Using Random Forest and Neural Network to predict win placement of PUBG

Xiaoyu Wang
Department of Electrical and Computer Engineering
Texas A&M University
College Station, Texas
wxy83320485@tamu.edu

Abstract

PlayerUnknown's Battlegrounds(PUBG) is a battle royale game. 100 players are dropped onto an island empty-handed and must explore, scavenge, and eliminate other players until only one is left standing, all while the play zone continues to shrink. I used Random Forest and Neural Network to predict player's win placement percent based on their gaming features, such as kill numbers, assists, heal items used numbers, etc. First, I obtained the data from Kaggle and described the variables. Second, simple exploratory data analysis(EDA) are applied to summarize the main features and find the most related features. Third, I selected features and divided original data into training and testing set for later experiments through feature engineering. Finally, percentile win placement were predicted by random forest and neural network, and the two results are analyzed and compared.

1 Introduction

Battle Royale-style video games have taken the world by storm. 100 players are dropped onto an island empty-handed and must explore, scavenge, and eliminate other players until only one is left standing, all while the play zone continues to shrink.

PlayerUnknown's BattleGrounds(PUBG) is an online multiplayer battle royale game developed and published by PUBG Corporation. The game was first fully released on December 20, 2017. Up to now, it has enjoyed massive popularity. With over 50 million copies sold, it's the fifth best selling game of all time, and has millions of active monthly players.

In this game, lots of features have an effect on players' win placement, not only shooting skills, but also driving skills, hiding positing selection, offensive and defensive strategy. Even some coincidence may change your "life". One of the most important reasons that leads to PUBG's success is its uncertainty. Although there is a lot randomness in this game, we can analyze the players data and using some related features to predict the win placement through AI and data mining knowledge.

In this paper, I propose to apply Random Forest and Neural Network to predict player's win placement percent based on their gaming features, such as kill numbers, assists, heal items used numbers, etc.

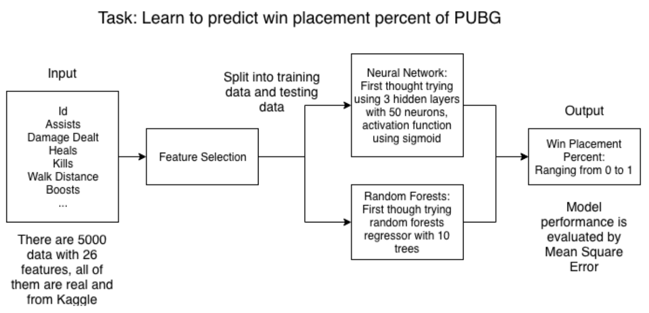

Task: Learn to predict win placement percent of PUBG
Input
Id
Assists
Damage Dealt
Heals
Kills
Walk Distance
Boosts
...
There are 5000 data with 26 features, all of them are real and from Kaggle
Feature Selection
Split into training data and testing data
Neural Network: First thought trying using 3 hidden layers with 50 neurons, activation function using sigmoid
Random Forests: First though trying random forests regressor with 10 trees
Output
Win Placement Percent: Ranging from 0 to 1
Model performance is evaluated by Mean Square Error

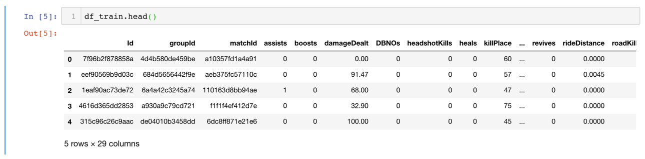
Figure 1. Program Architecture

Above is the architecture of the system, first I retrieved about 5000 data from Kaggle. Kaggle provides 4.36 real players' gaming data obtained from PUBG official developer API. For beginning, I will choose 5000 thousand of

them. 4000 will form training set and the rest 1000 will form testing set. Each tuple contains 29 features, details about data features will be introduced in Related Work section. Then I used exploratory data analysis and feature selection to filter unrelated features. I used Python library numpy, pandas and seaborn to check correlation between features, handle the missing value and group some features together to reduce noise and avoid overfitting. After that I applied neural network, which contains 3 hidden layers and each with 50 neurons, and random forest, which was first set to 10 trees for regression, to learn and predict the win placement. Keras will be used to build multi-layer neural network and Sklearn will be used to build random forest regressor. Mean square error is used to evaluate the predicted results. The values can range from 0(perfect) to 1(worst possible) since that all predictions are int he interval [0, 1]. Finally, I will analyze the prediction results and compare the performance between two methods, including time cost and accuracy.
The paper contains following parts: Abstract, Introduction, Related Work, AI Approach, Implementation, Evaluation, Lessons Learned and Conclusion.

