

PUBG Finish Placement Prediction

EE 660 Project Type: Individual (Kaggle Competition)

NAME: ISHAN MOHANTY

EMAIL: imohanty@usc.edu

DATE: December 6th, 2018

1. Abstract

The gaming industry today is a billion-dollar industry raking in more and more revenue through building amazing applications that gets its users addicted. The use of machine learning and AI have revolutionized the way video games are played, analyzed and developed. To study machine learning techniques and apply it to a real-world game is not only challenging but the mere thought of it seems extremely exciting. With that thought we dive into our gaming machine learning problem which is derived from a Kaggle competition titled: ***PUBG Finish Placement Prediction***.

PUBG also known as PlayerUnknown's Battlegrounds is a mega-gory bonanza where in 100 players or less battle it out to earn bragging rights. We need to predict the win placement percentile of player or a team to determine their survival rate through the use of machine learning techniques.

We follow the subsequent steps roughly to see how successful we are in predicting the output response variable. The plan of action is to first define our problem, perform preprocessing, feature engineering and dimensionality adjustment with a pre-training set that is born from the original training set. Then perform training using various regression techniques and finally fit the prediction on the test set. After this fine tuning will be employed and kernels would be run multiple times to obtain the best 'Mean Absolute Error' or MAE for a given model. These techniques are discussed in the sections to follow.

2. Introduction

2.1. Problem Type, Statement and Goals

PlayerUnknown's Battlegrounds also known as PUBG, is a multiplayer game where in 100 players battle it out on a diabolical island covered with abundant weaponry. During the warfare and the on-going blood-shed, the area of the playing zone decreases and forces players into direct confrontation. This continues until there is one player or one team standing that gets declared the victor. The PUBG dataset provided by Kaggle, gives all the game stats of matches ranging from solos, duos, squads and custom.

Our goal is to design a model to predict all players' winning placement percentage based on their stats in the range of 1(first place) to 0(last place).

The problem here falls in the 'regression' realm as the prediction is real-valued in the range of 0 to 1 inclusive. 'winPlacePerc' is the feature variable that is to be predicted. Our motivation to analyze this dataset also comes from the fact that the gaming world is thriving and using machine learning could prove to be a game-changer for revolutionizing AI and also improving games by understanding features importance. The fact that the data set is based on real-time data makes gamers all round the word better their strategy by studying the results of our experiment.

This problem presents a non – trivial property due to the reasons given below:

1. Exploratory data analysis and visualization is very challenging due to diverse and sometimes independent features that vary a lot.
2. The massive scale of data makes the problem very challenging in terms of training the model and deploying the results on the dataset. There is about 4.45 million training data points and about 1.93 million test data points.
3. Since the data contains real-time stats which can be used by gamers to devise their strategies.

2.3. Prior and Related Work: None

2.4. Overview of Approach

The machine learning problem was modelled using the basic linear regression, random forest (ensemble model) and LightGBM(gradient-boosting model). Here model selection is very dicey as there are no labels for the test-set which makes prediction really challenging. We choose the primitive linear regression as the benchmark, as it has the lowest performance in terms of the metrics measured and then move on to the superior algorithms to improve training and validation accuracy and decrease the 'mean average error' [MAE]. The metrics, 'mean absolute error' [MAE] was used to measure the performance.

3. Implementation

The implementation was carried out by using regression libraries from sci-kit learn like linearRegression, RandomForestRegressor and gradient-boosting libraries like lgbm developed by Microsoft. The whole implementation was done using the python language. All Dependencies include libraries like numpy, pandas, matplotlib, seaborn, lightgbm and sci-kit learn.

3.1. Data Set

The data set used is from Kaggle and has training data which has 4.45 million points and test data that has 1.93 million points. The data can be found using this URL: <https://www.kaggle.com/c/pubg-finish-placement-prediction/data>. There are a total of 28 features which are numeric or categorical, in the dataset and we need to predict the label ‘winPlacePerc’ which represents percentile winning placement, where 1 represents the first position and 0 represents the last position in a given game. The ‘winPlacePerc’ is present in the training data set but is absent in the test data set.

Sl.no	Feature	Type	Description
1	Id	object	A unique id to identify the entry
2	groupId	object	ID to identify a group in a match. A different group ID is generated each time, if the same player plays in different matches,
3	matchId	object	ID to identify a match played.
4	assists	int64	number of enemies damaged by a player, but killed by team mates.
5	boosts	int64	Count of boost items used.
6	damageDealt	float64	Total damage dealt, excluding self – inflicted damage
7	DBNOs	int64	Count of enemy players knocked.
8	headshotKills	int64	Count of enemy players killed by headshots.
9	heals	int64	Count of healing items used.
10	killPlace	int64	Ranks of number of enemy players killed in a match.
11	killPoints	int64	Ranking of a player based on the number of kills.
12	kills	int64	Max number of enemy players killed in a short time.
13	killStreaks	int64	Number of enemy players killed.

14	longestKill	float64	Longest distance between player and the other player killed at the time of death.
15	matchDuration	int64	Duration of a match in seconds
16	matchType	object	Type of match being played.
17	maxPlace	int64	Worst case placement of data in the match.
18	numGroups	int64	Number of groups we have data for in the match.
19	rankPoints	int64	Rank of the player
20	revives	int64	Number of times a player revived teammates.
21	rideDistance	float64	Total distance traveled in vehicles in meters.
22	roadKills	int64	Number of kills while in a vehicle.
23	swimDistance	float64	Total distance traveled by swimming in meters.
24	teamKills	int64	Number of times this player killed a teammate.
25	vehicleDestroys	int64	Count of the vehicles destroyed.
26	walkDistance	float64	Total distance traveled on foot in meters
27	weaponsAcquired	int64	Number of weapons picked up.
28	winPoints	int64	Win-based external ranking of player.

Figure 1: Table describing feature type and description

3.2. Preprocessing, Feature Extraction, Dimensionality Adjustment

1. Preprocessing

We perform pre-processing on raw data in order to obtain clean data for our training algorithm pipeline.

