MSBD 5012 Machine Learning

Placement Prediction of Battle Royale Games

2018-12-01

Group 19

Cao Pei
Guo Donghui
Lan Hao
Sun Yimiao
Zhu An Qi

Introduction

For our project, we decided to compete in a kaggle competition. The competition was about predicting players' performance of an online game called: Playerunknown's Battlegrounds.

Playerunknown's Battlegrounds, or PUBG, is a popular online battle royale game where players compete against each other in free-for-all combat. Players begin on a map empty-handed and must explore, scavenge, and eliminate other players until only one man is left standing.

We are given 65000 games' worth of data. And our task is predicting the final placement for each player in the game, by using a wide range of statistics generated by the players during gameplay.

In total there are 28 features in the dataset, with the target of prediction, winPlacePercentage, being the percentage chance of placing first. We've also highlighted some of the features that we think are more important than the rest. Some of the notable ones include: kills, killPlace, heals, walk distance, ride distance, and match duration.

Data Exploration

Due to the high volume of data and large numbers of features, we began with data exploration to have a more structured understanding of the dataset we are working with. Our approach is mainly divided into three parts: outlier detection, features normalization and identification of important features. To effectively gain a deeper understanding of the data, we used a lot of graphs and tables to help us visualize the dataset better, and hopefully gain insights that are otherwise not available.

Outlier Detection

orm	damageDealtNorm	maxPlaceNorm	matchDurationNorm	healsandboosts	totalDistance	killsWithoutMoving	headshot_rate
	842.0600	21.30	842.06	3	0.0	True	0.000000
	547.6280	17.38	2834.52	6	0.0	True	0.000000
	3132.5000	35.80	1607.42	5	0.0	True	0.200000
	200.4060	24.13	1014.73	0	0.0	True	0.000000
	158.0000	17.38	2834.52	0	0.0	True	1.000000
	789.5160	9.36	909.48	3	0.0	True	0.166667
	1708.4800	9.12	836.00	1	0.0	True	0.333333
	757.7570	21.45	856.57	11	0.0	True	0.285714
	179.1710	11.12	1017.48	7	0.0	True	0.500000
	125.7732	22.05	1051.05	0	0.0	True	0.000000

Figure 1. Outlier detection

We noticed that some rows in our dataset have weird characteristics. The players could be cheaters, maniacs or just anomalies. Removing these outliers will most likely improve results. For instance, one the most obvious sign of cheating in the game is killing without movement. It is fishy because we noticed that the travelled distance for some players remain zero during the whole game, but still managed to get kills. Except the small possibilities that these players are AFK or got killed early, it is most likely a cheater.

Features Normalization

In the dataset, although it seems we are already given many features, not all of them are important. Some potentially important features are not provided but can be interpolated. In this case, we believe it may be necessary to create more relevant features to improve the predictive quality of our models. For instance, in the following diagram, we plotted the number of players in each match. We noticed that the total number of players in each match can be very varied. This is likely a valuable feature for our model. We can use these values to compute the total number of players per match and use them as a basis to normalize other features.

