

PUBG Finish Placement Prediction

Abinaya Manimaran

EE 660 Project - December 6, 2018

Abstract

This project is taken up from Kaggle, an online community of data scientists and machine learners. The goal of the project is to predict PUBG player's win place percentage based on the player's statistics. Millions of players data is used for building a regression model. Players data within a group in a match is combined to reduce the dataset size. Some insightful derived features are added to improve the predictions. The best model using Light Gradient Method and Kaggle leaderboard score is reported.

1 Introduction

1.1 Problem Type, Statement and Goals

PlayerUnknown's Battlegrounds (PUBG) is one of the current trending online multiplayer battle Royale game. It is a player Vs player shooter game, in which up to one hundred players can begin to fight in a battle. Every few minutes, the game area starts to shrink while each player tries to eliminate other players in the battle field. The game ends when there is one last player and that her/she is declared as the winner. At the end of game, each player is given a rank based on how many other teams are still alive when the player got eliminated. The aim of this project is to predict player's finishing placement (Win Place Percentage) based on their stats given.

This is a **Regression** type problem since the target can vary on a scale between 0 (last place in a match) to 1 (first place). There are different strategies that a player can take up and follow. A player can try being aggressive and improve or a player be sneaky when it counts not wasting time or a player can learn pick goods in the game and jump to destination. The model has to understand winning strategy of the player and predict the player's final placement rank value. Some reasons that the problem in hand is non-trivial are:

- It is difficult to model each player's strategy to win the game, since it depends on the mood of the player while he is playing the game. He can use a unique strategy or a combination of many strategies. Also, a player can play in a way that the model has

not seen before at all. Hence some domain knowledge has to be applied on the given features to come up with the best model.

- The team at PUBG has made 65,000 games worth data available for public in Kaggle. This adds up to 6.38 million players data spanning both train and test. The size of data is really huge and it becomes really hard to handle them while training a regression model.
- Dimensionality of feature space is very less when compared to the number of data points. Most of the times, the model tends to underfit with less information for every player.

1.2 Prior and Related Work - None

1.3 Overview of Approach

The dataset was initially analyzed using pre-train data. Exploratory data analysis proved that the dataset size is huge when compared to the dimensionality. Hence, all the works were focussed on reducing the dataset size and increasing the dimensionality. The best model was found by grouping the players data with respect to *match id* and *group id*. Additional features derived from base features helped in improving the model. R^2 score and Mean Absolute Error (MAE) were used to compare different approaches and different Regression models. Results from the best model were uploaded to Kaggle leaderboard finally.

2 Implementation

2.1 Data Set

As mentioned before in section 1.1, there are a total of 6.38 million players data made available by Kaggle. This is already given separated as train and test data. The output *Win Place Percentage* is provided for train data, but not for test data. Since this is a competition, we need to predict the *Win Place Percentage* for the test data and submit to the Kaggle leaderboard. A brief description of the dataset is given in table 1

Number of training data points	4.45 million
Number of testing data points	1.93 million
Number of input variables (features)	28
Number of output variables (Regression target)	1

Table 1: Dataset description

A more detailed description of the features is given in table 2. Features are categorized either categorical or numerical. If a feature is categorical, the cardinality of train and test is given separately. If the features are numerical, the range of the values is shown.

S.No	Name	Type	Description	Cardinality / Range
1	Id	categorical	unique for every player	4.45m, 1.93m
2	groupId	categorical	unique for every group in a match	2.02m, 0.88 m
3	matchId	categorical	ID to identify match	47964, 20556
4	assists	numerical	Number of enemy players this player damaged that were killed by teammates	0 to 27
5	boosts	numerical	Number of boost items used	0 to 33
6	damage dealt	numerical	Total damage dealt	0 to 0.0 6616
7	DBNOs	numerical	Number of enemies knocked	0 to 59
8	head shot kills	numerical	Number of enemies killed with headshots	0 to 64
9	heals	numerical	Number of healing items used	0 to 80
10	kill place	numerical	Ranking in match of number of enemy players killed	1 to 100
11	kill points	numerical	Kills-based external ranking of player	0 to 2174
12	kills	numerical	Number of enemies killed	0 to 72
13	kill streaks	numerical	Max number of enemy players killed in a short amount of time	0 to 20
14	longest kill	numerical	Longest distance between player and player killed at time of death	0 to 1094
15	match duration	numerical	Duration of match in seconds	74 to 2237
16	match type	categorical	String identifying the game mode	16, 16
17	max place	numerical	Worst placement we have data for in the match	2 to 100
18	num groups	numerical	Number of groups we have data for in the match	1 to 100
19	rank points	numerical	Elo-like ranking of player	-1 to 5910
20	revives	numerical	Number of times this player revived teammates	0 to 39
21	ride distance	numerical	Total distance traveled in vehicles (m)	0 to 40710
22	road kills	numerical	Number of kills while in a vehicle	0 to 18
23	swim distance	numerical	Total distance traveled by swimming (m)	0 to 3823
24	team kills	numerical	Number of times this player killed a teammate	0 to 12
25	vehicle destroys	numerical	Number of vehicles destroyed	0 to 5
26	walk distance	numerical	Total distance traveled on foot (m)	0 to 25780
27	weapons acquired	numerical	Number of weapons picked up	0 to 236
28	win points	numerical	Win-based external ranking of player	0 to 2013
29	win place perc	numerical	Target for prediction	0 to 1

Table 2: Feature and target description

2.2 Preprocessing, Feature Extraction, Dimensionality Adjustment

Some of the features were highly correlated with the target variable, while some were not. The number of data points were huge when compared to the feature dimension. All the analysis related to preprocessing, feature extraction and dimensionality adjustment are described in the following sub sections.

