PUBG Player Placement Prediction

Author(s)
Rachit Shah : rshah25
Sourabh Sandanshi : ssandan
Udit Misra : umisra

1 Background And Introduction

Player Unknown Battle Ground's (PUBG) is an online Multiplayer battle royale game developed by PUBG Corporation. It is a player versus player action game in which no more than 100 player fight in a battle royale. Players can choose to enter the match solo, duo or with a small team up to 4 people. However, there are events and custom games which allow team sizes more than 4. We will address this discrepancy during feature engineering. The last person or the last team alive wins the match. The player has to knock down their opponents so as to survive till the last. During the game, players search buildings, ghost towns to find weapons, vehicles, armor, and other equipment.

The data is part of a Kaggle competition which has scraped 65000 games worth of data using an API. The goal is to predict the win percentage of the player based on 28 given features. For example, in a solo game of 100 players if a player got rank of 80, then its winPlacePerc will be (100-80)/(100-1)=0.204 using the formula winPlacePerc = (maxPlace-winPlace)/(maxPlace-1). The problem is then essentially a regression problem to predict the winPlacePerc of the player between [0,1].

These types of problems are solved by doing extensive exploratory data analysis and feature engineering before applying a regression model. We gained insights in the features and added new relevant features by exploring the game and the semantics of the features. We were provided with the Training Dataset so we divided it into Training and Validation Dataset in 70:30 ratio respectively.

We first used the LGBM (Light Gradient Boosting Machine) (5) Regressor to train our training dataset after doing exploratory analysis on the data. We compared 10 other models namely Neural Network, Multiple Linear Regression, Decision tree, etc. with the same dataset. We reported the accuracy of each model that we got on the validation data.

1.1 Objective

To predict the final ranking and placement from the game statistics and the initial player ratings.

The overview of the steps involved in the project are:
1. Data Cleaning and Exploratory Data Analysis
2. Visualizing the data and feature engineering
3. Training Models on Training data and then drawing comparison between them using validation data
4. Choosing the best model and fitting the Test data on it
5. Reporting the Accuracy of the final chosen model.

2 Method

The main objective is to predict the ranking of the player from the initial player ratings and statistics available in data. So first we are doing exploratory data analysis in which we focus primarily towards getting to know about how each attribute contribute towards the prediction. We use the help of seaborn package in python to plot various kind of graphs between attributes to get data insights. We use correlation to find out the attributes which are more related to the target attribute. We also form new attributes from the existing attributes which would give much more insight for prediction.

$$cor = \sum_{i=0}^{n} \frac{(x - x_i)(y - y_i)}{n - 1}$$

In our project we have used 11 different algorithms for our player placement prediction model. In those 11 algorithms, 4 were Regression models, 4 were ensemble models (out of which 3 were boosting models), 2 were Neural Network and a Decision Tree model. We have also used Grid Search for parameter tuning in the case of Boosting model which we will discuss further. The 4 Regression Models that we have used are:
1. Multiple Linear Regression
2. Ridge Regression
3. Lasso Regression
4. Elastic Net Regression

In Multiple Linear Regression (6) we tend to learn about multiple uncorrelated independent variable with the dependent variable or the criterion. In our project there were 28 dependent variables and 1 independent variable winPlacePerc(which gives the winning percentile of particular player). The equation is given as:

$$Y = \beta_0 + \beta_1.x_1 + \beta_2.x_2 + \beta_3.x_3 + ...$$

The other 3 Regression model that are used, uses the technique of Regularization (7) to reduce the magnitude of coefficients according to the effect they have on the target variable. They are also called as shrinkage models because they reduce the parameter coefficients, so as to prevent multicollinearity. The cost function of Ridge Regression is given as:

$$\sum_i (Y - x_i.\beta)^2 + \lambda \sum_j \beta_j^2$$

where lambda is the coefficient denoting the penalty term. higher the value of lambda more the coefficient decrease happens. Lasso Regression is also similar to the Ridge regression, but the difference here is that it also regularizes the coefficients in the way that it makes the magnitude of some coefficients as 0 according to the influence it has. This has the effect of doing automatic feature selection on the data and increasing the interpretability of the data. While ridge regression never makes the coefficient magnitude as zero. So Lasso regression performs well in the case where there are many attributes. Its equation is given as:

$$\sum_i (Y - x_i.\beta)^2 + \lambda \sum_j \beta_j$$

Now Elastic Net (8) is a combination of both Ridge and Lasso Regression. Both penalty terms of Ridge and Lasso are added to the cost function and is shown as following:

$$\sum_i (Y - x_i.\beta)^2 + \lambda_1 \sum_j \beta_j + \lambda_2 \sum_j \beta_j^2$$