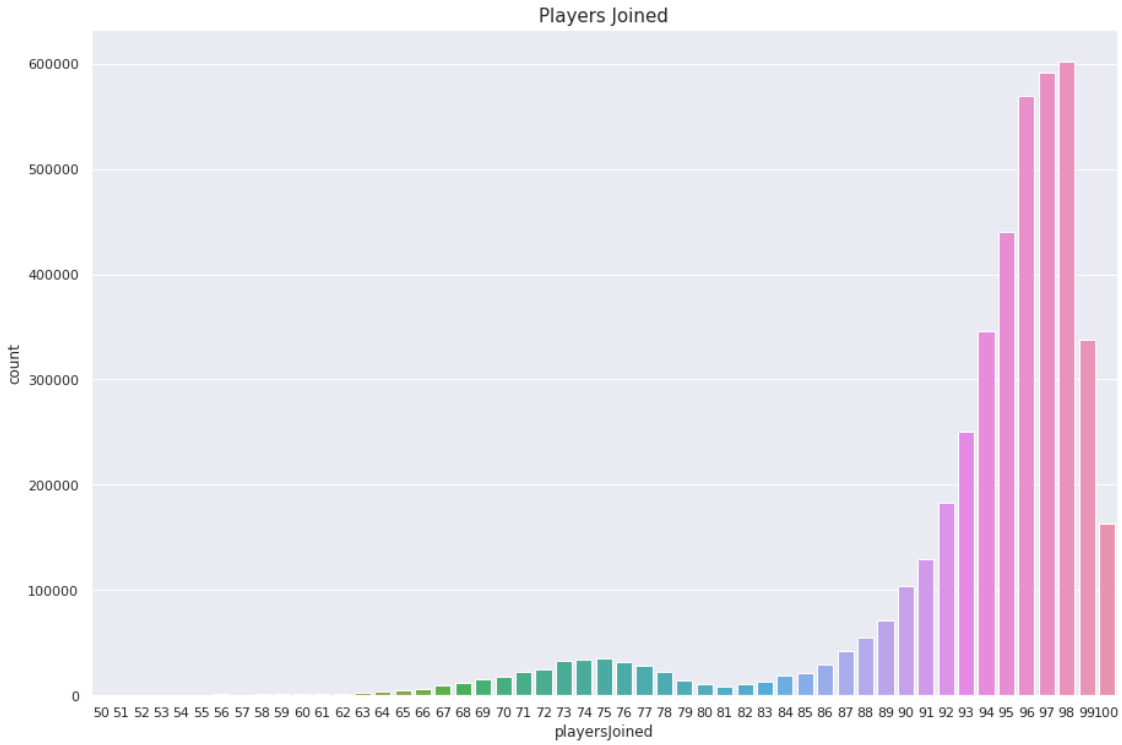

Figure 2. Features normalization

Identification of Important Features

To recognize how each feature affects the target prediction, we plotted different graphs to visualize their correlations and discovered some interesting patterns. For example, we found there is a strong relationship between the total distance travelled and the match ranking. From the following diagram, we can see the average distance travelled consistently increases with the player's placement. In other words, the better player you are, the further your distance travelled.

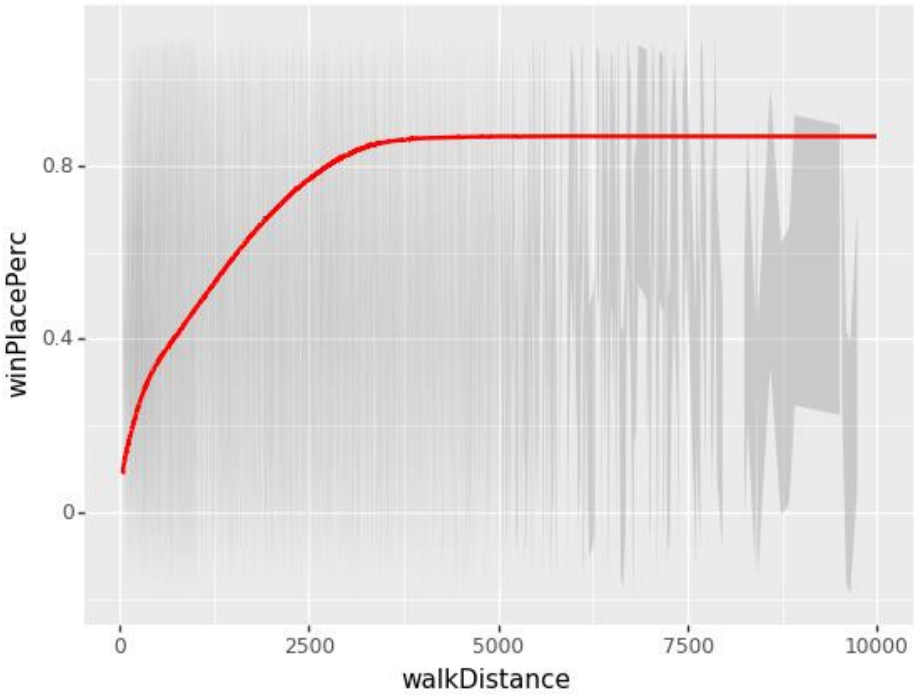

Figure 3. Important feature identification

Feature Engineering

We are provided 29 features in the dataset, including PUBG game stats of killing, moving distance, means of transportation and heal/boost. We believe that the features in our data are important to the predictive models we use and will influence the results we are going to achieve. The quality and quantity of the features will have great influence on whether the model is good or not. Therefore, after the exploration data analysis, we build a simple model to try various features on this model. And then we select some useful features to train a more complex model to get the result.

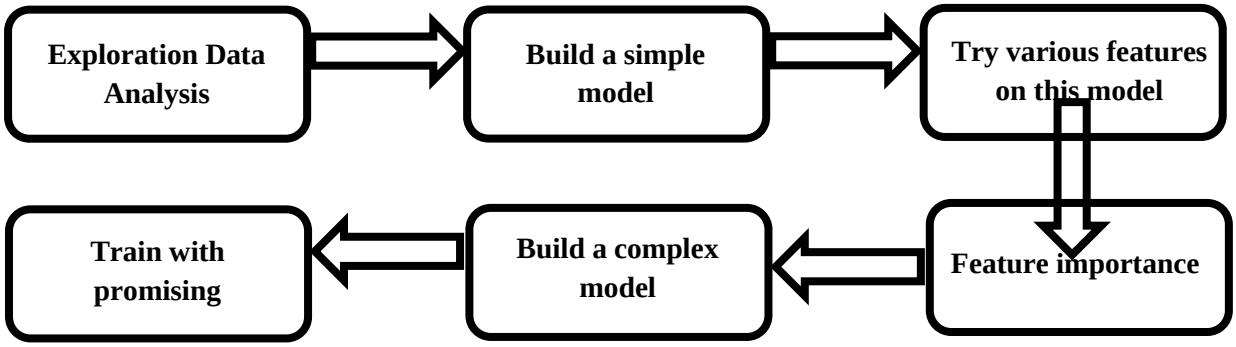

Figure 4. Feature Engineering Pipeline

Correlation

Firstly, we do a correlation work because this is the simplest way to see the relation between features. Here, if the value on the target is close to 0, it means that the feature may be irrelevant to the target.