2 Related Work

As mentioned before, training and testing data are provided by Kaggle. The original data set is a .csv file which contains 4.36 million gaming data from random players in PUBG. Each tuple contains 29 attributes.



```
In [5]:  df_train.head()
Out[5]:
```

	Id	groupId	matchId	assists	boosts	damageDealt	DBNOs	headshotKills	heals	killPlace	...	revives	rideDistance	roadKill
0	7f96b2f878858a	4d4b580de459be	a10357fd1a4a91	0	0	0.00	0	0	0	60	...	0	0.0000	
1	eef90569b9d03c	684d5656442f9e	aeb375fc57110c	0	0	91.47	0	0	0	57	...	0	0.0045	
2	1eaf90ac73de72	6a4a42c3245a74	110163d8bb94ae	1	0	68.00	0	0	0	47	...	0	0.0000	
3	4616d365dd2853	a930a9c79cd721	f1f1f4ef412d7e	0	0	32.90	0	0	0	75	...	0	0.0000	
4	315c96c26c9aac	de04010b3458dd	6dc8ff871e21e6	0	0	100.00	0	0	0	45	...	0	0.0000	

5 rows × 29 columns

Figure 2. Data Architecture

Above is part of the data from .csv file. “winPalcePerc" is the target feature. This is a percentile winning placement, where 1 corresponds to 1st place, and 0 corresponds to last place in the match. It is calculated off of maxPlace, not numGroups, so it is possible to have missing chunks in a match. The rest 28 attributes are potential factors affecting the target feature. Kaggle names each attribute and gives explanation as below:
“DBNOs”: Number of enemy players knocked. “assists”: Number of enemy players this player damaged that were killed by teammates. “boosts”: Number of boost items used. “damageDealt”: Total damage dealt. Note: Self inflicted damage is subtracted. “headshotKills”: Number of enemy players killed with headshots. “heals”: Number of healing items used. “Id”: Player’s Id. “killPlace”: Ranking in match of number of enemy players killed. “killPoints”: Kills-based external ranking of player. If there is a value other than -1 in rankPoints, then any 0 in killPoints should be treated as a “None”. “killStreaks”: Max number of enemy players killed in a short amount of time. “kills”: Number of enemy players killed. “longestKill”: Longest distance between player and player killed at time of death. This may be misleading, as downing a player and driving away may lead to a large longestKill stat. “matchDuration”: Duration of match in seconds. “matchId”: ID to identify match. There are no matches that are in both the training and testing set. “matchType”: String identifying the game mode that the data comes from. The standard modes are “solo”, “duo”, “squad”, “solo-fpp”, “duo-fpp”, and “squad-fpp”; other modes are from events or custom matches. “rankPoints”: Elo-like ranking of player. This ranking is inconsistent and is being deprecated in the API’s next version, so use with caution. Value of -1 takes place of “None”. “revives”: Number of times this player revived teammates. “rideDistance”: Total distance traveled in vehicles measured in meters. “roadKills”: Number of kills while in a vehicle. “swimDistance”: Total distance traveled by swimming measured in meters. ”teamKills”: Number of times this player killed a teammate. “vehicleDestroys”: Number of vehicles destroyed. “walkDistance”: Total distance traveled on foot measured in meters. “weaponsAcquired”: Number of weapons picked up. “winPoints”: Win-based external ranking of player. (Think of this as an Elo ranking where only winning matters.) If there is a value other than -1 in rankPoints, then any 0 in winPoints should be treated as a “None”. “groupId”: ID to identify a group within a match. If the same group of players plays in different matches, they will have a different groupId each time. “numGroups”: Number of groups we have data for in the match. “maxPlace”: Worst placement we have data for in the match. This may not match with numGroups, as sometimes the data skips over placements.
For EDA and feature engineering part, I referenced Rejasupotaro’s “Effective Feature Engineering” and Dimitrios Effrosynidis’s “EDA is Fun”. Rejasupotaro created a Kaggle Kernel and finished correlation, feature importances of Tree models, permutation importance, SHAP values, score gain on a simple model to select and group most related features to get the balance between accuracy and efficiency. Dimitrios Effrosynidis used statistical methods to analyze five common players in PUBG: killers, runners, drivers, swimmers, and healers. Then he studied different impact on win placement between different models: solo, duo and squad.

3 AI Approach

The task is to predict player's win placement percent based on their features, such as kill numbers, assists, heal items used numbers, etc. It is a typical supervised learning problem, where there is a set of examples and a set of labeled features, partitioned into input features and target features. Since the prediction value tends to be a continuous value ranging from 0 to 1, it is a regression type problem under supervised learning.
There are various models or methods for solving supervised problem. For basic models, there are learning decision tress and linear regression. For composite models, there are random forests and gradient boosting, and also, artificial neural network is good to solve this problem. Finally, I chose random forest and neural network to predict the win placement. Below is my analysis and comparison between different methods:

Regression tree(prediction tree)

Regression tree(prediction tree):The most common method for constructing regression tree is CART(Classification and Regression Tree), which is also known as recursive partitioning. The method starts by searching for every distinct values of all its predictors, and splitting the value of a predictor that minimizes the following statistic:

$$SSE = \sum_{i \in S_1} (y_i - \overline{y_1}) + \sum_{i \in S_2} (y_i - \overline{y_2})$$

For group S_1 and S_2, the method will recursively split the predictor values within groups. In practice, the method stops when the sample size of the split group falls below certain threshold. However, decision tree is unstable, a small change can lead to a large change in the structure of the optimal tree. Also, they are relatively inaccurate, it can be remedied by replacing a single decision tree with a random forest of decision trees, which has less explainability than simple decision tree.

