PREDICTION OF PUBG Finish Placement Prediction

Jin Zhang

zhang.jin2@husky.neu.edu

INFO 7245, Fall 2018, Northeastern University

1. Abstract

Player Unknow's Battle Grounds was launched in December 2017 and It becomes quite popular after several months. And I have been playing this game for a while. When I see this dataset, It got my attention immediately. I would like to go deeper and explore this dataset and see how the masterpiece players dominate this game.

Also, after the popularity of this game. now it is dying. maybe through this project, might give me some idea why this phenomenon is happening.

In this paper, my project will involve with a measurement on things like heals, killPoints, kills, longestKill, vehicleDestroys, winPoints, and other factors predict the winPlacePct which is the rank of a team. And an index will be generated for each factor, which could be seen as a guide to the team on where to put they emphasize during the match. Then a deep learning model will be implemented by several steps, the over 65,000 games' worth of anonymized player data will be reshaped in order to fit in the Machine Learning model.

I will apply several Machine Learning model including LightGBM, XGBoost, and Random Forest then I will create a Stacked Ensemble Super-Model to the dataset from a competition held by PUBG Finish Placement Prediction.

2. Introduction

Context

You are given over 65,000 games' worth of anonymized player data, split into training and testing sets, and asked to predict final placement from final in-game stats and initial player ratings.

What's the best strategy to win in PUBG? Should you sit in one spot and hide your way into victory, or do you need to be the top shot? Let's let the data do the talking!

Challenge

In a PUBG game, up to 100 players start in each match (matchId). Players can be on teams (groupId) which get ranked at the end of the game (winPlacePerc) based on how many other teams are still alive when they are eliminated. In game, players can pick up different munitions, revive downed-but-not-out (knocked) teammates, drive vehicles, swim, run, shoot, and experience all of the consequences -- such as falling too far or running themselves over and eliminating themselves.

You are provided with a large number of anonymized PUBG game stats, formatted so that each row contains one player's post-game stats. The data comes from matches of all types: solos, duos, squads, and custom; there is no guarantee of there being 100 players per match, nor at most 4 player per group.

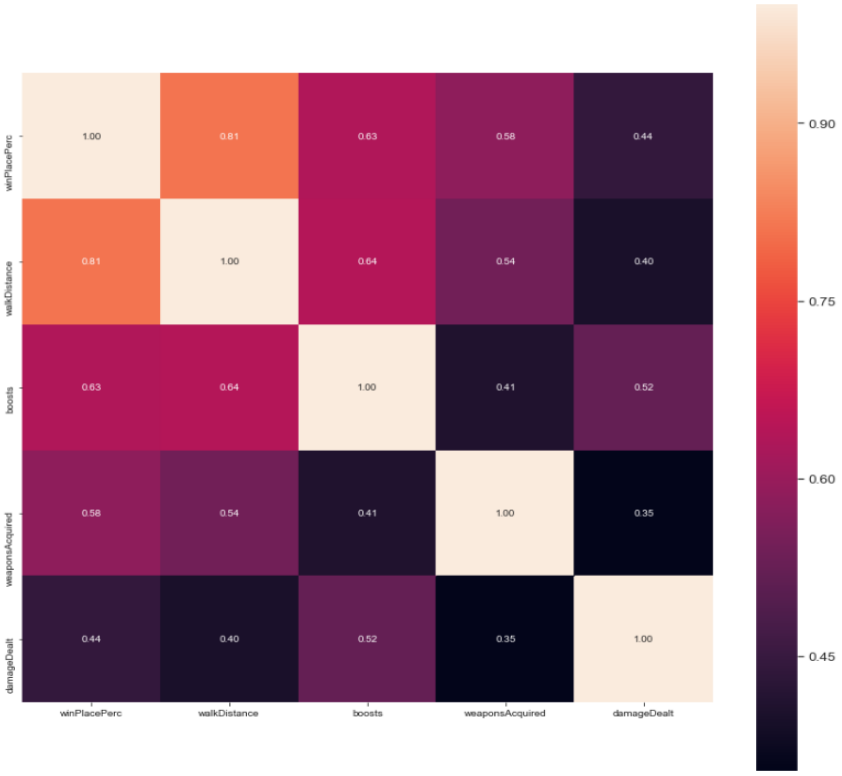
You must create a model which predicts players' finishing placement based on their final stats, on a scale from 1 (first place) to 0 (last place).