Other than Regression we have used 3 Boosting (10) Models namely AdaBoost, Gradient Boosting and Light Gradient Boosting Machine (LGBM). Firstly Boosting is an ensemble technique to create a strong classifier from a combination of weak classifiers. This occurs by creating a model from the train data. Then creating a second model that attempts to correct the errors made in the first model. Models are added till the training set is predicted accurately. So Adaboost retrains the algorithm iteratively by choosing the training set based on accuracy of previous training set. Each training data is given a particular weight according to the accuracy achieved and the algorithm runs again.

Gradient Boosting Model (9) is a supervised technique where the main objective is to minimize the Loss function i.e., the Mean Squared Error(MSE). It uses Gradient Descent to minimize the loss and uses a learning rate alpha to find the updated value of predicted values. LGBM is a type of gradient boosting algorithm which uses decision trees as its framework. The most inherent part of this model is that it can handle a very large size of data and takes very low memory to run. Due to its efficiency, accuracy and interpretability it has started to be widely used for very large data sets. Similarly it is not advised to use this model with small data set as it is prone to overfitting.

LGBM is an ensemble model of decision tree which learns by fitting negative gradients which are also known as residual errors. Suppose we have n identical and independent distributed (x1,x2,...xn) with dimension s in a Gradient Space. In every iteration of gradient boosting, the residuals of loss function with respect to output of the model are denoted by (g1,g2,...gn). So the model splits each node at the point where the Information Gain is the maximum.

The next model that we explored was Decision Trees. Decision Tree (11) works on the concept of dividing the tree based on the Information Gain attained from the attributes of the given dataset. Information Gain is based on the concept of entropy and information content. Entropy is the measure of randomness of an attribute with respect to target attribute. It is defined as:

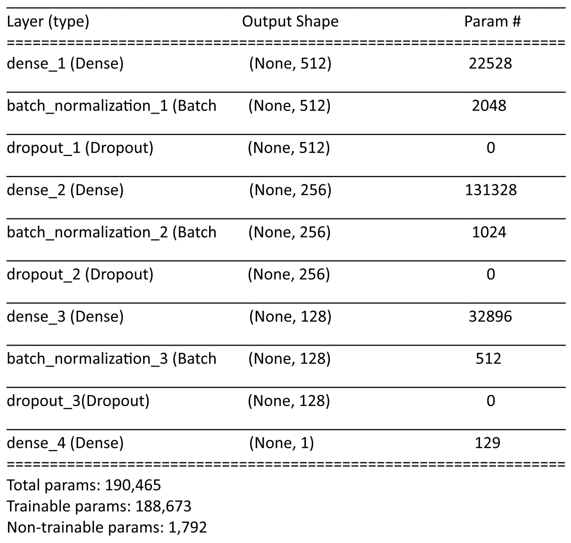
$$H(t) = -\sum_{i}^{j} p_i . \log_2(p_i)$$

where H(t) is the entropy of the atttribute. p is the probability of each class present in the child node. Now Information gain of the attribute is defined as:

$$IG(T/a) = H(T) - H(T/a)$$

Here H(t) represents entropy of the parent while H(T/a) is the weighted sum of entropy of all the children in the attribute. We also evaluated our result with the Random Forest Algorithm. Random forest (12) works similar to Decision tree but the catch here is that it makes multiple decision trees and merges them together to get a more accurate and stable predictions.

