance', 'roadKills',
               'swimDistance', 'teamKills', 'vehicleDestroys', 'walkDistance',
               'weaponsAcquired', 'winPoints', 'winPlacePerc', 'match_mean',
               'match_median'],
              dtype='object')
```

Figure 14: Dropping 0 value data

```
In [0]: #Read training data from csv and remove records with NA values
        train = pd.read_csv('train_V2.csv', header=0, sep=',', quotechar='"')
        train.dropna(inplace=True)
        print(len(train))
        pd.options.display.max_columns = 2000
        train.head()

        4446965
```

Out[0]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPla
0	7f96b2f878858a	4d4b580de459be	a10357fd1a4a91	0	0	0.00	0	0	0	60
1	eef90569b9d03c	684d5656442f9e	aeb375fc57110c	0	0	91.47	0	0	0	57
2	1eaf90ac73de72	6a4a42c3245a74	110163d8bb94ae	1	0	68.00	0	0	0	47
3	4616d365dd2853	a930a9c79cd721	f1f1f4ef412d7e	0	0	32.90	0	0	0	75
4	315c96c26c9aac	de04010b3458dd	6dc8ff871e21e6	0	0	100.00	0	0	0	45

```
In [0]: #Read test data from csv and remove records with NA values
        test = pd.read_csv('test_V2.csv', header=0, sep=',', quotechar='"')
        test.dropna(inplace=True)
        print(len(test))
        test.head()

        1934174
```

Out[0]:

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPla
0	9329eb41e215eb	676b23c24e70d6	45b576ab7daa7f	0	0	51.46	0	0	0	73
1	639bd0dcd7bda8	430933124148dd	42a9a0b906c928	0	4	179.10	0	0	2	11
2	63d5c8ef8dfe91	0b45f5db20ba99	87e7e4477a048e	1	0	23.40	0	0	4	49
3	cf5b81422591d1	b7497dbdc77f4a	1b9a94f1af67f1	0	0	65.52	0	0	0	54
4	ee6a295187ba21	6604ce20a1d230	40754a93016066	0	4	330.20	1	2	1	7

Figure 15: Training and testing data

LightGBM

```
In [0]: def calculate_error(cl,name):
          print(name)
          print('Mean Absolute Error is {:.5f}'.format(mean_absolute_error(y_val, cl.predict(X_val))))
          print('R2 score is {:.2%}'.format(r2_score(y_val, cl.predict(X_val))))

In [0]: # Create parameters to search
        gridParams = {
            'learning_rate': [0.05,0.1,0.3,0.002],
            'n_estimators': [50,250],
            'num_leaves': [6,10,16,200],
            'boosting_type' : ['gbdt','dart','goss','rf'],
            'objective' : ['mae'],
            }


        mdl = LGBMRegressor(boosting_type= 'gbdt',
                  objective = 'mae',
                  n_estimators=250,
                  learning_rate=0.3,
                  num_leaves=200,
                  n_jobs = 3, # Updated from 'nthread'
                  silent = True,
                  max_depth = -1,
                  verbose=2,
                  random_state=212)


        # To view the default model params:
        mdl.get_params().keys()

        # Create the grid
        grid = GridSearchCV(mdl, gridParams,
                            verbose=2,
                            n_jobs=2)
        # Run the grid
        grid.fit(X_train, y_train)

        # Print the best parameters found
        print(grid.best_params_)
        print(grid.best_score_)

        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)
        <ipython-input-20-ba31e491e78a> in <module>()
             25
             26 # Create the grid
        ---> 27 grid = GridSearchCV(mdl, gridParams,
             28                     verbose=2,
             29                     n_jobs=2)
```

Figure 16: LGBM

```
In [0]: mlp = MLPRegressor(activation = 'relu',
                                   max_iter=10000,learning_rate='adaptive',
                                   tol=0.0,warm_start=True,solver='adam', verbose=True)
        mlp.fit(X_train,y_train)
        calculate_error(mlp,"MLP")

/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:32: RuntimeWarning: overflow encountered in r
educe
  return umr_sum(a, axis, dtype, out, keepdims)
Iteration 1, loss = 22.39872526
Iteration 2, loss = 3.38558040
Iteration 3, loss = 2.06050984
Iteration 4, loss = 0.61622816
Iteration 5, loss = 0.62046392
Iteration 6, loss = 0.40559269
Iteration 7, loss = 1.28667318
Iteration 8, loss = 0.34655277
Iteration 9, loss = 0.22156778
Iteration 10, loss = 0.03770630
Iteration 11, loss = 0.02100452
Iteration 12, loss = 0.52963382
Iteration 13, loss = 0.44018969
Iteration 14, loss = 0.03515462
Training loss did not improve more than tol=0.000000 for two consecutive epochs. Stopping.
MLP
Mean Absolute Error is 0.08622
R2 score is -121.94%
```

Figure 17: MLP