3. Code with Documentation

Since our entire dataset is filled with numeric value, which players' finishing placement. I identify this problem to be a supervised learning Regression problem that can be best solved using Regression methods.

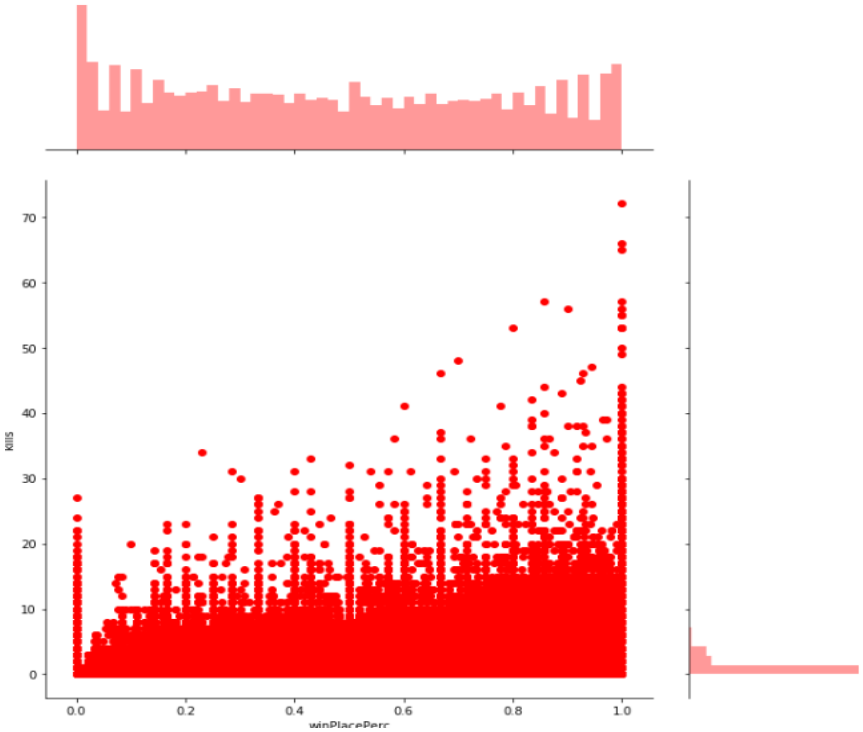
3.1 Exploratory Data Analysis

Exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task. [12]



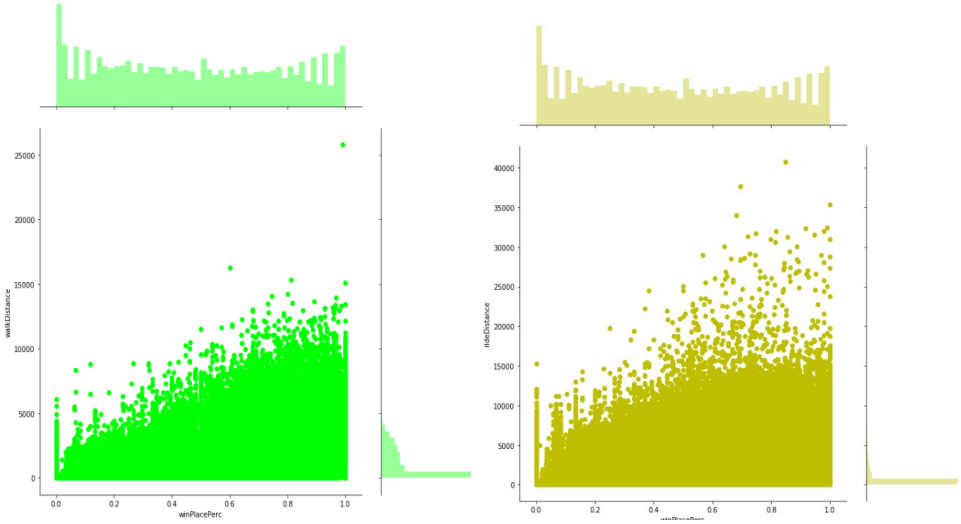
To understand the correlation between different parameters present in our dataset, we plotted a heat map and discovered that the correlation between the different parameters is not very strong.