We also used Neural Network. Neural network is composed of three layers, namely input layer, hidden layer and the output layer. The activation function is used to find the weighted input of each unit in the the layers. Hyper parameter tuning is done to achieve model optimization. We used Keras library to help us train the best model for our data. We extended one more step forward by using Deep learning for our dataset. Deep learning is also a type of Neural Network consisting of hierarchy of layers in which each layer further transforms the data into more abstract representation. In deep learning more the layers, higher will be the features learned by the model. The output layer combines all these features and make a prediction. This way it differs from Neural Network. While simple Neural Network uses only one hidden layer which is not suitable for learning complex features, deep learning uses multiple hidden layers to learn these complex features which might risk overfitting. To avoid this, we use Batch Normalization which normalizes and scales the output of the activation at each layer to avoid values going to the extremes. This allows each layer to learn differently from other layers, and hence avoids overfitting. It also has the effect of reducing the training time. We also use dropout at each hidden layer to avoid overfitting by ignoring a set amount of neurons output at each layer. Hence, deep learning can be very expensive and requires massive dataset to train itself on. The architecture of the model can be seen in figure 1.



```
Layer (type)                    Output Shape           Param #
==============================================================
dense_1 (Dense)                 (None, 512)            22528

batch_normalization_1 (Batch    (None, 512)            2048

dropout_1 (Dropout)             (None, 512)            0

dense_2 (Dense)                 (None, 256)            131328

batch_normalization_2 (Batch    (None, 256)            1024

dropout_2 (Dropout)             (None, 256)            0

dense_3 (Dense)                 (None, 128)            32896

batch_normalization_3 (Batch    (None, 128)            512

dropout_3(Dropout)              (None, 128)            0

dense_4 (Dense)                 (None, 1)              129
==============================================================
Total params: 190,465
Trainable params: 188,673
Non-trainable params: 1,792
```

Figure 1: Architecture of Deep Learning Model

3 Experiment Setup

3.1 Dataset Description

We are provided with training and test datasets consisting of 4.45million and 1.93million records respectively, each having 28 features. The data we got was from the Kaggle Competition PUBG Finish Placement Prediction (1). The attributes of the dataset is as follows:

ATTRIBUTES	DESCRIPTION	DATATYPE
Id	Player's ID	Category
groupId	ID to identify group in match	Category
DBNOs	Enemy Players Knocked	uint8
Assists	Enemy Player Damaged that were killed By Teammates	uint8
Boosts	Total Boosts item used	uint8
DamageDealt	Total Damage Dealt	float16
HeadShotKills	Total headshots taken by the player	uint8
heals	No. of healing items used	uint8
killPlace	Ranking in match of number of enemy players killed	uint8
killPoints	Kills-based external ranking of player	uint8
killStreaks	Maximum enemy killed in a short duration	uint8
kills	No. of enemy player killed	uint8
longestKill	Longest distance between player and player killed	float16
numGroups	Total groups in a match	uint8
matchDuration	Duration of match in seconds	uint8
matchId	The Match ID to trace match	Category
matchType	Types of match like solo, duo etc.	Category
maxPlace	The max rank a team/player got in that match	uint8
rankPoints	Elo-like ranking of player	uint8
revives	No. of times player revived teammates	uint8
rideDistance	Total distance covered in the form of ride	float16
roadKills	No. of kills while in a vehicle	uint8
swimDistance	Total distance in form of swimming	float16
teamKills	Total teammates killed in a game	uint8
vehicleDestroys	Total vehicle destroyed	uint8
walkDistance	Total distance walked in a game	float16
weaponsAcquired	Total weapons aquired throughout the game	uint8
winPoints	Win-based external ranking of player.(Elo-like)	uint8
winPlacePerc	The winning percentage of a player	float16

3.2 Data Insights

In order to find out patterns in the data and determine what features may be useful, We decided to perform Exploratory Data Analysis on the data. This would also help us create features which better depict the skill level of the players. We start with the distribution of winning percentile across all games. From Figure 2, we see that our percentile distribution isn't exactly as uniform as we expected. There seems to be more cases occurring at low percentiles. This indicates that overall, we have more losers than winners.

