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', 'pr
oduct_url', 'merchant_id', 'merchant_profile_picture', 'merchant_info_subtitle', 'mercha
nt_name', 'merchant_title', 'urgency_text', 'title_orig', 'shipping_option_name', 'curre
ncy_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', pro
duct_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|c
amel',
                                          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)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
#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))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

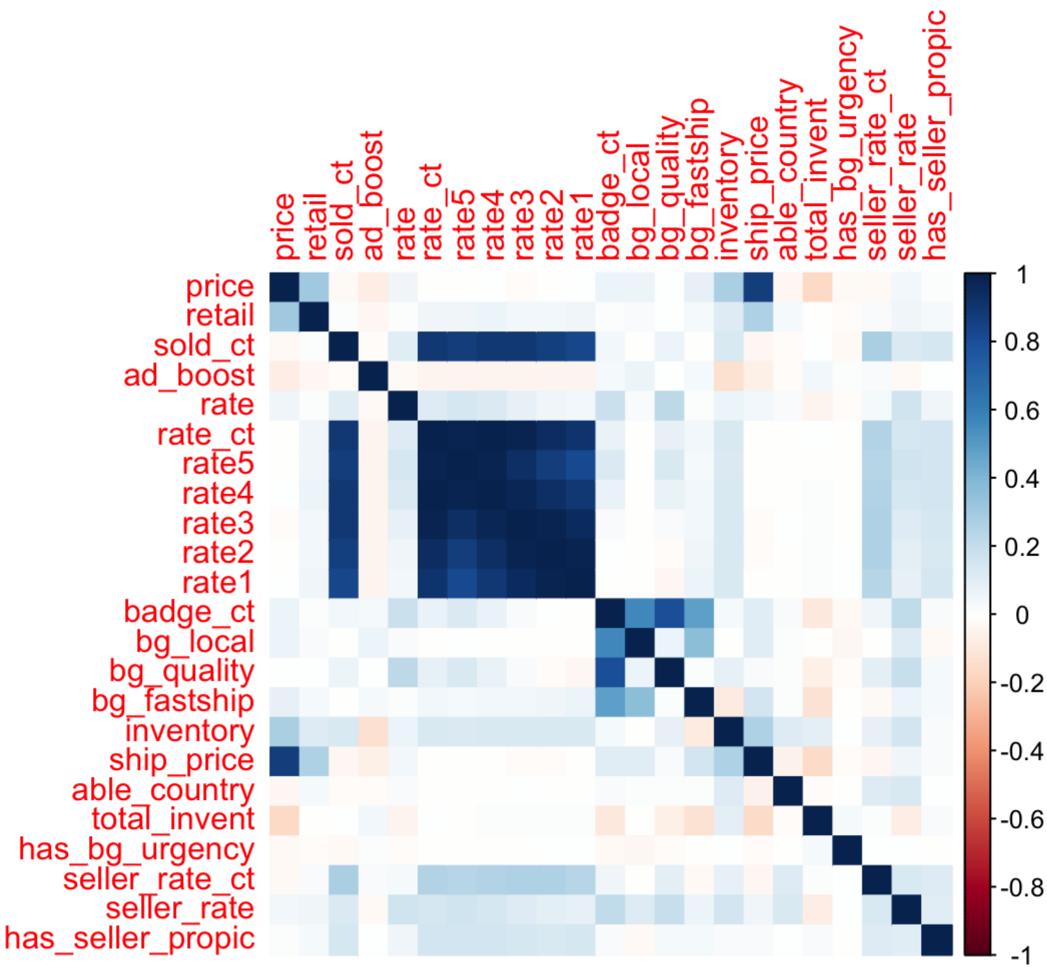
```
#convert column name to short version
origin_colname <- colnames(wish)
colnames(wish) <- c('price', 'retail', 'sold_ct', 'ad_boost', 'rate', 'rate_ct', 'rate5'
, 'rate4', 'rate3', 'rate2', 'rate1', 'badge_ct', 'bg_local', 'bg_quality', 'bg_fastshi
p', 'color', 'size', 'inventory', 'ship_price', 'able_country', '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')
```