We then plotted killing info. kills has a correlation with winning. more kills the `winPlacePerc` becomes higher. However, there are some outliers. Although those player got many kills but they didn't win. Maybe too focus on killing.



We then conducted analysis on the ‘move distance’ of the player.

Apparently walking has a high correlation with winPlacePerc.

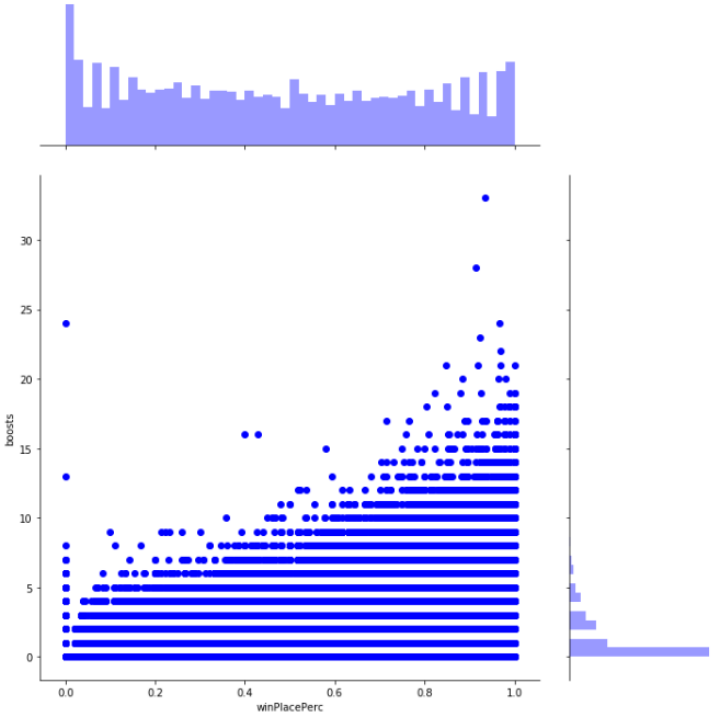


Apparently walking has a high correlation with `winPlacePerc.

There is a small correlation between rideDistance and winPlacePerc. But is not significant as the runner does, the slope is more gentle as previous plot

And there is another fact about the vehicle, In the game, if you destroy a vehicle, you will kill the player inside it as well. this means you got more kills this fact should increase the percentage of winning, Let's check. Is this true or not.

Then let’s take a look at the healing and boosting



This is obviously the healing and boosting, definitely are correlated with winPlacePerc. Boosting is more.

In every plot, there is an abnormal behavior when values are 0. This case, that player simplely like a waste the healing resources. I don't know.

And in this game the The matching mode in the game will also greatly affect the winPlacePerc.

There are 3 game modes in the game.