The higher counts at the percentile extremes is to be expected. There is always some someone getting the percentiles scores of zero and 1, the rest of the scores varying across different matches. Let's consider the average winning percentile per match and look at the distribution. As mentioned earlier, we should expect a normal distribution with mean 0.5. But, what we find is that the mean distribution is somewhat right-tailed.We can infer that some games have lower average percentiles. The reason for this can be players leaving the game before it finishes. This leads to more people and teams having lower ranks.

Next, we have a look at the correlation. From the plot in Figure 4, we can see that some features are highly correlated with each other and thus can either be combined or dropped. We also can extract attributes correlated to our target variable, win place percentage. A Principal Component Analysis

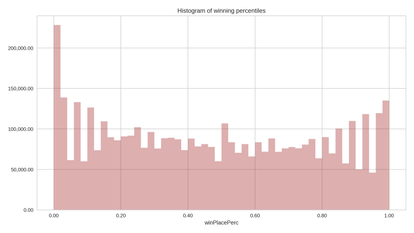


Figure 2: Histogram of Winning Percentiles

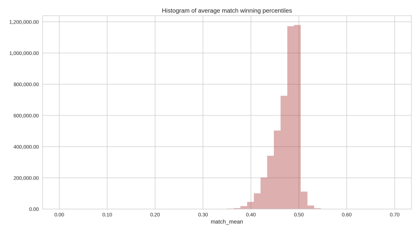


Figure 3: Histogram of Average match Winning Percentiles

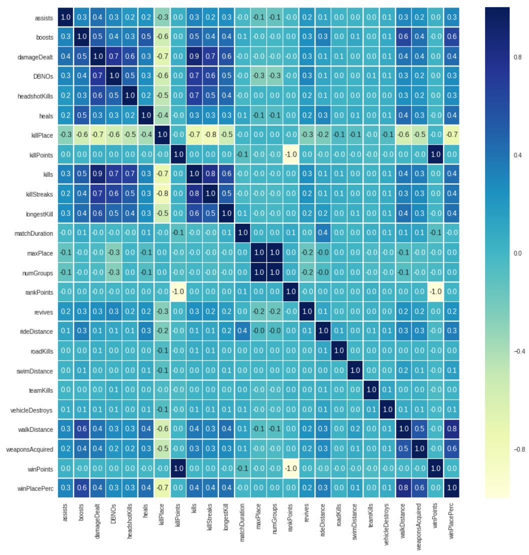


Figure 4: Correlation matrix

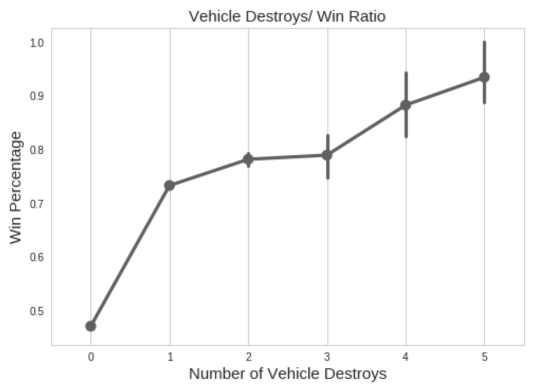


Figure 5: Vehicles Destroyed vs Win Place

corroborates this hypothesis. We also see that attributes like No. of vehicles destroyed, which have almost zero correlation with the target variable, actually does impact it. We also plot the number of Kills against the win place prediction to see a general trend of proportionality which stagnates after a certain level of kills.

