Predicting Simulated Data

Anson Hsu(301208466) Jiaqi Li(301274339) Miguel Quiza(301229372)

November 24, 2018

Introduction

This report is made to analyze the simulated data which has consist 203287 observations and 75 variables, and the response variable y is the Value. The simulated dataset has no meaning so we just consider each entry as a number. For the data cleaning, there are several missing value entry in both training and testing data set so we choose the median of the column to fill the missing value. In addition, we also find out that there are several columns consist -9 for every observations; we consider those columns as missing value column so we remove those columns. Because the dataset is large some modeling methods can not be run on our computer. Firstly, we extract a part of data from training set to build a randomforest and GBMs model. Secondly, we use XGBoost to run 100% of the training set. Consequently we find out that XGBoost with 100% of the training set have much better RMSE than other models with less training set. As a result of XGBoost has the lowest RMSE and the shortest running time, we decide to use XGboost model as our final model for prediction.

1 Data Exploration

1.1 Data Cleaning

In both the training and testing data set, there are missing values appeared. In the beginning, we use KNN and Mice to impute the missing value but because there are a column with too many missing value in the testing dataset, KNN and Mice can not perform well. Therefore, we choose to replace the missing values by impute the column median. Consequently, we find out that column median gave us better result than KNN and Mice. we did not perform this process in XGBoost since it will impute missing values by default.

There are some columns which are consist -9 for all the observations. The information from where the data provided, tells us that those are the error and they are all represent the NAN, missing value. Therefore, we choose to remove all these columns from the training and testing dataset. Moreover, a column B.017 is not in the numerical form and it can not be used to build model. Therefore, we transform the factor form to numeric form for column B.017.

1.2 The Problem of Large Dataset

This dataset consist 203287 observations and 75 variables so the running time for building the models and missing value imputation takes the long time to run. Considering about the efficiency, we decide to extract the part of the data from training set to build the model. At the beginning, we try 3 percent of the training dataset. Secondly, we try 25 percent of the training dataset. The general idea is that if we are able to use 100% of the training set, we may have very low RMSE for GBM model. However, considering about the running time, we abandon this concept and choose to use XGBoost to build our model with 100% of the training dataset.

2 Model Exploration & Application

Note: Some models (that we used) are not mentioned due to spacing purposes

2.1 GBMs

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It often provides predictive accuracy that cannot be beat. But GBMs will continue improving to minimize all errors. This can overemphasize outliers and cause overfitting. And also GBMs often require many trees (>1000) which can be time and memory exhaustive.

Since our cleaned training dataset is large (with 153287 observations), we started with 25% of training dataset to train our GBM model using n_tree=5000. It spent 30 minutes to get the result and the root mean square error (RMSE) is 8.6. Then we used the whole training dataset to train our GBM model and it seems impossible to get the result on our personal computer. So we decided to explore other efficient method.

2.2 XGBoost

Extreme Gradient Boosting is among the hottest libraries in supervised machine learning, especially prominent in Kaggle competitions. It supports various objective functions, including regression, classification, and ranking. The core algorithm is parallelizable, so it can use all the processing power of our machine and the machines in our cluster. Extreme Gradient Boosting can be more than 10 times faster than existing gradient boosting packages because the R-package can automatically do parallel computation on a single machine.

Also, we do not need to worry about the miss value problem because in XGBoost, when the data is sparse (i.e. contains missing values), the model permits instances to be classified in the default direction. In every branch, there are two possible choices (left or right of the split); where the optimal default directions are learned from the training data. Therefore, XGBoost will impute (or even avoid missing values) by default, and no other forms of imputation methods are required prior to the predictive modelling

Since the XGBoost can only accept numerical variables, the first step was converting the variable planet from factor variable into integer variable. Then we converted the training and testing sets into matrices (understandable by XGBoost functions). We also needed to decide the number of iterations to run and the specific method type in the parameter tuning within the xgb.cv function. Between the options 'gblinear' and 'gbtree,' the xgb.cv function outlined the optimal number of iteration to give us the root mean square error(RMSE) for each method. It spent around 5 minutes to get the results which is very efficient. The results are shown in Figure 1 and Figure 2 .