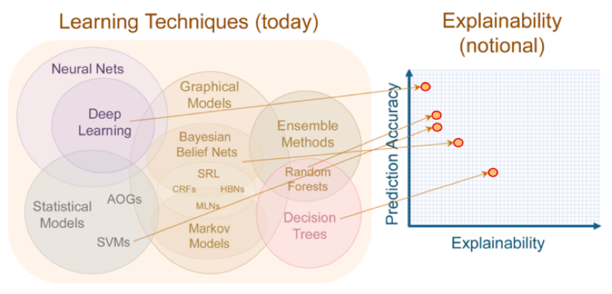


Figure 3. Model Explainability vs. Power. Credit: iapp

Linear regression

Linear regression is the problem of fitting a linear function to a set of training examples, in which the input and target features are numeric. However, it is limited to linear relationship, and it only looks at the mean of the dependent variable. Also, it has relatively less accuracy than other composite models.
The above figures shows the relative accuracy and explainability of different learning techniques. Basic model like decision tree and linear regression usually have good explainability but lack accuracy compared with more complex models. Good explainability means model has fewer parameters and is easier to interpret.
Decision trees, and linear functions provide the basis for many other supervised learning techniques.

Random forest

Random forest, based on simple decision tree, is to have a number of decision trees, each of which can make a prediction on each example, and aggregate the predictions of the trees to make a prediction of the forest each example. Once the trees have been trained, a prediction can use the average of the predictions of the tree for a probabilistic prediction. Alternatively, each tree can vote with its most likely prediction with the most votes can be used.

Gradient boosting

Gradient boosting produces a prediction model in the form of an ensemble of weak prediction models. It builds the model in stage-wise fashion, and it generalizes them by allowing optimization of an arbitrary detterxntiable loss function. Compared with random forest, gradient boosting has pros and cons, it is more sensitive to overfitting if the data is noisy. Since there exists much coincidence and even cheaters in the game, the original data I obtained should be noisy and it is hard to distinguish with these noisy data with others. Besides, it is much more difficult to tune and needs more care in setting up, while random forest is more robust, since boosting is based on weak learners(high bias, low variance) while random forest uses fully grown decision trees(low bias, high variance). Also, random forest can perform netter on small data sets; gradient boosting trees are data hungry. Since I only use 5000 data examples, I chose RM in the first place.

Artificial Neural Network(ANN)

Artificial Neural Network(ANN) is inspired by the information processing model of the mind/brain. The human brain consists of billion of neurons that link one another in an intricate pattern. Every neuron receives information from many other neurons, processes it, gets excited or not, and passes its state information to other neurons. There are many benefits of using ANN: it can deal with high nonlinear relationships without much work from the user or analyst. And there is no need to program the NN since it learns from examples. Importantly, it is tolerant of data quality issues and it does not restrict the data to follow strict normality or independence assumptions. Last but not least, they usually provides better results compared to statical counterparts, once it is trained enough.

Before applying the predictors to the data, we need to preprocess the original data. To learn from data, quality data needs to be efficiently gathered, cleaned and organized. Kaggle gathers the data for me, and there are seven main steps in data cleansing and preparation: removing duplicated data, filling in missing value, comparing data elements, binning continuous values into a few buckets, removing outlier data elements, ensuring that data is representative of the phenomena under analysis by correcting for any biases in the selection of data and increasing information density of data.

Mean square error(MSE) is used to evaluate the performance of the predictors using following function:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \overline{Y}_i)^2$$

3 Implementation

Program preparation

Language: Python 2.7
IDE: Jupyter Notebook 5.7.2
Below are needed Python libraries:
Data analysis library: pandas, numy, scipy, sklearn
Data visualization library: matplotlib, seaborn, plotly
Data set shape: train.csv - (5001, 29)