In this case we observe that there is just one data point where in the ‘winPlacePerc’ is NaN. We remove this data point to clean our data.

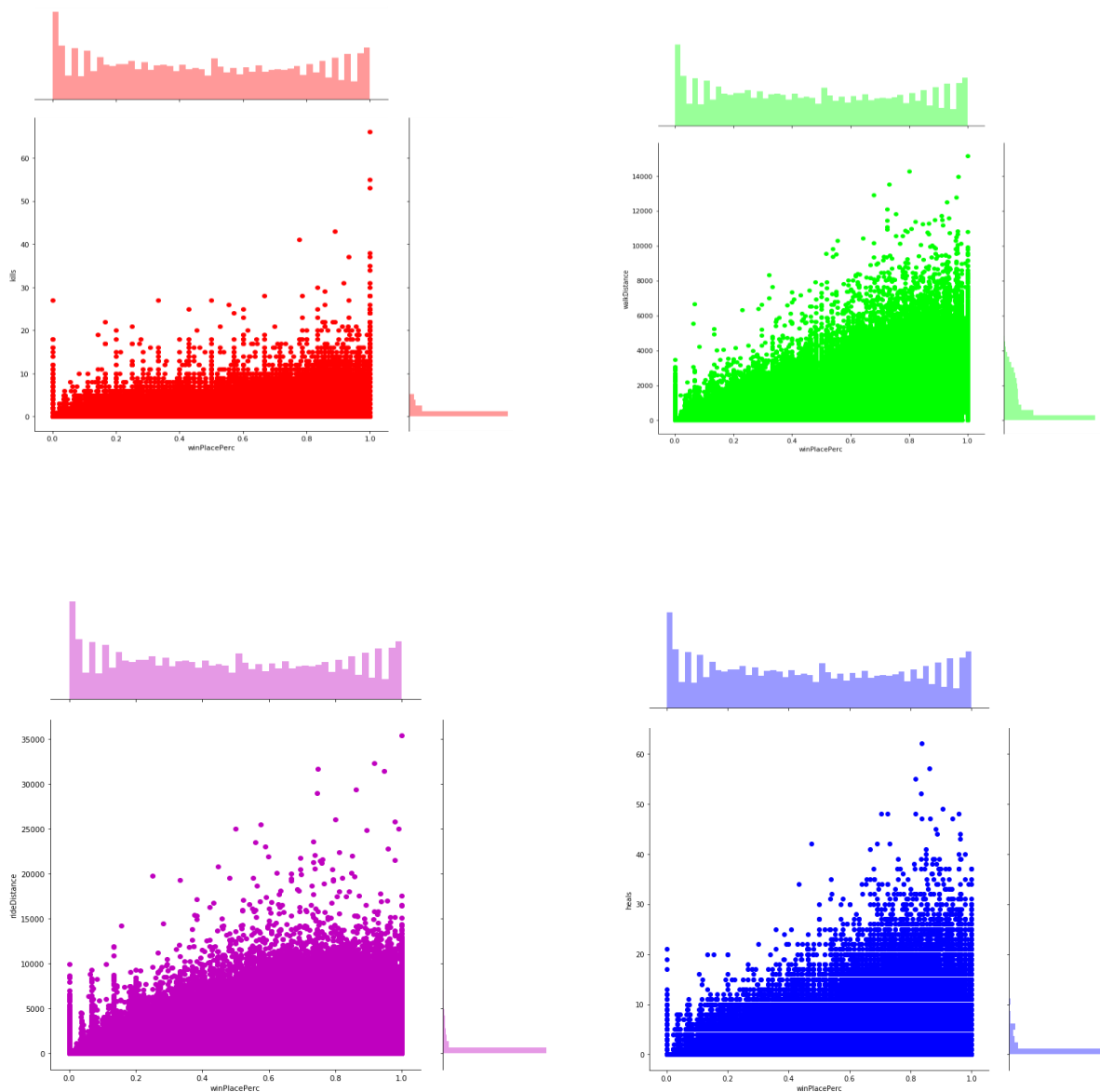
numGroups	rankPoints	revives	rideDistance	roadKills	swimDistance	teamKills	vehicleDestroys	walkDistance	weaponsAcquired	winPoints	winPlacePerc
1	1574	0	0.0	0	0.0	0	0	0.0	0	0	NaN

We then remove the following columns that are objects and don't count as features:

1. Id
2. matchId
3. groupId
4. winPlacePerc (response variable)

After this we go on to the process of exploratory data analysis also known as EDA.

The following shows some joint plots of the different features. Let us analyze them one at a time.



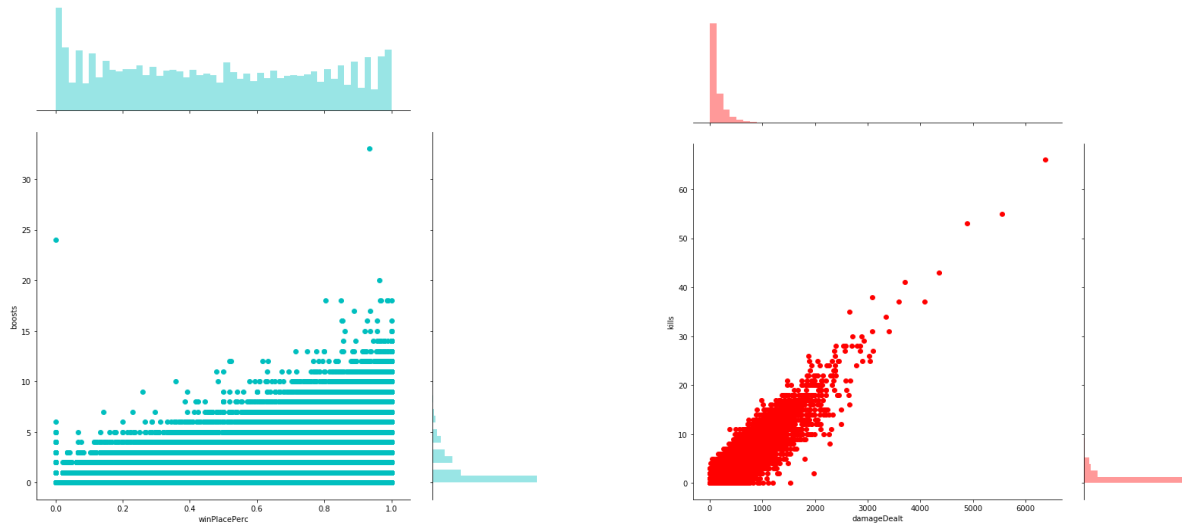


Figure 2: Joint plot between win placement percentage and different hand-picked features.

