954:534 Wish Project

Kanya Kreprasertkul, Yelin Shin and Zhe Ren

options(warn = -1)  
library(dplyr)  
library(tidyr)  
#library(tidyverse)  
library(GGally)  
library(plotly)  
library(cowplot)  
library(ggcorrplot)  
library(stringr)

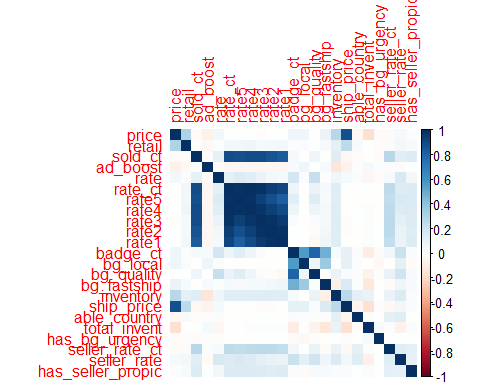
## Data pre-processing

wish <- read.csv('summer-products-with-rating-and-performance\_2020-08.csv')  
  
#dropping unnecessary columns   
drops <- c('title', 'tags', 'crawl\_month', 'theme', 'product\_id', 'product\_picture', 'product\_url', 'merchant\_id', 'merchant\_profile\_picture', 'merchant\_info\_subtitle', 'merchant\_name', 'merchant\_title', 'urgency\_text', 'title\_orig', 'shipping\_option\_name', 'currency\_buyer')  
wish <- wish[, !(names(wish) %in% drops)]  
  
#convert NA to 0  
wish$has\_urgency\_banner <- as.integer(wish$has\_urgency\_banner)  
wish$has\_urgency\_banner[which(is.na(wish$has\_urgency\_banner))] <- 0  
wish$rating\_five\_count[which(is.na(wish$rating\_five\_count))] <- 0  
wish$rating\_four\_count[which(is.na(wish$rating\_four\_count))] <- 0  
wish$rating\_three\_count[which(is.na(wish$rating\_three\_count))] <- 0  
wish$rating\_two\_count[which(is.na(wish$rating\_two\_count))] <- 0  
wish$rating\_one\_count[which(is.na(wish$rating\_one\_count))] <- 0  
wish$rating[which(wish$rating\_count == 0)] <- 0  
  
# cleaning size and color option  
wish <- wish %>%  
 mutate(product\_variation\_size\_id = tolower(product\_variation\_size\_id)) %>%  
 mutate(product\_variation\_size\_id = gsub(pattern = '.', replacement = '',  
 x = product\_variation\_size\_id, fixed = TRUE)) %>%  
 mutate(product\_variation\_size\_id = gsub(pattern = '(size-\*)|(size)', replacement = '',  
 x = product\_variation\_size\_id)) %>%  
 mutate(product\_variation\_size\_id = gsub(pattern = '.+[-]', replacement = '',  
 x = product\_variation\_size\_id)) %>%  
 mutate(product\_variation\_size\_id = ifelse(grepl(pattern = 'xl',product\_variation\_size\_id),  
 'xl', product\_variation\_size\_id)) %>%  
 mutate(product\_variation\_size\_id = ifelse(grepl(pattern = 'xs', product\_variation\_size\_id),  
 'xs', product\_variation\_size\_id)) %>%  
 mutate(product\_variation\_size\_id = str\_replace(product\_variation\_size\_id, ' ', '')) %>%  
 mutate(product\_variation\_size\_id = ifelse(product\_variation\_size\_id %in% c('s', 'xs', 'm', 'l', 'xl'),product\_variation\_size\_id, 'One-sized'))  
wish <- wish %>%   
 mutate(product\_color = tolower(product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'red|burgundy|claret|wine|jasper', product\_color),  
 'red', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'blue|navy', product\_color),  
 'blue', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'white', product\_color),  
 'white', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'green|army', product\_color),  
 'green', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'black', product\_color),  
 'black', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'yellow|leopard|gold', product\_color),  
 'yellow', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'pink|rose', product\_color),  
 'pink', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'grey|gray|silver', product\_color),  
 'gray', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'purple|violet', product\_color),  
 'purple', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'orange|apricot', product\_color),  
 'orange', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'beige|nude|ivory|coffee|brown|khaki|camel',  
 product\_color), 'khaki', product\_color)) %>%  
 mutate(product\_color = ifelse(grepl(pattern = 'floral|multicolor|camouflage|rainbow|star',  
 product\_color), 'multicolor', product\_color))  
  
#name blank category   
wish['product\_color'][wish['product\_color'] == ''] <- 'Not defined'  
wish['origin\_country'][wish['origin\_country'] == ''] <- 'Not defined'  
  
#shipping\_is\_express has too many zero, so we decided to exclude this column  
wish <- select(wish, -c(shipping\_is\_express))  
  
#Only 7 colors have more than 100 records so We decided to keep only 8 factors of color, i.e. black, white, blue, red, green, yellow, pink and others.  
color\_list <- c('black', 'white', 'blue', 'red', 'green', 'yellow', 'pink')  
wish$product\_color[!(wish$product\_color %in% color\_list)] <- 'others'  
  
wish %>%   
 group\_by(product\_color) %>%  
 summarise(no\_rows = length(product\_color)) %>%  
 arrange(desc(no\_rows)) %>%  
 filter(no\_rows > 100)  
  
#We decided to change origin to CN and others.  
wish$origin\_country <- as.character(wish$origin\_country)  
wish$origin\_country[which(wish$origin\_country != 'CN')] <- 'others'  
wish$origin\_country[is.na(wish$origin\_country)] <- 'others'  
  
wish %>%   
 group\_by(origin\_country) %>%  
 summarise(no\_rows = length(origin\_country)) %>%  
 arrange(desc(no\_rows))   
  
#convert column name to short version  
origin\_colname <- colnames(wish)  
colnames(wish) <- c('price', 'retail', 'sold\_ct', 'ad\_boost', 'rate', 'rate\_ct', 'rate5', 'rate4', 'rate3', 'rate2', 'rate1', 'badge\_ct', 'bg\_local', 'bg\_quality', 'bg\_fastship', 'color', 'size', 'inventory', 'ship\_price', 'able\_country', 'total\_invent', 'has\_bg\_urgency', 'origin', 'seller\_rate\_ct', 'seller\_rate', 'has\_seller\_propic')

library(corrplot)

## corrplot 0.84 loaded

# finding correlation between numeric columns and charges  
  
numeric.column <- sapply(wish, is.numeric)  
corr <- cor(wish[, numeric.column]) #, use = 'pairwise.complete.obs'  
corrplot(corr, method = 'color')



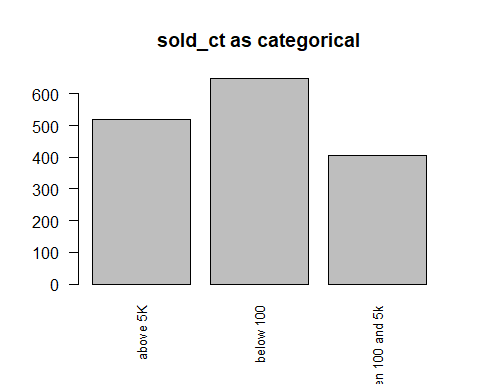
#convert the y (sold\_ct) to categorical. Also since it is unbalanced we group some category together.  
table(wish['sold\_ct']) # very unbalaned