```
In [0]: RANDOM_STATE=212
        #train_weights = (1/X_train.teamSize)
        #validation_weights = (1/X_test.teamSize)
        TARGET = 'winplaceperc'
        TRAIN_SIZE = 0.9
        EARLY_STOP_ROUNDS = 10
        time_0 = datetime.datetime.now()

        lgbm = LGBMRegressor(objective='mae', n_estimators=250,
                             learning_rate=0.3, num_leaves=200,
                             n_jobs=-1,  random_state=RANDOM_STATE, verbose=1)

        lgbm.fit(X_train, y_train,
                 eval_set=[(X_val, y_val)],
                 eval_metric='mae', early_stopping_rounds=EARLY_STOP_ROUNDS,
                 verbose=1)

        time_1  = datetime.datetime.now()

        print('Training took {} seconds. Best iteration is {}'.format((time_1 - time_0).seconds, lgbm.best_iterati
        on_))

[1]     valid_0's l1: 0.19286
Training until validation scores don't improve for 10 rounds.
[2]     valid_0's l1: 0.144091
[3]     valid_0's l1: 0.112807
[4]     valid_0's l1: 0.0921878
[5]     valid_0's l1: 0.0795694
[6]     valid_0's l1: 0.0716605
[7]     valid_0's l1: 0.0662104
[8]     valid_0's l1: 0.062826
[9]     valid_0's l1: 0.0609746
[10]    valid_0's l1: 0.0597035
[11]    valid_0's l1: 0.0588803
[12]    valid_0's l1: 0.0582907
[13]    valid_0's l1: 0.057764
[14]    valid_0's l1: 0.0570726
[15]    valid_0's l1: 0.0568547
[16]    valid_0's l1: 0.0566102
[17]    valid_0's l1: 0.0563797
[18]    valid_0's l1: 0.0562149
[19]    valid_0's l1: 0.0560829
[20]    valid_0's l1: 0.0560049
[21]    valid_0's l1: 0.0558181
[22]    valid_0's l1: 0.0557295
[23]    valid_0's l1: 0.0555972
[24]    valid_0's l1: 0.0555206
[25]    valid_0's l1: 0.0554388
[26]    valid_0's l1: 0.0553767
[27]    valid_0's l1: 0.0553458
[28]    valid_0's l1: 0.0553195
[29]    valid_0's l1: 0.0552607
[30]    valid_0's l1: 0.0552483
[31]    valid_0's l1: 0.0551807
[32]    valid_0's l1: 0.0551564
[33]    valid_0's l1: 0.0551277
[34]    valid_0's l1: 0.0551011
[35]    valid_0's l1: 0.0550945
[36]    valid_0's l1: 0.055037
```

Figure 18: Random state iteration goes on till 250

```
[214]   valid_0's l1: 0.0541534
[215]   valid_0's l1: 0.0541472
[216]   valid_0's l1: 0.0541403
[217]   valid_0's l1: 0.0541265
[218]   valid_0's l1: 0.0541252
[219]   valid_0's l1: 0.054122
[220]   valid_0's l1: 0.0541219
[221]   valid_0's l1: 0.0541179
[222]   valid_0's l1: 0.0541077
[223]   valid_0's l1: 0.0541066
[224]   valid_0's l1: 0.0541025
[225]   valid_0's l1: 0.0540998
[226]   valid_0's l1: 0.0540972
[227]   valid_0's l1: 0.0540788
[228]   valid_0's l1: 0.0540558
[229]   valid_0's l1: 0.054053
[230]   valid_0's l1: 0.054046
[231]   valid_0's l1: 0.0540425
[232]   valid_0's l1: 0.0540296
[233]   valid_0's l1: 0.0540207
[234]   valid_0's l1: 0.0540146
[235]   valid_0's l1: 0.0540144
[236]   valid_0's l1: 0.0540142
[237]   valid_0's l1: 0.0540115
[238]   valid_0's l1: 0.0540065
[239]   valid_0's l1: 0.0539957
[240]   valid_0's l1: 0.0539954
[241]   valid_0's l1: 0.0539809
[242]   valid_0's l1: 0.05398
[243]   valid_0's l1: 0.0539799
[244]   valid_0's l1: 0.0539667
[245]   valid_0's l1: 0.0539563
[246]   valid_0's l1: 0.0539543
[247]   valid_0's l1: 0.053954
[248]   valid_0's l1: 0.0539532
[249]   valid_0's l1: 0.0539427
[250]   valid_0's l1: 0.053942
Did not meet early stopping. Best iteration is:
[250]   valid_0's l1: 0.053942
Training took 222 seconds. Best iteration is 250
```

Figure 19: Validation of LGBM

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 3, Issue 12, ISSN No. 2455-2143, Pages 107-118
Published Online April 2019 in IJEAST (http://www.ijeast.com)

Deep Learning