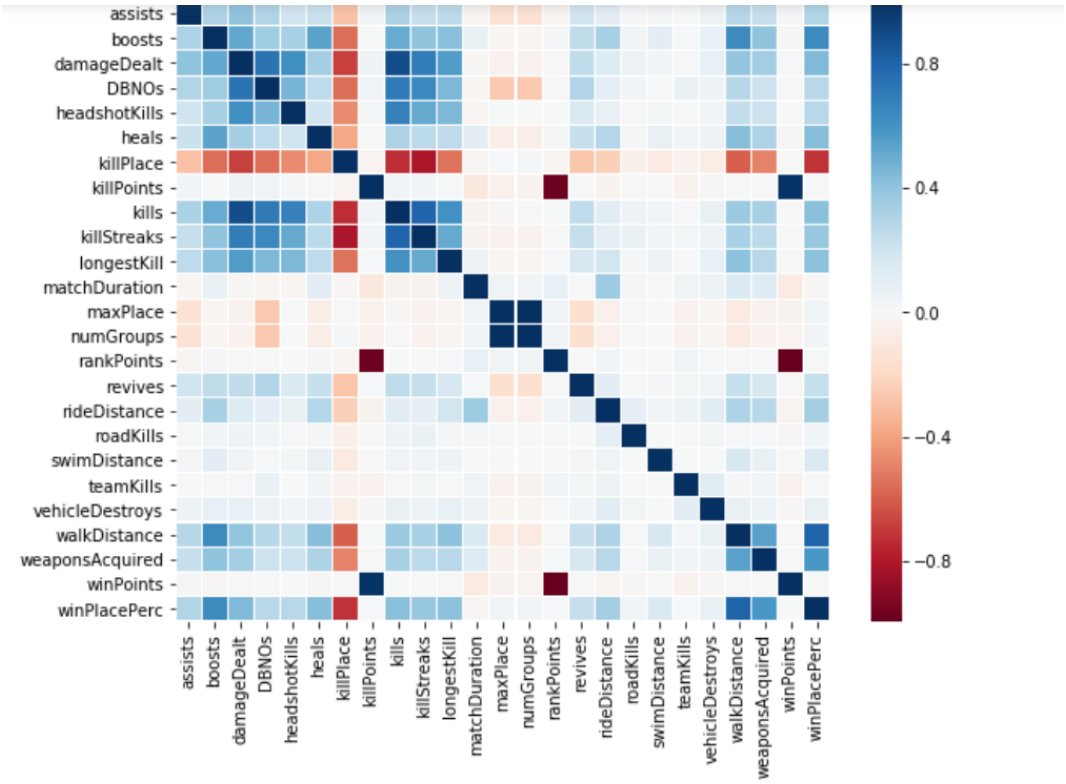

Figure 5. Correlation Heatmap

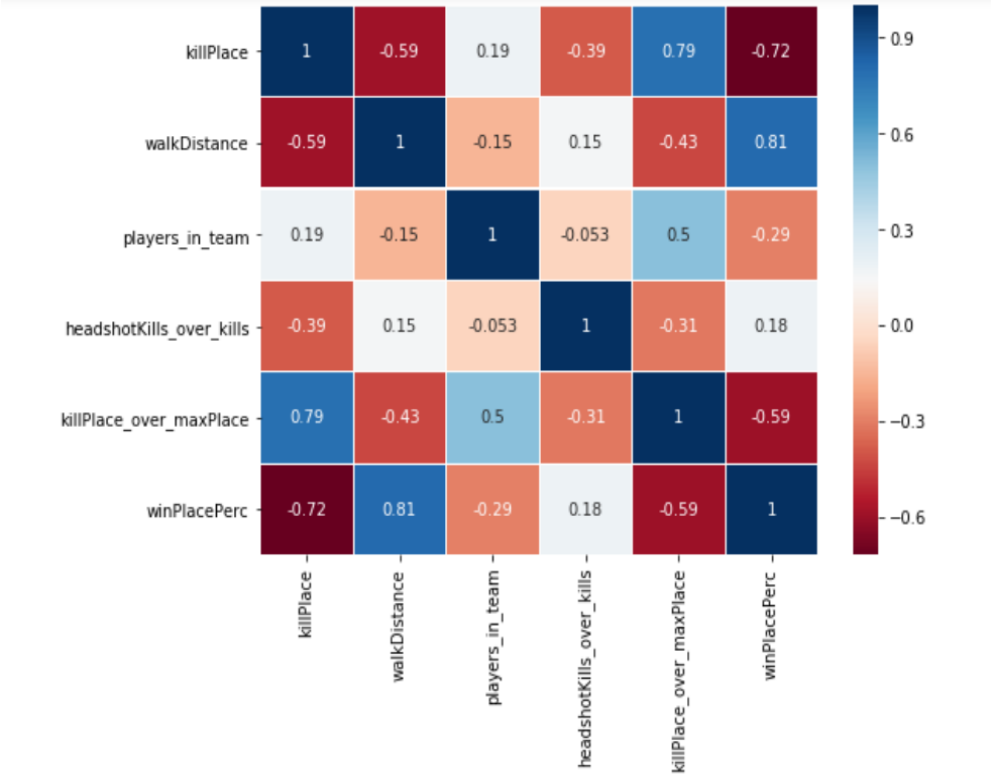

Fig 6. Correlation Heatmap

Score Gain on a Simple Model

We used linear regression during feature engineering since it's simple and fast. It'd be enough if you just want to see the impact of the new feature you added. Also, it's better to split dataset by match since we predict results by group in match. We add one feature to the original features for each time and test it with a linear model.

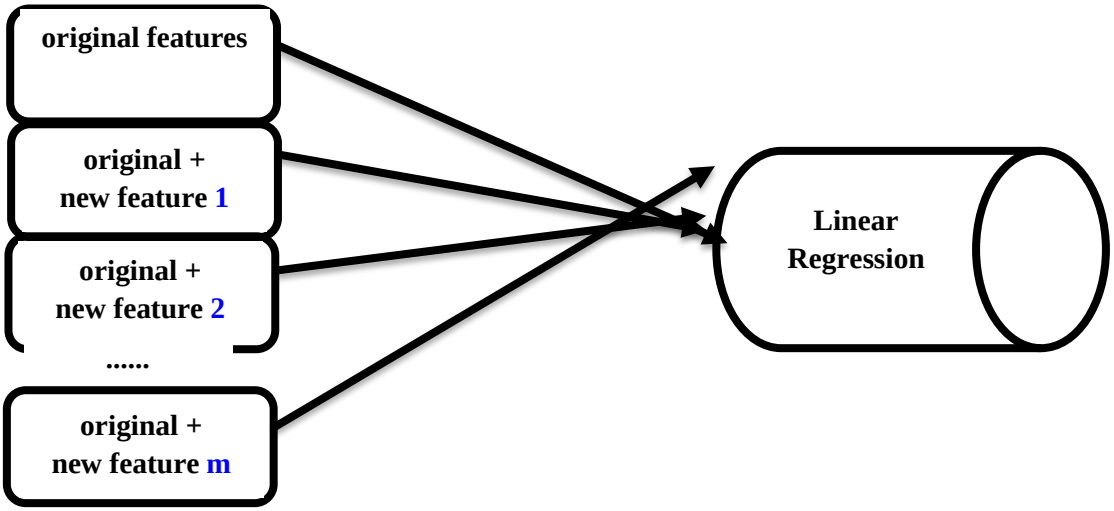

Figure 7. Feature Engineering by using LR

Player-level Generated Features

Table 1 Player-level Generated Features

Player-level Feature Name	Meaning
items	heals + boosts
players_in_team	the number of players in the team
total_distance	ride distance + swim distance + walk distance
headshotKills_over_kills	head shot kills/kills
killPlace_over_maxPlace	kill place/max Place
walkDistance_over_heals	walk distance/heals
walkDistance_over_kills	walk distance/kills
teamwork	assists + revives

We can see the score and the execution time.