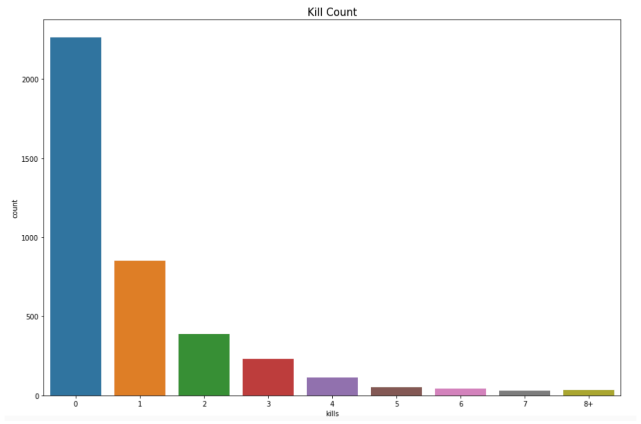


Figure 4. kills distribution

Exploratory Data Analysis

At first, I thought kills should be the most important factor about win placement. By exploring the data, I found that the average person kills is 0.9480 players, 99% of people have 7.0 kills or less, while the most kills ever recorded is 21. Only this cannot confirm that it is firmly related to the target value. Then I plotted the distribution figure of kills, more than 60% of players cannot make a kill during one game, it seems like kills cannot be the only factor. Moreover, 20 players (0.0000%) have won without a single kill and 4 players (0.0000%) have won without dealing damage. Then I used joint plot to see the correlation between kills and win placement:

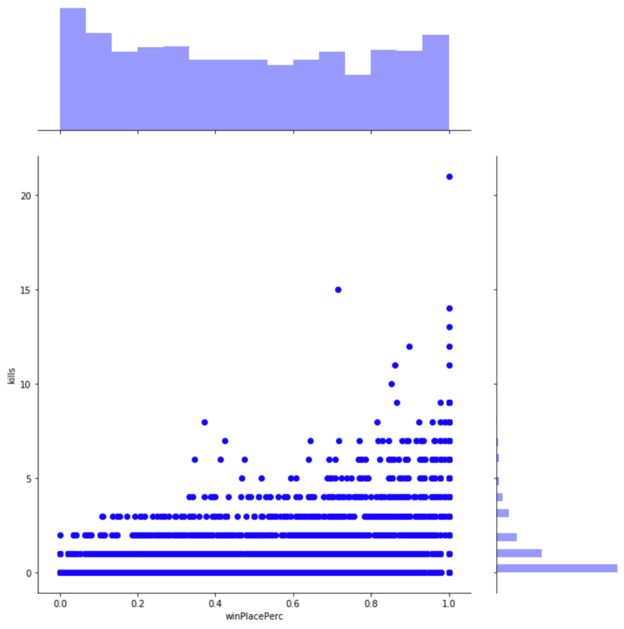


Figure 5. Joint distribution between kills and win placement

From above joint distribution diagram, it is obvious that kills has a correlation with winning. When the number of kills increase, the probability of win placement will increase too. Therefore killing should have a positive effect on winning. However, most people kill no players, the data is too centralized, then I thought kill place - rank of killing players might interpret the relation better. So I plotted the joint distribution between kill place and win placement to see if there is correlation. From the digram, apparently kill place has a greater negative effect on winning, which is reasonable - rank 1 means you are the best killer and rank last 1 means you are a green hand.

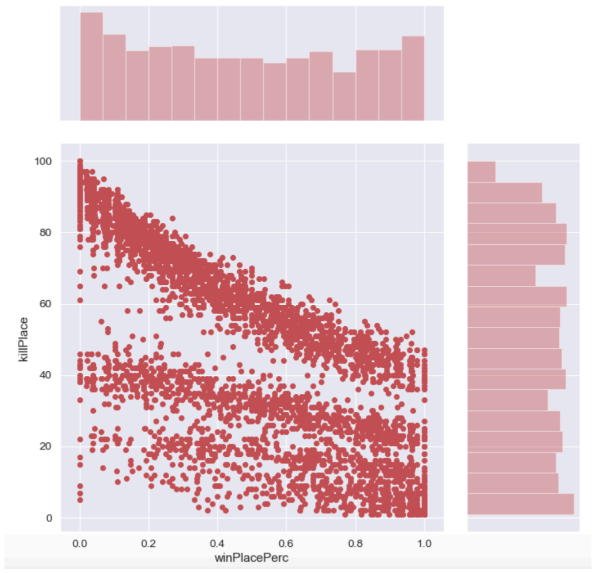


Figure 6. Joint distribution between kill place and win placement

Secondly, walking distance should be an important factor as well. Since the walking speed is invariant, longer distance might mean longer life, which indirectly represents the win placement. According to the training data, the average walking distance is 1146.3m, 99% of players have walked 4250.06m or less, while the marathon champion walked for 9325.0m.

