InstacartTreebasedModels

Yihong Qiu

6/8/2020

ALY 6040 Data Mining Applications

Assignment 3: Instacart Tree Based Models and Model Optimization

Shivangi Vashi

Yihong Qiu

Md Tajrianul Islam

Instructor: Kasun Samarasinghe

Spring 2020

June 8 2020

Northeastern University

Introduction

This week we use Instacart Maket Basket dataset to conduct analysis and to predict outcomes by using the tree based models. There are two types of tree-based models: regression and classification. A regression tree is used for a continuous dependent variable, while a classification tree is used for a categorical dependent variable. In this report,we will use Decision Tree, Random Forest and K-means Clustering to test the models in our analysis.

First, reading the dataset and importing relevant libraries.

```
library(plyr)
library(tidyverse)
library(data.table)
library(rpart)
library(RColorBrewer)
library(rattle)
library(randomForest)
library(NbClust)
library(dplyr)
library(ggfortify)
library(factoextra)


#using fread because it reads data very fast
#aisles<-fread("instacart-market-basket-analysis/aisles.csv")
#departments<-fread("instacart-market-basket-analysis/departments.csv")
order_products_prior<-fread("instacart-market-basket-analysis/order_products__prior.csv")
order_products_train<-fread("instacart-market-basket-analysis/order_products__train.csv")
orders<-fread("instacart-market-basket-analysis/orders.csv")
#products<-fread("instacart-market-basket-analysis/products.csv")
```

Data Wrangling

Data Preparation Since the dataset is very large, with Prior orders having 32 million rows, we subset the data to reduce calculation time. We did this by randomly sampling users, then only keeping their orders and prior order information by performing inner joins with the order prior and train datasets.

```
set.seed(123)
user_fraction <- 0.1
users <- unique(orders$user_id)
sample_users <- sample(users, round(user_fraction * length(users)))

cat('Number of orders (before): ', nrow(orders))
```

```
## Number of orders (before):  3421083
```

```
orders <- orders[user_id %in% sample_users]
cat('Number of orders (after): ', nrow(orders))
```

```
## Number of orders (after):  346739
```

```
# Training dataset
OrderProductPrior<-orders%>%inner_join(order_products_prior)
OrderProductPrior<-drop_na(OrderProductPrior)
OrderProductPrior<-OrderProductPrior[-c(1,2,3,9)]

#Testing dataset
OrderProductTrain<-orders%>%inner_join(order_products_train)
OrderProductTrain<-drop_na(OrderProductTrain)
OrderProductTrain<-OrderProductTrain[-c(1,2,3,9)]

dim(OrderProductPrior)
```

```
## [1] 3061150       6
```

```
dim(OrderProductTrain)
```

```
## [1] 137284      6
```

```
head(OrderProductPrior)
```

```
##    order_number order_dow order_hour_of_day days_since_prior_order product_id
## 10            2         0                17                       1      36216
## 11            2         0                17                       1       4461
## 12            2         0                17                       1       5876
## 13            2         0                17                       1        810
## 14            2         0                17                       1      31717
## 15            3         1                19                       8      36216
##    reordered
## 10         1
## 11         0
## 12         0
## 13         0
## 14         0
## 15         1
```

```
head(OrderProductTrain)
```

