Case: Multinomial Regression for Data Analytics at Chow Tai Fook

MSBA7002: Business Statistics

Contents

1. Problem Background

Chow Tai Fook Jewellery Group (CTF) was one of the world's largest jewelers, with a Greater China–focused retail network of over 3,100 points of sale (POS) globally as of 31 March 2019. CTF offered a wide range of products in four major categories, including gem- set jewelry, gold products (999.9 pure gold), platinum/karat gold products, and watches. Reflecting its multibrand strategy to serve customers' needs during their different life stages, CTF offered a wide array of brands and products for every occasion. It rolled out new collections frequently to keep the display window fresh and to render a sense of uniqueness to each customer. Unlike fast-moving consumer goods, the high value of inventory balances and relatively slow turnover of individual SKUs in the jewelry industry made good inventory management the key to ensuring healthy profitability.

To proactively respond to the dynamic market changes, CTF deployed big data analytics to improve inventory management as part of its "Smart+2020" strategy. Under the lead of Bobby Liu, the executive director in charge of innovation and technology at CTF, the company formed the strategic business office (SBO) and delegated the SBO to leverage insights from data and assist store managers to optimize inventory assortment and planning at each POS. The SBO was responsible for preparing the quarter-end reports for the senior management team members, who would use the reports for decision making throughout CTF. To compile the report, the SBO needed to first gather available data and then distill useful information by aggregating the data along different dimensions and creating variables. The SBO then chose appropriate analytics methods, developed efficient algorithms, and finally generated prescriptions and insights.

1.1 The Role of SBO

"It is our passion to make use of latest advanced analytics and technology innovation to bring in creativity and support growth and sustainability of the Group." –Jade Lee, General Manager, Analytics and Technology Application, Sustainability and Innovations Centre, Chow Tai Fook

Jade Lee led the SBO team, which had two key functions. First, to deliver advanced analytics to make smarter decisions. Second, to drive the implementation of the latest technology applications in the Group. Jade had over 20 years of experience in advanced analytics carrying out consulting and business management roles with leading business analytics corporations. "Given the uniqueness of the jewelry industry, we need to work out complicated optimization solutions for our inventory management." said Jade.

"The retail industry, however, is very different from the finance sector. We need to respond more swiftly and creatively to customer preferences." –Cher Ng, Associate Director of Hong Kong and Macau Strategic Business Office, Chow Tai Fook.

Cher Ng graduated from HKU in a statistics postgraduate program in 2000. She first worked in the finance sector for a few years. Cher said, "After the financial crisis, a lot of regulations came out, and we were working more on the regulatory side with limited creativity." Thus, she decided to apply her analytical skills to the more dynamic retail industry.

At CTF, Cher worked at the SBO and focused in the data analytics stream. Instead of dealing with the operations, the SBO leveraged the operational data to assess historical performance and predict future trends in the jewelry industry. SBO reports went directly to the senior management.

1.2 CTF's business and challenges

Now you are asked to take Cher's standpoint and summarize the main analytics challenges for jewelry demand forecasting. Which of the following is not a major analytics challenge for CTF's demand forecasting?
A. There were many categories and SKUs, but the sales number for a single product was small.
B. Because the products were slow-moving, CTF rarely introduced new products.
C. The sales or demand of different products were substitutable.
D. The viewing history was not directly linked to any purchase records.

The true analytics challenges for demand forecasting are A, C, and D.

First, because jewelry products are of high value and slow-moving, most products have single digit sales numbers per month or per year. For most days or weeks of a year, the sales number of a product is zero, so it is difficult to capture the correlation between the sales number and any other variables. Thus, it is almost impossible to build a meaningful model to predict demand of a product on a daily or weekly basis. (If we build a linear regression model, the predictions in most cases should be close to zero and could even be negative sometimes.) If we aggregate the data on a yearly basis, then we will have very few data points for each product.

Second, the sales or demand of different products are substitutable. This is because customers may accept more than one design in a category, but they just want to buy one piece. If the best choice is not available, then a customer may buy the second-best choice. Hence, the sales of a product may depend on the availability of other products. The demand forecasting model should be able to describe the relationship among different related products.