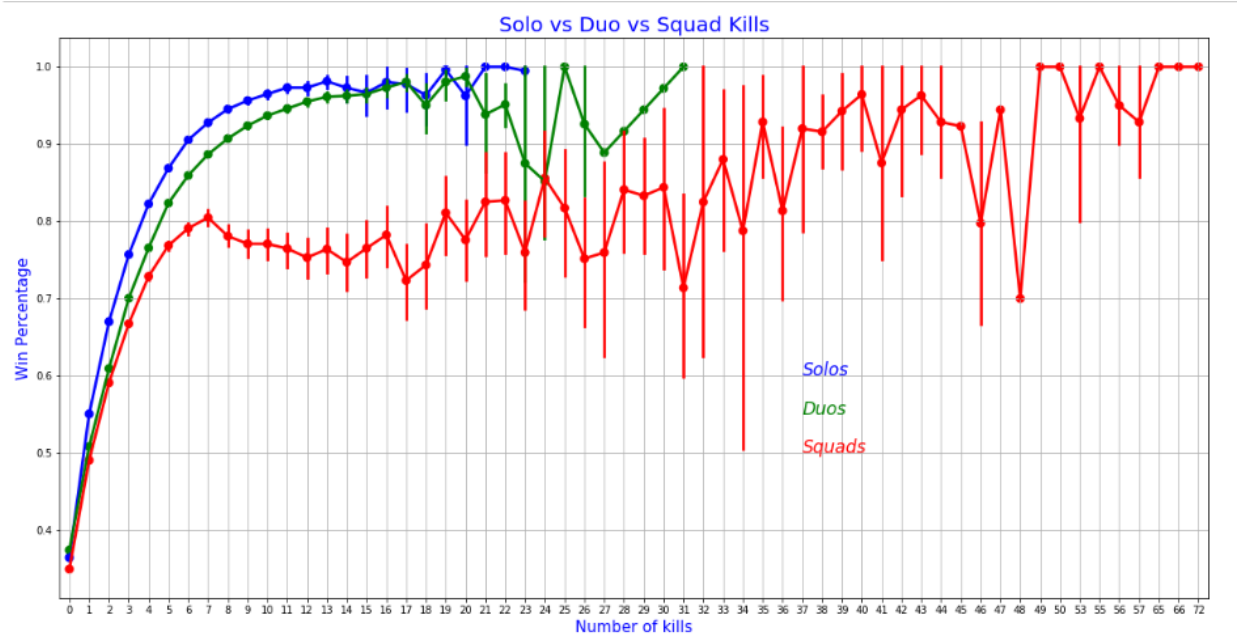
One can play solo,

Or with a friend (duo),

Or with 3 other friends (squad).

100 players join the same server, so in the case of duo the max teams are 50 and in the case of squads the max teams are 25

I plot the behavior of all model to see the different.



Very interesting. Solos and duos behave the same, but when playing squads kills don't matter that much.

The attribute DBNOs means enemy players knocked. A "knock" can happen only in duos or squads because the teammates have the chance to "revive" the knocked player in a given time. So a knocked player can be revived or die. If he is revived, the next time he will be knocked, his teammates will have less time to revive him.

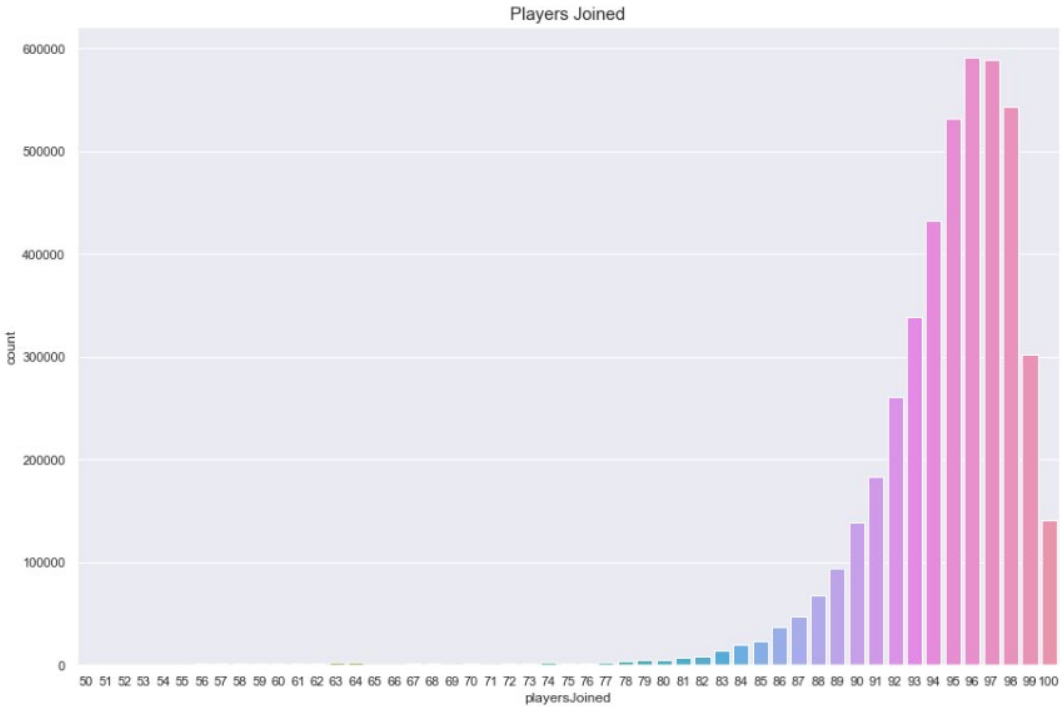
The attribute assist can also happen only in duos or squads. It generally means that the player had an involvement in a kill.