2.2.1 Exploratory Data Analysis and Preprocessing

I present some initial exploratory data analysis on the training dataset. For this purpose, I set aside 5% of the train dataset. All the analysis below were obtained from the pre-training dataset, which was omitted later and was not included for the actual training process. More information in section 2.3.

Figure 1 shows correlation matrix color coded between features. The last variable considered is the target. So last row and last column indicates the correlation of all features with the target *Win Place Percentage*. Other values in the matrix, shows the correlation of one feature with other features. The matrix is symmetric for obvious reasons.

- Some features are highly positively correlated with the target like *boosts*, *damage dealt*, *heals*, *kills*, *kill streaks*, *longest kill*, *walk distance* and *weapons acquired*. Hence these

features are of high importance to contribute to the regression target. Figure 2 (a) to (g) visualizes the positive correlation.

- Some features have zero correlation, which might indicate that they do not have importance in regression.
- *Kill place* is the only feature highly negatively correlated. Though it is negatively correlated, this is also an important feature since there is a trend in change of *kill place* with target. Figure 2 (h) visualizes the negative correlation.

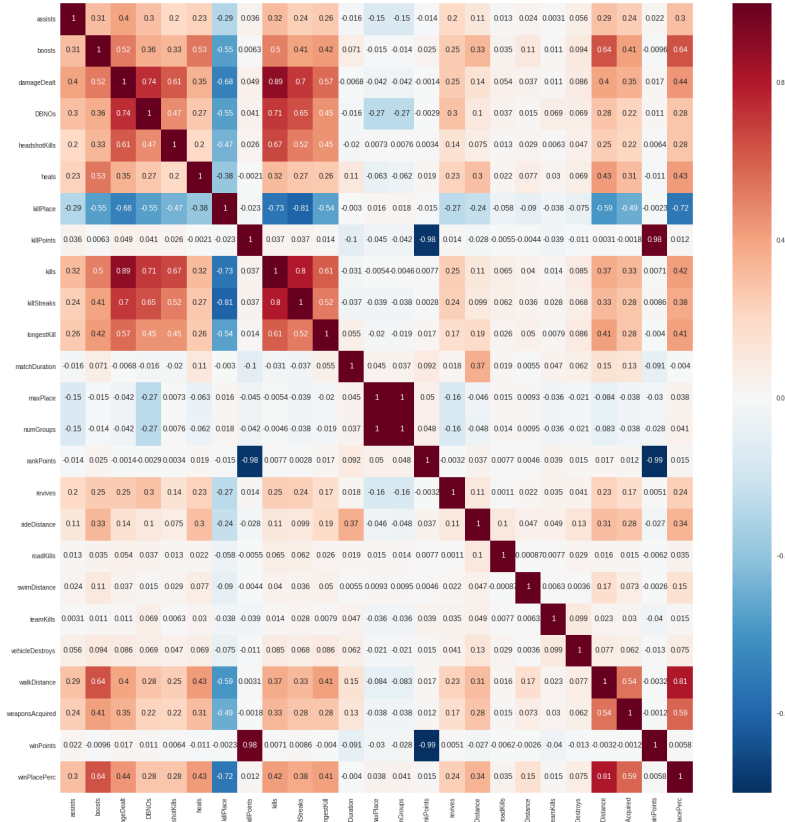


Figure 1: Correlation matrix between features

There were no missing values in the data considered for this project. The data was pretty clean and hence, there was no need of data cleaning.

Figure 3 is shown to show the importance of preprocessing of continuous and categorical features. 3(a) to (c) shows that the continuous features are of different ranges. For example features in (a) lies in the order of hundreds, (b) lies in the order of tens, while (c) lies in the order of ones. Thus standardization (features have mean 0 and standard deviation 1) or normalization (features forced to lie in the range of 0 and 1) is really important to use them with equal importance. I used standardization of numerical features since it gave good results when compared to normalization. I used *sklearn.preprocessing.StandardScaler* and *sklearn.preprocessing.MinMaxScaler* for implementation.