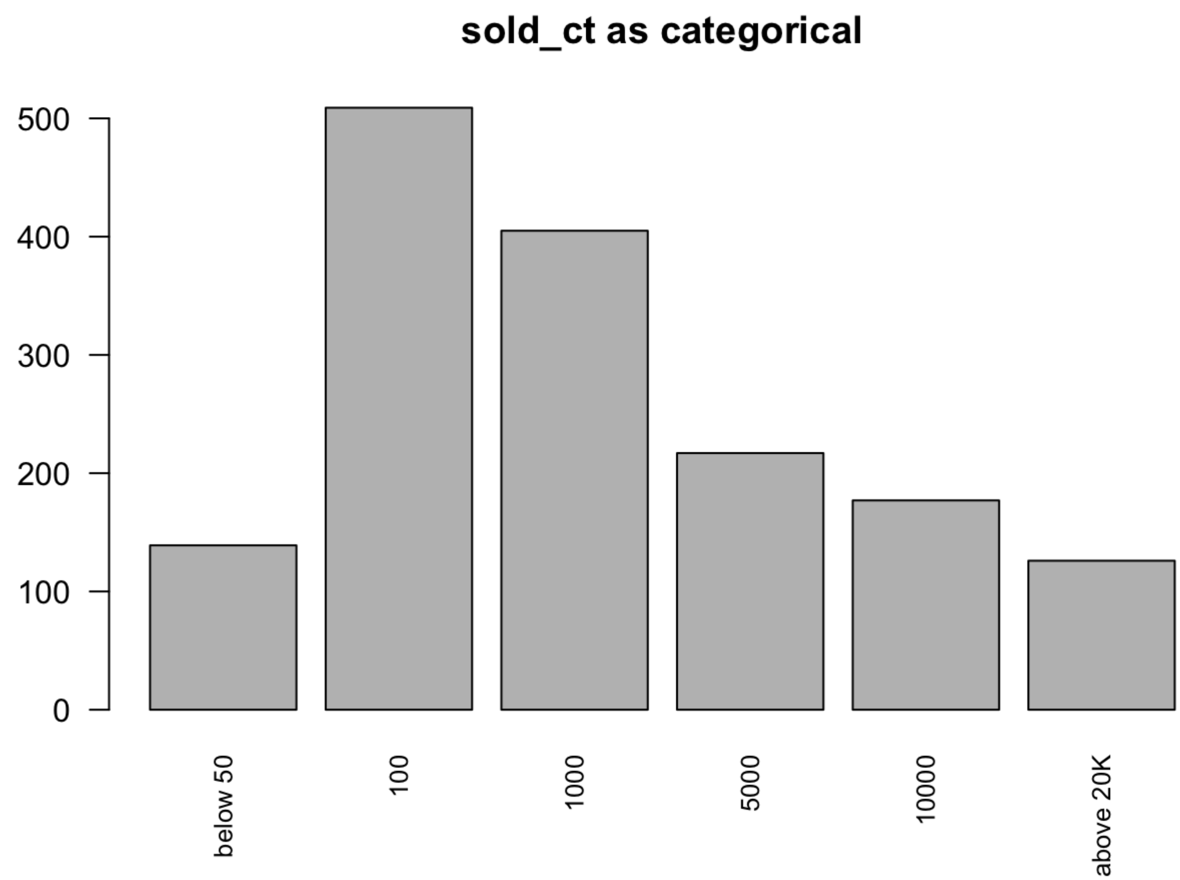

```
#convert the y (sold_ct) to categorical. Also since it is unbalanced we group some categ
ory together.
table(wish['sold_ct']) # very unbalaned
```

```
##
##      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 <= 50)] <- 'below 50'
wish_cate$sold_ct_cate[which(wish_cate$sold_ct >= 20000)] <- 'above 20K'
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
```

```
##
##       100      1000     10000      5000 above 20K  below 50
##       509       405       177       217       126       139
```

```
x1 <- factor(wish_cate$sold_ct_cate, levels = c("below 50", "100", "1000", "5000", "1000
0", "above 20K"))
tb <- table(x1)
barplot(tb, names.arg = row.names(tb), cex.names = 0.8, main = "sold_ct as categorical",
las = 2)
```




```
str(wish_cate)
```

```
## 'data.frame':    1573 obs. of  26 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 ...
##  $ rate_ct          : int  54 6135 14 579 20 1 6742 286 15 687 ...
##  $ rate5            : num  26 2269 5 295 6 ...
##  $ rate4            : num  8 1027 4 119 4 ...
##  $ rate3            : num  10 1118 2 87 2 ...
##  $ rate2            : num  1 644 0 42 2 0 490 18 1 68 ...
##  $ rate1            : num  9 1077 3 36 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/ 6 levels "100","1000","10000",..: 1 5 1 4 1 6 5 2 1 4
...
```

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, ]
```

Tree Models

```
library(tree)

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] "rate_ct" "rate3"
## Number of terminal nodes:  8
## Residual mean deviance:  1.188 = 1485 / 1250
## Misclassification error rate: 0.2647 = 333 / 1258
```