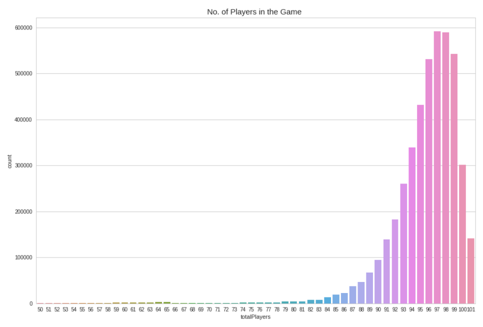


Figure 6: Number of Players

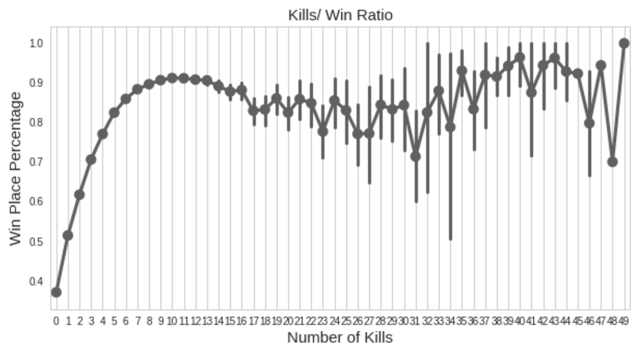


Figure 7: Kills vs Win Place

Next, We have a look at the total number of players in a game. From Figure 6, We observe that the number of players vary in each game. When there are 100 players in the game it might be easier to find and kill someone, than when there are 70 players. Hence, we need to normalize certain attributes which signify a player's skill level so that we can compare them.

3.3 Feature Engineering

We analyzed the data and engineered more features from the already existing features which we believe could help for a better fit for our models. We create a **'totalPlayers'** feature by groupby on 'matchId' giving total players who played in that match. Useful for normalizing data for diff. no. of players. Similarly, we also groupby on 'groupId' to get actual **'teamSize'** feature of that player. Since it is not necessary that a squad game will all have teams of 4. Next we create a **'normMatchType'** feature for categorizing in solo,duo,squad,other values. Though there is a matchType column, it has many discrepancies as custom matches / events are also in the data. We make a new feature based on actual teamSize and matchType. Since distance travelled has a high correlation with the 'winPlacePerc', we add a new feature **'totalDistance'** which is a sum over the 'swim','ride' and 'walk' distances of the player. Next, we create **'maxPossibleKills'** which calculates the max no. of kills the team can perform. For example, in a match of 100 players, if the player is in a team of 4 players, then maximum players they can kill is 100-4=96. **'itemsUsed'** - combination of all the items used like boosts, heals and weapons, since number of items used might indicate the player stayed longer in the game and hence might contribute to winPlace. We also get the rate / frequency of 'kills','damageDealt' and 'itemsUsed' by dividing them by 'totalDistance' to get **'itemsPerDistance'**, **'killsPerDistance'** and **'damageDealtPerDistance'**. Next we find team statistics like **'maxTeamKills'** - the maximum kills a teammate of the player has performed, **'totalTeamKills'** - total Kills the player's team has performed combined, **'itemsUsedPerTeam'** - total items the team has used combined, **'percTeamKills'** - ratio of kills the team performed combined by max possible kills, **'meanTeamKillPlace'** - mean of all teammates rank for killing which essentially finds the overall performance of the whole team in general. Similarly, we also find **'percKill'** - the ratio of kills performed by player divided by max possible kills and **'headshotKillRate'** which gives the skill of the player to kill a player by a single headshot. You can see the 15 new features in the word cloud shown in figure 8. The size of each new feature in the word cloud is ordered according to its correlation with the output column "winPlacePerc".