##   
## 1 2 3 6 7 8 10 50 100 1000 5000   
## 3 2 2 1 2 4 49 76 509 405 217   
## 10000 20000 50000 100000   
## 177 103 17 6

wish\_cate <- wish  
wish\_cate$sold\_ct\_cate <- wish\_cate$sold\_ct  
wish\_cate$sold\_ct\_cate[which(wish\_cate$sold\_ct <= 100)] <- 'below 100'  
wish\_cate$sold\_ct\_cate[which(wish\_cate$sold\_ct >= 5000)] <- 'above 5K'  
wish\_cate$sold\_ct\_cate[which(wish\_cate$sold\_ct > 100 & wish\_cate$sold\_ct < 5000)] <- 'between 100 and 5k'  
wish\_cate <- select(wish\_cate, -sold\_ct)  
wish\_cate$sold\_ct\_cate <- as.factor(wish\_cate$sold\_ct\_cate)  
wish\_cate$color <- as.factor(wish\_cate$color)  
wish\_cate$size <- as.factor(wish\_cate$size)  
wish\_cate$origin <- as.factor(wish\_cate$origin)  
table(wish\_cate$sold\_ct\_cate) # much better

##   
## above 5K below 100 between 100 and 5k   
## 520 648 405

x1 <- factor(wish\_cate$sold\_ct\_cate)  
tb <- table(x1)  
barplot(tb, names.arg = row.names(tb), cex.names = 0.8, main = "sold\_ct as categorical", las = 2)



#percentage of each rate count   
wish\_cate$rate5\_pct <- wish\_cate$rate5/wish\_cate$rate\_ct  
wish\_cate$rate4\_pct <- wish\_cate$rate4/wish\_cate$rate\_ct  
wish\_cate$rate3\_pct <- wish\_cate$rate3/wish\_cate$rate\_ct  
wish\_cate$rate2\_pct <- wish\_cate$rate2/wish\_cate$rate\_ct  
wish\_cate$rate1\_pct <- wish\_cate$rate1/wish\_cate$rate\_ct  
  
drops <- c('rate\_ct', 'rate5', 'rate4', 'rate3', 'rate2', 'rate1')  
wish\_cate <- wish\_cate[, !(names(wish\_cate) %in% drops)]  
  
wish\_cate <- wish\_cate %>% drop\_na(rate5\_pct)   
wish\_cate <- wish\_cate %>% drop\_na(price)   
  
summary(wish\_cate)

## price retail ad\_boost rate   
## Min. : 1.000 Min. : 1.00 Min. :0.0000 Min. :1.000   
## 1st Qu.: 5.830 1st Qu.: 7.00 1st Qu.:0.0000 1st Qu.:3.530   
## Median : 8.000 Median : 10.00 Median :0.0000 Median :3.830   
## Mean : 8.335 Mean : 23.27 Mean :0.4332 Mean :3.786   
## 3rd Qu.:11.000 3rd Qu.: 26.00 3rd Qu.:1.0000 3rd Qu.:4.090   
## Max. :49.000 Max. :252.00 Max. :1.0000 Max. :5.000   
##   
## badge\_ct bg\_local bg\_quality bg\_fastship   
## Min. :0.0000 Min. :0.00000 Min. :0.00000 Min. :0.00000   
## 1st Qu.:0.0000 1st Qu.:0.00000 1st Qu.:0.00000 1st Qu.:0.00000   
## Median :0.0000 Median :0.00000 Median :0.00000 Median :0.00000   
## Mean :0.1086 Mean :0.01898 Mean :0.07657 Mean :0.01309   
## 3rd Qu.:0.0000 3rd Qu.:0.00000 3rd Qu.:0.00000 3rd Qu.:0.00000   
## Max. :3.0000 Max. :1.00000 Max. :1.00000 Max. :1.00000   
##   
## color size inventory ship\_price   
## black :304 l : 53 Min. : 1.00 Min. : 1.000   
## others :278 m :204 1st Qu.: 6.00 1st Qu.: 2.000   
## white :268 One-sized: 87 Median :50.00 Median : 2.000   
## blue :167 s :669 Mean :33.22 Mean : 2.345   
## red :150 xl : 56 3rd Qu.:50.00 3rd Qu.: 3.000   
## green :138 xs :459 Max. :50.00 Max. :12.000   
## (Other):223   
## able\_country total\_invent has\_bg\_urgency origin   
## Min. : 6.00 Min. : 1.00 Min. :0.000 CN :1472   
## 1st Qu.: 31.00 1st Qu.:50.00 1st Qu.:0.000 others: 56   
## Median : 40.00 Median :50.00 Median :0.000   
## Mean : 40.45 Mean :49.82 Mean :0.301   
## 3rd Qu.: 43.00 3rd Qu.:50.00 3rd Qu.:1.000   
## Max. :140.00 Max. :50.00 Max. :1.000   
##   
## seller\_rate\_ct seller\_rate has\_seller\_propic sold\_ct\_cate  
## Min. : 3 Min. :2.941 Min. :0.0000 above 5K :520   
## 1st Qu.: 2116 1st Qu.:3.919 1st Qu.:0.0000 below 100 :603   
## Median : 8194 Median :4.041 Median :0.0000 between 100 and 5k:405   
## Mean : 26667 Mean :4.033 Mean :0.1466   
## 3rd Qu.: 24616 3rd Qu.:4.160 3rd Qu.:0.0000   
## Max. :2174765 Max. :4.578 Max. :1.0000   
##   
## rate5\_pct rate4\_pct rate3\_pct rate2\_pct   
## Min. :0.0000 Min. :0.0000 Min. :0.0000 Min. :0.00000   
## 1st Qu.:0.3889 1st Qu.:0.1603 1st Qu.:0.1082 1st Qu.:0.04147   
## Median :0.4715 Median :0.1879 Median :0.1429 Median :0.06667   
## Mean :0.4680 Mean :0.1861 Mean :0.1406 Mean :0.07467   
## 3rd Qu.:0.5537 3rd Qu.:0.2143 3rd Qu.:0.1693 3rd Qu.:0.09140   
## Max. :1.0000 Max. :0.6667 Max. :0.5000 Max. :1.00000   
##   
## rate1\_pct   
## Min. :0.00000   
## 1st Qu.:0.07021   
## Median :0.11073   
## Mean :0.13069   
## 3rd Qu.:0.16874   
## Max. :1.00000   
##

str(wish\_cate)