```
tree.pred <- predict(tree.wish, wish.test, type = "class")

table(tree.pred, wish.test$sold_ct_cate)
```

```
##
## tree.pred   100 1000 10000 5000 above 20K below 50
##   100        72    9     0    0         0        2
##   1000       14   56     0    4         0        0
##   10000       0    0    24   14        10        0
##   5000        0   16     5   24         0        0
##   above 20K   0    0     0    0        16        0
##   below 50   21    1     0    0         0       27
```

```
print("Misclassification error rate on test set: ")
```

```
## [1] "Misclassification error rate on test set: "
```

```
1 - ((table(tree.pred, wish.test$sold_ct_cate)[1] + table(tree.pred, wish.test$sold_ct_c
ate)[8] + table(tree.pred, wish.test$sold_ct_cate)[15] + table(tree.pred, wish.test$sold
_ct_cate)[22] + table(tree.pred, wish.test$sold_ct_cate)[29] + table(tree.pred, wish.tes
t$sold_ct_cate)[36]) / nrow(wish.test))
```

```
## [1] 0.3047619
```

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 = 25, importance = TR
UE)
bag.wish
```

```
##
## Call:
##  randomForest(formula = sold_ct_cate ~ ., data = wish.train, mtry = 25,      importan
ce = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 25
##
##         OOB estimate of  error rate: 21.54%
## Confusion matrix:
##           100 1000 10000 5000 above 20K below 50 class.error
## 100       359   20     0    0         0       23   0.1069652
## 1000       39  253     0   31         0        0   0.2167183
## 10000       0    4    97   34        13        0   0.3445946
## 5000        1   37    23  114         0        0   0.3485714
## above 20K   0    0    24    1        75        0   0.2500000
## below 50   21    0     0    0         0       89   0.1909091
```

```
bag.pred <- predict(bag.wish, wish.test)

table(bag.pred, wish.test$sold_ct_cate)
```

```
##
## bag.pred    100 1000 10000 5000 above 20K below 50
##   100        97    8     0    0         0        4
##   1000        3   68     0    7         0        0
##   10000       0    0    22   11         4        0
##   5000        0    6     6   24         2        0
##   above 20K   0    0     1    0        20        0
##   below 50    7    0     0    0         0       25
```

```
print("Misclassification error rate on test set: ")
```

```
## [1] "Misclassification error rate on test set: "
```

```
1 - ((table(bag.pred, wish.test$sold_ct_cate)[1] + table(bag.pred, wish.test$sold_ct_cat
e)[8] + table(bag.pred, wish.test$sold_ct_cate)[15] + table(bag.pred, wish.test$sold_ct_
cate)[22] + table(bag.pred, wish.test$sold_ct_cate)[29] + table(bag.pred, wish.test$sold
_ct_cate)[36]) / nrow(wish.test))
```

```
## [1] 0.1873016
```

Random forest

```
set.seed(123)

rf.wish <- randomForest(sold_ct_cate ~ ., data = wish.train, mtry = 25 / 3, importance =
TRUE)
rf.wish
```

```
##
## Call:
##  randomForest(formula = sold_ct_cate ~ ., data = wish.train, mtry = 25/3,      import
ance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 21.14%
## Confusion matrix:
##           100 1000 10000 5000 above 20K below 50 class.error
## 100       361   20     0    0         0       21   0.1019900
## 1000       37  259     0   27         0        0   0.1981424
## 10000       0    4   102   35         7        0   0.3108108
## 5000        1   38    23  113         0        0   0.3542857
## above 20K   0    0    26    1        73        0   0.2700000
## below 50   26    0     0    0         0       84   0.2363636
```

```
rf.pred <- predict(rf.wish, wish.test)

table(rf.pred, wish.test$sold_ct_cate)
```

```
##
## rf.pred     100 1000 10000 5000 above 20K below 50
##   100        98    9     0    0         0        3
##   1000        3   65     0    7         0        0
##   10000       0    0    22   10         5        0
##   5000        0    8     6   25         2        0
##   above 20K   0    0     1    0        19        0
##   below 50    6    0     0    0         0       26
```

```
print("Misclassification error rate on test set: ")
```

```
## [1] "Misclassification error rate on test set: "
```

```
1 - ((table(rf.pred, wish.test$sold_ct_cate)[1] + table(rf.pred, wish.test$sold_ct_cate)
[8] + table(rf.pred, wish.test$sold_ct_cate)[15] + table(rf.pred, wish.test$sold_ct_cat
e)[22] + table(rf.pred, wish.test$sold_ct_cate)[29] + table(rf.pred, wish.test$sold_ct_c
ate)[36]) / nrow(wish.test))
```

```
## [1] 0.1904762
```