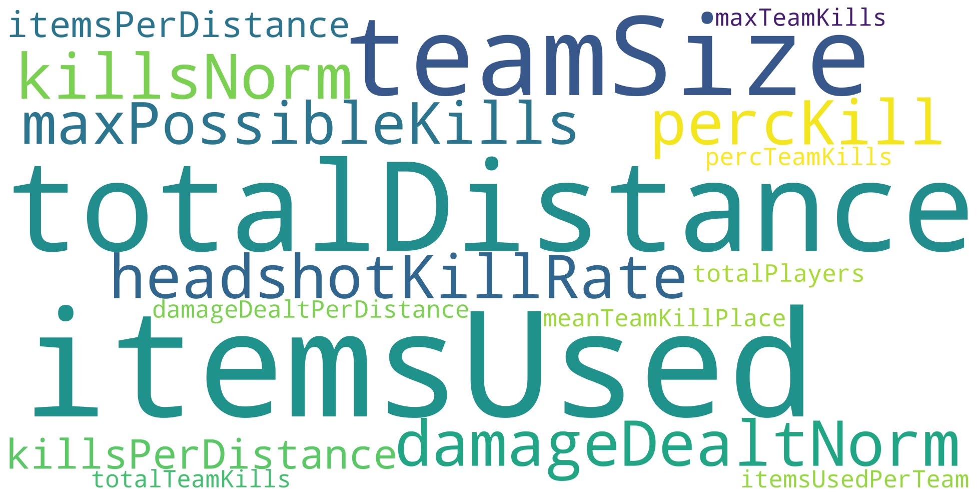


Figure 8: Word Cloud of New Features

4 Results

For our training dataset we tried fitting onto 11 different models as shown in table 2. The MAE values and R2 values that we got were from the validation data that we applied on each model. We found that boosting models outperformed Regression and Neural Network. Linear Gradient Boosting machine,

Table 2: Model Results

Model	MAE	R^2	Parameters
Linear Multiple Regression	0.08755	84.83%	default
Ridge Regression	0.08755	84.83%	default
LASSO Regression	0.12084	74.42%	default
Elastic Net Regression	0.113	77.06%	default
AdaBoost	0.0974	82.79%	learning rate = 0.8
Gradient Boosting	0.05861	93.01%	learning rate = 0.8
Random Forest	0.05751	93.00%	n_estimators = 0.8
Decision Tree	0.07711	86.92%	default
LGBM	0.05392	93.70%	learning rate = 0.3, n_estimators = 250, num_leaves = 200, objective = mae
Simple MLP	0.08630	85.42%	learning rate = adaptive, epochs = 23,
Deep Learning	0.0623	90.71%	hidden layers = 4, Optimizer -Adam, learning rate = 0.01, epsilon = 1e-8, decay = 1e-4, epochs = 23, Activation - ReLu for hidden and sigmoid for output layer

which is an extremely fast and efficient gradient boosting algorithm based on decision trees gave the best result out of the boosting models of 93.01% Mean Absolute Error. The parameters that we used for LGBMRegressor were, [learningrate:0.3, nestimators:250, numleaves:200, objective:'mae', earlystoppingrounds:10] and which are the best parameters we found by applying Grid Search Cross Validation with 3 folds and parameter grid: 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'],] The training took 624 seconds with the **Mean Absolute Error = 0.05437** and **R2 Score=93.22%**. The MAE vs epoch graph and the feature scores is shown below in figure 9 and figure 10 respectively.

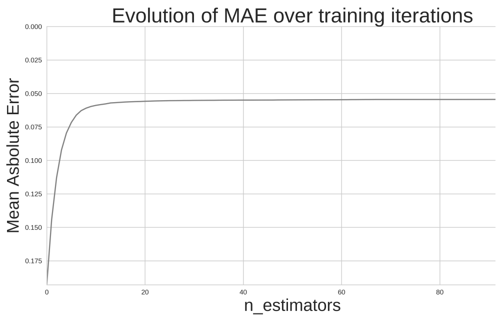


Figure 9: MAE vs Training Iterations

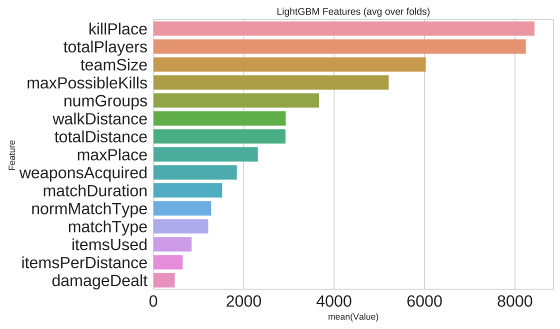


Figure 10: LGBM Top 15 Feature Scores

The MAE Score without our feature engineering was 0.0607 proving the importance of our new features which is further attested by the feature scores of LGBM. Out of the top 15 important features determined by LGBM, 7 of them are the new features we added. We then ran our final model on the test dataset and uploaded the predictions to the kaggle competition and the final Test MAE returned was 0.0539.