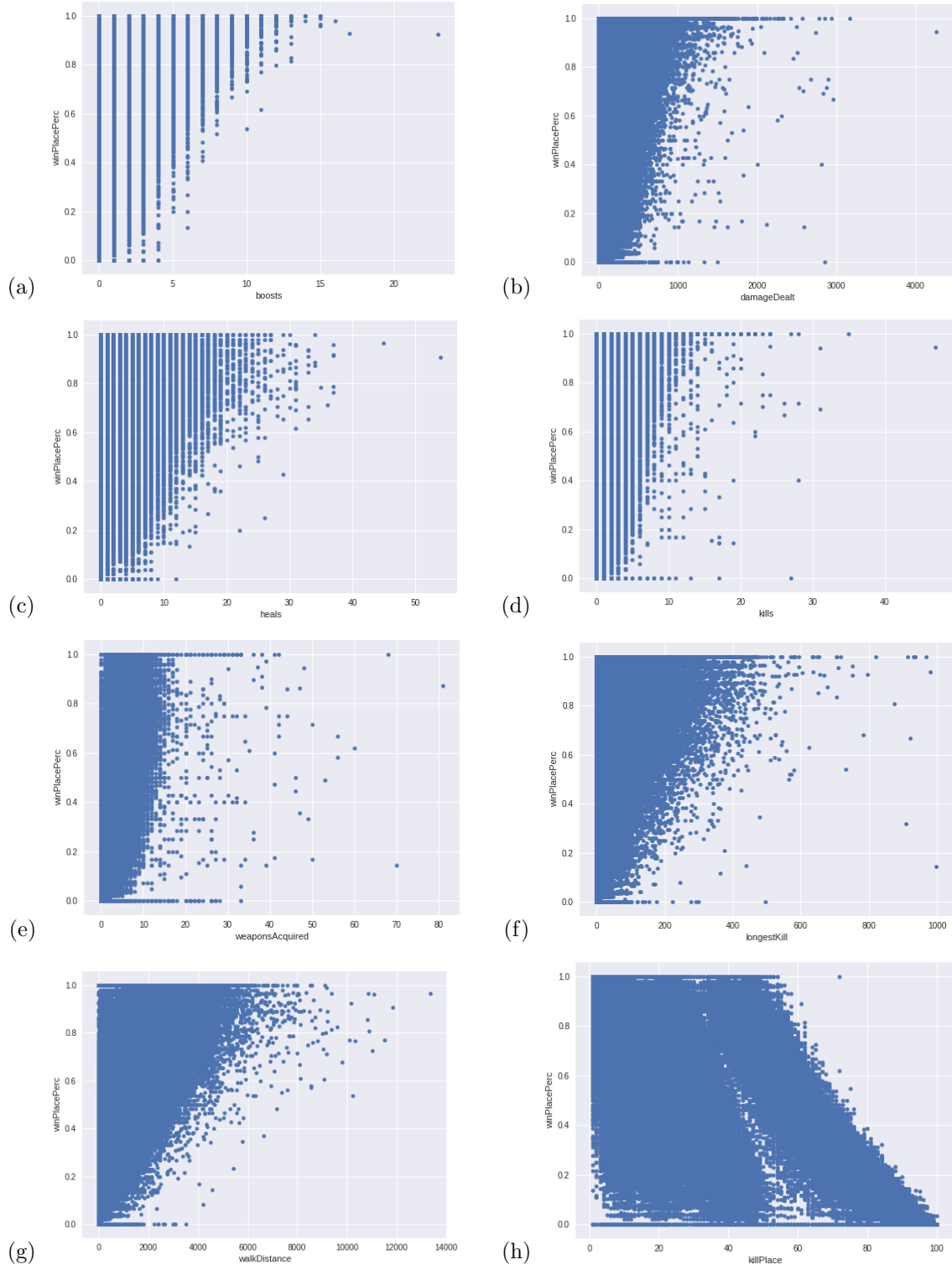


Figure 2: Features Vs Target (high positive and high negative correlation features considered)

- walk distance over heals = walk distance / heals
- walk distance over kills = walk distance / kills
- kills per walk distance = kills / walk distance
- skill = head shot kills + road kills

Example correlation matrix of the derived features from other base features are shown in figure 4. This analysis was also done on the pre-trained dataset. As we can see, the derived features are highly correlated with the target and base features, I drop the base feature *heal* and *kills* and retain the derived features. This is done in an effort to remove correlation between the features and reduce redundancy.

Also, I do not use *id*, *match id* and *group id* when I train the model. Section 2.2.3 explains how I use *group id* and *match id* to subsample the dataset and reduce the number of data points. Table 3 shows details of final feature dimension that I considered for the rest of the project.

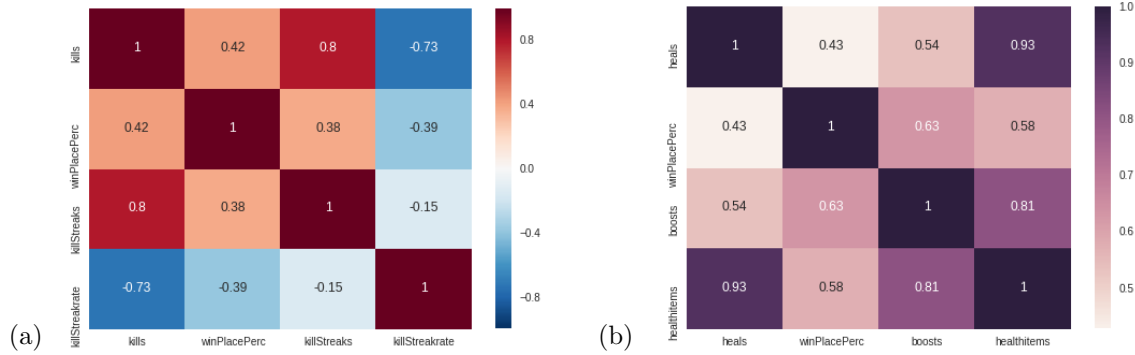


Figure 4: Sample correlation matrix of derived feature with base features and target

Original number of features	28
Number of features dropped	6
Number of features added due to one hot encoding	16
Number of extra features added	11
Total number of features	49

Table 3: Final feature dimension

2.2.3 Datacount and Dimensionality Adjustment

After adding some derived features as explained in the previous section, still the dimension was less compared to the number of data points. So I carry implemented the following two methods to reduce dataset size and increase number of features.

- Group the entire dataset by *match id* and *group id*. This is done mainly on the observation that the target (*win place percentage*) remains the same for all players within a group. This is illustrated in figure 5. This is a squad type match, which means groups within a match contain 4 player. We can see that the each target value remains the same for all 4 players in a group. Thus number of data points reduce to unique number of group ids, each data point representing the group based on statistics of the players within the group. Thus the dataset size reduced by approximately two in both train and test group.
- Since each datapoint represents a group, it opens up a new direction for calculation of statistics of a group. I calculate min, mean and max values of all the features of players within a group. I also calculate ranks of each group based on the group's min, mean and max values for every feature. Thus the dimensionality increased by 6 times. A total of $49 \times 6 = 294$ features were considered.