Moving from left to right sequentially in a row-wise manner we observe that the number of kills has a very high correlation with the winning placement percentile and hence proves to be a useful feature to determine win rate. Observing further, we see that walking has a high correlation with win placement percentile. The ride distance has a low correlation with the winning placement and hence, shows that it may not be that useful of a feature. We see that heals and boosts are very highly correlated with the win placement and boosts seems to have an upper hand as it is much more highly correlated than heals. Both great features, nevertheless. We see that the kills are quite strongly correlated to the damage dealt signifying more damage to a player results in eventual death.

2. Feature Extraction

In this section we use a correlation map to determine what all features are really important and the ones that are not. The more highly correlated a feature is with another, independent behavior is lost due to redundancy. Hence, the least correlated features must be included as independent features and the highly correlated ones should be grouped together. We attempt to understand these properties through the below heat map.

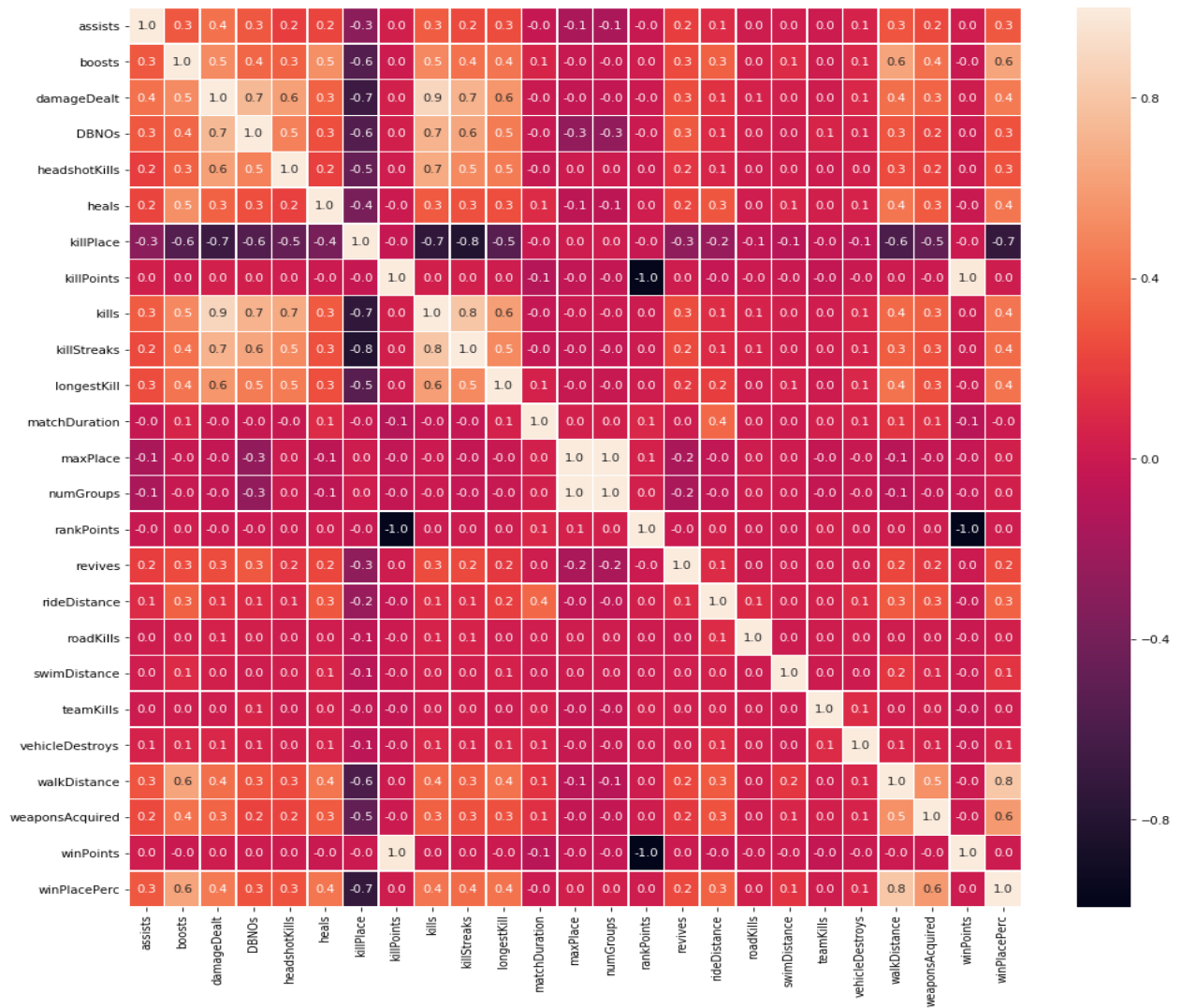


Figure 3: Correlation Heat Map between different features

We observe from the above correlation heat map shows us the most correlated features which could be highly positively or negatively correlated as well as low correlated features. Walking distance with respect to win place percentile is most highly correlated and hence, walking distance in the island does determine survival rate. This reaffirms the above joint plot. We then see the following have high positive correlations with win place percentile: weapons acquired, boosts, kills, heals, killstreaks and longest kill. Kill place seems to be highly negatively correlated with win place percentile and it just goes to show that coordinate location of the island determines which player gets killed rather than who survives and hence is also an important feature to consider. Now we take a decision on how to combine the highly correlated features to execute