## 'data.frame': 1528 obs. of 25 variables:  
## $ price : num 16 8 8 8 2.72 3.92 7 12 11 5.78 ...  
## $ retail : int 14 22 43 8 3 9 6 11 84 22 ...  
## $ ad\_boost : int 0 1 0 1 1 0 0 0 1 0 ...  
## $ rate : num 3.76 3.45 3.57 4.03 3.1 5 3.84 3.76 3.47 3.6 ...  
## $ badge\_ct : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ bg\_local : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ bg\_quality : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ bg\_fastship : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ color : Factor w/ 8 levels "black","blue",..: 7 3 8 1 8 2 7 2 1 4 ...  
## $ size : Factor w/ 6 levels "l","m","One-sized",..: 2 6 6 2 4 6 6 2 2 4 ...  
## $ inventory : int 50 50 1 50 1 1 50 50 50 50 ...  
## $ ship\_price : int 4 2 3 2 1 1 2 3 2 2 ...  
## $ able\_country : int 34 41 36 41 35 40 31 139 36 33 ...  
## $ total\_invent : int 50 50 50 50 50 50 50 50 50 50 ...  
## $ has\_bg\_urgency : num 1 1 1 0 1 0 0 0 1 0 ...  
## $ origin : Factor w/ 2 levels "CN","others": 1 1 1 1 1 1 1 1 1 1 ...  
## $ seller\_rate\_ct : int 568 17752 295 23832 14482 65 10194 342 330 5534 ...  
## $ seller\_rate : num 4.13 3.9 3.99 4.02 4 ...  
## $ has\_seller\_propic: int 0 0 0 0 0 0 1 0 0 0 ...  
## $ sold\_ct\_cate : Factor w/ 3 levels "above 5K","below 100",..: 2 1 2 1 2 2 1 3 2 1 ...  
## $ rate5\_pct : num 0.481 0.37 0.357 0.509 0.3 ...  
## $ rate4\_pct : num 0.148 0.167 0.286 0.206 0.2 ...  
## $ rate3\_pct : num 0.185 0.182 0.143 0.15 0.1 ...  
## $ rate2\_pct : num 0.0185 0.105 0 0.0725 0.1 ...  
## $ rate1\_pct : num 0.1667 0.1756 0.2143 0.0622 0.3 ...

## Methodology

#### 80:20 split for train and test set

set.seed(123)  
  
train\_rows <- sample(1:nrow(wish), 0.8 \* nrow(wish))  
wish.train <- wish\_cate[train\_rows, ] # wish training set  
wish.test <- wish\_cate[-train\_rows, ]  
  
wish.train <- wish.train %>% drop\_na(price)

### Multinomial Regression

set.seed(123)  
library(nnet)  
multinomial.mod <- multinom(sold\_ct\_cate ~ ., data = wish.train) #, na.action = na.roughfix

## # weights: 108 (70 variable)  
## initial value 1339.208380   
## iter 10 value 1255.977483  
## iter 20 value 1189.343320  
## iter 30 value 1164.442293  
## iter 40 value 1161.648181  
## iter 50 value 1160.944105  
## iter 60 value 1160.541882  
## iter 70 value 1159.267178  
## iter 80 value 1158.778710  
## final value 1158.151626   
## converged

summary(multinomial.mod)

## Call:  
## multinom(formula = sold\_ct\_cate ~ ., data = wish.train)  
##   
## Coefficients:  
## (Intercept) price retail ad\_boost rate  
## below 100 61.52575 0.05993473 0.007125439 0.02511886 52.47417  
## between 100 and 5k 162.13209 0.05669671 0.006771135 0.19507681 8.30567  
## badge\_ct bg\_local bg\_quality bg\_fastship colorblue  
## below 100 -0.09765252 -1.0448405 -0.07811868 1.02530663 -0.17996180  
## between 100 and 5k 0.01625992 -0.2818895 0.24682887 0.05132056 -0.03807248  
## colorgreen colorothers colorpink colorred colorwhite  
## below 100 0.2602755 -0.187931448 -0.0856353 0.1882070 -0.27893981  
## between 100 and 5k 0.3371423 -0.000843693 0.4433978 0.6170049 -0.04267305  
## coloryellow sizem sizeOne-sized sizes sizexl  
## below 100 0.9822648 0.1970586 1.5743267 0.41382825 0.8979739  
## between 100 and 5k 1.2630431 -0.3892427 0.5845379 0.01774944 -0.3659442  
## sizexs inventory ship\_price able\_country total\_invent  
## below 100 1.3233542 -0.011385347 -0.17951453 0.01114518 -4.493798  
## between 100 and 5k 0.5343495 -0.006466085 -0.05803779 0.01434421 -4.365661  
## has\_bg\_urgency originothers seller\_rate\_ct seller\_rate  
## below 100 0.09241284 1.0077496 -1.677742e-05 -1.6910850  
## between 100 and 5k 0.17423052 0.1463904 -1.292792e-05 -0.5705007  
## has\_seller\_propic rate5\_pct rate4\_pct rate3\_pct rate2\_pct  
## below 100 -0.33584439 -93.12020 -40.06576 8.845931 65.00529  
## between 100 and 5k 0.07444447 15.54658 24.04949 32.023675 39.89597  
## rate1\_pct  
## below 100 120.86049  
## between 100 and 5k 50.61638  
##   
## Std. Errors:  
## (Intercept) price retail ad\_boost  
## below 100 8.607617e-05 0.0005094232 0.002561476 5.506705e-05  
## between 100 and 5k 9.088986e-05 0.0006262238 0.002580111 6.270339e-05  
## rate badge\_ct bg\_local bg\_quality  
## below 100 0.0003177981 4.290554e-06 1.452266e-06 5.115773e-06  
## between 100 and 5k 0.0003372213 9.832451e-06 3.080471e-06 8.086474e-06  
## bg\_fastship colorblue colorgreen colorothers  
## below 100 1.197296e-06 6.700695e-06 1.170119e-05 2.473794e-05  
## between 100 and 5k 8.439712e-07 6.012753e-06 1.348777e-05 2.312286e-05  
## colorpink colorred colorwhite coloryellow  
## below 100 7.596063e-06 1.558492e-05 2.029067e-05 1.036194e-05  
## between 100 and 5k 8.749608e-06 1.770766e-05 1.659566e-05 1.452224e-05  
## sizem sizeOne-sized sizes sizexl  
## below 100 9.245807e-06 4.306001e-06 2.445879e-05 2.957718e-06  
## between 100 and 5k 4.759126e-06 6.895911e-06 3.000295e-05 1.796344e-06  
## sizexs inventory ship\_price able\_country  
## below 100 8.797572e-05 0.003763135 0.0001556580 0.004116648  
## between 100 and 5k 8.126724e-05 0.004002453 0.0001837391 0.004186555  
## total\_invent has\_bg\_urgency originothers seller\_rate\_ct  
## below 100 0.004305804 2.088730e-05 2.187938e-06 2.664673e-06  
## between 100 and 5k 0.004542852 2.247189e-05 1.212141e-06 2.407260e-06  
## seller\_rate has\_seller\_propic rate5\_pct rate4\_pct  
## below 100 0.0003359766 1.283221e-05 3.756664e-05 1.633464e-05  
## between 100 and 5k 0.0003572466 2.017580e-05 3.987934e-05 1.703798e-05  
## rate3\_pct rate2\_pct rate1\_pct  
## below 100 1.258024e-05 7.464148e-06 1.229044e-05  
## between 100 and 5k 1.458081e-05 6.645865e-06 1.281971e-05  
##   
## Residual Deviance: 2316.303   
## AIC: 2448.303

multinomial.pred\_train <- predict(multinomial.mod, wish.train)   
multinomial.pred\_test <- predict(multinomial.mod, wish.test)  
# training error  
print("Misclassification rate on the training set:")

## [1] "Misclassification rate on the training set:"

mean(as.character(multinomial.pred\_train) != as.character(wish.train$sold\_ct\_cate))