```
In [0]: filepath="weights-improvement-{epoch:02d}-{val_loss:.2f}.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

def compare(best, new):
  return best.losses['val_loss'] > new.losses['val_loss']

def path(new):
  if new.losses['val_loss'] < 0.1:
    return 'pubg_%s.h5' % new.losses['val_loss']

cb = [
      checkpoint,
      utils.GDriveCheckpointer(compare,path),
      keras.callbacks.TensorBoard(log_dir=os.path.join(utils.LOG_DIR,'PUBG'))
]
```

```
--2018-12-02 20:25:38--  https://raw.githubusercontent.com/Zahlii/colab-tf-utils/master/utils.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.0.133, 151.101.64.133, 151.101.1
28.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.0.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6935 (6.8K) [text/plain]
Saving to: 'utils.py.1'

utils.py.1          100%[===================>]   6.77K  --.-KB/s    in 0s

2018-12-02 20:25:38 (85.6 MB/s) - 'utils.py.1' saved [6935/6935]
```

```
In [0]: def build_model():
  model = Sequential()
  model.add(Dense(512, kernel_initializer='he_normal', input_dim=X_train.shape[1], activation='relu'))
  model.add(BatchNormalization())
  model.add(Dropout(0.1))
  model.add(Dense(256, kernel_initializer='he_normal', activation='relu'))
  model.add(BatchNormalization())
  model.add(Dropout(0.1))
  model.add(Dense(128, kernel_initializer='he_normal', activation='relu'))
  model.add(BatchNormalization())
  model.add(Dropout(0.1))
  model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
  optim = optimizers.Adam(lr=0.01, epsilon=1e-8, decay=1e-4, amsgrad=False)
  model.compile(optimizer=optim, loss='mse', metrics=['mae'])
  #model.summary
  #history = model.fit(X_train, y_train, epochs=70,batch_size=100000)
  return model
```

```
In [0]: downloader=GDriveSync()
filename='pubg_0.007696801999306263.h5'
drive_file_path=downloader.find_items(filename)[0]
downloader.download_file_to_folder(drive_file_path,filename)
```

```
Downloading file pubg_0.007696801999306263.h5 to pubg_0.007696801999306263.h5: 100%|██████████| 100.0/100
[00:00<00:00, 317.21it/s]
```

Figure 20: Deep Learning

International Journal of Engineering Applied Sciences and Technology, 2019
Vol. 3, Issue 12, ISSN No. 2455-2143, Pages 107-118
Published Online April 2019 in IJEAST (http://www.ijeast.com)

```
In [0]: deep.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 512)               22528
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_6 (Batch (None, 128)               512
_________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 129
=================================================================
Total params: 190,465
Trainable params: 188,673
Non-trainable params: 1,792
_________________________________________________________________

In [0]: deep.load_weights("weights-improvement-06-0.01.hdf5")

In [0]: history = deep.fit(X_train, y_train,
                  validation_data=(X_val, y_val),
                  initial_epoch=20,
                  epochs=30,
                  callbacks=cb,
                  verbose=1)

        calculate_error(deep,"Deep")

Train on 3112875 samples, validate on 1334090 samples
Epoch 21/30
3112875/3112875 [==============================] - 1500s 482us/step - loss: 0.0075 - mean_absolute_error:
0.0636 - val_loss: 0.0085 - val_mean_absolute_error: 0.0657

Epoch 00021: val_loss did not improve from 0.00811
No improvement.
Epoch 22/30
3112875/3112875 [==============================] - 1441s 463us/step - loss: 0.0075 - mean_absolute_error:
0.0635 - val_loss: 0.0083 - val_mean_absolute_error: 0.0654

Epoch 00022: val_loss did not improve from 0.00811
No improvement.
```

Figure 21: Deep Learning Architecture