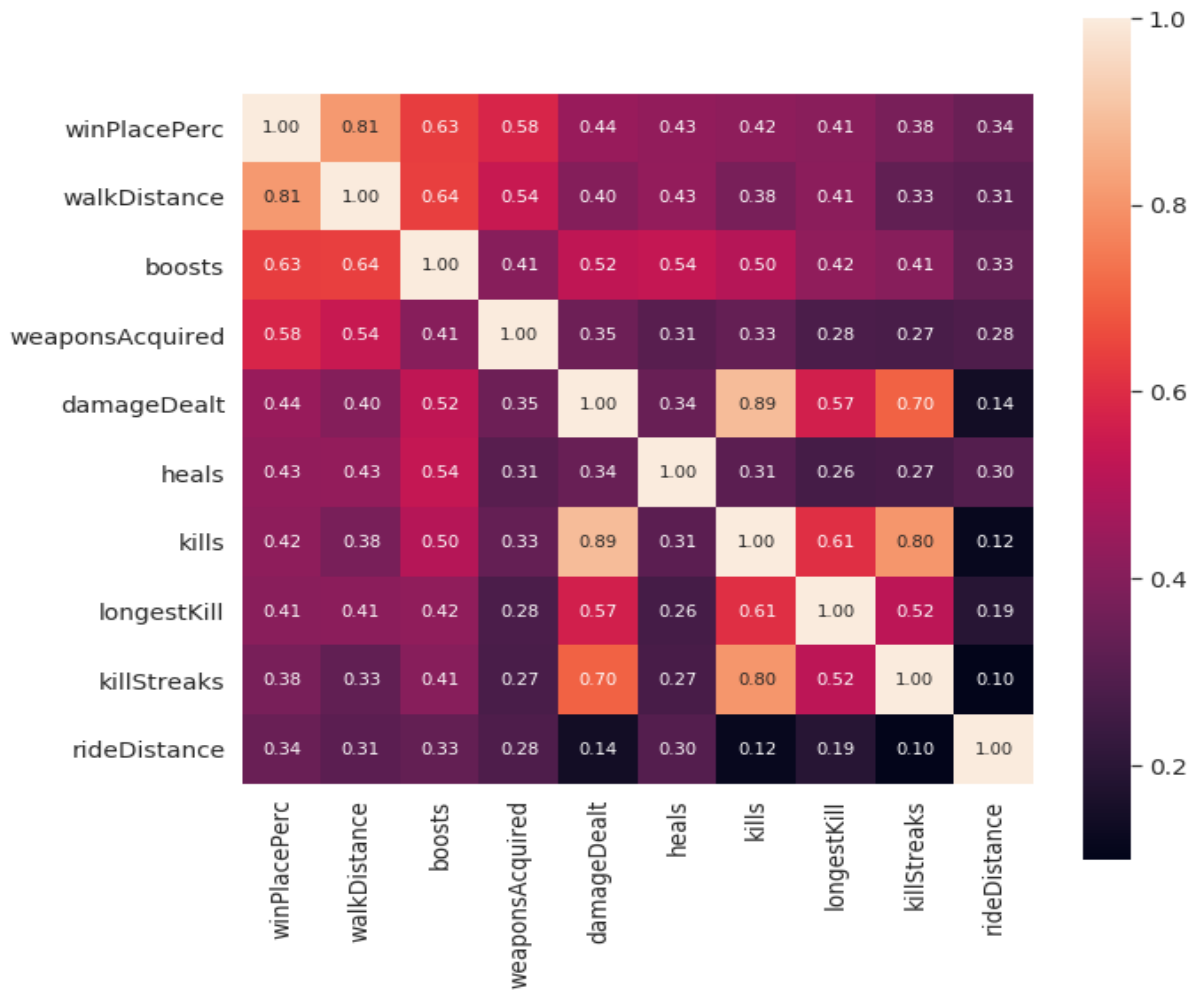


Figure 4: Correlation Heat Map between 8 highly positively correlated features with winPlacePerc

We see above the top 8 highly positively correlated features with win place percentile.

3. Dimensionality Adjustment

With such a large data set and sparse number of features. We would need to adjust the dimensionality of the features to get good results on the training and validation data. We do this by expanding our feature space by combining new features formed from highly correlated features by some designed formula.

We add the following features to the feature space, these are as follows:

1. $\text{headShotRate} = \text{kills} / \text{headshotKills}$
2. $\text{killStreakRate} = \text{killStreaks} / \text{kills}$
3. $\text{healthItems} = \text{heals} / \text{boosts}$
4. $\text{totalDistance} = \text{rideDistance} + \text{walkDistance} + \text{swimDistance}$

5. $\text{killPlace_over_maxPlace} = \text{killPlace} / \text{maxPlace}$
6. $\text{headshotKills_over_kills} = \text{headshotKills} / \text{kills}$
7. $\text{distance_over_weapons} = \text{totalDistance} / \text{weaponsAcquired}$
8. $\text{walkDistance_over_heals} = \text{walkDistance} / \text{heals}$
9. $\text{walkDistance_over_kills} = \text{walkDistance} / \text{kills}$
10. $\text{killsPerWalkDistance} = \text{kills} / \text{walkDistance}$
11. $\text{items} = \text{heals} + \text{boosts}$