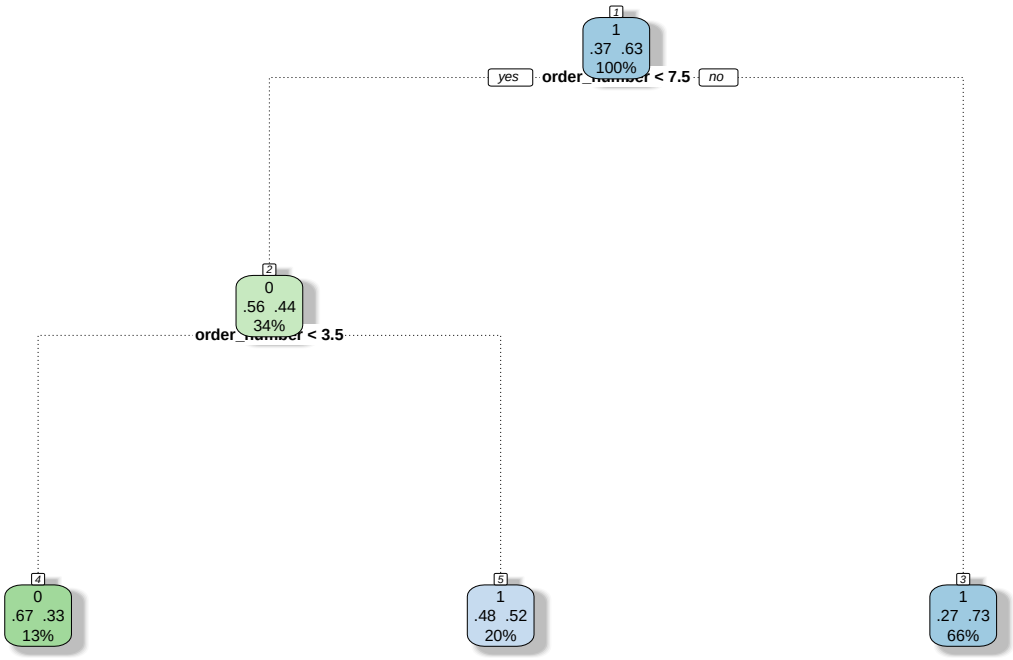
```
##   order_number order_dow order_hour_of_day days_since_prior_order product_id
## 1            7         6                9                       7      36216
```

```
## 2             7         6                 9                      7       47546
## 3             7         6                 9                      7       21137
## 4             7         6                 9                      7        5450
## 5             7         6                 9                      7        8518
## 6             7         6                 9                      7       22031
##   reordered
## 1         1
## 2         1
## 3         1
## 4         0
## 5         0
## 6         1
```

Analysis

Building Decision Tree Model We build the decision tree to seek when the reordered is 1, how order_number will be distributed, when order_number is less than 7.5, it will go to more specific like whether it is less than 3.5 as shown in the decision tree plot. The result shows that 66% data are order more than 7.5, 20% order data are between 3.5 to 7.5

```
#Create the decision tree model
OrderProductTree<- rpart(reordered~., data = OrderProductPrior, method = 'class')

# Plot the model
fancyRpartPlot(OrderProductTree, cex = 0.5)
```



Rattle 2020−Jun−10 23:48:54 venusqiu

Prediction Then we do the prediction of the decision tree model. The dataset correctly predict 272,953 won't reorder and 1,802,414 out of 2,653,957 are reordered. The accurary of the test is 68.79%, which is pretty good.

```
pred <-predict(OrderProductTree, OrderProductPrior, type = 'class')

Table<-table(OrderProductPrior$reordered, pred)
Table
```

```
##    pred
##           0       1
##   0  272953  851543
##   1  134240 1802414
```

```
Accuracy<-sum(diag(Table)) / sum(Table)

print(paste('Accuracy for test', Accuracy))
```

```
## [1] "Accuracy for test 0.67796971726312"
```

Advantages and Disadvantages of Decision Tree Advantage of Decision Tree: Decision tree split from the top down, grouping data into the most homogeneous "sub-nodes" based on their characteristics, so it perform well with categorical variables. It can process missing values quite well. Besides, it is easy to understand, interpret and visualize. Disadvantage of Decision Tree: It might be intorelant for a small change in data cause the model to shift and the outliers have a big impact. Also, it would go too deep and get overfitting for the result.

Random Forest In the Random Forest outcome, we get a 33.75% estimate of error rate, which is okay. Based on the confusion matrix, the dataset predicts 29,506 won't reorder, and 61,449 will reorder. As we know, the higher Mean Decrease Accuracy and Mean Decrease Gini we get, the higher accuracy of the variables we choose. From the results, variables order_number, order_dow, order_hour_of_day, days_since_prior_order and product_id perform well in this model.

```
OrderProductTrain$reordered<-as.factor(OrderProductTrain$reordered)

RF <- randomForest(reordered ~., data=OrderProductTrain, ntree=500,
                           keep.forest=FALSE, importance=TRUE)
print(RF)
```

