Random Forest or Gradient Boosting?
- a practical comparison between two most popular ensemble methods in predicting German used cars prices in R

He Jin hj2479

I. Introduction

Among all the ensemble methods, Random Forest and Gradient Boosting Methods tend to be two most well-discussed methods. Whereas random forests build an ensemble of deep independent trees based on bagging, GBM builds an ensemble of shallow and weak successive trees with each tree learning and improving on the previous based on boosting.

Questions always come that which of one is better than the other and the answer is debatable. Generally, Gradient Boosting Methods tends out to have better model performance compared to Random Forest. However, Random Forest might have other benefits that people might care about in certain circumstances. Therefore, here a real-world dataset was carried out to test and compare the discrepancy between these two ensemble methods and see what are the detailed benefits and weaknesses for each method.

II. Data and Method

1. Data Source

The dataset is from Kaggle (https://www.kaggle.com/orgesleka/used-cars-database), which initially contains more than 370,000 entries with 20 columns scraped with Scrappy from eBay-Kleinanzeigen, a German used car website. Since the used cars' prices change a lot in the market, the factors influencing the used car sales has become a topic of great interest to both car dealers and buyers. Therefore, we will use supervised learning methods, in this cases the ensemble learning methods, to help to predict those used car prices and select the most important factors corresponding to the outcome. Our particular goal in this project is to compare the performance of the two most popular ensemble learning models, that is, Random Forest and Gradient Boosting, in dealing regression problems in this particularly used car dataset.

2. Data Preprocessing and Sampling

The original dataset was downloaded as CSV file and read in R. Since it was a dataset from the German website, firstly some of the keywords must be translated into English. Next, we screened certain variables and created another variable we might be interested called car age. Then, we filter those observations with apparent errors and those outliers, such as power < 50, price < 1000, etc. Only complete cases were kept because of the relatively sufficient large dataset. After the first round data cleaning, there are 205,337 entries and nine variables, which is 57.89% of the total original data in the dataset.

205,337 observations was still a huge number. Therefore, further shrinkage is necessary. To better represent the concept of German used car, here, six most famous German car brand were selected from all the other car brands. They were Audi, BMW, Mercedes Benz, Opel, Volkswagen, and Porsche. Those selected brands were used to present the entire German market in this case since they could cover from the entry level to luxury level. Also, buses were excluded from the dataset. After the second round data preprocessing, 119,802 observations were remaining in the dataset.

Although the dataset here is tidy enough, it could still be considered relatively abundant. In order to further save the potential computational time, another round of sampling is somehow necessary. Here, one out of ten of the original data were randomly picked to composite the new sample which contains 12000 observations. Among those 12000 observations, 10000 were randomly selected to form the training dataset while the rest of 2000 were used to form the test dataset.

3. Random Forest

There are three types of R packages needed to conduct the Random Forest model in this dataset, which was randomForest, Ranger, and h2o. randomForest was probably the most widely known (or used) implementation of Random Forest method in R, but due to its single-threaded feature, it is usually less efficient compared to the other random forest packages. Ranger is a fast implementation of random forest or recursive partitioning using C++ backend, particularly suited for high dimensional data. h2o is a powerful Java-based platform which supports multiple machine learning methods.

Firstly, randomForest package was used to perform the random forest regression to the data without any parameter tuning. Then, to better improve the performance of the random forest model, two different tuning procedures were conducted using Ranger and h2o since they proved to be much more efficient than the basic package. The reason why we would use two different tuning methods is that we would like to compare two methods in both different package environment and same package environment.

In Ranger, we tried hyperparameter combinations like the followings: mtry started from 2 to 6 by 1; node size started from 3 to 9 by 2; sample size in .55, .632, .70, .80. The total number of the combinations in Ranger was 80. Since Random Forest naturally has the out of bag sample, we would use that as our validation set to select the best model, with RMSE as the criteria.