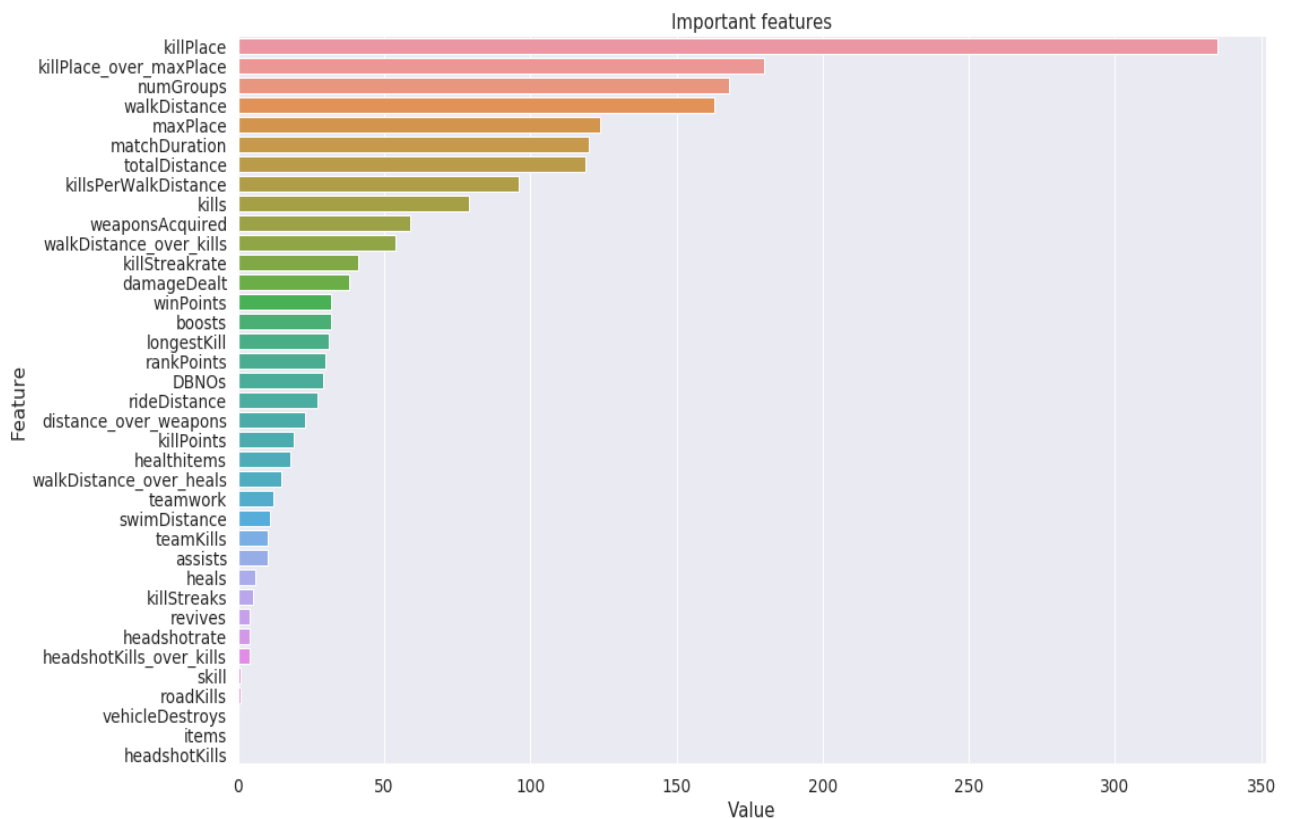


Figure 5: Feature importance with new expanded feature set

The important features are obtained by running the Light GBM technique on the pre-training data and validation data. The important features are sorted and displayed in descending order. We can see the new features added here. We see that killplace bags the most important feature award and headshotKills the least important feature.