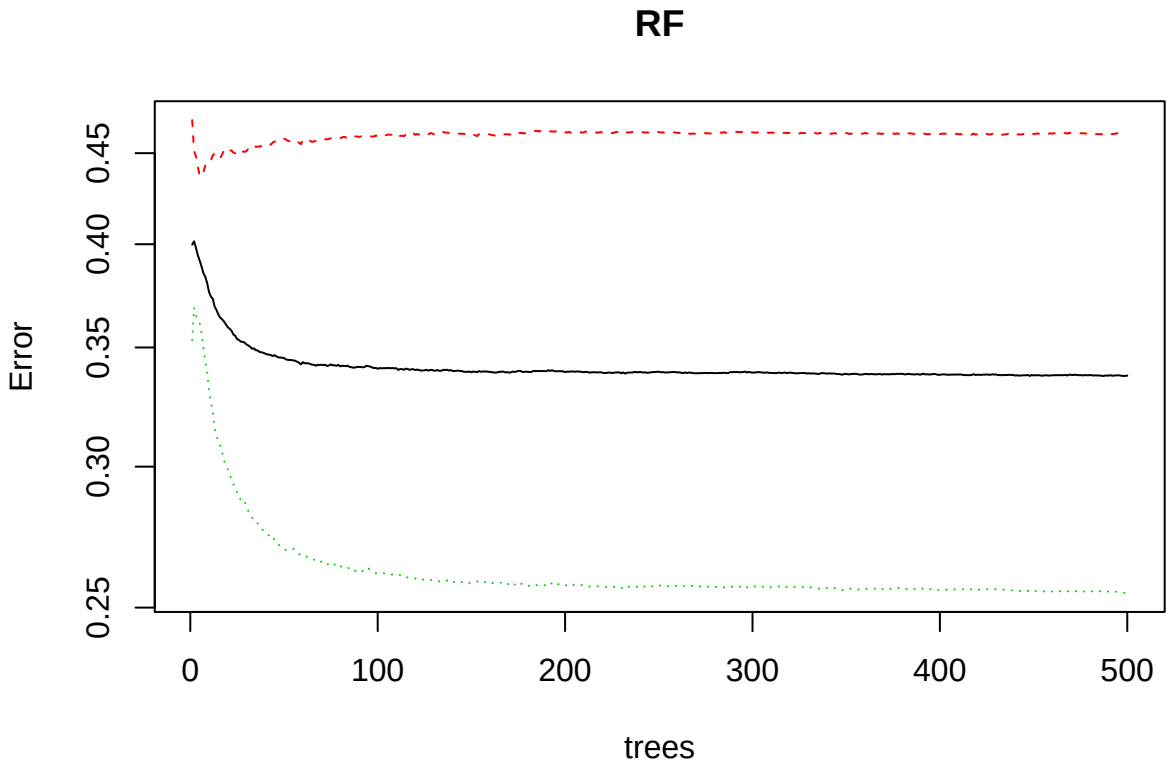
```
##
## Call:
##  randomForest(formula = reordered ~ ., data = OrderProductTrain,      ntree = 500, keep.forest = FALS
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 33.75%
## Confusion matrix:
##       0     1 class.error
## 0 29506 25302   0.4616479
## 1 21027 61449   0.2549469
```

```
importance(RF)
```

```
##                                0         1 MeanDecreaseAccuracy
## order_number           200.34385 201.26907             258.74271
## order_dow              100.88156 112.41843             128.94184
## order_hour_of_day      111.72136 112.39366             130.47859
## days_since_prior_order 148.30181 124.99507             173.64575
## product_id              23.36255  17.99741              28.31051
```

```
##                         MeanDecreaseGini
## order_number                    8437.953
## order_dow                       3947.134
## order_hour_of_day               6141.016
## days_since_prior_order          6705.436
## product_id                     18419.259
```

```
plot(RF, log="y")
```




```
#randomForest(reordered ~ ., OrderProductTrain, keep.forest=FALSE, ntree=100)
```

Advantages and Disadvantages of Random Forest Advantages of Random Forest: It usually have very good performance and easy to understand. Also it provides a reliable feature importance estimate because it applied a large number of individual decision trees. Random forest can solve both type of problems that is classification and regression and does a decent estimation at both fronts. There is no pre-processing required. It is robust to outliers. Disadvantages of Random Forest: It is less interpretable than an individual decision tree. It can become slow on large datasets. Although it is more accurate, but it cannot compete with advanced boosting algorithms. Training a large number of deep trees can have high computational costs and use a lot of memory.