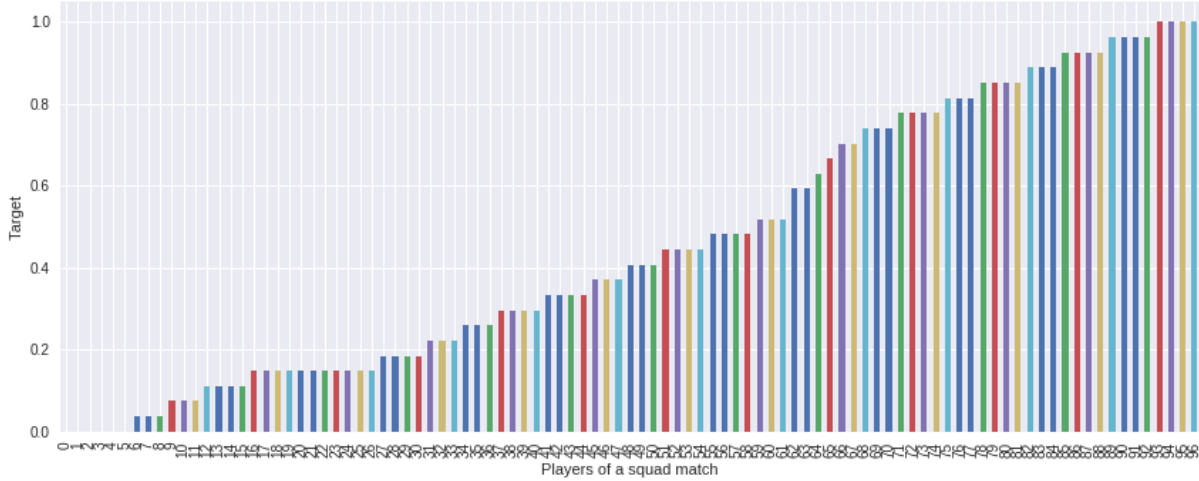


Figure 5: Sample player win place percentages from a match

2.3 Dataset Methodology

Dataset methodology is given below. Figure 6 shows flowchart of training process. 5% of training data was set aside for exploratory data analysis. All the plots for analysis in section 2.2.1 were generated using this 5% of pretrain data. Once initial analysis were performed, it was never used in the training process. The remaining data was again split into pseudo train and pseudo test. Pseudo test data is needed because the project is a Kaggle competition and there is no ground truth available for the actual test data. Pseudo train data was further considered for training and pseudo test data was never used in the training process. Pseudo test data was only used in evaluation like actual test data. As mentioned in section 2.2.3, the data was grouped by *match id* and *group id*, obtaining a single data point for every group. Each group's features were calculated by taking min,

min rank, mean, mean rank, max and max rank on players within that group. This reduces the size by two. Even then, the training data set size was 2.1 million.

Cross validation was difficult to perform in such a huge number of data points. Thus, I randomly sub sampled 40% of the grouped pseudo train data for cross validation. 5-fold cross validation was used to find the best parameters of various models like lambda in ridge and lasso regression models, number of estimators in random forest and light GBM. Then, I combined and used all the grouped pseudo train data to retrain the model on best chosen parameters.

Figure 7 shows flowchart for test data. Pseudo test data or actual test was not used anywhere in the training phase. Test data was grouped on *match id* and *group id* (as followed in training process). The target *win place percentage* for every group was predicted using the trained model obtained from training phase. Finally, all players within the group are traced and assigned with the predicted *win place percentages*.

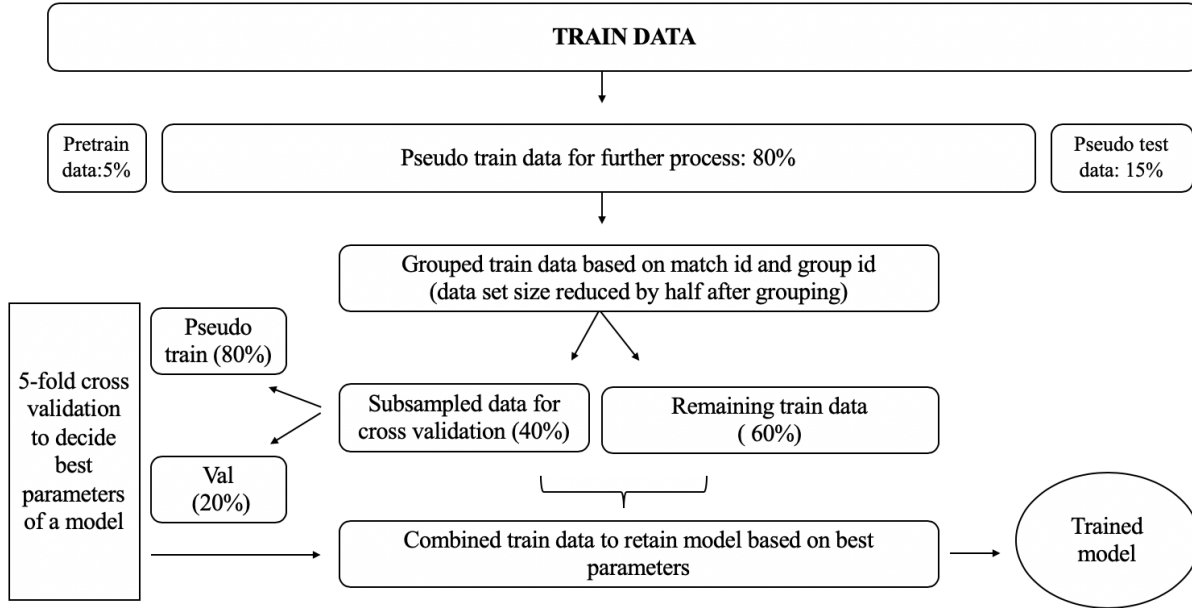


Figure 6: Train dataset methodology

2.4 Training Process

I tried various machine learning models on the data. They are linear, ridge, lasso regressions, random forest regressor, light gradient boost method for regression and xgboost. I describe all of them in detail below.