## [1] 0.4577523

# test error  
print("Misclassification rate on the test set:")

## [1] "Misclassification rate on the test set:"

mean(as.character(multinomial.pred\_test) != as.character(wish.test$sold\_ct\_cate))

## [1] 0.4854369

confusion.matrix <- table(wish.test$sold\_ct\_cate, multinomial.pred\_test)  
print(confusion.matrix)

## multinomial.pred\_test  
## above 5K below 100 between 100 and 5k  
## above 5K 61 28 15  
## below 100 27 90 7  
## between 100 and 5k 26 47 8

accuracy.percent <- 100\*sum(diag(confusion.matrix))/sum(confusion.matrix)  
above5k.precent <- 100\*confusion.matrix[1,1]/sum(confusion.matrix[1,])  
print(paste("Test accuracy:",accuracy.percent,"%"))

## [1] "Test accuracy: 51.4563106796116 %"

print(paste("Above 5k accuracy:",above5k.precent,"%"))

## [1] "Above 5k accuracy: 58.6538461538462 %"

### Dicision Tree Models

library(tree)

## Registered S3 method overwritten by 'tree':  
## method from  
## print.tree cli

set.seed(123)  
  
tree.wish <- tree(sold\_ct\_cate ~ ., data = wish.train)   
summary(tree.wish)

##   
## Classification tree:  
## tree(formula = sold\_ct\_cate ~ ., data = wish.train)  
## Variables actually used in tree construction:  
## [1] "rate2\_pct" "seller\_rate\_ct" "rate3\_pct" "rate4\_pct"   
## Number of terminal nodes: 10   
## Residual mean deviance: 1.418 = 1715 / 1209   
## Misclassification error rate: 0.3158 = 385 / 1219

tree.pred <- predict(tree.wish, wish.test, type = "class")  
  
# table(tree.pred, wish.test$sold\_ct\_cate)  
  
# print("Misclassification error rate on test set: ")  
  
confusion.matrix <- table(wish.test$sold\_ct\_cate, tree.pred)  
print(confusion.matrix)

## tree.pred  
## above 5K below 100 between 100 and 5k  
## above 5K 88 5 11  
## below 100 13 98 13  
## between 100 and 5k 42 20 19

accuracy.percent <- 100\*sum(diag(confusion.matrix))/sum(confusion.matrix)  
above5k.precent <- 100\*confusion.matrix[1,1]/sum(confusion.matrix[1,])  
print(paste("Test accuracy:",accuracy.percent,"%"))

## [1] "Test accuracy: 66.3430420711974 %"

print(paste("Above 5k accuracy:",above5k.precent,"%"))

## [1] "Above 5k accuracy: 84.6153846153846 %"

### Bagging

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':  
##   
## margin

## The following object is masked from 'package:dplyr':  
##   
## combine

set.seed(123)  
  
bag.wish <- randomForest(sold\_ct\_cate ~ ., data = wish.train, mtry = length(wish.train) - 1, importance = TRUE, na.action = na.roughfix)  
bag.wish

##   
## Call:  
## randomForest(formula = sold\_ct\_cate ~ ., data = wish.train, mtry = length(wish.train) - 1, importance = TRUE, na.action = na.roughfix)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 24  
##   
## OOB estimate of error rate: 27.07%  
## Confusion matrix:  
## above 5K below 100 between 100 and 5k class.error  
## above 5K 329 10 77 0.2091346  
## below 100 20 419 40 0.1252610  
## between 100 and 5k 122 61 141 0.5648148

bag.pred <- predict(bag.wish, wish.test)  
  
confusion.matrix <- table(wish.test$sold\_ct\_cate, bag.pred)  
print(confusion.matrix)

## bag.pred  
## above 5K below 100 between 100 and 5k  
## above 5K 84 3 17  
## below 100 3 111 10  
## between 100 and 5k 24 12 45

accuracy.percent <- 100\*sum(diag(confusion.matrix))/sum(confusion.matrix)  
above5k.precent <- 100\*confusion.matrix[1,1]/sum(confusion.matrix[1,])  
print(paste("Test accuracy:",accuracy.percent,"%"))

## [1] "Test accuracy: 77.6699029126214 %"

print(paste("Above 5k accuracy:",above5k.precent,"%"))

## [1] "Above 5k accuracy: 80.7692307692308 %"

### Random forest

set.seed(123)  
  
rf.wish <- randomForest(sold\_ct\_cate ~ ., data = wish.train, mtry = (length(wish.train) - 1) / 3, importance = TRUE, na.action = na.roughfix)  
rf.wish

##   
## Call:  
## randomForest(formula = sold\_ct\_cate ~ ., data = wish.train, mtry = (length(wish.train) - 1)/3, importance = TRUE, na.action = na.roughfix)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 8  
##   
## OOB estimate of error rate: 27.24%  
## Confusion matrix:  
## above 5K below 100 between 100 and 5k class.error  
## above 5K 331 13 72 0.2043269  
## below 100 23 419 37 0.1252610  
## between 100 and 5k 122 65 137 0.5771605

rf.pred <- predict(rf.wish, wish.test)  
  
  
confusion.matrix <- table(wish.test$sold\_ct\_cate, rf.pred)  
print(confusion.matrix)

## rf.pred  
## above 5K below 100 between 100 and 5k  
## above 5K 83 2 19  
## below 100 4 108 12  
## between 100 and 5k 23 12 46

accuracy.percent <- 100\*sum(diag(confusion.matrix))/sum(confusion.matrix)  
above5k.precent <- 100\*confusion.matrix[1,1]/sum(confusion.matrix[1,])  
print(paste("Test accuracy:",accuracy.percent,"%"))

## [1] "Test accuracy: 76.6990291262136 %"

print(paste("Above 5k accuracy:",above5k.precent,"%"))

## [1] "Above 5k accuracy: 79.8076923076923 %"

### SVM

#### Linear

library(e1071)  
# summary(wish.train)  
# summary(wish.test)  
  
set.seed(123)  
  
tuned <- tune(svm, sold\_ct\_cate ~ ., data = wish.train, kernel = "linear", ranges = list(cost = append(seq(0.01, 10, by = 0.5), 10)))  
summary(tuned)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 8.51  
##   
## - best performance: 0.4814862   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 0.01 0.5053109 0.04242946  
## 2 0.51 0.4839385 0.04506527  
## 3 1.01 0.4839656 0.04133876  
## 4 1.51 0.4839724 0.04068346  
## 5 2.01 0.4814998 0.03790904  
## 6 2.51 0.4815066 0.03637876  
## 7 3.01 0.4814998 0.04201858  
## 8 3.51 0.4831459 0.03843672  
## 9 4.01 0.4831391 0.04081346  
## 10 4.51 0.4839588 0.04090609  
## 11 5.01 0.4847785 0.04098030  
## 12 5.51 0.4831324 0.04218639  
## 13 6.01 0.4847853 0.03975657  
## 14 6.51 0.4839520 0.04349467  
## 15 7.01 0.4831256 0.04569449  
## 16 7.51 0.4831324 0.04340741  
## 17 8.01 0.4839520 0.04434453  
## 18 8.51 0.4814862 0.04346441  
## 19 9.01 0.4823127 0.04347480  
## 20 9.51 0.4814862 0.04380657  
## 21 10.00 0.4831324 0.04357905

print("The best cost:")