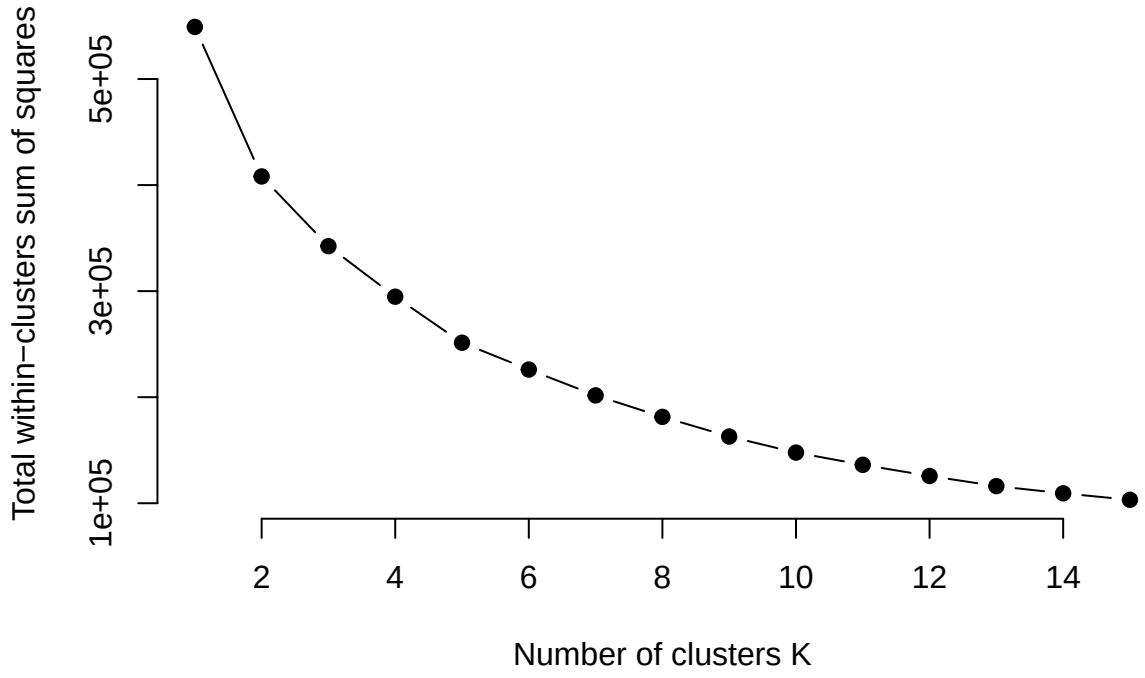
```
Product_cluster <- OrderProductTrain %>%
  mutate(days_since_prior_order = as.numeric(days_since_prior_order)) %>%
  transmute(product_id=product_id,order_hour=order_hour_of_day,days_since_prior_order, reordered= as.nu


#Scaling Data
ProductScaled <- scale(Product_cluster[, -1])

# Set max number of clusters as 15
```

```
k.max <- 15
# Compute and plot wss for k = 2 to k = 15.
wss <- sapply(1:k.max,
              function(k){
                kmeans(ProductScaled, k, nstart=50,iter.max = 15 )$tot.withinss})
wss
```

Clustering

```
##  [1] 549132.0 408130.4 342418.9 294749.3 251352.2 226045.1 201681.4 181413.0
##  [9] 162907.9 147748.5 136221.6 125706.5 116131.9 109364.7 103227.2
```