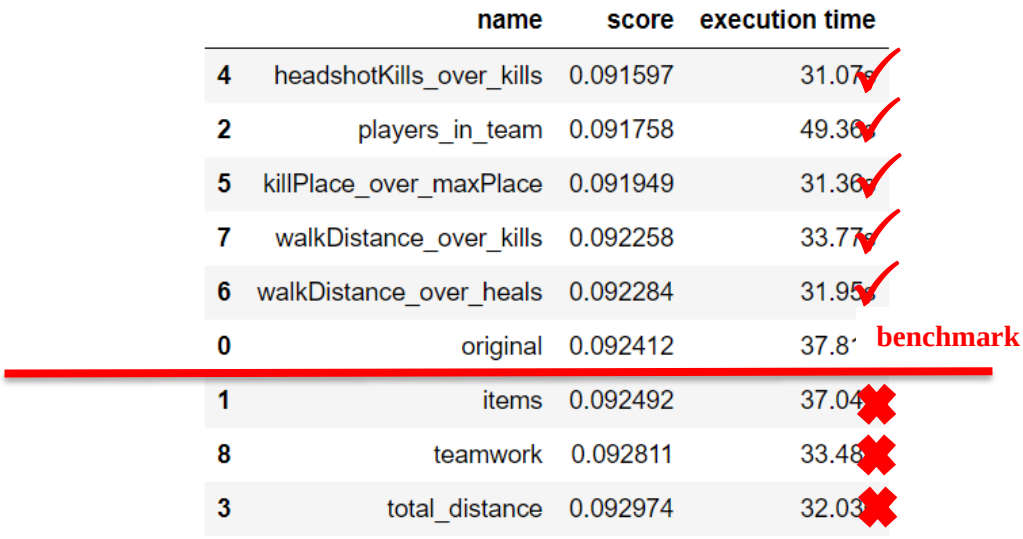

Figure 8. Player-level Generated Features Score Gain on LR Model

Aggregated Features

Aggregated Feature Name	Meaning
min_by_team	Find the min of features for each team in all competitions
max_by_team	Find the max of features for each team in all competitions
sum_by_team	Find the sum of features for each team in all competitions
median_by_team	Find the median of features for each team in all competitions
mean_by_team	Find the mean of features for each team in all competitions
rank_by_team	Find the rank of features for each team in all competitions

Table 2. Aggregated Features

	name	score	execution time
6	rank_by_team	0.057774	56.49s
4	median_by_team	0.075883	49.03s
5	mean_by_team	0.077083	47.63s
2	max_by_team	0.078436	48.35s
3	sum_by_team	0.088642	48.55s
1	min_by_team	0.089845	47.63s
0	original	0.092959	25.88 **benchmark**

Figure 9. Aggregated Generated Features Score Gain on LR Model

You can see how important rank features are in this dataset. Based on the score of the linear model by using original features, we select those features that have positive contributions to the model.

Feature importance of Tree models

Tree models can output feature importance. We build a LightGBM model to see feature importance.

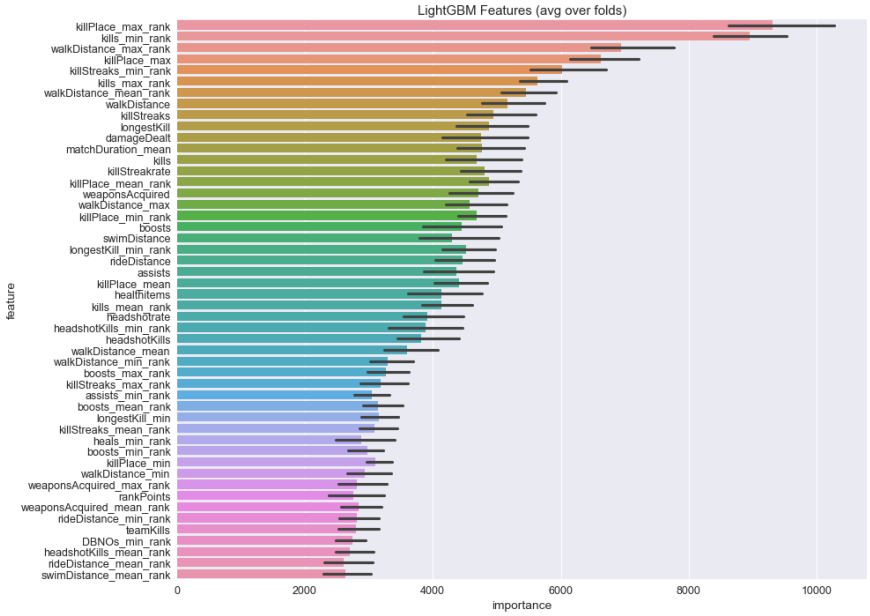

Figure 10. LightGBM Model Feature Importance

Model Selection

After feature engineering, we try to select an appropriate machine learning model by comparing several models with a sub-sampled dataset of 10,000. 5-fold cross validation is applied and evaluated based on mean absolute error.