## [1] "The best cost:"

tuned$best.parameter$cost

## [1] 8.51

lin.svm <- svm(sold\_ct\_cate ~ ., kernel = "linear", type = "C-class", data = wish.train, cost = tuned$best.parameter$cost)  
  
train\_pred <- predict(lin.svm, wish.train, na.action = na.exclude)  
table <- table(wish.train$sold\_ct\_cate, train\_pred)  
  
print(paste("accuracy with cost =", tuned$best.parameter$cost, "for train: "))

## [1] "accuracy with cost = 8.51 for train: "

1-(sum(table)-sum(diag(table))) / (sum(table))

## [1] 0.5348646

test\_pred <- predict(lin.svm, wish.test)  
table <- table(wish.test$sold\_ct\_cate, test\_pred)  
  
print(paste("accuracy with cost =", tuned$best.parameter$cost, "for test: "))

## [1] "accuracy with cost = 8.51 for test: "

1-(sum(table)-sum(diag(table))) / (sum(table))

## [1] 0.5048544

print(paste("above 5k group - accuracy with cost =", tuned$best.parameter$cost, ": "))

## [1] "above 5k group - accuracy with cost = 8.51 : "

sum(table[1,1]) /sum(table[1,])

## [1] 0.5384615

# we cannot plot SVM classification plot since we have more than 2 columns  
table(wish.test$sold\_ct\_cate, test\_pred)

## test\_pred  
## above 5K below 100 between 100 and 5k  
## above 5K 56 35 13  
## below 100 18 91 15  
## between 100 and 5k 26 46 9

#### Radial

# names(wish.train)  
set.seed(123)  
  
tuned <- tune(svm, sold\_ct\_cate ~ ., data = wish.train, kernel = "radial", ranges = list(cost = append(seq(0.01, 15, by = 0.5), 10)))  
summary(tuned)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 12.01  
##   
## - best performance: 0.3675112   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 0.01 0.6070519 0.04260201  
## 2 0.51 0.4298537 0.03659364  
## 3 1.01 0.4159193 0.03849578  
## 4 1.51 0.4077293 0.03480182  
## 5 2.01 0.3995258 0.03606130  
## 6 2.51 0.3970532 0.03736618  
## 7 3.01 0.3904891 0.04562588  
## 8 3.51 0.3921284 0.04731593  
## 9 4.01 0.3863907 0.04793160  
## 10 4.51 0.3822924 0.03879900  
## 11 5.01 0.3814862 0.03992683  
## 12 5.51 0.3823059 0.03877121  
## 13 6.01 0.3814795 0.03870461  
## 14 6.51 0.3781940 0.03609755  
## 15 7.01 0.3790137 0.03875255  
## 16 7.51 0.3814659 0.03912204  
## 17 8.01 0.3839249 0.03433190  
## 18 8.51 0.3822788 0.03786632  
## 19 9.01 0.3781872 0.03880695  
## 20 9.51 0.3749085 0.03738736  
## 21 10.01 0.3757282 0.03752909  
## 22 10.51 0.3699837 0.03683100  
## 23 11.01 0.3683376 0.03636024  
## 24 11.51 0.3683308 0.03789495  
## 25 12.01 0.3675112 0.03489759  
## 26 12.51 0.3691505 0.03202574  
## 27 13.01 0.3716095 0.03338185  
## 28 13.51 0.3732557 0.03371330  
## 29 14.01 0.3691573 0.03347686  
## 30 14.51 0.3675180 0.03184827  
## 31 10.00 0.3749085 0.03718716

print("The best cost:")

## [1] "The best cost:"

tuned$best.parameter$cost

## [1] 12.01

rad.svm <- svm(sold\_ct\_cate ~ ., kernel = "radial", data = wish.train, cost = tuned$best.parameter$cost)  
  
train\_pred <- predict(rad.svm, wish.train, na.action = na.exclude)  
table <- table(wish.train$sold\_ct\_cate, train\_pred)  
  
print(paste("accuracy with cost =", tuned$best.parameter$cost, "for train: "))

## [1] "accuracy with cost = 12.01 for train: "

1-(sum(table)-sum(diag(table))) / (sum(table))

## [1] 0.8630025

test\_pred <- predict(rad.svm, wish.test)  
table <- table(wish.test$sold\_ct\_cate, test\_pred)  
  
print(paste("accuracy with cost =", tuned$best.parameter$cost, "for test: "))

## [1] "accuracy with cost = 12.01 for test: "

1-(sum(table)-sum(diag(table))) / (sum(table))

## [1] 0.6343042

print(paste("above 5k group - accuracy with cost =", tuned$best.parameter$cost, ": "))

## [1] "above 5k group - accuracy with cost = 12.01 : "

sum(table[1,1]) /sum(table[1,])

## [1] 0.6730769

table(wish.test$sold\_ct\_cate, test\_pred)

## test\_pred  
## above 5K below 100 between 100 and 5k  
## above 5K 70 8 26  
## below 100 11 93 20  
## between 100 and 5k 36 12 33

The result shows that there is overfitting issue. (traing error is getting low, but test error is getting higher)

#### Polynomial

set.seed(123)  
tune.poly <- tune(svm, sold\_ct\_cate ~ ., data = wish.train, kernel = "poly", degree = 3, ranges = list(cost = append(seq(0.01, 15, by = 0.5), 10)))  
summary(tuned)

##   
## Parameter tuning of 'svm':  
##   
## - sampling method: 10-fold cross validation   
##   
## - best parameters:  
## cost  
## 12.01  
##   
## - best performance: 0.3675112   
##   
## - Detailed performance results:  
## cost error dispersion  
## 1 0.01 0.6070519 0.04260201  
## 2 0.51 0.4298537 0.03659364  
## 3 1.01 0.4159193 0.03849578  
## 4 1.51 0.4077293 0.03480182  
## 5 2.01 0.3995258 0.03606130  
## 6 2.51 0.3970532 0.03736618  
## 7 3.01 0.3904891 0.04562588  
## 8 3.51 0.3921284 0.04731593  
## 9 4.01 0.3863907 0.04793160  
## 10 4.51 0.3822924 0.03879900  
## 11 5.01 0.3814862 0.03992683  
## 12 5.51 0.3823059 0.03877121  
## 13 6.01 0.3814795 0.03870461  
## 14 6.51 0.3781940 0.03609755  
## 15 7.01 0.3790137 0.03875255  
## 16 7.51 0.3814659 0.03912204  
## 17 8.01 0.3839249 0.03433190  
## 18 8.51 0.3822788 0.03786632  
## 19 9.01 0.3781872 0.03880695  
## 20 9.51 0.3749085 0.03738736  
## 21 10.01 0.3757282 0.03752909  
## 22 10.51 0.3699837 0.03683100  
## 23 11.01 0.3683376 0.03636024  
## 24 11.51 0.3683308 0.03789495  
## 25 12.01 0.3675112 0.03489759  
## 26 12.51 0.3691505 0.03202574  
## 27 13.01 0.3716095 0.03338185  
## 28 13.51 0.3732557 0.03371330  
## 29 14.01 0.3691573 0.03347686  
## 30 14.51 0.3675180 0.03184827  
## 31 10.00 0.3749085 0.03718716

print("The best cost:")