GBM

```
set.seed(123)

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:         18 minutes 29 seconds
##     H2O cluster timezone:       America/New_York
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.32.0.2
##     H2O cluster version age:    17 days
##     H2O cluster name:           H2O_started_from_R_RZhe_osn291
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.96 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     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, XGBoost, Algos, AutoML, Core V3, TargetEnc
oder, Core V4
##     R Version:                  R version 4.0.3 (2020-10-10)
```

```
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"

# Build and train the model:
gbm.wish <- h2o.gbm(x = predictors,
                    y = response,
                    nfolds = 5,
                    distribution = "multinomial",
                    keep_cross_validation_predictions = TRUE,
                    training_frame = wish.train.h2o)
```

```
##
  |
  |                                                                      |   0%
  |
  |====================                                                  |  28%
  |
  |==============================================                        |  66%
  |
  |=================================================================     |  93%
  |
  |======================================================================| 100%
```

```
h2o.confusionMatrix(gbm.wish)
```

	1...	1000	10000	5000	above 20K	below 50	Error	Rate
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
100	401	1	0	0	0	0	0.002487562	1 / 402
1000	2	321	0	0	0	0	0.006191950	2 / 323
10000	0	0	148	0	0	0	0.000000000	0 / 148
5000	1	0	0	174	0	0	0.005714286	1 / 175
above 20K	0	0	0	0	100	0	0.000000000	0 / 100
below 50	0	0	0	0	0	110	0.000000000	0 / 110
Totals	404	322	148	174	100	110	0.003179650	4 / 1,258

7 rows

```
print("Misclassification error rate on training set: ")
```

```
## [1] "Misclassification error rate on training set: "
```

```
h2o.confusionMatrix(gbm.wish)['Totals','Error']
```

```
## [1] 0.00317965
```

```
h2o.confusionMatrix(gbm.wish, wish.test.h2o)
```

	100	1000	10000	5000	above 20K	below 50	Error	Rate
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
100	99	4	0	0	0	4	0.07476636	8 / 107
1000	8	69	0	5	0	0	0.15853659	13 / 82
10000	0	1	21	6	1	0	0.27586207	8 / 29

	100 <dbl>	1000 <dbl>	10000 <dbl>	5000 <dbl>	above 20K <dbl>	below 50 <dbl>	Error <dbl>	Rate <chr>
5000	0	7	11	24	0	0	0.42857143	18 / 42
above 20K	0	0	3	2	21	0	0.19230769	5 / 26
below 50	4	0	0	0	0	25	0.13793103	4 / 29
Totals	111	81	35	37	22	29	0.17777778	56 / 315

7 rows

```
print("Misclassification error rate on test set: ")
```

```
## [1] "Misclassification error rate on test set: "
```

```
h2o.confusionMatrix(gbm.wish)['Totals','Error']
```

```
## [1] 0.00317965
```