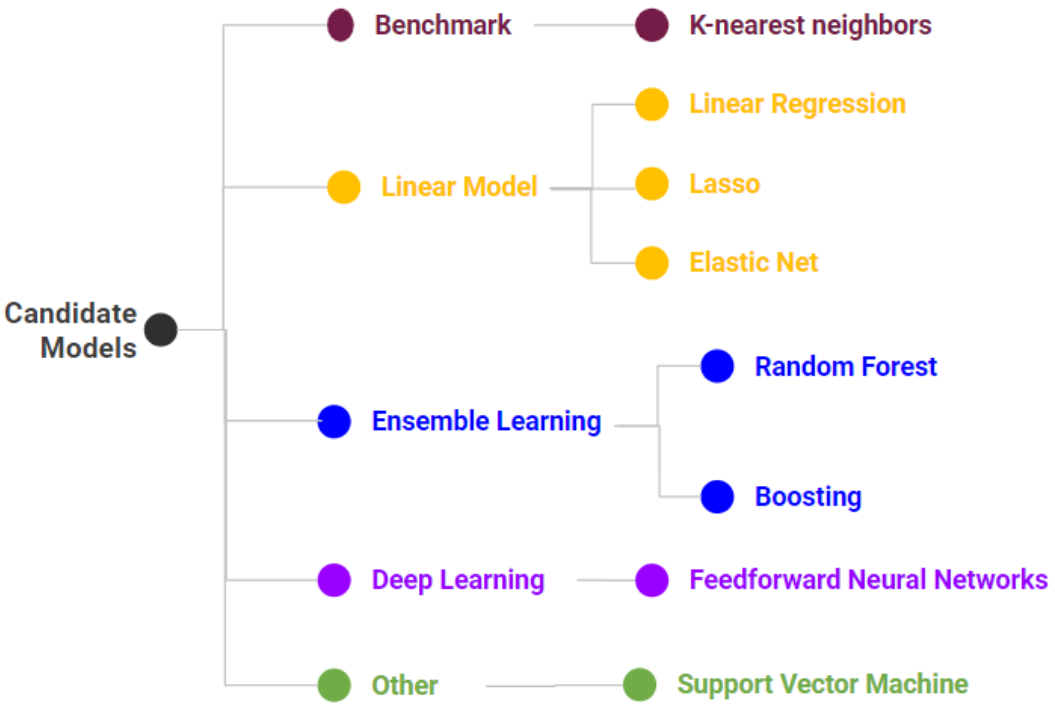

Figure 11. Candidate models

We divide our candidate models into five groups: K-nearest neighbor method is used as a benchmark with k equal to 5; linear models which include linear regression, LASSO and elastic net; ensemble learning method; deep learning and support vector machine.

Model name	MAE cv_mean	MAE cv_std
Linear Regression	0.0920	0.0041
Elastic Net	0.0899	0.0015
Lasso	0.0900	0.0015
K-Nearest Neighbors Regressor	0.1094	0.0026
Support Vector Regression	0.2670	0.0022
Random Forest Regressor	0.0698	0.0023
CatBoost Regressor	0.0662	0.0013

Gradient Boosting Regressor	0.0714	0.0016
XGBoost Regressor	0.0715	0.0015
LightGBM Regressor	0.0651	0.0012

Table 3. Performance of models

As we can see, traditional machine learning model such as linear regression, LASSO and elastic net generate better result than k-nearest neighbor method. However, ensemble learning method is the one that gives significantly improved result. Besides random forest method, we compare four types of boosting methods, namely gradient boosting, xgboost, catboost and lightgbm. Among all models, we find that LightGBM Regressor gives the best result in terms of both mean value and standard deviation of mean absolute error. Therefore, we choose LightGBM as one of our candidate models to train the whole dataset.

LightGBM

Another reason that we choose LightGBM as one of our models is that it has won many Kaggle competitions in recent years and has shown its superior performance.

LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It has many excellent characteristics. For example, its training speed is fast and efficient with low memory usage. It can handle large-scale data with the support of parallel and GPU learning.

We first used 10,000 samples and 5-fold cross validation to tune the parameters of LightGBM. After that, we use the whole training set to train LightGBM and use the trained model to generate predictions. The time spent on training and prediction with full data set is 16,562 seconds on Kaggle cloud. The final mean absolute error of test set is 0.0246 after we submit the result to Kaggle competition and so far, we rank 178 among 780 groups on the public leaderboard.

Feedforward Neural Network

Since we have access to a high end GPU and have lots of data to train, it makes sense to try a neural network model. We used the keras package offered in Tensorflow's GPU installation for deep learning.

We began our exploratory analysis for deep learning with a simple baseline model, which is a 2-layer feedforward network with densely connected layers. The first layer has 40 neurons in order to match the training feature dimension. In addition, we used Relu as the activation function for the hidden nodes, and Adam gradient boosting method for weight updates.

For hyper parameter tuning, we explored different configurations of network structure. Such as Deep vs shallow, wide vs narrow, different regularization schemes as well as different dropout percentages.

We first explored the effect of different number of layers on the model performance. We did this by comparing the performance of the baseline model with a deep model and a shallow model. These models have similar configurations to the baseline model, with the deep model having 6 layers instead of 2, and the shallow model simply having one layer.