In h2o, the hyperparameter combinations became: mtry from started from 2 to 6 by 1; max tree depth started from 20 to 40 by 5; min rows(node size) started from 1 to 5 by 2; nbins (number of bins) started from 10 to 30 by 5; sample size in .55, .632, .70, .80. This large grid search in h2o contains 1800 hyperparameter combinations which might be impossible for us to run each out. Consequently, h2o provides an additional grid search path called “RandomDiscrete”, which will jump from one random combination to another and stop once a certain level of improvement has been made, a certain amount of time has been exceeded, or a certain amount of models have been run. Then we create a random grid search that will stop if none of the last ten models have managed to have a 0.5% improvement in MSE compared to the best model before that. If we continue to find improvements, then we cut the grid search off after 1200 seconds (20 minutes) to save time and computational memory.

4. Gradient Boosting

For Gradient Boosting, only two packages would be used in fitting the model which were gbm and h2o. The gbm R package is an implementation of extensions to Freund and Schapire’s AdaBoost algorithm and Friedman’s gradient boosting machine, which is the original R implementation of GBM using the C++ backend. Since it has relatively good efficiency, we would still use it in our first round hyperparameter gird searching. The h2o package used here was exactly the same platform as used before.

When using the gbm package tuning the hyperparameter, we tried the following combinations: shrinkage in .01, .05 and .1; interaction depth in 5, 7, 9; min observations in each nodes in 3, 5; bag.fraction in .65, .8, 1. The total number of combinations in this step was 54. Due to the computational limit of the laptop we have, the original cross-validation method used to select the model is not appropriate to use here. Therefore, we set a training fraction here equal to 0.75, indicating we would use the first 75% percent of the training data as the true training dataset and use the rest 25% as the validation dataset, to pick the best model.

Similar to Random Forest, we would use h2o to conduct the second round hyperparameter search. h2o provides many parameters for GBM that can be adjusted and here we will only focus on those common hyperparameters, which includes: ntrees: number of trees to train; max_depth: depth of each tree; min_rows: Fewest observations allowed in a terminal node; learn_rate: rate to descend the loss function gradient; learn_rate_annealing: allows to have a high initial learn_rate, then gradually reduce as trees are added (speeds up training); sample_rate: row sample rate per tree; col_sample_rate: column sample rate per tree. The total number of combinations in this step was 480. The criteria that we used for Random Forest in h2o above ("RandomDiscrete", ten models with 0.05% improve in MSE, max time of 20 minutes) would also be used here, including the principle of 75% training set and 25% validation set.

III. Results Analysis

1. Random Forest results

- *Direct fitting in Random Forest package*

The first random forest model was directly fitted without tuning any parameters. Here, the only parameter we need to concern about is mtry, and mtry = 2 (8/3 then rounding) in this scenario since the default mtry for regression in the Random Forest model is p/3. It took R 102.324 seconds to finish the model fitting. After fitting this initial model, the prediction of the initial model on the test dataset was then followed. The test RMSE here was given as 3868.886.

- *Tuning in Ranger*

Then, a larger hyperparameter search was conducted using Ranger. In order to save the computational time, here the number of trees was defined as the default value 500. The best model was given as: mtry = 3, node_size = 3, sampe_size = 0.8 and the Out of Bag RMSE here was 3654.051 while the test RMSE was 3804.952. It took R around 15 minutes to finish tuning procedure and 3 second to run the final model.

- *Tuning in h2o*

Our Random Forest grid search in h2o assessed **60** models and the best model was given as: max_depth = 25, min_rows = 1, mtries = 3, nbins = 20, ntrees = 500, sample_rate = 0.7. The optimal model achieved an OOB RMSE of 3504.894. The optimal model took 19.6 seconds to run and the test RMSE was 3677.937.