3.3. Dataset Methodology

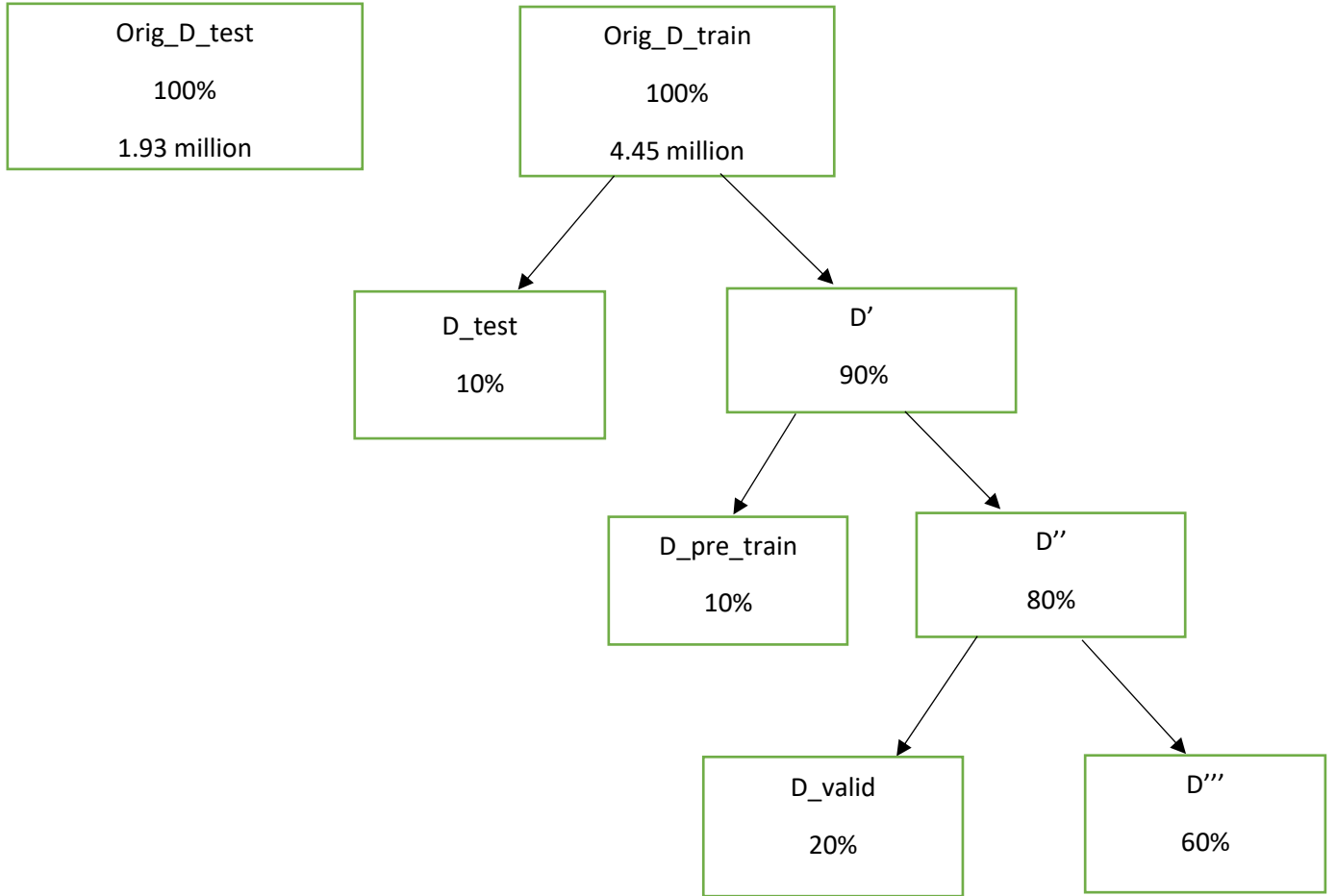


Figure 5: Dataset division for training and testing

The PUBG data set has 2 datasets, the training and test data set. We go about splitting the original training data **Orig_D_train** into a test dataset **D_test**, which is 10% of the training data and another sampled training dataset as **D'** which is 90% of the original dataset. **D'** was further split into pre-train data which was 10%, **D_pre_train** and another **D''** as train subsampled data which was 80% of **D'**. After that the **D''** was split into **D'''** which is the final training set comprising 60% data out of **D''** and validation set **D_valid** which was 20%.

The pre-train data, **D_pre_train** was used for feature engineering and data analysis, **D'''** and **D_valid** for training. The validation results were an important indicator of best model to be fitted on **D_test** for prediction of the output response 'winPlacePerc'. After that the model was fitted on **Orig_D_test**.

Cross-validation for parameter tuning was not performed due to time constraints as each kernel ran for atleast 3-4 hours before getting results due to the massive size of the data.

The above procedure resulted in extremely poor MAE performance in the range of 0.6-0.7. hence, this data methodology was rejected to avoid wasting kernel running time and instead we trained on the whole Orig_D_train and divided it into 80% train and 20% validation data. After this we fitted the prediction onto the entire Orig_D_test after memory reduction to get the best results. The pre_train in the above figure was still used the same way for feature engineering and data analysis.

The models described in the training process was run on the data division described as follows:

1. Linear regression and Random Forest Regressor was trained on Orig_D_train and prediction was done on Orig_D_test as they take a very long time to run otherwise.
2. 80% of Orig_D_train as train data and 20% of Orig_D_train for validation data was used to train Light GBM and the prediction was fitted onto the entire Orig_D_test to get best results.

3.4 Training Process

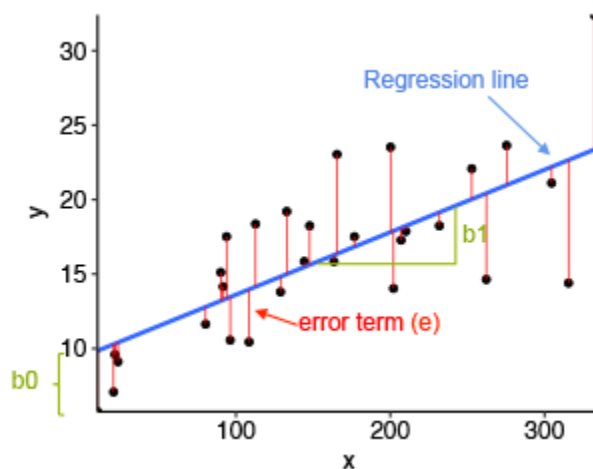
1. Linear Regression algorithm

We choose linear regression as the baseline model here. The problem statement demands us to predict 'winPlacePerc' which is a real-valued variable, hence we start out with the primitive linear regression model to observe the training behavior. Linear regression is a form of supervised learning. The assumption that this algorithm takes in is that there is a linear relationship between the training data and its output label.

Linear regression takes the mathematical form:

$$Y = W_0 + W_i X_i, \text{ where } i = 1, 2, 3 \dots n$$

Here, W_i are the different weights that have to be learned by the algorithm, W_0 is the slope-intercept and Y is the output response.



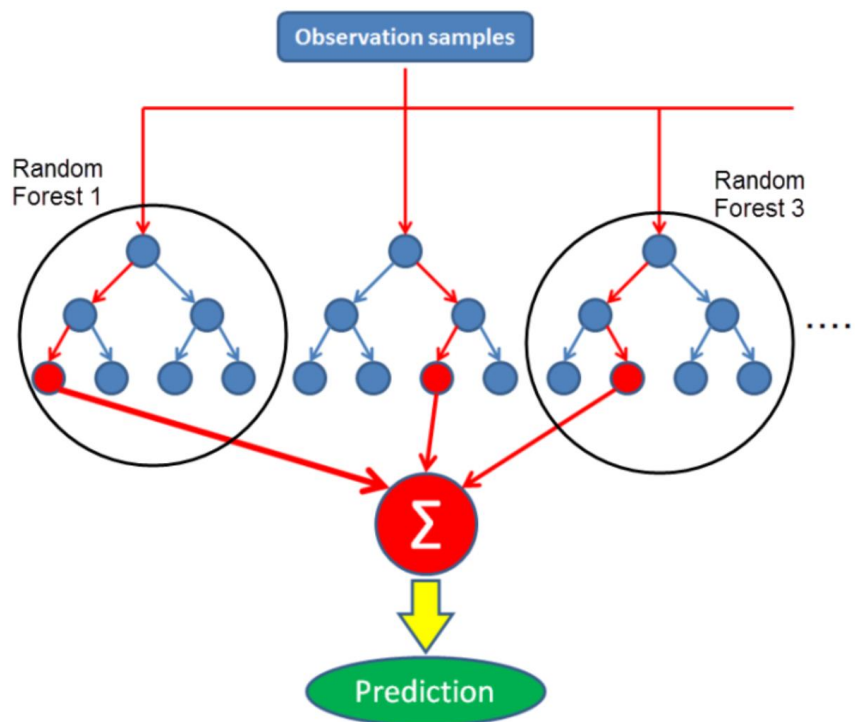
[5] Figure 6: Linear regression graphical model

After the model is trained, the best fit is determined for the data and this generates a line that minimizes the sum of squared distances between predicted labels \hat{Y} and output response labels Y .