Multinomial regression

```
set.seed(123)
library(nnet)
multinomial.mod <- multinom(sold_ct_cate ~ ., data = wish.train)
```

```
## # weights:  222 (180 variable)
## initial  value 2254.033412
## iter  10 value 2082.917563
## iter  20 value 1706.950647
## iter  30 value 1674.506550
## iter  40 value 1636.530083
## iter  50 value 1602.333617
## iter  60 value 1516.842452
## iter  70 value 1284.733458
## iter  80 value 1092.324716
## iter  90 value 923.684260
## iter 100 value 830.168190
## final  value 830.168190
## stopped after 100 iterations
```

```
summary(multinomial.mod)
```

```
## Call:
## multinom(formula = sold_ct_cate ~ ., data = wish.train)
##
## Coefficients:
##           (Intercept)      price       retail   ad_boost       rate
## 1000      -27.6647102 -0.1610435  0.002018544  0.3988605  0.7506249
## 10000     -11.9468907 -0.2900361 -0.022770741  1.7606111  3.0288554
## 5000      -24.8856809 -0.2077328 -0.009378203  0.6326620  1.4756729
## above 20K   0.7890865 -0.3098477 -0.022605468  3.8576692  5.2712954
## below 50   18.1019566 -0.1547209  0.001828351 -1.4127596 -0.8476303
##                rate_ct      rate5        rate4        rate3        rate2
## 1000       0.004748629 0.03194677 -0.04742378 -0.07185554 -0.07205429
## 10000      0.014590877 0.03978816 -0.06638209 -0.09980726 -0.04970919
## 5000       0.009595458 0.03834941 -0.06619161 -0.07099634 -0.07553097
## above 20K  0.015784076 0.04506234 -0.07777596 -0.08828677 -0.07701518
## below 50  -0.004029510 0.04646389 -0.09902672  0.05707937  0.04174828
##                 rate1     badge_ct   bg_local bg_quality bg_fastship
## 1000       0.16413548  -0.06502733  1.3211531 -0.5043531  -0.8818273
## 10000      0.19070129  -1.00577928 -0.1433377 -1.6827942   0.8203526
## 5000       0.18396497  -5.20845185  4.8693592  3.9710469 -14.0488580
## above 20K  0.21379963  -4.80248235  6.2177891  1.3401197 -12.3603912
## below 50  -0.05029433 -10.70071869  2.8934426 -5.1726132  -8.4215481
##               colorblue  colorgreen colorothers   colorpink   colorred
## 1000       0.0008876128 -0.01082212   0.1064402  0.01712693  0.1365742
## 10000      0.6105289230 -0.27892996   1.0371048  1.05584888  0.4907939
## 5000      -0.0191599963 -0.29749197   0.6052281  0.53095451  0.5448338
## above 20K -0.1781153619  0.83929410   1.6836057 -0.58358197  0.4767851
## below 50   0.6891023704 -0.38387724   0.5406745  0.40275676 -2.2474095
##           colorwhite coloryellow     sizem sizeOne-sized       sizes
## 1000      0.01207121   0.4209092 -1.818938   -2.35523300 -1.38838690
## 10000     0.90907634   0.4073694 -1.746427   -2.77908551 -2.59325262
## 5000      0.33478497  -0.1998063 -1.264702   -2.91614100 -1.51007935
## above 20K 0.86887934  -5.1504027 -6.269321  -13.16865877 -7.42436381
## below 50  0.38532870   0.2081955 -2.366422    0.06170327 -0.08700506
##                sizexl    sizexs    inventory  ship_price able_country
## 1000       -3.6073828 -1.989208  0.003325556  0.31980087  0.003758126
## 10000      -8.3790711 -3.497371  0.037501870 -0.30810262 -0.015669301
## 5000       -5.6711373 -2.047250  0.010051262  0.04897388  0.002977891
## above 20K -15.7517324 -9.153472  0.058131504 -0.92501674 -0.066597282
## below 50   -0.8681239 -0.138710 -0.018169788  0.90231704  0.009546501
##           total_invent has_bg_urgency originothers seller_rate_ct seller_rate
## 1000         0.4878338     0.07807921   -0.7294529  -1.044367e-05  0.09378708
## 10000        0.1656005     0.30279710   -0.1592076  -2.212731e-06 -3.65715589
## 5000         0.3236319     0.14815549   -0.2248420  -4.047016e-06 -0.01229375
## above 20K   -0.4083287     1.45900421    2.0559379  -1.497055e-06 -3.12655195
## below 50    -0.2486529     0.28903506    0.6927080   2.689096e-07 -1.16089905
##           has_seller_propic
## 1000              0.8757460
## 10000             1.3665213
## 5000              0.9772515
## above 20K         1.9429725
## below 50         -0.5757087
##
```