2.4.1 Linear Regression

This is a simple model based on ordinary least squares. The assumption is that the target variable can be expressed in terms of linear combination of features. The loss function of

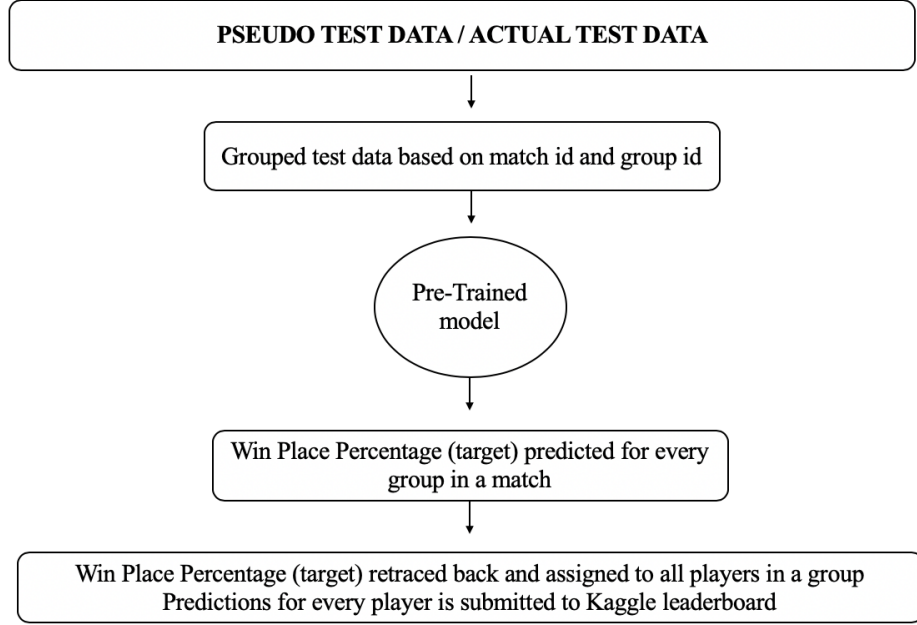


Figure 7: Test dataset methodology

linear regression is given by Residual Sum of Squares (RSS) as in 1. This is derived by maximizing the likelihood (Maximum Likelihood Estimate)

$$J(w) = \frac{1}{2} \sum_{i=1}^N (y_i - \underline{w}^T \underline{x}_i)^2 \quad (1)$$

On solving for weights algebraically, we get the ordinary least squares form. There are no extra parameters that need to be decided or tuned. Thus I did not use cross validation for Linear Regression. I used *sklearn.linear_model.LinearRegression* for implementation.

$$\hat{\underline{w}}_{OLS} = (\underline{X}^T \underline{X})^{-1} \underline{X}^T \underline{y} = \underline{X}^{-1} \underline{y} \quad (2)$$

2.4.2 Ridge Regression

Ridge is similar to linear regression but it is derived by maximizing the posterior instead of likelihood (Maximum A-posterior Estimate). Equation 4 is derived by assuming a gaussian prior for the weights. One can observe that this equation is similar to 1 but with an extra l2 term added to the right. This is the penalty term or regularization strength added to the linear regression loss to control over fitting or under fitting. I used *sklearn.linear_model.RidgeRegression* for implementation.

$$J(w) = \frac{1}{2} \sum_{i=1}^N (y_i - \underline{w}^T \underline{x}_i)^2 + \lambda \|\underline{w}\|_2^2 \quad (3)$$

I tried Ridge regression since, the data was huge in number when compared to the number of features leading to under fitting most of the time. Larger the lambda value, larger is the regularization to prevent over fitting. But in my case, I had to reduce the lambda value to prevent under fitting. I used cross validation on the subsampled data to find the best lambda value from the range of 0.00001 to 1. The best lambda value was 0.01 giving the lowest error.

2.4.3 Lasso Regression

Lasso regression is again derived by maximizing the posterior. The difference between ridge and lasso regression is that the prior for weights is assumed to be a laplacian distribution. The penalty term reduces to be a l1 norm of the weights. Since the assumption is laplacian, the probability of weights to be zero is higher when compared to gaussian probability. Thus the weights become sparse.

$$J(w) = \frac{1}{2} \sum_{i=1}^N (y_i - \underline{w}^T \underline{x}_i)^2 + \lambda \|\underline{w}\|_1 \quad (4)$$

This method is also generally used to prevent over fitting. I reduced the lambda values instead of usual higher values to prevent under fitting. Similar to ridge regression, I used cross validation to find the best value for lambda between 0.00001 to 1. The best lambda value was 0.001 after cross validation. I used *sklearn.linear_model.LassoRegression* for implementation.

2.4.4 Random Forest Regression

Random forest is a type of additive model, that makes predictions based on average of individual models in case of regression. In random forest based method, there are B trees built and they are combined to make it a forest of decision trees. Random sampling of dataset is done to build a tree and in each iteration of region split, only a fraction d features out D features are selected. Once B trees are built, the regression estimate is the average of all the trees predictions as shown in equation 5.

$$\hat{f}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}_b(x) \quad (5)$$

$$\hat{f}(x) = \frac{1}{B} \sum_{b=1}^B \sum_{m=1}^M w_m^{(b)} \mathbb{1}(\underline{x} \in R_m^{(b)}) \quad (6)$$

Random forest regressor is very good at handling non-linear interaction between the features unlike the linear model. Since features are very few hundreds in number and the data does not have many sparse features, random forest model is expected to work very well. I used *sklearn.ensemble.RandomForestRegressor* for implementation. I used cross

validation on the subsampled data to decide number of estimators (B - number of trees). I varied the values from 20 to 100, and the best value was 40.

2.4.5 Gradient Boost Regression

Like any other boosting algorithm, gradient boosting combines many weak learners into a strong learning in an iterative manner. I tried two types of Gradient Boost implementation in Python.