We follow the data is split according to our 'Dataset methodology' and trained and after that applied to our final test set.

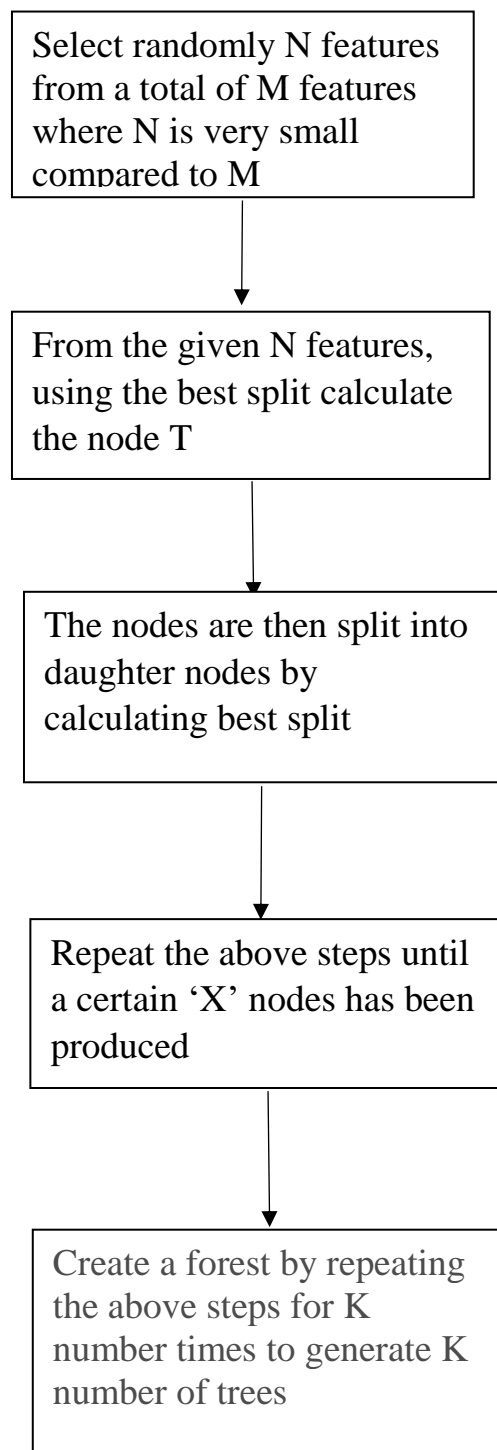
2. Random Forest Regressor algorithm

Random forest is an ensemble model used for classification and regression. It improves the overfitting hurdle that the decision trees are plagued with. This model is an excellent choice as it can handle the non-linear relationships between the features and output variable. They work better than expected for both categorical as well as numerical features for massive datasets. They can be used for studying and analyzing feature importance which is a game-changer in the machine learning universe. One disadvantage with this technique rather with all tree-based techniques is that they do not function well when feature dimensions are low compared to the amount of data. Therefore, for large data with sparse features, this technique would not be very effective.



[3] Figure 7: Random Forest graphical model

The Random forest flowchart is explained below:



[4] Figure 7: Flow chart diagram of random forest algorithm

The parameters that have been tuned for random forest regression in our implementation is as follows:

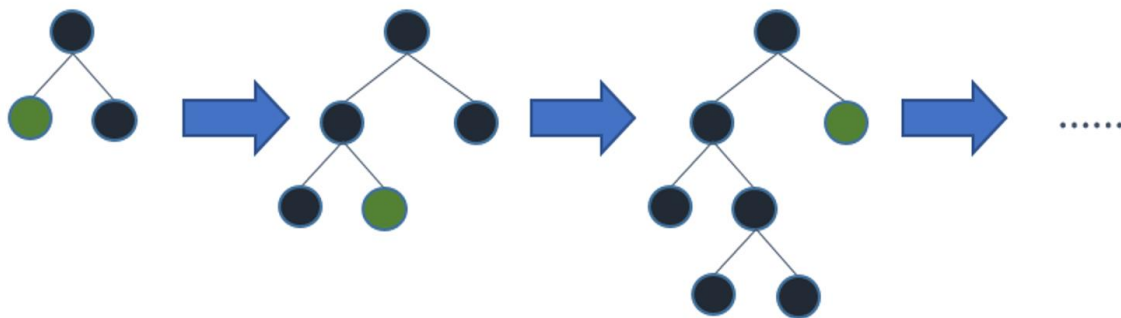
1. **n_estimators** : Regulates the number of trees in the eventual selected model. Generally speaking, more the trees, greater is the accuracy but the downside to this is as the number of trees increases, so will the training and prediction time. 30 estimators were used during training time.
2. **criterion**: Used to measure the performance of the split by using error analysis techniques. MAE was used here.
3. **max_depth**: determines the depth of each distinct tree so that overfitting is ruled out.

3. Light GBM

Boosting in the machine learning cosmos is a sort of ensemble technique used mainly for reducing the variance and bias. It's main ideology questions whether weak learners can be made into strong ones just like a professor's untiring endeavour to motivate the students at the bottom of his class.

The Light GBM algorithm is developed by Microsoft in an effort to achieve rapid training times and steep efficiency. It is a gradient boosting algorithm that makes use of the decision tree methodology for classification, regression and learning assignments.

The primary difference between the tree-based technique as opposed to Light GBM is that tree algorithms split the tree level-wise whereas the boastful LightGBM splits the tree depth-wise or level-wise generated better overall results in terms of performance.



[2] Figure 8: A Light GBM tree demonstrating leaf-wise growth

The parameters that were tuned are as follows:

1. **Objective**: Type of learning task. Regression in our case.
2. **Metric**: Performance measure. MAE here.
3. **n_estimators**: number of trees in the model. Here we have set it to 18000 to get a good accuracy.