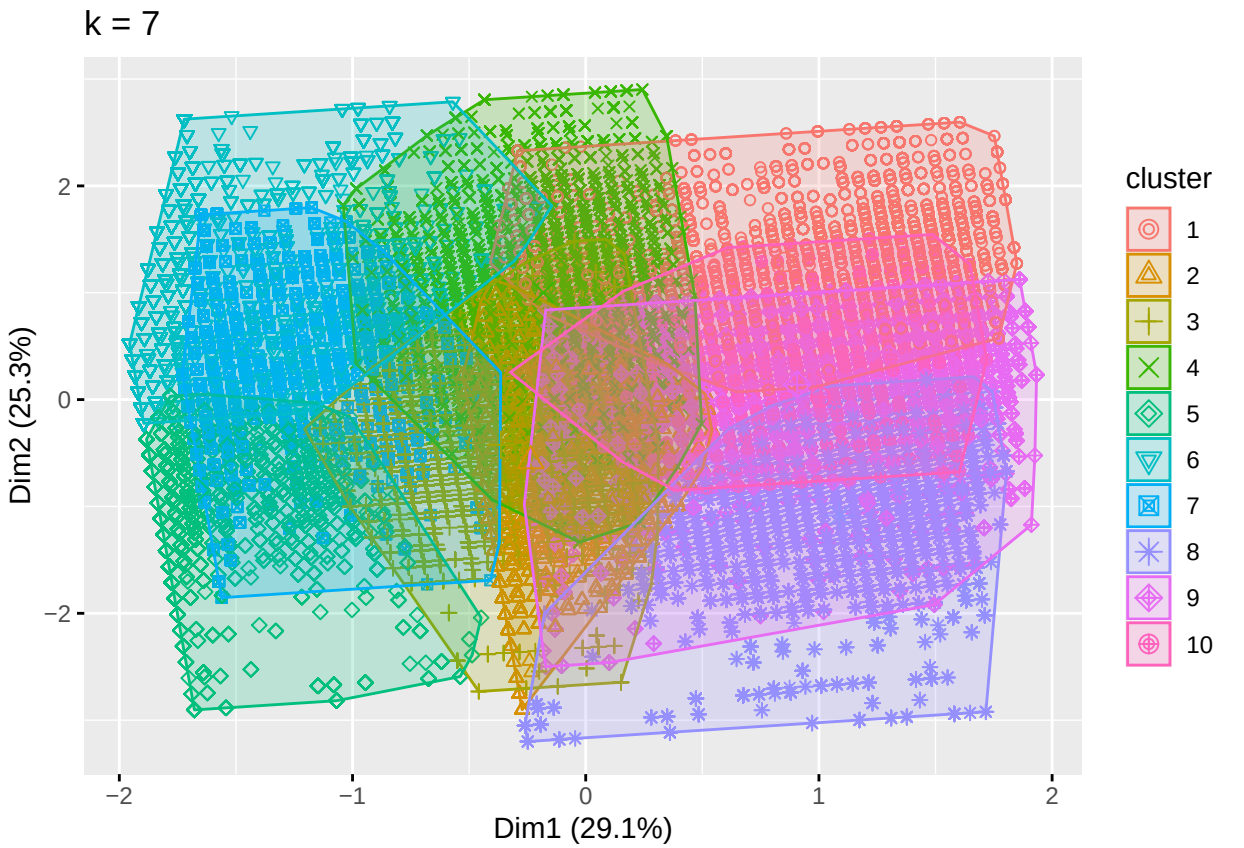
```
plot(1:k.max, wss, type="b", pch = 19, frame = FALSE, xlab="Number of clusters K",ylab="Total within-clu
```




```
#From the plot you can see that the elbow is at n=4 hence number of clusters= 5

fitK <- kmeans(ProductScaled, 10)
str(fitK)
```

```
## List of 9
##  $ cluster     : int [1:137284] 9 9 9 4 4 9 4 4 4 9 ...
##  $ centers     : num [1:10, 1:4] 0.8718 0.0202 -0.0046 0.0586 -0.3594 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:10] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:4] "order_hour" "days_since_prior_order" "reordered" "order_day"
##  $ totss       : num 549132
##  $ withinss    : num [1:10] 20637 25992 13611 15894 8744 ...
##  $ tot.withinss: num 155208
##  $ betweenss   : num 393924
##  $ size        : int [1:10] 15325 20921 11707 12253 11543 8675 10449 18624 15368 12419
##  $ iter        : int 5
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

```
fviz_cluster(fitK, geom = "point", data = ProductScaled) + ggtitle("k = 7")
```



So we are trying to clusters the products that are often bought together. So if we are trying to build a recommendation system to suggest similar products with the least euclidean distance from the 1st add to cart product.

We use the elbow plot to find out how many clusters or k's to set for the model. Right where the curve bends, ie at the elbow is where the optimum number of clusters is. For our data it is at n=10. Which is pretty much understandable, considering we are talking about a grocery store it is possible to have 10 clusters of product groups that are mostly purchased together considering which day, which time and whether it was ordered before or not. The model does not perform well, and we can conclude that k means clustering does not make sense for this data- there is not much structure to it.