- XGBoost is a scalable end-to-end tree boosting system which is highly efficient, flexible and portable. XGboost uses pre-sorted algorithm to find the best split. Hence it enumerates over all the features and sorts the feature values for every feature. Once sorting is done, a linear scan is done to find the best split. Again the best split among all the features is obtained. I used *XGBoost* for implementation. Since each XGBoost regressor was taking hours to run even for the subsampled data for cross validation, I could not use cross validation. Instead, I used the XGBoost parameters which had best results posted in Kaggle discussion forum. I used 1000 estimator, max depth of 9 with learning rate 0.007.
- Light Gradient Boost Method (LGBM) is gaining more popularity in the regression world. Light GBM uses Gradient-based One-Side Sampling (GOSS) to find the best split value. This runs in high speed when compared to other boosting methods, can handle very large number of data and also takes very less memory to run. It also gives improved accuracy in results. LightGBM grows regression trees vertically unlike other methods which grow trees level wise. The tree with maximum delta loss to grow is chosen. This helps in reduction in loss more than the level wise methodology. I used *lightGBM* package in python for implementation. The subsampled data for cross validation was used in parameter tuning. I varied number of estimators from 1000 to 30k in steps of 5k and learning rate from 0.001 to 1 in steps of 0.05 to find the best parameters. The best model found have 20k estimators with a learning rate of 0.05 along with other default parameters.

2.4.6 Multi-Layer Perceptrons for Regression

Multi-Layer Perceptrons (MLPs) can be used for regression having single output unit with Identity function as activation function. It uses back propagation method to optimize weights in every layers. Some hyper parameters that are important in MLP are number of hidden layers, number of hidden units in a layer, learning rate, activation function for hidden layers, dropouts etc. MLPs for regression performs best when there is huge amount of data like any other deep learning frameworks and it can capture even complex non-linear relationships between the features. I used *keras* for python implementation. The best chosen model after various trial and error experiments is given in table 4.

Number of input units	294
Number of output unit	1 (Identity activation)
Number of hidden layers	3
Number of hidden units	256, 512, 256
Activation function for hidden layers	ReLU
Loss	MAE
Learning rate	0.01
Epochs	1000
batch size	128

Table 4: Mult-Layer Perceptrons Architecture

2.4.7 R^2 Score and Mean Absolute Error

For all the above mentioned regression methods, I used R^2 score and Mean Absolute Error (MAE) to validate the model. R^2 score is the coefficient of determination is generally used for regression. Best possible score is 1 which means the model is perfect. If the score is zero, then it predicted the mean values of target (which can be considered as baseline). If the score is negative, the model is performing worst than the baseline.

$$R^2 = 1 - \frac{RSS}{TSS} \quad (7)$$

where,

$$RSS = \sum_i (y_i - f_i)^2 \quad (8)$$

$$TSS = \sum_i (y_i - \bar{y}_i)^2 \quad (9)$$

$$\bar{y} = \frac{1}{N} \sum_{i=1}^N y_i \quad (10)$$

Though R^2 helps us to understand the performance of our model with respect to the baseline, I used MAE since that was commonly used in the Kaggle discussion forums. Also the leaderboard also gave MAE as the score. This gives the average of absolute difference between the actual target and predicted regression value.

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (11)$$

Different regression methods and different ways of approaching the problem (refer section 2.5) were compared by using both R^2 score and MAE values. Both were calculated using `sklearn.metrics.r2_score` and `sklearn.mean_absolute_error` in python.

2.5 Model Selection and Comparison of Results

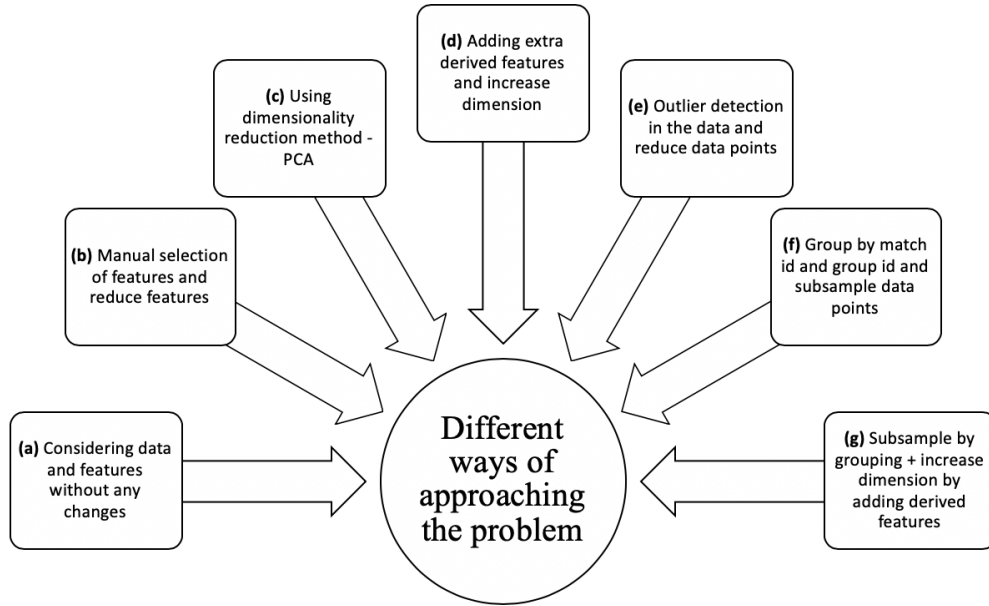


Figure 8: Different ways the problem was approached

I had multiple potential models in the beginning. I will explain them in detail in this section. Figure 8 show all different ways I approached the problem with. The approaches I considered are given in the same temporal order from left to right, the left most being the initial model and the right most one being the final model with best results.