```
## Std. Errors:
##            (Intercept)        price      retail     ad_boost         rate
## 1000      8.134035e-05 0.0004630351 0.003079637 7.033372e-05 0.0003084177
## 10000     5.853166e-05 0.0004504221 0.003206519 5.031983e-05 0.0002306347
## 5000      5.338372e-05 0.0004182991 0.003924067 4.130580e-05 0.0002076405
## above 20K 2.362325e-05 0.0002439419 0.003402751 2.290133e-05 0.0000947457
## below 50  4.869266e-05 0.0003362180 0.003689204 2.824574e-05 0.0001944642
##                rate_ct       rate5       rate4       rate3        rate2
## 1000      0.0009252869 0.002610177 0.004133245 0.001708015 0.0005496741
## 10000     0.0008627750 0.001648768 0.003286139 0.003248780 0.0012971244
## 5000      0.0008549785 0.001756894 0.002654546 0.002254971 0.0007089916
## above 20K 0.0008450768 0.001812987 0.002753778 0.001941288 0.0011125132
## below 50  0.0016010876 0.004631441 0.001505957 0.002316585 0.0010399621
##                  rate1     badge_ct     bg_local   bg_quality  bg_fastship
## 1000      0.003483198 1.941985e-05 6.677781e-06 1.484766e-05 3.390835e-06
## 10000     0.002385706 8.272079e-06 5.608737e-06 9.516577e-06 2.047785e-08
## 5000      0.002147667 1.040146e-05 6.394490e-06 1.048542e-05 1.077099e-14
## above 20K 0.002645374 9.113099e-06 3.974735e-06 6.492284e-06 6.074309e-09
## below 50  0.001217330 3.793831e-09 2.189973e-09 4.467612e-09 1.867055e-14
##              colorblue   colorgreen  colorothers    colorpink     colorred
## 1000      1.138153e-05 1.504265e-05 2.363437e-05 1.540724e-05 1.851938e-05
## 10000     1.060125e-05 1.067117e-05 1.554791e-05 9.664728e-06 7.590704e-06
## 5000      1.071562e-05 1.251115e-05 1.364521e-05 1.558000e-05 1.431976e-05
## above 20K 3.424368e-06 4.608186e-06 6.248537e-06 2.124927e-06 8.240810e-06
## below 50  8.061175e-06 1.167466e-05 9.085852e-06 6.825160e-06 2.546989e-06
##             colorwhite  coloryellow        sizem sizeOne-sized        sizes
## 1000      2.107494e-05 1.864294e-05 1.791062e-05  1.198041e-05 2.604003e-05
## 10000     1.588328e-05 2.944319e-06 1.970989e-05  7.637517e-06 2.866046e-05
## 5000      1.412096e-05 4.314154e-06 1.470741e-05  5.031051e-06 2.695692e-05
## above 20K 6.527963e-06 3.544504e-06 4.864702e-06  6.212806e-06 1.473113e-05
## below 50  1.920308e-05 1.517617e-05 6.314416e-06  5.292974e-06 4.666599e-05
##                 sizexl       sizexs   inventory   ship_price able_country
## 1000      3.147489e-06 8.973705e-05 0.004078485 1.384518e-04  0.003940297
## 10000     4.753395e-06 1.115115e-05 0.003930140 1.241897e-04  0.003835487
## 5000      3.428585e-06 3.530876e-05 0.004712102 1.172323e-04  0.004038155
## above 20K 2.131742e-06 4.952342e-06 0.001498382 6.705321e-05  0.001101779
## below 50  7.117187e-06 8.330519e-05 0.005532497 9.999560e-05  0.002991580
##           total_invent has_bg_urgency originothers seller_rate_ct  seller_rate
## 1000       0.004070173   2.521215e-05 2.361273e-06   3.691981e-06 3.176325e-04
## 10000      0.002926579   2.834100e-05 4.195225e-06   5.545282e-06 2.348904e-04
## 5000       0.002669182   1.729939e-05 2.905866e-06   5.369303e-06 2.143025e-04
## above 20K  0.001181160   1.892074e-05 3.027637e-06   6.094110e-06 9.592144e-05
## below 50   0.002431574   2.174217e-05 7.180488e-06   4.048173e-06 1.894113e-04
##           has_seller_propic
## 1000           1.375699e-05
## 10000          1.185569e-05
## 5000           1.338053e-05
## above 20K      7.826075e-06
## below 50       2.445400e-06
##
## Residual Deviance: 1660.336
## AIC: 2000.336
```

```r
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:"
```

```r
mean(as.character(multinomial.pred_train) != as.character(wish.train$sold_ct_cate))
```

```
## [1] 0.2615262
```