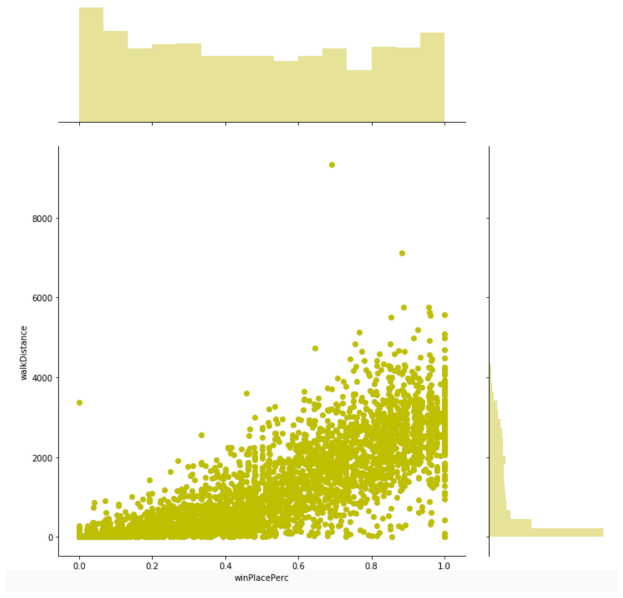


Figure 7. Joint distribution between walking distance and win placement

According to the joint plot of walking distance and win placement, they are correlated as well, walking distance also has a positive effect on winning.

Using the same method, I found that swimming distance and driving distance have small correlation with winning then kills and walking distance. Since the map is an island, and most combats are finished on ground, there are few rivers or vehicles on the island so most players walk to death or winning. However, destroying a vehicle will increase the winning probability dramatically.

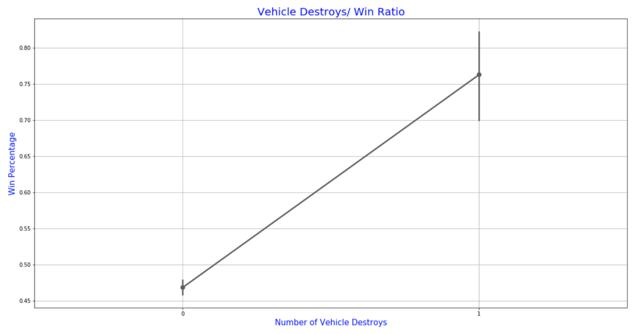


Figure 8. Point plot between vehicle destroy and win placement

Then boosts and healers were considered. Healers are first aid kit and medicine which can recover players' lots HP and heal them continuously. While boosts will give players better weapons and equipments. The average players use 1.4 heal items, 99% of people use 12.02 or less, while the most used 29. The average players use 1.2 boost items, 99% of people use 8.0 or less, while the doctor used 11.

According to the point plot of boosts and healers. Both of them has positive effects on winning, while boost is more correlated.

Next I analyzed the correlation between different features, so similar features can be eliminated or merged. Using seaborn library to plot the heat map between 29 attributes:

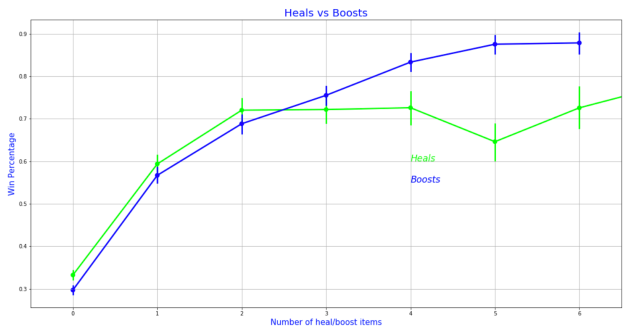


Figure 9. Point plot between healers, boosts and win placement

In terms of the target variable (winPlacePerc), there are a few variables high medium to high correlation. The highest positive correlation is walkDistance which is 0.8 and the highest negative the killPlace which is -0.7. To be more specific, I took a close look on the top 5 most positive features heat map with win placement.

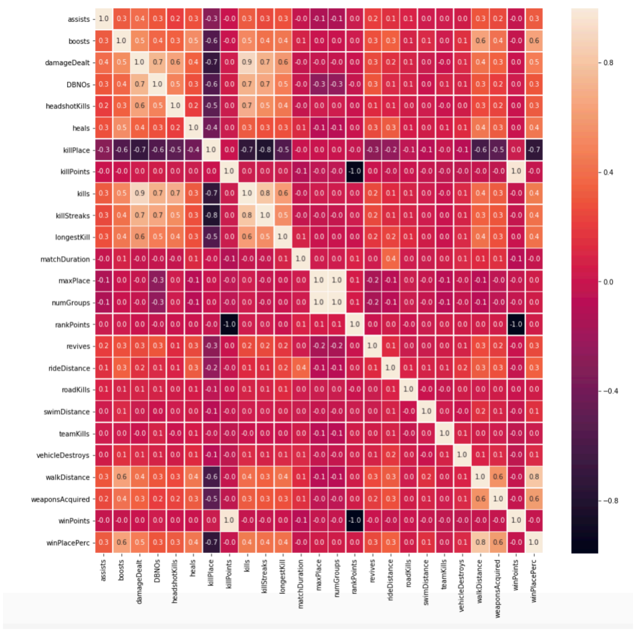


Figure 10. Correlation heat map between 29 attributes

From above diagram, walk distance(0.8), boosts(0.6), weapons acquired(0.6), damage dealt(0.5) and kills(-0,7) are most correlated with winning, which are corresponding to the former joint distribution analysis. The following step is feature engineering.