## [1] "The best cost:"

tuned$best.parameter$cost

## [1] 12.01

poly.svm <- svm(sold\_ct\_cate ~ ., kernel = "poly", data = wish.train, degree = 3, cost = tuned$best.parameter$cost)  
  
train\_pred <- predict(poly.svm, wish.train, na.action = na.exclude)  
table <- table(wish.train$sold\_ct\_cate, train\_pred)  
  
print(paste("accuracy with cost =", tuned$best.parameter$cost, "for train: "))

## [1] "accuracy with cost = 12.01 for train: "

1-(sum(table)-sum(diag(table))) / (sum(table))

## [1] 0.733388

test\_pred <- predict(poly.svm, wish.test)  
table <- table(wish.test$sold\_ct\_cate, test\_pred)  
  
print(paste("accuracy with cost =", tuned$best.parameter$cost, "for test: "))

## [1] "accuracy with cost = 12.01 for test: "

1-(sum(table)-sum(diag(table))) / (sum(table))

## [1] 0.5566343

print(paste("above 5k group - accuracy with cost =", tuned$best.parameter$cost, ": "))

## [1] "above 5k group - accuracy with cost = 12.01 : "

sum(table[1,1]) /sum(table[1,])

## [1] 0.8461538

table(wish.test$sold\_ct\_cate, test\_pred)

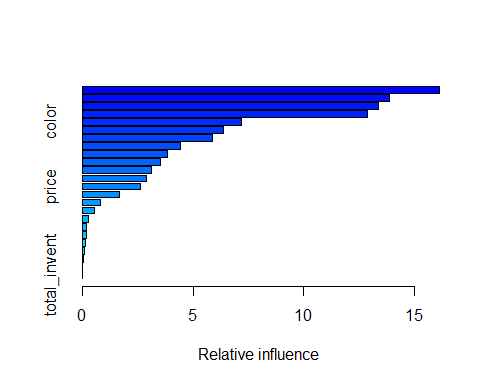
## test\_pred  
## above 5K below 100 between 100 and 5k  
## above 5K 88 5 11  
## below 100 35 74 15  
## between 100 and 5k 59 12 10

### GBM

library(gbm)

## Loaded gbm 2.1.8

boost.wish = gbm(sold\_ct\_cate ~ ., data = wish.train, distribution = "multinomial", n.trees = 10000, shrinkage = 0.01, interaction.depth = 4)  
summary(boost.wish)



## var rel.inf  
## rate2\_pct rate2\_pct 16.12485958  
## rate4\_pct rate4\_pct 13.86494973  
## rate3\_pct rate3\_pct 13.34848772  
## seller\_rate\_ct seller\_rate\_ct 12.84795272  
## color color 7.17406831  
## rate1\_pct rate1\_pct 6.38769545  
## seller\_rate seller\_rate 5.85917016  
## rate5\_pct rate5\_pct 4.41883467  
## retail retail 3.84763373  
## able\_country able\_country 3.50756573  
## rate rate 3.14135646  
## size size 2.90831988  
## price price 2.64017935  
## inventory inventory 1.67281080  
## ship\_price ship\_price 0.82420334  
## ad\_boost ad\_boost 0.53543846  
## has\_bg\_urgency has\_bg\_urgency 0.26273503  
## bg\_quality bg\_quality 0.18791776  
## badge\_ct badge\_ct 0.17240086  
## has\_seller\_propic has\_seller\_propic 0.12808411  
## origin origin 0.08833278  
## bg\_local bg\_local 0.05700338  
## bg\_fastship bg\_fastship 0.00000000  
## total\_invent total\_invent 0.00000000

boost.predP <- predict(boost.wish, wish.test, n.trees = 10000, type = 'response')  
  
classification <- c("above 5K", "below 100", "between 100 and 5k")  
boost.pred <- apply(boost.predP, 1, which.max)  
boost.pred <- classification[boost.pred]  
confusion.matrix <- table(wish.test$sold\_ct\_cate, boost.pred)  
print(confusion.matrix)

## boost.pred  
## above 5K below 100 between 100 and 5k  
## above 5K 82 3 19  
## below 100 4 108 12  
## between 100 and 5k 20 10 51

accuracy.percent <- 100\*sum(diag(confusion.matrix))/sum(confusion.matrix)  
above5k.precent <- 100\*confusion.matrix[1,1]/sum(confusion.matrix[1,])  
print(paste("Test accuracy:",accuracy.percent,"%"))

## [1] "Test accuracy: 77.9935275080906 %"

print(paste("Above 5k accuracy:",above5k.precent,"%"))

## [1] "Above 5k accuracy: 78.8461538461538 %"

### XGBoost

library(xgboost)  
# Create numeric labels with one-hot encoding  
set.seed(123)  
train\_labs <- as.numeric(wish.train$sold\_ct\_cate) - 1   
val\_labs <- as.numeric(wish.test$sold\_ct\_cate) - 1   
  
# options(na.action='na.pass')  
new\_train <- model.matrix(~ . + 0, data = subset(wish.train, select = -sold\_ct\_cate))  
new\_val <- model.matrix(~ . + 0, data = subset(wish.test, select = -sold\_ct\_cate))  
  
# Prepare matrices  
xgb\_train <- xgb.DMatrix(data = new\_train, label = train\_labs)  
xgb\_val <- xgb.DMatrix(data = new\_val, label = val\_labs)  
  
params <- list(booster = "gbtree", objective = "multi:softprob", num\_class = 4, eval\_metric = "mlogloss")  
  
# Calculate # of folds for cross-validation  
xgbcv <- xgb.cv(params = params, data = xgb\_train, nrounds = 100, nfold = 5, showsd = TRUE, stratified = TRUE, print\_every\_n = 10, early\_stop\_round = 20, maximize = FALSE, prediction = TRUE)

## [1] train-mlogloss:1.128172+0.005632 test-mlogloss:1.169920+0.009762   
## [11] train-mlogloss:0.433942+0.010224 test-mlogloss:0.684675+0.038885   
## [21] train-mlogloss:0.230444+0.008175 test-mlogloss:0.632566+0.043106   
## [31] train-mlogloss:0.140213+0.007266 test-mlogloss:0.634191+0.042982   
## [41] train-mlogloss:0.088632+0.005966 test-mlogloss:0.638861+0.046647   
## [51] train-mlogloss:0.060192+0.003827 test-mlogloss:0.654579+0.053184   
## [61] train-mlogloss:0.042890+0.002283 test-mlogloss:0.673031+0.057878   
## [71] train-mlogloss:0.032716+0.001623 test-mlogloss:0.691112+0.056010   
## [81] train-mlogloss:0.025648+0.001521 test-mlogloss:0.707691+0.058055   
## [91] train-mlogloss:0.020897+0.001250 test-mlogloss:0.723342+0.062724   
## [100] train-mlogloss:0.017756+0.001176 test-mlogloss:0.734206+0.060495

# Function to compute classification error  
classification\_error <- function(conf\_mat) {  
 conf\_mat = as.matrix(conf\_mat)  
   
 error = 1 - sum(diag(conf\_mat)) / sum(conf\_mat)  
   
 return (error)  
}  
  
# Mutate xgb output to deliver hard predictions  
xgb\_train\_preds <- data.frame(xgbcv$pred) %>% mutate(max = max.col(., ties.method = "last"), label = train\_labs + 1)  
  