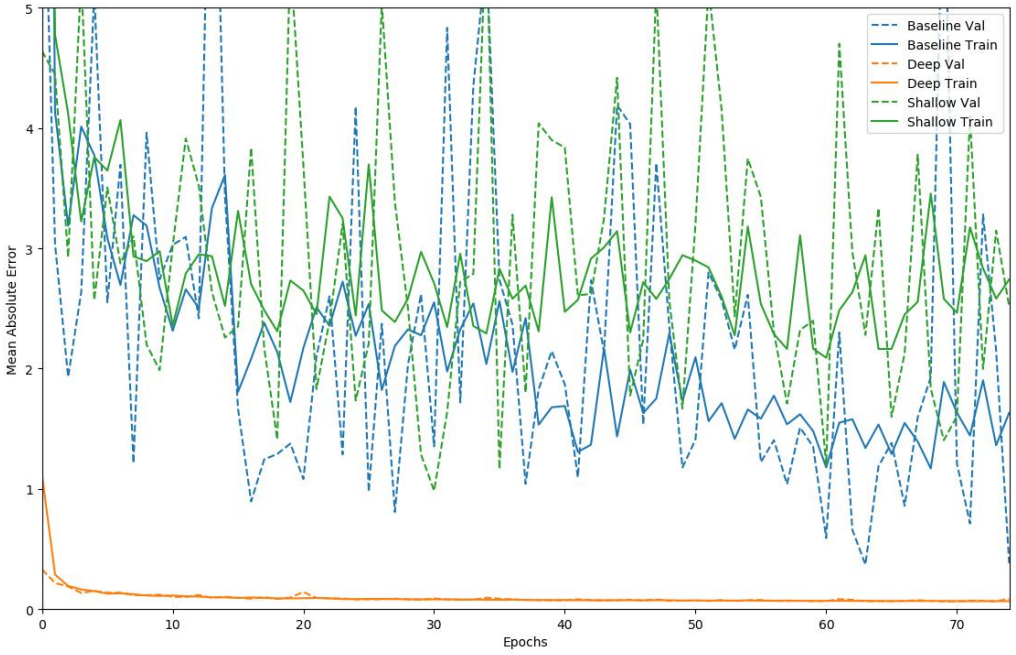

Figure 8. Deep vs Shallow comparison

We plotted the mean absolute error for training and validation for 75 iterations for all three models. As shown in the graph, the deep model outperformed all other models by a large margin.

We explore the effect of different numbers of neurons per layer. For this we used a wide and a narrow network. The wide network contains more neurons at each layer than the baseline and the narrow network fewer. We plotted the training and validation loss for these 3 models. We can see that the wide network outperformed the other 2 models for later iterations.

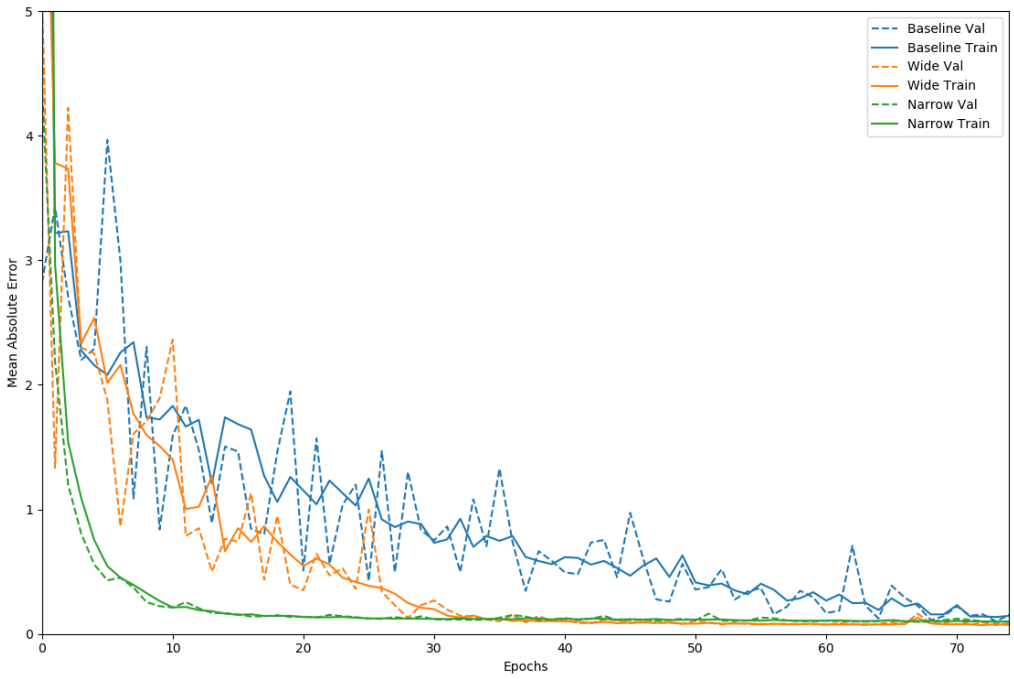

Figure 9. Wide vs narrow comparison

Our next goal is to ensure that our models don't overfit as it tries to fit the training data. For this we explored different methods of regularization and dropout.

For regularization, we explored different schemes such as l1, l2, and a combination of l1 and l2. We also explored applying regularization at different layers in our network.

Our findings indicate that we couldn't achieve better performances with different regularization schemes as the validation loss for our baseline model outperformed all other models with regularization applied to them.