- I initially tried just considering all the features and all the data points without making any changes in them. The results were not as good as expected.
- Then I went on improving the model. Since many features were highly correlated with one another as shown in figure 1, I thought reduction in features can help improve the model. Thus, I tried automatic reduction of features using Principal Component Analysis (PCA).
- I also tried manual selection like omitting features that were of zero correlation with target and selecting one from the highly correlated feature pairs.
- Since feature reduction were not helping in model improvement, I had to either subsample the huge dataset or add extra features to increase dimensionality. To achieve increase in number of features, I added extra derived features as given in section 2.2.2.
- I also tried reducing the dataset by removing outliers in the data, which was done by looking omitting data points that are lesser than 5th percentile value and greater than 95th percentile value of every feature. This was improving the results, but not to a great extent.

- I went on improving the methodology of subsampling by grouping data points with respect to match id and group id. This improved my results.
- Thus my final model was a combination of subsampling by grouping and increasing dimensionality by adding extra features.

Each of the above mentioned approach were trained using all the regression models discussed in section 2.4. The comparison of results is given in table 5. All the results are given for the pseudo test data.

Approach used	Metric considered	Regression Model						
		Linear	Ridge	Lasso	RF	Light GBM	XGBoost	MLP
(a)	R^2	0.821	0.832	0.831	0.921	0.944	0.829	0.967
	MAE	0.091	0.088	0.087	0.061	0.054	0.061	0.059
(b)	R^2	0.808	0.808	0.808	0.908	0.905	0.900	0.901
	MAE	0.097	0.097	0.098	0.055	0.067	0.054	0.069
(c)	R^2	0.789	0.791	0.789	0.892	0.901	0.865	0.899
	MAE	0.114	0.114	0.115	0.069	0.057	0.082	0.059
(d)	R^2	0.856	0.852	0.851	0.901	0.954	0.856	0.978
	MAE	0.074	0.075	0.074	0.059	0.058	0.052	0.054
(e)	R^2	0.824	0.824	0.821	0.894	0.920	0.813	0.919
	MAE	0.077	0.077	0.077	0.063	0.051	0.070	0.052
(f)	R^2	0.942	0.948	0.951	0.974	0.981	0.957	0.075
	MAE	0.049	0.048	0.045	0.357	0.029	0.419	0.039
(g)	R^2	0.953	0.958	0.951	0.973	0.989	0.959	0.078
	MAE	0.046	0.044	0.045	0.351	0.027	0.410	0.037

Table 5: Comparison of Results. Approaches are labeled from (a) to (g), refer figure 8

Some observations on different regression models are:

- Linear regression was giving decent results, but not as better as other complex models that capture non-linear dependencies. This is expected since our problem is not a simple one.
- Ridge and Lasso regression were mostly worsening the linear regression results. Though the lambda value were lesser, adding penalty term worsens the performance.
- Random forest regressor gave better results when compared to Linear, Ridge, and Lasso regressor. This is also expected since Random Forest model can capture both linear and non-linear dependencies.
- The expectation that XGBoost is one of the best model did not work in my case. Since the number of data points was huge, I couldn't fine tune the parameters properly even after subsampling. Also, most of the times Kaggle kernel died before it could converge. These could be the reason for degraded performance.

- MLP was one of the best performing model. This is expected since as number of data points increases, deep learning models tend to perform better.
- Light Gradient Boost method was the best model. This had various advantages like it was faster than complex methods like XGBoost and MLP. Even after subsampling the data by grouping, there was 2.1 million data points. Light GBM was able to handle such huge amount of data without compromising on the results.

Scatter plot between the actual and predicted pseudo test target data from the best model is given in figure 9. The points lie in the orientation of 45 degree line. Which shows that the predicted and actual are very close.

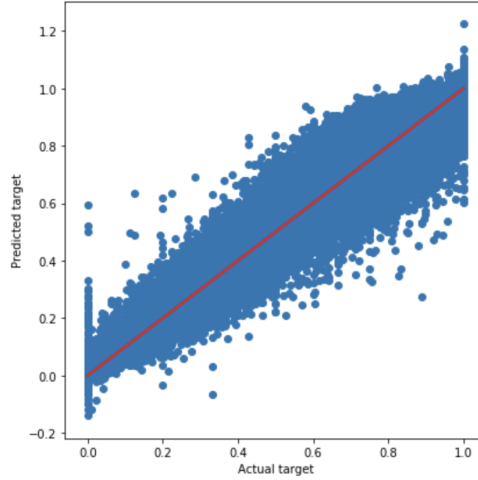


Figure 9: Actual Vs Predicted - Pseudo test

2.5.1 Post-Processing to Improve Results

After seeing other people's leaderboard score, I found that some people were using post-processing methods to improve the score by 0.003. I adopted their approach to improve my leaderboard score. Feature *max place* gives the worst placement for a match. Hence each player's predicted ranking can be adjusted based on the *max place* of a given match. This is given by equations 12 and 13

$$gap = \frac{1.0}{maxplace - 1.0} \quad (12)$$

$$newPrediction = \frac{oldPrediction}{gap} \times gap \quad (13)$$

3 Final Results and Interpretation

The best model decided from comparisons of various approaches and regression models is given in table 6. This model was used to get predictions on the actual Kaggle test data.

The predictions were submitted to the leaderboard and the score is reported in the same table. This value is hence same for all the players with a match.

Data points	grouped by match id and group id
Operation on group by	Min, Mean, Max and their ranks
Features	Base features + Derived features
Regression Model	Light Gradient Boost Method
Num of estimators	1500
Num of leaves	31
Learning rate	0.05
Bagging fraction	0.09
Metric	MAE
Pseudo train MAE (in sample performance)	0.023
Pseudo test MAE (out sample performance)	0.027
Leaderboard Score (without post processing)	0.0247
Leaderboard Score (with post processing)	0.0214

Table 6: Final Model

Since ground truth for the actual test data is not available, I compare the distributions of target *win place percentage* between the train and actual test data. The distribution of the target should remain the same irrespective of train or test data. This is proved in figure 10 since the distributions remain visually same.

Some final observations from the results are listed below.