# Examine output  
head(xgb\_train\_preds)

## X1 X2 X3 X4 max label  
## 1 2.557963e-01 0.394735545 3.491999e-01 2.682663e-04 2 3  
## 2 9.978531e-01 0.001962732 1.680852e-04 1.610581e-05 1 1  
## 3 8.626200e-05 0.999510407 4.009536e-04 2.427827e-06 2 2  
## 4 1.106577e-04 0.997806489 2.068996e-03 1.382771e-05 2 2  
## 5 3.321923e-02 0.934673190 3.199683e-02 1.107415e-04 2 3  
## 6 8.847255e-06 0.999940276 5.025148e-05 6.239600e-07 2 2

xgb\_conf\_mat <- table(true = train\_labs + 1, pred = xgb\_train\_preds$max)  
  
# Error   
cat("XGB Training Classification Error Rate:", classification\_error(xgb\_conf\_mat), "\n")

## XGB Training Classification Error Rate: 0.2657916

# predicting / testing on test dataset  
xgb\_model <- xgb.train(params = params, data = xgb\_train, nrounds = 100)  
  
# Predict for validation set  
xgb\_val\_preds <- predict(xgb\_model, newdata = xgb\_val)  
  
xgb\_val\_out <- matrix(xgb\_val\_preds, nrow = 4, ncol = length(xgb\_val\_preds) / 4) %>%   
 t() %>%  
 data.frame() %>%  
 mutate(max = max.col(., ties.method = "last"), label = val\_labs + 1)   
  
# Confustion Matrix  
xgb\_val\_conf <- table(true = val\_labs + 1, pred = xgb\_val\_out$max)  
  
cat("XGB Validation Classification Error Rate:", 1-classification\_error(xgb\_val\_conf), "\n")

## XGB Validation Classification Error Rate: 0.7508091

cat("XGB Validation Classification Error Rate - above 5k:", xgb\_val\_conf[1,1]/sum(xgb\_val\_conf[1,]), "\n")

## XGB Validation Classification Error Rate - above 5k: 0.8076923

### Stacked Ensembles

# we already have gbm.wish for GBM, now build RF model  
# https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/stacked-ensembles.html  
# Stacked Ensemble model's performance is not so different from those of base learners'   
library(h2o)

##   
## ----------------------------------------------------------------------  
##   
## Your next step is to start H2O:  
## > h2o.init()  
##   
## For H2O package documentation, ask for help:  
## > ??h2o  
##   
## After starting H2O, you can use the Web UI at http://localhost:54321  
## For more information visit https://docs.h2o.ai  
##   
## ----------------------------------------------------------------------

##   
## Attaching package: 'h2o'

## The following objects are masked from 'package:stats':  
##   
## cor, sd, var

## The following objects are masked from 'package:base':  
##   
## %\*%, %in%, &&, ||, apply, as.factor, as.numeric, colnames,  
## colnames<-, ifelse, is.character, is.factor, is.numeric, log,  
## log10, log1p, log2, round, signif, trunc

h2o.init()

## Connection successful!  
##   
## R is connected to the H2O cluster:   
## H2O cluster uptime: 14 minutes 33 seconds   
## H2O cluster timezone: America/New\_York   
## H2O data parsing timezone: UTC   
## H2O cluster version: 3.32.0.1   
## H2O cluster version age: 2 months and 7 days   
## H2O cluster name: H2O\_started\_from\_R\_maxmo\_plv200   
## H2O cluster total nodes: 1   
## H2O cluster total memory: 7.00 GB   
## H2O cluster total cores: 12   
## H2O cluster allowed cores: 12   
## H2O cluster healthy: TRUE   
## H2O Connection ip: localhost   
## H2O Connection port: 54321   
## H2O Connection proxy: NA   
## H2O Internal Security: FALSE   
## H2O API Extensions: Amazon S3, Algos, AutoML, Core V3, TargetEncoder, Core V4   
## R Version: R version 3.6.2 (2019-12-12)

wish.train.h2o <- as.h2o(wish.train)

## | | | 0% | |======================================================================| 100%

wish.test.h2o <- as.h2o(wish.test)

## | | | 0% | |======================================================================| 100%

predictors <- c(colnames(wish.train)[1:length(wish.train) - 1])  
response <- "sold\_ct\_cate"  
  
set.seed(123)  
  
gbm.wish <- h2o.gbm(x = predictors,  
 y = response,  
 nfolds = 5,  
 distribution = "multinomial",  
 keep\_cross\_validation\_predictions = TRUE,  
 training\_frame = wish.train.h2o, seed=1)

## | | | 0% | |=============== | 22% | |======================================================================| 100%

rf.wish <- h2o.randomForest(x = predictors,  
 y = response,  
 training\_frame = wish.train.h2o,  
 ntrees = 50,  
 nfolds = 5,  
 keep\_cross\_validation\_predictions = TRUE,  
 seed = 1)

## | | | 0% | |========== | 14% | |============================================================ | 85% | |======================================================================| 100%

ensemble <- h2o.stackedEnsemble(x = predictors,  
 y = response,  
 training\_frame = wish.train.h2o,  
 base\_models = list(gbm.wish, rf.wish))

## | | | 0% | |======================================================================| 100%

perf <- h2o.performance(ensemble, newdata = wish.test.h2o)  
  
# Compare to base learner performance on the test set  
perf\_gbm\_test <- h2o.performance(gbm.wish, newdata = wish.test.h2o)  
perf\_rf\_test <- h2o.performance(rf.wish, newdata = wish.test.h2o)  
baselearner\_best\_auc\_test <- max(h2o.auc(perf\_gbm\_test), h2o.auc(perf\_rf\_test))  
ensemble\_auc\_test <- h2o.auc(perf)  
print(sprintf("Best Base-learner Test AUC: %s", baselearner\_best\_auc\_test))

## [1] "Best Base-learner Test AUC: -Inf"

print(sprintf("Ensemble Test AUC: %s", ensemble\_auc\_test))

## character(0)

perf

## H2OMultinomialMetrics: stackedensemble  
##   
## Test Set Metrics:   
## =====================  
##   
## MSE: (Extract with `h2o.mse`) 0.1869437  
## RMSE: (Extract with `h2o.rmse`) 0.4323699  
## Logloss: (Extract with `h2o.logloss`) 0.5548341  
## Mean Per-Class Error: 0.2670902  
## Null Deviance: (Extract with `h2o.nulldeviance`) 669.9311  
## Residual Deviance: (Extract with `h2o.residual\_deviance`) 342.8875  
## AIC: (Extract with `h2o.aic`) NaN  
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)  
## =========================================================================  
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class  
## above 5K below 100 between 100 and 5k Error Rate  
## above 5K 85 2 17 0.1827 = 19 / 104  
## below 100 4 107 13 0.1371 = 17 / 124  
## between 100 and 5k 29 10 42 0.4815 = 39 / 81  
## Totals 118 119 72 0.2427 = 75 / 309  
##   
## Hit Ratio Table: Extract with `h2o.hit\_ratio\_table(<model>, <data>)`  
## =======================================================================  
## Top-3 Hit Ratios:   
## k hit\_ratio  
## 1 1 0.757282  
## 2 2 0.961165  
## 3 3 1.000000