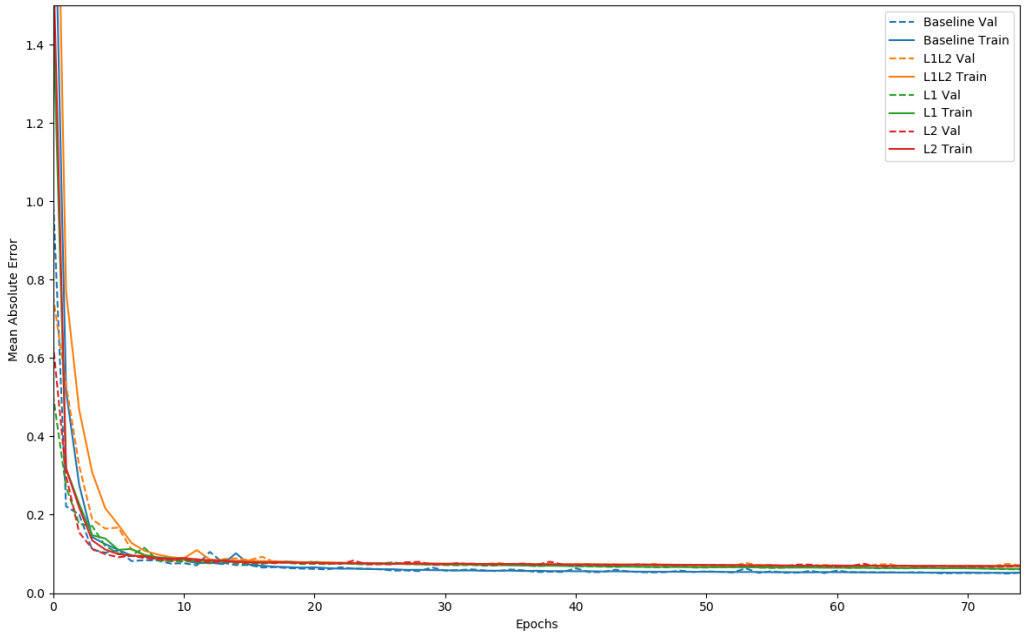

Figure 10. Regularization comparison

Similarly, for dropout, we explored different percentages of masks as well applying dropout at different layers in our network. We couldn't achieve any significant improvement with dropout applied.

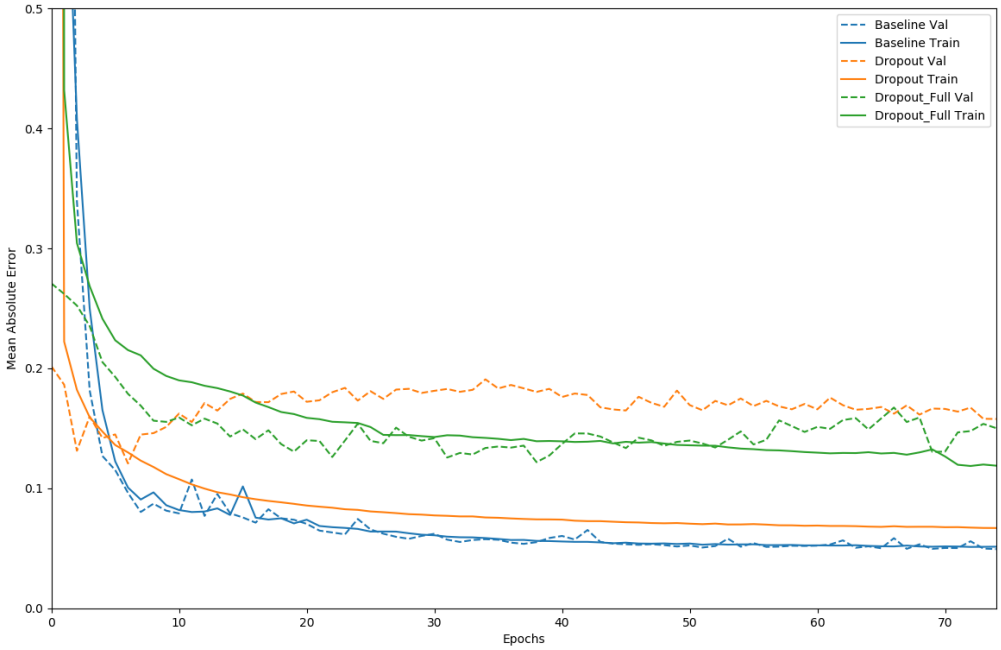

Figure 11. Dropout comparison

Our final network configuration contains 6 layers with the first layer having 120 neurons and gradually decreasing number for the next layers, and we were able to achieve a mean absolute loss of 0.0448 in the competition.

Contributions

Zhu An Qi: data exploration
Sun Yimiao: feature engineering
Cao Pei: model selection and LightGBM
Guo Donghui: deep learning part
Lan Hao: tune parameters