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

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

```r
mean(as.character(multinomial.pred_test) != as.character(wish.test$sold_ct_cate))
```

```
## [1] 0.3301587
```

SVM

```r
library(e1071)

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) # cost = 3.51 is the best
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  4.51
##
## - best performance: 0.2583746
##
## - Detailed performance results:
##     cost     error dispersion
## 1   0.01 0.4889333 0.02599852
## 2   0.51 0.2798540 0.04567908
## 3   1.01 0.2711175 0.04075831
## 4   1.51 0.2679429 0.04103799
## 5   2.01 0.2591746 0.04324969
## 6   2.51 0.2663238 0.04205503
## 7   3.01 0.2663238 0.04222112
## 8   3.51 0.2623429 0.04039357
## 9   4.01 0.2607556 0.03838902
## 10  4.51 0.2583746 0.04214592
## 11  5.01 0.2583810 0.04267620
## 12  5.51 0.2591873 0.04483738
## 13  6.01 0.2583873 0.04157809
## 14  6.51 0.2615683 0.03918624
## 15  7.01 0.2599810 0.03859511
## 16  7.51 0.2607810 0.04191514
## 17  8.01 0.2599873 0.04279232
## 18  8.51 0.2591937 0.04249829
## 19  9.01 0.2599873 0.04146328
## 20  9.51 0.2599937 0.04203670
## 21 10.00 0.2592000 0.04273174
```

```
lin.svm <- svm(sold_ct_cate ~ ., kernel = "linear", type = "C-class", data = wish.train,
cost = 3.51)

train_pred <- predict(lin.svm, wish.train)
table <- table(wish.train$sold_ct_cate, train_pred)

print("training error with cost = 3.51: ")
```

```
## [1] "training error with cost = 3.51: "
```

```
(sum(table)-sum(diag(table))) / (sum(table))
```

```
## [1] 0.2082671
```

```
test_pred <- predict(lin.svm, wish.test)
table <- table(wish.test$sold_ct_cate, test_pred)

print("test error with cost = 3.51: ")
```

```
## [1] "test error with cost = 3.51: "
```

```
(sum(table)-sum(diag(table))) / (sum(table))
```

```
## [1] 0.2730159
```

```
# we cannot plot SVM classification plot since we have more than 2 columns
table(wish.test$sold_ct_cate, test_pred)
```

```
##            test_pred
##             100 1000 10000 5000 above 20K below 50
##   100        96    6     0    0         0        5
##   1000       15   59     0    8         0        0
##   10000       0    0    20    8         1        0
##   5000        0    7    11   24         0        0
##   above 20K   0    0     3    2        21        0
##   below 50   20    0     0    0         0        9
```

```
set.seed(123)

tuned <- tune(svm, sold_ct_cate ~ ., data = wish.train, kernel = "radial", ranges = list
(cost = append(seq(0.01, 10, by = 0.5), 10)))
summary(tuned) # cost = 10 is best
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.2996762
##
## - Detailed performance results:
##     cost     error dispersion
## 1   0.01 0.6804762 0.03230922
## 2   0.51 0.4213651 0.04927598
## 3   1.01 0.3887619 0.04450719
## 4   1.51 0.3760444 0.04446184
## 5   2.01 0.3593270 0.03565937
## 6   2.51 0.3505714 0.03869834
## 7   3.01 0.3466032 0.04113565
## 8   3.51 0.3442222 0.04441861
## 9   4.01 0.3315111 0.04498217
## 10  4.51 0.3275365 0.04495818
## 11  5.01 0.3235302 0.04675360
## 12  5.51 0.3211238 0.04721553
## 13  6.01 0.3171619 0.04981243
## 14  6.51 0.3163683 0.04859089
## 15  7.01 0.3171556 0.05332027
## 16  7.51 0.3123810 0.05105239
## 17  8.01 0.3084063 0.04625207
## 18  8.51 0.3044254 0.05034701
## 19  9.01 0.3020571 0.05167599
## 20  9.51 0.3020571 0.05288085
## 21 10.00 0.2996762 0.04935219
```

```
table(wish.test$sold_ct_cate, test_pred)
```

```
##            test_pred
##              100 1000 10000 5000 above 20K below 50
##   100         96    6     0    0         0        5
##   1000        15   59     0    8         0        0
##   10000        0    0    20    8         1        0
##   5000         0    7    11   24         0        0
##   above 20K    0    0     3    2        21        0
##   below 50    20    0     0    0         0        9
```

```
rad.svm <- svm(sold_ct_cate ~ ., kernel = "radial", data = wish.train, cost = 10)

train_pred <- predict(rad.svm, wish.train)
table <- table(wish.train$sold_ct_cate, train_pred)

print("radical svm - training error with cost = 10: ")
```

```
## [1] "radical svm - training error with cost = 10: "
```

```
(sum(table)-sum(diag(table))) / (sum(table))
```

```
## [1] 0.1136725
```

```
test_pred <- predict(rad.svm, wish.test)
table <- table(wish.test$sold_ct_cate, test_pred)

print("radical svm - test error with cost = 10: ")
```

```
## [1] "radical svm - test error with cost = 10: "
```

```
(sum(table)-sum(diag(table))) / (sum(table))
```

```
## [1] 0.2920635
```

```
table(wish.test$sold_ct_cate, test_pred)
```

```
##              test_pred
##                100 1000 10000 5000 above 20K below 50
##   100           96    8     0    0         1        2
##   1000          21   50     1   10         0        0
##   10000          0    0    20    6         3        0
##   5000           0    8     8   25         1        0
##   above 20K      0    0     6    1        19        0
##   below 50      16    0     0    0         0       13
```

