Machine Learning Project Report

SX1916115 Jingtang Zhang*

E-mail: jingtangzhang@nuaa.edu.cn

About the Task

Our task is to finish a competition on *kaggle*, which is a platform for machine learning competitions. Specifically, our task is based on a famous computer game called *PLAYERUNKNOWN'S BATTLEGROUNDS (PUBG)*. In this game, up to 100 players will be dropped onto an island called Erangel empty-handed. They must explore, scavenge, and eliminate other players until only one of them is left standing, while the play zone continues to shrink.

We are given some anomymized player data from 65000 games. The data is split into training set and testing set. In the training set, we have the data describing the player's performance, like various kinds of killing count, or the moving distance on a car or under water. Also, we have a winning placement percentage **winPlacePerc**, ranging from 0 to 1, where 1 corresponds to the 1st place, and 0 corresponds to the last place. This is the target of prediction. In the testing set, we have all of the columns in the training set, except the **winPlacePer** column. Our task is to predict this column as precisely as possible. The final score is calculated by the mean absolute error between the prediction and the groud-truth.

Analysis and Motivation

Version 1

Given the training data, we can see that it either describes the behavior of a player or the metadata of a match. According to my experience of playing computer games when I was a teenager, the final score of a match is strongly related to the match type. For different match types, there are different rules and strategies, which may affect the player's data like walking distance or killing count. So my first motivation is to split the training set by different match types.

Also, during different matches, one player tends to perform differently. He may shoot well in one game, while missing anything he shoot in another game. My second motivation is to train a single model for each single game.

From the Machine Learning course, I have learned about the effectiveness of ensemble learning. So I tried to use all models of the same match type to predict a record from the corresponding match type, and get the average value as the final prediction result.

Version 2

In the algorithm of version 1, the data for training a model comes from a single game, containing less than 100 rows, which is too few for training a model empirically. So I tried to involve more data for each model's training process. However, the amount of data for each match type is totally different. As a result, it is impossible to train models for match types without enough data.

My solution is to train models by different types of match. I split the whole training set into 100 parts, and use each part to train a model. Each part contains about 45000 rows of data which belong to different match types. While predicting a record, I use all of the 100 models to predict and use the average value as the final result.

Algorithm and Implementation

Version 1

For the first version of my algorithm, I choose the following features: [”walkDistance”, ”killStreaks”, ”rideDistance”, ”kills”, ”heals”, ”boosts”, ”damageDealt”, ”weaponsAcquired”, ”headshotKills”, ”teamKills”, ”roadKills”, ”swimDistance”, ”revives”, ”assists”, ”killPlace”, ”longestKill”, ”vehicleDestroys”]. The reason is that, these features are all used to describe a player, instead of a match. In my opinion, since I train a model for each single match, the features describing the match are meaningless, because they are all the same for the same match.

I choose **Decision Tree regressor** as the algorithm. The reason is that it can save the efforts of decomposition, since I don’t know which feature is more important for the result. The algorithm can judge the best feature for me automatically.

In order to train a model from a match, I sorted the whole training data by match type and matchID. For each matchID, I trained a model using all rows with this matchID, and added the model to the model set of the corresponding match type. During prediction, I used all the models of that match type to calculate the average prediction value, which is an application of ensemble learning: considering the matches are inherently independent to each other, the model generated from each match can do prediction independently.

Specifically, to speed up the prediction, for match types with much more data, which corresponds to more models, I randomly sampled part of the models for prediction with a max predictor threshold. Also, I implemented the program in a multi-processes way, which can fully utilize the multi-core CPU. The source code is available at my GitHub.

Version 2

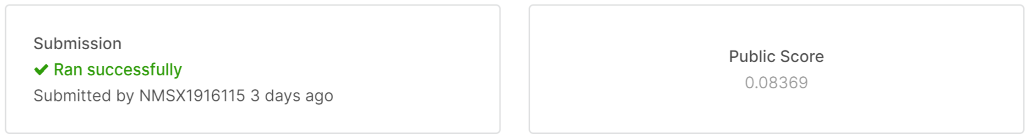
For the second version of my algorithm, I trained 100 models with about 45000 rows of data each, and used all models together to predict a record. I tried several ensumble learning algorithms, including **XGBoost regressor**, **Random Forest regressor**, **AdaBoost regressor** and **Gradient Boosting regressor**. They are all using ensemble method inside. I integrated all the models together for a second time of ensemble prediction. The source code is available at my GitHub.

Results and Evaluation

Version 1

For the version 1 algorithm, I tried different parameters of max predictor threshold. If a match type contains more models than the threshold, then only models of the threshold would get involved into the prediction.

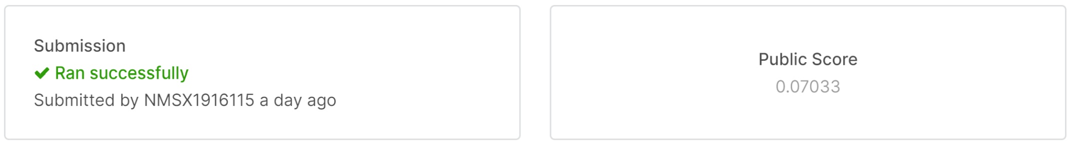
At first, I introduced this parameter to speed up the prection process. However, according to the experiment, I found a greater threshold would not get a better score. I used the threshold of 500, 1000, 1500, 2000, among which 1500 got the following best score, while 1000 was not better than 500 and 2000 was not better than 1500.


Submission
✓ Ran successfully
Submitted by NMSX1916115 3 days ago

Public Score
0.08369

Version 2

For the version 2 algorithm, I just tried different ensemble learning algorithms: **XGBoost regressor**, **Random Forest regressor**, **AdaBoost regressor** and **Gradient Boosting regressor**, among which Random Forest regressor got the following best score.


Submission
✓ Ran successfully
Submitted by NMSX1916115 a day ago

Public Score
0.07033

Thoughts

I didn't do much work about the features in my experiment, which should be an important part in machine learning. In fact, I tried to use *Principal Components Analysis* to cut off some features, but it didn't work well. I referred to some high-score solutions after my experiment, and noticed that their feature engineering was much more complex. Some of them even generated more features from the given training data, and got a very good result. I guess the better feature engineering is why they can get a higher score.