2. GBM results

- *Direct fitting gbm package*

Similar to the procedure above, firstly, a direct model without any tuning step was fitted using the gbm package. Here, the tree size was also set as 500 for comparison purpose and five-fold cross-validation was used. It only took R 1.89 seconds to run the model. Apparently, with such little running time, the default result from GBM had a much worse performance compared to the Random Forest, with a test RMSE 8826.957.

- *Tuning in gbm*

After some preliminary tuning, 500 trees were proved to be too relatively small the model fitting. To better achieve a sufficient performance of GBM, we used 5000 trees in this case. After trying all 54 combinations of the hyperparameter in 45 minutes, the best model tuning hyperparameter in gbm was given as: n.trees = 4925, interaction.depth = 5, shrinkage = 0.01, n.minobsinnode = 5 and bag.fraction = 1, with a validation RMSE 4069.965 and a test RMSE 3595.808. The optimal model took R 23 second to run.

- *Tuning in h2o*

The GBM grid search in h2o assessed 99 models and the best model was 23th model, given as: col_sample_rate = 0.8 learn_rate = 0.01, learn_rate_annealing = 1, max_depth = 5, min_rows =4 and sample_rate = 4, with a validation RMSE 3684.87. Once the preferred model was founded, the parameter would be extracted and retrain a new model with the full training data. We will use the best model from the full grid search and perform a five-fold Cross-Validation to get a robust estimate of the expected error, which took 135.24 seconds to run. The CV RMSE was 3479.135 and the test RMSE was 3451.929.

Initial fitting	**Random Forest**	**Gradient Boosting**
Test RMSE	3868.886	8826.957
Running time	102.324	1.89
First Round Tuning	**Random Forest (Ranger)**	**Gradient Boosting (gbm)**
Tuning Validation RMSE (OOB for Random Forest)	3654.051	4069.965
Tuning time (approximate)	15 minutes	45 minutes
Test RMSE	3804.952	3595.808
Running time (optimal model)	3.312	23.808
Second Round Tuning in h2o	**Random Forest (h2o)**	**Gradient Boosting (h2o)**
Tuning Validation RMSE (OOB for Random Forest)	3504.894	3684.87
Tuning time	20 minutes	20 minutes
Test RMSE	3677.937	3451.929
Running time (optimal model)	19.6	135.24

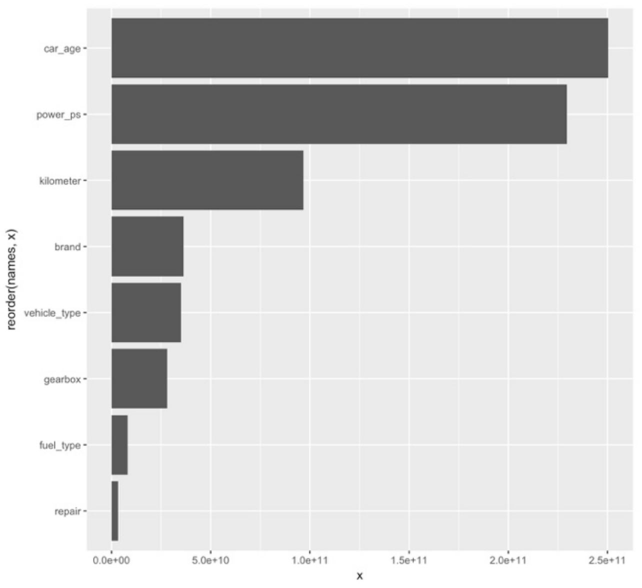
Table 1. Comparison of Test RMSE, Validation RMSE, tuning and running time between Random Forest and GBM

3. Comparison between Random Forest and GBM

In the initial fitting round, Random Forest tended to be superior to GBM under the situation where tree size is equal to 500 and without any hyperparameter tuning (Table 1). With such a small number of trees, GBM took only 1.89 seconds to fit the model, and the result was consequently not good enough. Random Forest tends to have stable performance, the RMSE of the test dataset is 3868.886 without any tuning, which was only slightly higher than our optimal model found later.