The attribute revive also happens in duos or squads.

3.2 Feature Engineering

After I done the EDA part, It is critical important to do extract the high correlate feature from the dataset.

A game in PUBG can have up to 100 players fighting each other. But most of the times a game isn't "full". There is no variable that gives me the number of players joined. I create one for this case.



Based on the playersJoined feature we can create (or change) a lot of others to normalize their values.

This is a really good assumption.

create the killsNorm and damageDealtNorm features.

When there are 100 players in the game it might be easier to find and kill someone, than when there are 90 players.

This case, normalize the kills in a way that a kill in 100 players will score 1 (as it is)

In 90 players it will score (100-90)/100 + 1 = 1.1

```
df_train['killsNorm'] = df_train['kills']*((100-df_train['playersJoined'])/100 + 1)
df_train['damageDealtNorm'] = df_train['damageDealt']*((100-df_train['playersJoined'])/100 + 1)
df_train[['playersJoined', 'kills', 'killsNorm', 'damageDealt', 'damageDealtNorm']][5:8]
```

	playersJoined	kills	killsNorm	damageDealt	damageDealtNorm
5	95	1	1.05	100.000	105.00000
6	97	0	0.00	0.000	0.00000
7	96	0	0.00	8.538	8.87952

- **Another simple feature is the sum of `heals` and `boosts`.**
- **Also the sum of `total distance` travelled.**

```
df_train['healsAndBoosts'] = df_train['heals']+df_train['boosts']
df_train['totalDistance'] = df_train['walkDistance']+df_train['rideDistance']+df_train['swimDistance']
```

When using boosting items you run faster. They also help to stay out of the zone and loot more (meaning walking more).

So l create a feature boosts per walking distance.boostsPerWalkDistance.

Heals don't make you run faster, but they also help to stay out of the zone and loot more.

I create the same feature for heals also.healsPerWalkDistance

And for both heals and boost: healsAndBoostsPerWalkDistance.

After I done a lot of similar work, finally I can go for the modeling part.

3.3 Modeling

I used for powerful algorithm. To train my dataset. Flowing are the details of them. First , Let me introduce the Evaluation method.

MAE: Mean Absolute Error

- Submissions are evaluated on Mean Absolute Error between your predicted winPlacePerc and the observed winPlacePerc.

The Mean Absolute Error is given by:

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}$$

Lasso (L1)

- Regularization consists in adding a penalty on the different parameters of the model to reduce the freedom of the model. Hence, the model will be less likely to fit the noise of the training data and will improve the generalization abilities of the model. Lasso or L1 has the property that is able to shrink some of the coefficients to zero. It adds penalty equivalent to absolute value of the magnitude of coefficients. Therefore, that feature can be removed from the model.

- This model may be very sensitive to outliers. in order to made it more robust on them.
- I use the sklearn's Robustscaler() method on pipeline
- See how this model works

In [23]:
```
m_lasso = make_pipeline(RobustScaler(), Lasso(alpha =0.0005, random_state=1))
m_lasso.fit(X_train, y_train)
print_score(m_lasso)
```

```
['MAE Train: ', 0.08945645052839803, 'MAE Validation: ', 0.08968774850994174]
```

Random decision forests

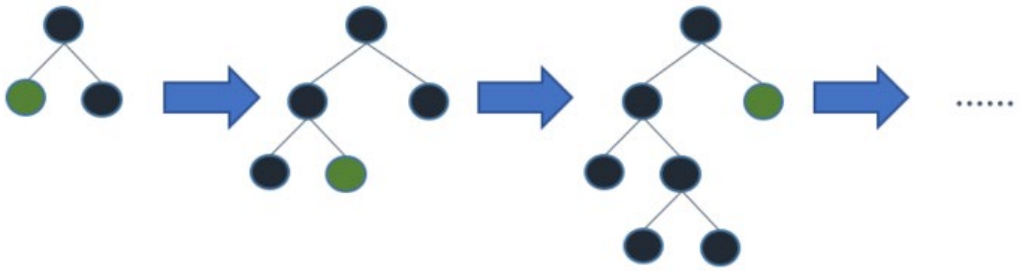
- This is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