4. **early_stopping**: This can boost analysis time by halting if accuracy of the validation data does not improve. We use 180 rounds here.
5. **num_leaves**: The count of the leaves in the whole tree. We set it here to 21 to prevent overfitting, by default it is 31.
6. **learning_rate**: Determines the rate of change in the learning rate. The smaller the learning rate and the more the iterations, the better will be the accuracy. Here we set it to 0.05.
7. **bagging_fraction**: sets the amount of data to be taken for each round and can be very helpful in decreasing training time so as to avoid overfitting. We set it to 0.7 in our model.

The other advantages of Light GBM are as follows:

1. uses less memory.
2. Generates better accuracy compared to other tree-based techniques.
3. Sponsors parallel processing and learning through GPU.
4. Has the ability to handle massive data.

3.5. Model Selection and Comparison of Results

The models in the above section were compared based on the ‘Mean Absolute Error’ metric. The Mean Absolute Error is described as the average of absolute differences between predictions and observations of the test data.

$$\text{MAE} = \frac{1}{n} \sum_{j=1}^n |y_j - \hat{y}_j|$$

S.No	Model	MAE
1	Linear Regression	0.0353
2	RF Regressor	0.0256
3	Light GBM	0.0205

Linear regression sets the benchmark or baseline performance for the PUBG dataset problem and scores the least in terms of performance as it has the highest MAE. The tree-based algorithm, Random forest regressor does much better by lowering it’s MAE to 0.0256. The

best model we have is the gradient-boosting Light GBM algorithm which performs incredibly well by scoring the lowest MAE of 0.0205 and getting the best performance.

4. Final Results and Interpretation

The Light GBM model got the best E_{out} of 0.0205 on the PUBG Kaggle competition data set. The system was trained on the following parameters to achieve the above E_{out} or MAE:

Light GBM Parameters:

1. **Objective:** Type of learning task. Regression in our case.
2. **Metric:** Performance measure. MAE here.
3. **n_estimators:** number of trees in the model. Here we have set it to 18000 to get a good accuracy.
4. **early_stopping:** This can boost analysis time by halting if accuracy of the validation data does not improve. We use 180 rounds here.
5. **num_leaves:** The count of the leaves in the whole tree. We set it here to 21 to prevent overfitting, by default it is 31.
6. **learning_rate:** Determines the rate of change in the learning rate. The smaller the learning rate and the more the iterations, the better will be the accuracy. Here we set it to 0.05.
7. **bagging_fraction:** sets the amount of data to be taken for each round and can be very helpful in decreasing training time so as to avoid overfitting. We set it to 0.7 in our model.

Below are the MAE for the train and validation data sets for each iteration after training the Light GBM model.

Iterations	MAE Train	MAE Validation
1000	0.0283149	0.0289317
2000	0.0270329	0.0282068
3000	0.0262151	0.0279032
4000	0.0255496	0.0277235
5000	0.0249809	0.0276105

6000	0.0244836	0.0275341
7000	0.0240213	0.0274711
8000	0.0235885	0.0274273
9000	0.0231791	0.0273794
10000	0.0227967	0.0273489
11000	0.0224268	0.0273174
12000	0.0220802	0.0272913
13000	0.0217434	0.0272699
14000	0.0214174	0.027251
15000	0.0211142	0.0272396
16000	0.020807	0.0272282
17000	0.0205098	0.0272186






Figure 9: Table showing training and validation MAE results for different iterations

Early stopping, best iteration is:

[1700] Average of Train MAE: 0.019769 , Average of Validation MAE: 0.0271842

Hence, Light GBM showed superior performance over the baseline linear regression model and the intermediate tree-based random forest model.

I scored 104 when I had initially submitted but now my ranking has climbed down 4 spots to 108. The score with username: **ishanmohanty**, is attached for reference below.

107	new	cantguessit		0.0205	3	3d
108	new	ishanmohanty		0.0205	8	7h
Your Best Entry  Your submission scored 0.0205, which is not an improvement of your best score. Keep trying!						
109	▼ 31	Martin Liu		0.0205	28	6d
110	▼ 30	Ranojoy Barua		0.0205	43	18d

The following kernel links are displayed below:

1. Linear regression: <https://www.kaggle.com/ishanmohantym/trial-by-linear-regression-version>
2. Random forest: <https://www.kaggle.com/ishanmohantym/trial-with-randomforest-regressor>
3. Light GBM: <https://www.kaggle.com/ishanmohantym/trial-by-lgbm/notebook>
4. EDA: <https://www.kaggle.com/ishanmohantym/eda-basic-version/edit>

5. Summary and conclusions

We have analyzed and studied the effective behavior of gradient boosting techniques such as Light GBM and have been able to verify its superior performance compared to simple regression models like linear regression and complex tree-based models like random forest regression. The best MAE for the Light GBM is 0.0205. Hence, we conclude that Light GBM is the best model that could be used for predicting 'winPlacePerc' for the PUBG dataset.

Interesting results can be seen if we try gradient boosting techniques like XGBOOST and also try to play with the parameters further in Light GBM with powerful hardware resources at our disposal. These are the approaches we can consider for future improvements of these existing models.

6. References

- [1] <https://www.kaggle.com/c/pubg-finish-placement-prediction> -- kaggle dataset competition
 - [2] <https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc> --- Light GBM analysis
 - [3] <https://thedataclass.com/2018/04/17/random-forest/> --- Random forest diagram
 - [4] <https://syncedreview.com/2017/10/24/how-random-forest-algorithm-works-in-machine-learning/> --- Random Forest flow chart help
 - [5] <http://www.sthda.com/english/articles/40-regression-analysis/167-simple-linear-regression-in-r/> --- linear regression diagram
 - [6] <https://www.kaggle.com/chocozzz/lightgbm-baseline> - by Hyun woo kim
- This wonderful kernel helped me understand the approach and enabled me to build my code on top of this kernel.
- [7] <https://www.kaggle.com/rejasupotaro/effective-feature-engineering> --- referred to this kernel for doing EDA

[8] <https://www.kaggle.com/deffro/eda-is-fun> --- referred to this kernel for doing EDA