On the other hand, in both first round and second round tuning process, GBM tended to be superior to Random Forest, with an average 200 RMSE reduction in the test dataset (Table 1). However, the cost was that GBM took relatively more prolonged time(both the tuning time and the optimal model running time) to find and tune the proper hyperparameter grid due to its various parameter settings. Nonetheless, it also might become a benefit of using GBM – meaning the model could have lots of flexibility. On the other hand, Random Forest could have a relatively good model performance with minimal tuning required and relatively fast implementation. Under the same package environment with same fixed tuning time, GBM tends to have better model performance.

Regarding the variance importance, there is a slight difference between the results from Random Forest and GBM (figure1). In Random Forest, kilometer is more important than brand while in GBM the result is opposite. This little discrepancy is acceptable since these two methods have different algorithm mechanism, one is bagging and the other is boosting. Fortunately, both of the method successfully mark the most important variable, which was the car age.



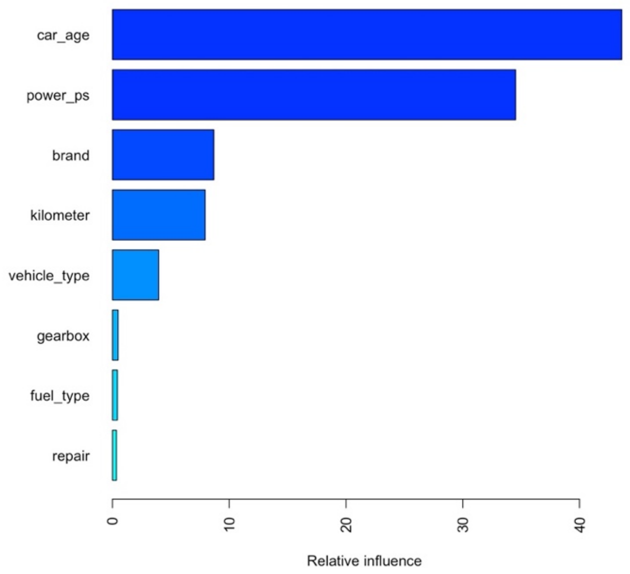


Figure1. Variance important plot for the best model selected from Random Forest and Gradient Boosting Method in the first round tuning. (Random Forest on the left, GBM on the right)

IV. Conclusion and Discussion

Two different ensemble methods of predicting the price of German brand used car was developed and compared. Generally, Gradient Boosting Method tended to have a better performance regarding the test dataset with the price of a longer tuning time and a more complex tuning procedure. With the proper parameter tuning, the improvement of GBM is significant. Random Forest is not bad overall, especially considered its stable performance with less hyperparameter tuning. The most important features of the data were car age, power of vehicle, brand and kilometer. Through this systematic analysis, our purpose of comparison between these two methods in this particular German brand used car has accomplished.

However, our works might still have many places to improve. For instance, due to the limitation of computing speed, we only sampled one of ten of the whole dataset to conduct the analysis. In the future, it may worth trying more powerful computer to conduct the analysis, especially for the hyperparameter grid part, since it is more complicated, intricate and computationally expensive (e.g. using Cross validation when tuning GBM). Also, we might try Classification method in this scenario by merely divided our price range into specific categories to get a more intuitive result. Also, there are certain steps which might need some clarifications — for example, the reason why we select ntrees = 500 for Random Forest while ntrees = 5000 for GBM is that Random Forest is an average result. Conceptually, when the ntrees is large enough, the results will not change much. However, for GBM, each new trees is fixing the difference of the entire system and with the enlarging of the numbers of trees, the model could have considerable improvement. To better illustrate the performance of GBM, ntrees = 5000 was chosen here.

GBM or Random Forest, which is better? Just like the classic problem of bias-variance trade-off, you always need to sacrifice certain things in order to get the optimal model performance. To sum up, with the prerequisite of sufficient computational resources and enough time, we might conclude that GBM is better than Random Forest in terms of model performance.