In [32]:
```
### Train basic model
m_rf = RandomForestRegressor(n_estimators=40, min_samples_leaf=3, max_features='sqrt',n_jobs=-1)
m_rf.fit(X_train, y_train)
print_score(m_rf)
```

```
['MAE Train: ', 0.03982842659005754, 'MAE Validation: ', 0.0635736304816094 7]
```

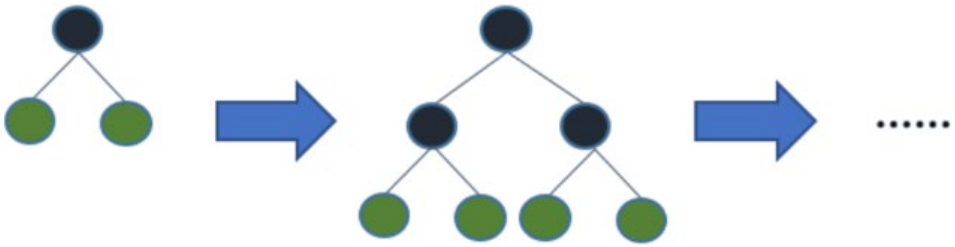
Light GBM

- Light GBM is a gradient boosting framework that uses tree based learning algorithm.
- How it differs from other tree based algorithm?
- Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.
- Below diagrams explain the implementation of LightGBM and other boosting algorithms.



Leaf-wise tree growth

Explains how LightGBM works



Level-wise tree growth

Why Light GBM is gaining extreme popularity?

- The size of data is increasing day by day and it is becoming difficult for traditional data science algorithms to give faster results. Light GBM is prefixed as 'Light' because of its high speed. Light GBM can handle the large size of data and takes lower memory to run. Another reason of why Light GBM is popular is because it focuses on accuracy of results. LGBM also supports GPU learning and thus data scientists are widely using LGBM for data science application development.
- Can we use Light GBM everywhere?
- No, it is not advisable to use LGBM on small datasets. Light GBM is sensitive to overfitting and can easily overfit small data. There is threshold on the number of rows but my experience suggests me to use it only for data with 10,000+ rows.

In [33]:

```
m_lgb = lgb.LGBMRegressor(objective='regression',num_leaves=5,
                          learning_rate=0.05, n_estimators=720,
                          max_bin = 55, bagging_fraction = 0.8,
                          bagging_freq = 5, feature_fraction = 0.2319,
                          feature_fraction_seed=9, bagging_seed=9,
                          min_data_in_leaf =6, min_sum_hessian_in_leaf = 11)
m_lgb.fit(X_train, y_train)
print_score(m_lgb)
```

```
['MAE Train: ', 0.06521944013909331, 'MAE Validation: ', 0.06525478522828515]
```

XGBoost

- XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.
- Why Use XGBoost?
- The two reasons to use XGBoost are also the two goals of the project:
- Execution Speed.

In [34]:

```
m_xgb = xgb.XGBRegressor(colsample_bytree=0.4603, gamma=0.0468,
                         learning_rate=0.05, max_depth=3,
                         min_child_weight=1.7817, n_estimators=2200,
                         reg_alpha=0.4640, reg_lambda=0.8571,
                         subsample=0.5213, silent=1,
                         random_state =7, nthread = -1)
m_xgb.fit(X_train, y_train)
print_score(m_xgb)
```

```
['MAE Train: ', 0.05604655303741984, 'MAE Validation: ', 0.05667289773881197]
```

The objective of my work is creating a Stacked Ensemble Super-Model. To improve the performance of all my models

Stacking models

- Simplest Stacking approach: Averaging base models
- I begin with this simple approach of averaging base models.
- I build a new class to extend scikit-learn with our model and also to laverage encapsulation and code reuse (inheritance)
- Averaged base model class