Gbtree

iter	train_rmse_mean	train_rmse_std	test_rmse_mean	test_rmse_std
197	5.043935	0.08039585	6.365258	0.6741455

Figure 1: RMSE results

Gblinear

iter	train_rmse_mean	train_rmse_std	test_rmse_mean	test_rmse_std
150	8.981273	0.1206263	10.02336	0.5157931

Figure 2: RMSE results

3 Model Decision and Conclusion

We decided to continue with the well performing XGBoost model, achieving a final 6.4024 RMSE score in the private leaderboard in Kaggle. Three advantages we learned from XGBoost are:

1. Its ability to very accurately impute missing values.
2. The combination of using XGBoost with cross validation can lead to minimize overfitting.
3. The XGBoost is much more efficient than the other methods we used before (Random Forest, GBM).

References

[1] Beginners Tutorial on XGBoost and Parameter Tuning in R.
https://www.hackerearth.com/zh/practice/machine-learning/machine-learning-algorithms/beginners-tutorial-on-xgboost-parameter-tuning-r/ tutorial/

[2] Simple xgboost with linear model in R.
https://www.kaggle.com/talcapa/simple-xgboost-with-linear-model-in-r

[3] Gradient Boosting Machines.
http://uc-r.github.io/gbm_regression

R Code

````
---
title: "module3-2"
author: "Jiaqi Li"
date: "2018??11??21??"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```{r}
rm(list = ls(all = TRUE))

# load plotting library
library(ggplot2)
library(mice)
library(xgboost)
library(Matrix)
library(methods)
library(caret)
library(randomForest)
library(dplyr)

# close all previous plots
while (!is.null(dev.list())) dev.off()

# read test/train and solution databases
xtrain = read.csv("D:/stat440/module3/module3/Xtrain.txt", sep=' ', row.names = NULL,
header = TRUE)
ytrain = read.table("D:/stat440/module3/module3/Ytrain.txt", sep=',', row.names = NULL,
header = TRUE)
test = read.csv("D:/stat440/module3/module3/Xtest.txt", sep=' ', row.names = NULL, header
= TRUE)




```

```{r}
xtrain
ytrain
test
```
````

```{r}
library(DMwR)
xtrain[,"X.B17"]=as.numeric(as.character(xtrain[,"X.B17"]))




xtrain1=cbind(ytrain,xtrain)
xtrain1
xtrain1=xtrain1[,-c(1,3,66:78)]


test[,"X.B17"]=as.numeric(as.character(test[,"X.B17"]))


Xtest1 = test[,-c(1,64:76)]
```

```{r}
xtrain1
Xtest1
```

```{r}



test2=as.matrix(Xtest1)
train1=as.matrix(xtrain1)


dtrain <- xgb.DMatrix(data = train1[,2:63], label= train1[,1])

dtest <- xgb.DMatrix(data = test2[,1:62])

```

```{r}
params <- list(booster = "dart", objective = "reg:linear")
xgbcv <- xgb.cv( params = params, data = dtrain, nrounds =200, nfold = 5, showsd = T,
stratified = T, print_every_n = 10, early_stop_round = 20,
                 maximize =
F,eta=0.3,gamma=5,colsample_bytree=0.9,subsample=0.8,alpha=5,max_depth=6,min_child
_weight=1)
print(xgbcv$evaluation_log[which.min(xgbcv$evaluation_log$test_rmse_mean)])
```

```

```{r}
xgb1 <- xgb.train (params = params, data = dtrain, nrounds = 300, print_every_n = 10,
early_stop_round = 10, maximize = F , eval_metric =
"rmse",eta=0.3,gamma=5,colsample_bytree=0.9,subsample=0.8,alpha=5,max_depth=5,min
_child_weight=1)

Value=predict(xgb1,dtest)

```

```{r}
result=data.frame(Value)


result=cbind(Id=test$Id,result)
result

```

```{r}
write.csv(result,file='result1.csv',row.names = FALSE)
```
```