Feature Engineering

Since I found both boosts and healers are correlated with win placement, I decided to group these two together and create new feature "boostAndheals". Besides, I used dealt damage rather than kills for predicting because they both represent similar attribute - output damage while dealt damage has larger correlation than kills. And also kill place represents kills indirectly. Finally, I chose five features for predicting: walkDistance, boostAndheals, weaponsAcquired, damageDealt and killPlace.
There is no NULL values in the data, which means no missing data.

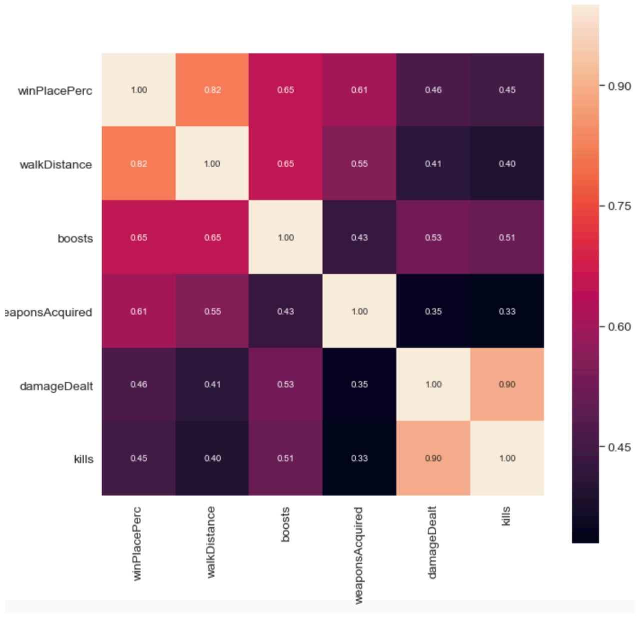
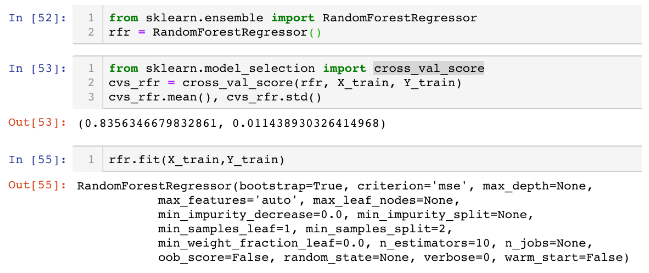


Figure 11. Top 5 factors correlation heat map with win placement

Predictor Model

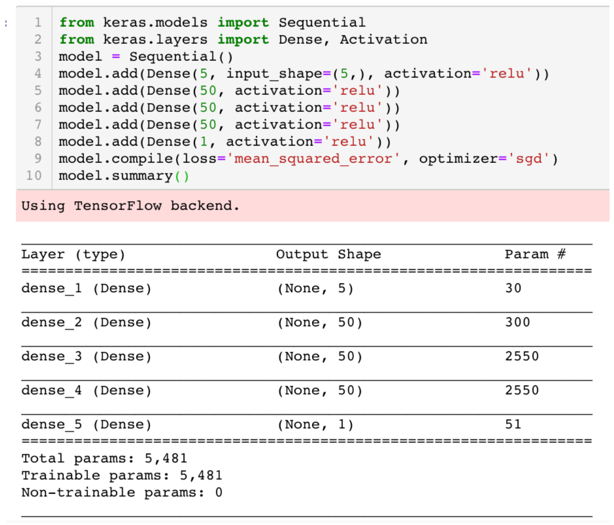
For Random Forest, sklearn.ensemble.RandomForestRegressor are used to generate trees, n_estimators(number of trees in forest) was set to 10, mean square error is used for criterion. After applying sklearn.model_selection.cross_val_score, I got a score of 0.8356346679832861, which means the model and feature selection is correct.



```
In [52]: from sklearn.ensemble import RandomForestRegressor
         rfr = RandomForestRegressor()

In [53]: from sklearn.model_selection import cross_val_score
         cvs_rfr = cross_val_score(rfr, X_train, Y_train)
         cvs_rfr.mean(), cvs_rfr.std()

Out[53]: (0.8356346679832861, 0.011438930326414968)

In [55]: rfr.fit(X_train,Y_train)

Out[55]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                    oob_score=False, random_state=None, verbose=0, warm_start=False)
```