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

```
tune.poly <- tune(svm, sold_ct_cate ~ ., data = wish.train, kernel = "poly", degree = 3,
ranges = list(cost = append(seq(0.01, 10, by = 0.5), 10)))
summary(tuned) # cost = 10 is best
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.2996762
##
## - Detailed performance results:
##     cost     error dispersion
## 1   0.01 0.6804762 0.03230922
## 2   0.51 0.4213651 0.04927598
## 3   1.01 0.3887619 0.04450719
## 4   1.51 0.3760444 0.04446184
## 5   2.01 0.3593270 0.03565937
## 6   2.51 0.3505714 0.03869834
## 7   3.01 0.3466032 0.04113565
## 8   3.51 0.3442222 0.04441861
## 9   4.01 0.3315111 0.04498217
## 10  4.51 0.3275365 0.04495818
## 11  5.01 0.3235302 0.04675360
## 12  5.51 0.3211238 0.04721553
## 13  6.01 0.3171619 0.04981243
## 14  6.51 0.3163683 0.04859089
## 15  7.01 0.3171556 0.05332027
## 16  7.51 0.3123810 0.05105239
## 17  8.01 0.3084063 0.04625207
## 18  8.51 0.3044254 0.05034701
## 19  9.01 0.3020571 0.05167599
## 20  9.51 0.3020571 0.05288085
## 21 10.00 0.2996762 0.04935219
```

```
poly.svm <- svm(sold_ct_cate ~ ., kernel = "poly", data = wish.train, degree = 3, cost =
10)

train_pred <- predict(poly.svm, wish.train)
table <- table(wish.train$sold_ct_cate, train_pred)

print("poly svm - training error with cost = 10: ")
```

```
## [1] "poly svm - training error with cost = 10: "
```

```
(sum(table)-sum(diag(table))) / (sum(table))
```

```
## [1] 0.290938
```

```
test_pred <- predict(poly.svm, wish.test)
table <- table(wish.test$sold_ct_cate, test_pred)

print("poly svm - test error with cost = 10: ")
```

```
## [1] "poly svm - test error with cost = 10: "
```

```
(sum(table)-sum(diag(table))) / (sum(table))
```

```
## [1] 0.4412698
```

```
table(wish.test$sold_ct_cate, test_pred)
```

```
##            test_pred
##              100 1000 10000 5000 above 20K below 50
##   100         81   24     0    0         0        2
##   1000        31   47     0    4         0        0
##   10000        0   14    11    4         0        0
##   5000         1   33     4    4         0        0
##   above 20K    0    4     2    1        19        0
##   below 50    14    1     0    0         0       14
```

XGBoost

```
library(xgboost)
# Create numeric labels with one-hot encoding
train_labs <- as.numeric(wish.train$sold_ct_cate) - 1
val_labs <- as.numeric(wish.test$sold_ct_cate) - 1

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 = 8, eval_met
ric = "mlogloss")

# Calculate # of folds for cross-validation
xgbcv <- xgb.cv(params = params, data = xgb_train, nrounds = 100, nfold = 5, showsd = TR
UE, stratified = TRUE, print_every_n = 10, early_stop_round = 20, maximize = FALSE, pred
iction = TRUE)
```

```
## [1]  train-mlogloss:1.341189+0.006812    test-mlogloss:1.452198+0.024134
## [11] train-mlogloss:0.257475+0.007266    test-mlogloss:0.652524+0.038305
## [21] train-mlogloss:0.111341+0.003895    test-mlogloss:0.627687+0.044540
## [31] train-mlogloss:0.060830+0.002556    test-mlogloss:0.647127+0.046695
## [41] train-mlogloss:0.038761+0.001365    test-mlogloss:0.672499+0.052703
## [51] train-mlogloss:0.027742+0.001070    test-mlogloss:0.696814+0.060135
## [61] train-mlogloss:0.021398+0.000788    test-mlogloss:0.717969+0.062493
## [71] train-mlogloss:0.017425+0.000493    test-mlogloss:0.739182+0.067279
## [81] train-mlogloss:0.014889+0.000369    test-mlogloss:0.753347+0.070806
## [91] train-mlogloss:0.013131+0.000275    test-mlogloss:0.769416+0.071068
## [100]    train-mlogloss:0.011942+0.000220    test-mlogloss:0.778177+0.072006
```

```
# 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 = "las
t"), label = train_labs + 1)

# Examine output
head(xgb_train_preds)
```

	X1 <dbl>	X2 <dbl>	X3 <dbl>	X4 <dbl>	X5 <dbl>	X6 <dbl>	
1	9.995453e-01	0.0003030366	1.863318e-05	1.896994e-05	2.201454e-05	7.396194e-05	9.04
2	9.969054e-01	0.0028352684	3.497161e-05	7.203760e-05	3.648415e-05	7.369727e-05	2.10
3	8.797839e-01	0.1154617816	9.670191e-04	7.834939e-04	1.020273e-03	1.223267e-03	3.80
4	4.973701e-05	0.9504509568	1.319222e-04	4.921312e-02	4.128080e-05	4.189618e-05	3.55
5	2.813710e-04	0.9624082446	1.920006e-03	3.459550e-02	2.309796e-04	2.181582e-04	1.72
6	9.987790e-01	0.0003156056	8.051770e-05	1.462003e-04	8.252206e-05	5.313216e-04	3.23

6 rows | 1-8 of 11 columns

```
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: 0.2352941
```

```
# 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 = 8, ncol = length(xgb_val_preds) / 8) %>%
               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:", classification_error(xgb_val_conf),
"\n")
```

```
## XGB Validation Classification Error Rate: 0.2126984
```