In [44]:

```
class AveragingModels(BaseEstimator, RegressorMixin, TransformerMixin):
    def __init__(self, models):
        self.models = models

    # we define clones of the original models to fit the data in
    def fit(self, X_train, y_train):
        self.models_ = [clone(x) for x in self.models]

        # Train cloned base models
        for model in self.models_:
            model.fit(X_train, y_train)

        return self

    #Now we do the predictions for cloned models and average them
    def predict(self, X_valid):
        predictions = np.column_stack([
            model.predict(X_valid) for model in self.models_
        ])
        return np.mean(predictions, axis=1)
```

- Averaged base models score.
- Because the L1 model did not do will.
- I just average three models here Random forest, XGBoost,and LightGBM.

```
In [42]:
averaged_models = AveragingModels(models = (m_xgb , m_lgb , m_rf))

averaged_models.fit(X_train, y_train)
# m_xgb.fit(X_train, y_train)
print_score(averaged_models)
# score = print_score(averaged_models)

# print(" Averaged base models score: {:.4f} ({:.4f})\n".format(score.mean(), score.std()))
['MAE Train: ', 0.05179337622463965, 'MAE Validation: ', 0.05970013212909626]
```

My stacking model improve the score a little bit.

Finally, I get the results of all my model. And make a good prediction.

4. Results

Since this is a regression problem, I worked on my dataset with multiple models and preset their hyperparameters. The accuracy from each of the methods along with the hyperparameters used is mentioned in the table below

#	Method	Hyperparameters	Accuracy
1	Random forest	n_estimators=40, min_samples_leaf=3, max_features='sqrt',n_jobs=-1	70%
2	Lasso(L1)	RobustScaler(), Lasso(alpha =0.0005, random_state=1	65.30%
3	LightGBM	objective='regression',num_leaves=5, learning_rate=0.05, n_estimators=720, max_bin = 55, bagging_fraction = 0.8, bagging_freq = 5, feature_fraction = 0.2319, feature_fraction_seed=9, bagging_seed=9, min_data_in_leaf =6, min_sum_hessian_in_leaf = 11	79.50%
4	XGboost	colsample_bytree=0.4603, gamma=0.0468, learning_rate=0.05, max_depth=3, min_child_weight=1.7817, n_estimators=2200, reg_alpha=0.4640, reg_lambda=0.8571, subsample=0.5213, silent=1, random_state =7, nthread = -1	82.7%
5	Stacking Super Model	m_xgb , m_lgb , m_rf , m_L1	80%
		m_xgb , m_lgb , m_rf	84.70%

5. Discussion

From EDA, I was able to conclude that working distance is the most relevant feature for the winning percentage. The features that are most helpful to this prediction

In this paper, I implemented four different supervised regression models to provide solution for my regression problem of predicting players' finishing placement. I experimented with various

hyperparameters of all the models to achieve a better accuracy score. As compared to Kaggle's best accuracy score of 82%, we have achieved accuracy score of 84% with Stacking Super Model.

Future work:

- Run the model on a lager dataset.
- Improve the stacking method.
- Apply DNN to my regression problem.
- Creating a recommendation system for my research, to show what strategy can help player to win the game etc.

Acknowledgement

Thanks for the guidance and support of the professor.

Thanks for the guidance and support of TA.

6. Referance

[1] https://www.kaggle.com/c/pubg-finish-placement-prediction/data

[2] https://www.kaggle.com/deffro/eda-is-fun

[3] https://en.wikipedia.org/wiki/Mean_absolute_error

[4] https://www.kaggle.com/c/pubg-finish-placement-prediction#evaluation

[5] https://www.kaggle.com/raviprakash438/lasso-and-ridge-regularisation

[7] https://en.wikipedia.org/wiki/Random_forest

[8] https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

[9] https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

[10] http://deeplearning.net/tutorial/mlp.html

[11] http://www.helsinki.fi/~ahonkela/dippa/node41.html

[12] https://en.wikipedia.org/wiki/Exploratory_data_analysis

[13] http://www.statsoft.com/Textbook/Support-Vector-Machines

[14] https://www.kaggle.com/lusob04/titanic-rnn