- Just considering the given features and all the data points isn't sufficient to get the best model. There needs to be some subsampling method to reduce dataset size. Also given basic features aren't sufficient to get a good model. Number of features have to be improved.
- As we saw some features were highly correlated with other features. There is redundancy in the features available. Also some features were not correlated with the target. When these features were manually selected and removed, it worsened the performance of the model.
- Similar to manual selection of features, dimensionality reduction was reducing the model performances. This adds to our intuition that the number of features are very less and even more reducing them is a bad technique. We need to increase the number of features.
- Adding some derived features improved all the models when compared to the previous results. This proves that adding more features improves the models given huge number of data points.
- To reduce the data points, instead of random sampling, outlier based removal helped improve the model. This proves that subsampling of dataset and reducing the size helps in getting better results.

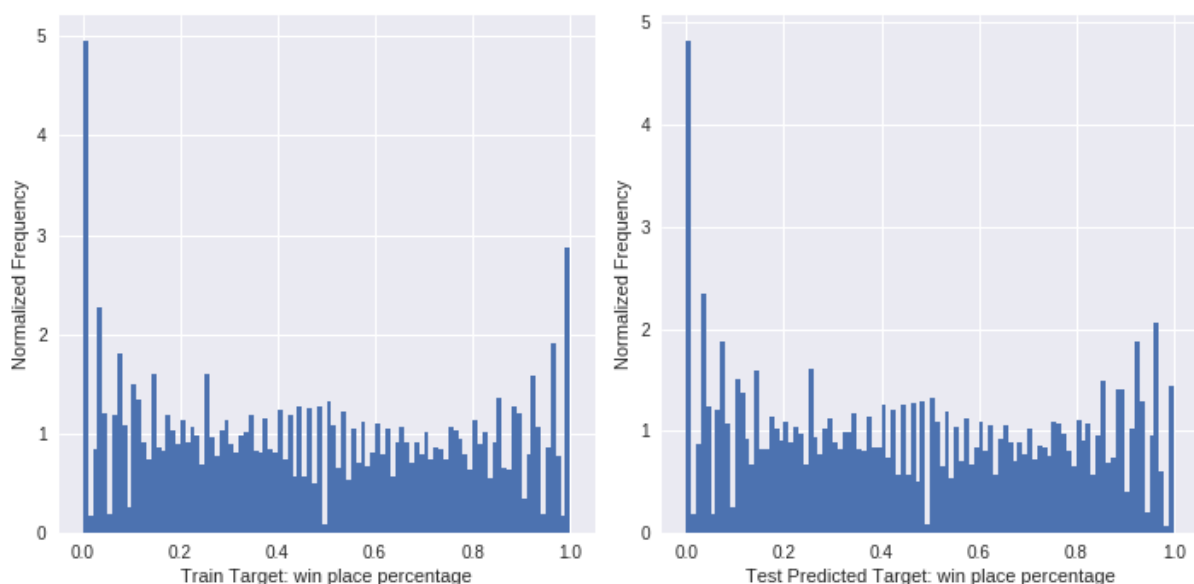


Figure 10: Comparison of target distributions between train and test data

- Subsampling by grouping players within the same match and group reduced the dataset size by two. But the model performance largely improved giving the least MAE when compared to all the models previously considered.
- Since subsampling by grouping and adding derived features were best models when compared to other models, combining them gave the best results.

4 Contributions of each team member

This project was done individually.














5 Summary and conclusions

The project was very motivating since this was my first time in a Kaggle competition. I mainly learnt how to handle huge datasets and to work with Kaggle kernels. This is my first regression project too on a real-world data and I implemented many regression models. After trying out many approaches to solve the problem, I was able to successfully build a model that predicts the final win place percentage of every player in PUBG game. Grouping the players data with respect to match and group was the key criterion that provided the best model. Adding features also helped in improving the regression results.

Since there are two more months for the Kaggle competition to end, I would like to work on this project more in the winter break to improve my leaderboard score.

6 References

- EE 660 class lecture notes
- Kaggle discussion forums
- <https://www.kaggle.com/c/pubg-finish-placement-prediction>
- <https://xgboost.readthedocs.io/en/latest/>
- <https://lightgbm.readthedocs.io/en/latest/>
- <https://www.kaggle.com/anycode/simple-nn-baseline-4>

Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions
144	new							0.0206 8 3d
145	▼ 42							0.0208 34 24d
146	▼ 41							0.0210 15 8d
147	▲ 7							0.0213 9 6d
148	▼ 41							0.0213 11 3d
149	new							0.0214 3 ~10s
Your Best Entry ↑ Your submission scored 0.0214, which is an improvement of your previous score of 0.0247. Great job!  Tweet this!								
150	new							0.0217 6 2d
151	▼ 43							0.0218 20 18d
152	▼ 43							0.0221 29 1mo
153	▼ 43						 +4	0.0222 18 1mo
154	▼ 43							0.0224 4 1mo
155	▼ 43							0.0224 2 18d

Overview	Data	Kernels	Discussion	Leaderboard	Rules	Team	My Submissions	Submit Predictions
of your full submission — your public score is only a rough indication of what your final score is. You should thus choose submissions that will most likely be best overall, and not necessarily on the public subset.								
3 submissions for Abinaya Manimaran							Sort by	Most recent ▼
All Successful Selected								
Submission and Description							Public Score	Use for Final Score
dec_5_script (version 1/1) 3 minutes ago by Abinaya Manimaran From "dec_5_script" Script							0.0214	<input type="checkbox"/>
my_final_model_script (version 1/1) 2 days ago by Abinaya Manimaran From "my_final_model_script" Script							0.0247	<input type="checkbox"/>
my_final_model (version 1/2) 2 days ago by Abinaya Manimaran From "my_final_model" Script							0.0260	<input type="checkbox"/>
No more submissions to show								

Figure 11: Kaggle submission results