perf\_gbm\_test

## H2OMultinomialMetrics: gbm  
##   
## Test Set Metrics:   
## =====================  
##   
## MSE: (Extract with `h2o.mse`) 0.2065348  
## RMSE: (Extract with `h2o.rmse`) 0.454461  
## Logloss: (Extract with `h2o.logloss`) 0.6037602  
## Mean Per-Class Error: 0.2878935  
## R^2: (Extract with `h2o.r2`) 0.6518089  
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)  
## =========================================================================  
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class  
## above 5K below 100 between 100 and 5k Error Rate  
## above 5K 87 2 15 0.1635 = 17 / 104  
## below 100 3 103 18 0.1694 = 21 / 124  
## between 100 and 5k 30 13 38 0.5309 = 43 / 81  
## Totals 120 118 71 0.2621 = 81 / 309  
##   
## Hit Ratio Table: Extract with `h2o.hit\_ratio\_table(<model>, <data>)`  
## =======================================================================  
## Top-3 Hit Ratios:   
## k hit\_ratio  
## 1 1 0.737864  
## 2 2 0.948220  
## 3 3 1.000000

perf\_rf\_test

## H2OMultinomialMetrics: drf  
##   
## Test Set Metrics:   
## =====================  
##   
## MSE: (Extract with `h2o.mse`) 0.2127525  
## RMSE: (Extract with `h2o.rmse`) 0.461251  
## Logloss: (Extract with `h2o.logloss`) 0.6123574  
## Mean Per-Class Error: 0.3100205  
## R^2: (Extract with `h2o.r2`) 0.6413268  
## Confusion Matrix: Extract with `h2o.confusionMatrix(<model>, <data>)`)  
## =========================================================================  
## Confusion Matrix: Row labels: Actual class; Column labels: Predicted class  
## above 5K below 100 between 100 and 5k Error Rate  
## above 5K 84 4 16 0.1923 = 20 / 104  
## below 100 4 106 14 0.1452 = 18 / 124  
## between 100 and 5k 33 15 33 0.5926 = 48 / 81  
## Totals 121 125 63 0.2783 = 86 / 309  
##   
## Hit Ratio Table: Extract with `h2o.hit\_ratio\_table(<model>, <data>)`  
## =======================================================================  
## Top-3 Hit Ratios:   
## k hit\_ratio  
## 1 1 0.721683  
## 2 2 0.948220  
## 3 3 1.000000

# Generate predictions on a test set   
pred <- h2o.predict(ensemble, newdata = wish.test.h2o)

## | | | 0% | |======================================================================| 100%

### Neural network

library(neuralnet)

##   
## Attaching package: 'neuralnet'

## The following object is masked from 'package:dplyr':  
##   
## compute

library(nnet)  
y\_index = grep("sold\_ct\_cate", names(wish.train))  
nn.train = cbind(wish.train[, -y\_index], wish.train[, y\_index])  
names(nn.train)[names(nn.train) == 'wish.train[, y\_index]'] <- 'sold\_ct\_cate'  
  
  
nn.train = cbind(nn.train[, 1:8],class.ind(as.factor(nn.train$color)),nn.train[, 11:15],nn.train[17:24],class.ind(as.factor(nn.train$size)),class.ind(as.factor(nn.train$origin)), class.ind(as.factor(nn.train$sold\_ct\_cate)))  
#normalized by scaling data  
scl <- function(x){ (x - min(x))/(max(x) - min(x)) }  
nn.train[, 1:29] <- data.frame(lapply(nn.train[, 1:29], scl))  
  
names(nn.train)[names(nn.train) == 'One-sized'] <- 'one\_sized'  
names(nn.train)[names(nn.train) == 'above 5K'] <- 'above\_5k'  
names(nn.train)[names(nn.train) == 'below 100'] <- 'below\_100'  
names(nn.train)[names(nn.train) == 'between 100 and 5k'] <- 'btw\_100\_5k'  
  
name <- names(nn.train)  
f <- as.formula(paste("above\_5k + below\_100 + btw\_100\_5k ~", paste(name[!name %in% c("above\_5k","below\_100","btw\_100\_5k")], collapse = " + ")))  
f

## above\_5k + below\_100 + btw\_100\_5k ~ price + retail + ad\_boost +   
## rate + badge\_ct + bg\_local + bg\_quality + bg\_fastship + black +   
## blue + green + others + pink + red + white + yellow + inventory +   
## ship\_price + able\_country + total\_invent + has\_bg\_urgency +   
## seller\_rate\_ct + seller\_rate + has\_seller\_propic + rate5\_pct +   
## rate4\_pct + rate3\_pct + rate2\_pct + rate1\_pct + l + m + one\_sized +   
## s + xl + xs + CN + others

set.seed(123)  
nn <- neuralnet(f,  
 data = nn.train,  
 hidden = c(37, 15,3),  
 act.fct = "logistic",  
 linear.output = FALSE,stepmax = 3000)

plot(nn)

nn.test = cbind(wish.test[, -y\_index], wish.test[, y\_index])  
names(nn.test)[names(nn.test) == 'wish.test[, y\_index]'] <- 'sold\_ct\_cate'  
  
nn.test = cbind(nn.test[, 1:8], class.ind(as.factor(nn.test$color)), nn.test[, 11:15], nn.test[17:24], class.ind(as.factor(nn.test$size)), class.ind(as.factor(nn.test$origin)), class.ind(as.factor(nn.test$sold\_ct\_cate)))  
#normalized by scaling data  
scl <- function(x){ (x - min(x))/(max(x) - min(x)) }  
nn.test[, 1:29] <- data.frame(lapply(nn.test[, 1:29], scl))  
  
names(nn.test)[names(nn.test) == 'One-sized'] <- 'one\_sized'  
names(nn.test)[names(nn.test) == 'above 5K'] <- 'above\_5k'  
names(nn.test)[names(nn.test) == 'below 100'] <- 'below\_100'  
names(nn.test)[names(nn.test) == 'between 100 and 5k'] <- 'btw\_100\_5k'  
  
#train accuracy  
nn.train\_pred <- compute(nn, nn.train[, 1:37])  
nn.train\_pred <- nn.train\_pred$net.result  
true\_y.train <- max.col(nn.train[, 38:40])  
predicted\_y.train <- max.col(nn.train\_pred)  
  
#test accuracy  
nn.test\_pred <- compute(nn, nn.test[, 1:37])  
nn.test\_pred <- nn.test\_pred$net.result  
true\_y.test <- max.col(nn.test[, 38:40])  
predicted\_y.test <- max.col(nn.test\_pred)  
  
table.train <- table(true\_y.train, predicted\_y.train)  
table.test <- table(true\_y.test, predicted\_y.test)  
  
  
print("accuracy for train: ")

## [1] "accuracy for train: "

print(1-(sum(table.train)-sum(diag(table.train))) / (sum(table.train)))

## [1] 0.9589828

print("accuracy for test: ")

## [1] "accuracy for test: "

print(1-(sum(table.test)-sum(diag(table.test))) / (sum(table.test)))

## [1] 0.5210356

print("above 5k group accuracy:")

## [1] "above 5k group accuracy:"

sum(table.test[1,1]) /sum(table.test[1,])

## [1] 0.6153846