Figure 12. Random Forest Model

For Neural Network, Keras is used to build the network. The model was built in sequential mode, the first layer is input layer, there are five neurons, input shape is (5,). The next three layers are all hidden layers, each with 50 neurons. All the layers are using RELU as activation function.



```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential()
model.add(Dense(5, input_shape=(5,), activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(1, activation='relu'))
model.compile(loss='mean_squared_error', optimizer='sgd')
model.summary()
```

```
Using TensorFlow backend.
```

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 5)                 30
_________________________________________________________________
dense_2 (Dense)              (None, 50)                300
_________________________________________________________________
dense_3 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_4 (Dense)              (None, 50)                2550
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 51
=================================================================
Total params: 5,481
Trainable params: 5,481
Non-trainable params: 0
_________________________________________________________________
```

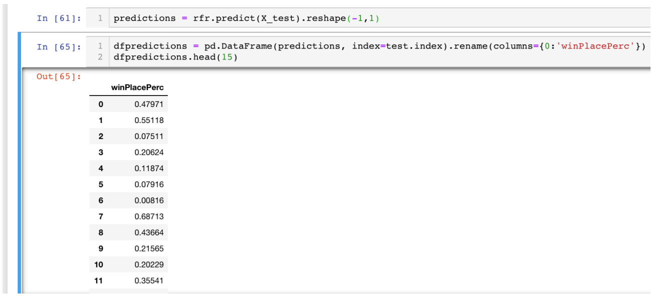
Figure 13. ANN Model

The model is compiled by mean square error, sgd for optimizer.

4 Evaluation

Random Forest

In this experiment, I split the data set into two parts: training data(4000, 80%) and testing data(1000, 20%). During EDA and feature engineering, 5 main features: walkDistance, boostAndheals, weaponsAcquired, damageDealt and killPlace are used for tuning Random Forest. After fitting the training set features and labels, I got the predictions of the test data:



```
In [61]:  predictions = rfr.predict(X_test).reshape(-1,1)

In [65]:  dfpredictions = pd.DataFrame(predictions, index=test.index).rename(columns={0:'winPlacePerc'})
          dfpredictions.head(15)

Out[65]:
```

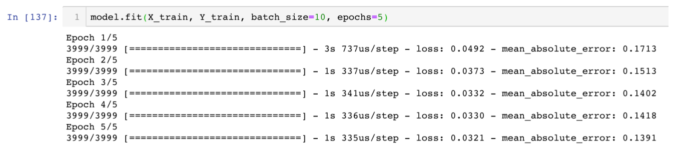
	winPlacePerc
0	0.47971
1	0.55118
2	0.07511
3	0.20624
4	0.11874
5	0.07916
6	0.00816
7	0.68713
8	0.43664
9	0.21565
10	0.20229
11	0.35541

Figure 14. Random Forest Prediction Results

As mentioned before, MSE is used to evaluated model's performance. For these 1000 testing data, the MSE is 0.01430938005544915.

Neural Network

At first, I used the architecture from proposal: model was built in sequential mode, the first layer is input layer, there are five neurons, input shape is (5,). The next three layers are all hidden layers, each with 50 neurons. All the layers are using RELU as activation function. The model is compiled by mean square error, sgd for optimizer. However, the training process is always unsatisfied, the fitting loss is 0.3223 and the testing error is about 0.33. No matter how I changed the hidden layers, neurons, activation function, the result did not change much. After doing more experiments, I found that it was due to overfitting, it occurs when the learner makes predictions based on regularities that appear in the training examples but do not appear in the test examples or in the world from which the data is taken. Then I found two methods to solve that problem: dropout and batch normalization:
Dropout: consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting. Firstly I set dropout value to 0.1, which means after each layer, 10% neurons connection will be cut off. This method helped a lot: in 5 epochs, the fitting loss decreased from 0.3223 to 0.151. However, this result still cannot compare with random forest.
Batch Normalization:it normalizes the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. BN layer has many effects: (1) Accelerate convergence (2) Control over-fitting, can use less or no Dropout and regularity (3) Reduce network insensitivity to initialization weights (4) Allow large learning rate. I added a BN layer next to the hidden layer , the fitting loss decreased dramatically to 0.0321.