5 Conclusion

For this dataset, we have observed that Boosting models like LGBM, AdaBoost and Random Forest perform better than traditional Multiple Regression models and Neural Networks. This is perhaps because boosting algorithms are very robust to noise and outliers. They optimize the model via gradient descent using generic differentiable loss functions and also has various benefits like automatic null-handling, in-built feature selection and scale invariance. Since our data is not necessarily linearly separable, multiple linear regression models performed worse. There are multiple features with non-linear relationships between them, which boosting algorithms can recognize better using the ensemble of multiple weak learners and tuning the parameters to control model complexity (avoid overfitting). Also, while neural networks are good for learning complex features at a higher level by doing automatic feature engineering at each layer which would be helpful for data involving image, text or audio where we don't have many features (just the data itself), our problem has tabular data with multiple features already given. With tabular data, often the relationship between features is shallow and there really is no need to explore complex features. As a result, we find that the boosting algorithms can learn non-linear relationships better than linear regression and work better on tabular data than neural networks. Along with providing better results, they are faster to train as compared to neural or deep neural networks. This makes them a must use approach for regression problems of this nature. We also understood the process and importance of hyperparameter optimization and achieving it through grid search.

We also learned the importance of Exploratory Data Analysis and Feature engineering in Machine Learning. EDA helps us find the trends in the data and understand what type of pre-processing is needed. We can also use these insights to add new features to enhance the robustness of our model.

The deep learning model used could have performed better if it was given more time to train. We could also have improved it by hyper tuning other relevant parameters to achieve better results. Even for the best performing model LGBM, a randomized grid search could have given better results than providing grid parameters.

We restricted ourselves to the kaggle problem statement of predicting the Win place percentile for all the players in the dataset. We could have predicted other statistics like No. of kills, Walk distance, headshots given previous data. This prediction can be done for the whole dataset or at a finer scale such as per team or per match.

We have uploaded our project to this GitHub repository: https://github.com/rachit-shah/pubg-player-placement-prediction

References

[1] *"PUBG Finish Placement Prediction (Kernels Only) | Kaggle"*, Kaggle.com, 2018. [Online]. Available: https://www.kaggle.com/c/pubg-finish-placement-prediction/data. [Accessed: 18- Nov- 2018].

[2] Hodge, V.J., Devlin, S, Sephton, N., Block, F., Drachen, A. & Cowling, P.I. (2017) *Win Prediction in Esports: Mixed-Rank Match Prediction in Multi-player Online Battle Arena Games.* (1711.06498 ed.) arXiv.

[3] Ding, Y. (2018).*Research on operational model of PUBG. MATEC Web Of Conferences*, 173, 03062.

[4] Wang, N., Li, L., Xiao, L., Yang, G., & Zhou, Y. (2018). *Outcome prediction of DOTA2 using machine learning methods.*, *International Conference on Mathematics and Artificial Intelligence*, pp. 61-67

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu. *"LightGBM: A Highly Efficient Gradient Boosting Decision Tree"*. Advances in Neural Information Processing Systems 30 (NIPS 2017), pp. 3149-3157.

[6] Preacher, K. J., Curran, P. J., Bauer, D. J. (2006). *Computational tools for probing interactions in multiple linear regression, multilevel modeling, and latent curve analysis.* Journal of educational and behavioral statistics, 31(4), 437-448.

[7] Tibshirani, R. (1996). *Regression shrinkage and selection via the lasso.* Journal of the Royal Statistical Society. Series B (Methodological), 267-288.

[8] Zou, H., Hastie, T. (2005). *Regularization and variable selection via the elastic net.* Journal of the Royal Statistical Society: Series B (Statistical Methodology), 67(2), 301-320.

[9] Friedman, J. H. (2001). *Greedy function approximation: a gradient boosting machine.* Annals of statistics, 1189-1232.

[10] Schapire, R. E. (2003). *The boosting approach to machine learning: An overview. In Nonlinear estimation and classification (pp. 149-171).* Springer, New York, NY.

[11] Quinlan, J. R. (1986). *Induction of decision trees. Machine learning, 1(1), 81-106.*

[12] Liaw, A., Wiener, M. (2002). *Classification and regression by randomForest.* R news, 2(3), 18-22.