```
In [137]:  model.fit(X_train, Y_train, batch_size=10, epochs=5)

Epoch 1/5
3999/3999 [==============================] - 3s 737us/step - loss: 0.0492 - mean_absolute_error: 0.1713
Epoch 2/5
3999/3999 [==============================] - 1s 337us/step - loss: 0.0373 - mean_absolute_error: 0.1513
Epoch 3/5
3999/3999 [==============================] - 1s 341us/step - loss: 0.0332 - mean_absolute_error: 0.1402
Epoch 4/5
3999/3999 [==============================] - 1s 336us/step - loss: 0.0330 - mean_absolute_error: 0.1418
Epoch 5/5
3999/3999 [==============================] - 1s 335us/step - loss: 0.0321 - mean_absolute_error: 0.1391
```

Figure 15. ANN Predictions Results

And For these 1000 testing data, the MSE is 0.02727401074800811.
For the predicted win placement and the testing 28 features, I plotted the correlation heat map again, the result showed the corresponding relation I analyzed in the former

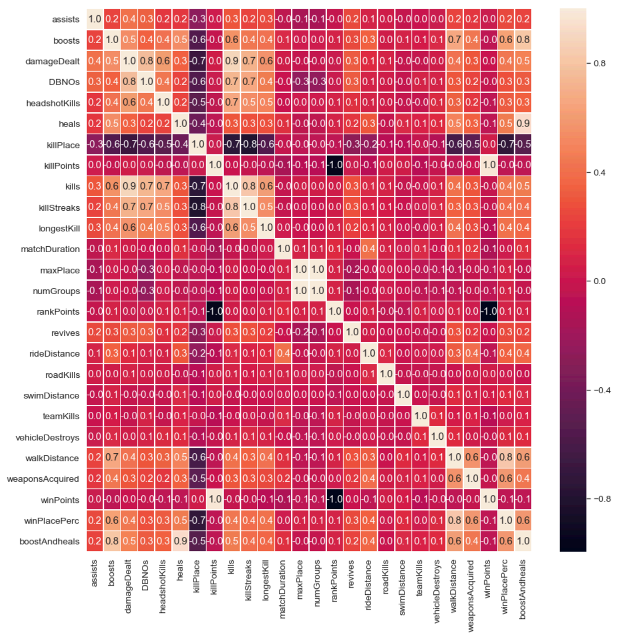


Figure 16. Correlation Heat Map of Testing Data

section: walk distance(0.8), boostAndheasl(0.6), weapons acquired(0.6), damage dealt(0.4) and kills(-0,7) are most correlated with winning, which are corresponding to the former EDA results. Therefore, it is believed that these five attributes affect players' winning in PUBG most.

5 Lesson learned

From this project, I learned a lot from tracking book: Data Analytics Made Accessible.
Data mining and machine learning contains three main part: data preprocessing, supervised learning and result analysis.
In data preprocessing, gathering and selecting data is the first step, to learn from data, quality data needs to be effectively gathered, cleaned and organized and then efficiently mined. Most organizations develop an enterprise data model(EDM) to organize their data. The second step is to clean and prepare data. There are seven main ways for preparation: 1. remove duplicated data, 2. fill missing values, 3. make data elements comparable, 4. bin continuous data into a few buckets, 5. remove outlier data elements, 6. ensure that the data is representative of the phenomena under analysis by correcting for any biases in the selection of data, 7. select data to increase information density. Besides, I learned about exploratory data analysis and feature engineering: EDA is an approach analyzing data sets to summarize their main characteristics, often with visual methods. It is used for seeing what the data can tell us beyond the formal modeling or hypothesis testing task. Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is fundamental to the application of machine learning, and is both difficult and expensive. The need for manual feature engineering can be obviated by automated feature learning. Feature engineering is an informal topic, but it is considered essential in applied machine learning.
In supervised learning, I learned that supervised learning is to predict the values of the target features from the input features based on the given examples. There are lots of models and methods foe solving this problem: decision tree, linear regression, random forest, boosting, artificial neural network. I learned the each model's architecture, pros and cons. During building and training ANN, I learned more about overfitting and how to deal with that. Overfitting occurs when the learner makes predictions based on regularities that appear in the training examples but do not appear in the test examples or in the world from which the data is taken. It typically happens when the model tries to find signal in randomness – there are spurious correlations in the training data that are not reflected in the problem domain as a whole – or when the learner becomes overconfident in its model. Pseudocounts, regularization and cross-validation can be used to detect and avoid overfitting.

6 Conclusion

In this paper, I tackled the problem of predicting the win placement of PUBG based on players gaming data. To achieve this, I adapted Random Forest and Neural Network to handle this problem. Before fitting training data and predicting testing data. I applied simple EDA and feature engineering for data preprocessing. The original data set is obtained from Kaggle, I chose 4000 for training, 1000 for testing. I got 0.0143 MSE for random forest and 0.0272 MSE for neural network.
For future work, I will extend data set, applying more examples for prediction. And I will try other methods such as gradient boosting. Finally, I will try to reduce data processing time and training time.

7 References

David Poole, Alan Mackworth. 2017. *Artificial Intelligence: Foundations of Computational Agents, second edition, Cambridge University Press*

Anil Maheshwari. 2018. *Data Analytics Made Accessible*

Sergey Ioffe, Christian Szegedy. *2015. Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift*

Wikipedia. *exploratory data analysis, feature engineering, mean sure error, gradient boosting, decision tree, playunknown's battlegrounds*