Third, there are 16 product categories with more than 15,000 distinct products offered by CTF, but a customer normally only chooses from a small subset of these products. To better capture the utilities of different products, it is important to know what products a customer considered when s/he made the purchase decision. However, at that time, though the viewing history was recorded, the viewing data was not directly linked to the purchase data.

As a result, when we train a demand forecasting model, we should not just use the sales data of a particular product or use the aggregate sales numbers of related products. To build the demand forecasting model, we need to make the following assumptions.
(1) Each category has its intrinsic demand. For example, each potential customer may want to buy a ring and some customers would like to buy a necklace.
(2) There is no intrinsic demand for any single product. This is because customers rarely visit the jewelry stores and thus they have no idea what products are available in a store. They visit a store because they want to buy a product from a certain category, and they will make a choice after they see the products in

the choice set. This is quite different from the traditional retailing business where customers know what toothpastes are in a supermarket.

Based on the above assumptions, the instructor can propose a two-step approach: First, forecast the demand of a category in a given period, and then allocate the demand to each product based on the assortment and choice model. For the first step, CTF may focus on the aggregate sales number of a category and use a regression model with time series components for the forecast. This document focuses on the second step and the choice model in this case. We are going to use the multinomial regression to predict customers' choice.

1.3 Data

```
my_data <- read.csv('chow_tai_fook.csv')
my_data$baseDate <- as.Date(my_data$baseDate)
my_data$productID <- as.factor(my_data$productID)
my_data$mode <- as.logical(my_data$mode)
summary(my_data)
```

```
##    individual       baseDate           branchID_16      branchID_26
##  Min.   :  1.00   Min.   :2005-01-12   Min.   :0.0000   Min.   :0.0000
##  1st Qu.: 79.75   1st Qu.:2006-01-07   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :158.50   Median :2006-07-15   Median :0.0000   Median :0.0000
##  Mean   :158.50   Mean   :2006-05-31   Mean   :0.4019   Mean   :0.2848
##  3rd Qu.:237.25   3rd Qu.:2006-11-15   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :316.00   Max.   :2007-01-03   Max.   :1.0000   Max.   :1.0000
##
##   branchID_222          ct           productID      r_info1_11
##  Min.   :0.0000   Min.   : 131.0   213    : 316   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.: 259.8   290    : 316   1st Qu.:0.0000
##  Median :0.0000   Median : 339.0   6204   : 316   Median :0.0000
##  Mean   :0.3133   Mean   : 390.9   6605   : 316   Mean   :0.1714
##  3rd Qu.:1.0000   3rd Qu.: 504.0   6713   : 316   3rd Qu.:0.0000
##  Max.   :1.0000   Max.   :1088.0   70565  : 316   Max.   :1.0000
##                                    (Other):9164
##  r_info1_113_276  r_info1_111_126   r_info3_1        r_info3_23
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.0000   Median :0.0000   Median :0.0000   Median :0.0000
##  Mean   :0.4571   Mean   :0.1429   Mean   :0.4571   Mean   :0.3714
##  3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:1.0000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##
##   r_info3_4567    common_info1_0     mode            noStock
##  Min.   :0.0000   Min.   :0.0000   Mode :logical   Min.   :0.000
##  1st Qu.:0.0000   1st Qu.:1.0000   FALSE:10744     1st Qu.:0.000
##  Median :0.0000   Median :1.0000   TRUE :316       Median :1.000
##  Mean   :0.1714   Mean   :0.7714                   Mean   :1.272
##  3rd Qu.:0.0000   3rd Qu.:1.0000                   3rd Qu.:2.000
##  Max.   :1.0000   Max.   :1.0000                   Max.   :7.000
##
##    noDisplay         noSold
##  Min.   :0.0000   Min.   :0.00000
##  1st Qu.:0.0000   1st Qu.:0.00000
##  Median :0.0000   Median :0.00000
##  Mean   :0.0359   Mean   :0.02857
```

```
##  3rd Qu.:0.0000   3rd Qu.:0.00000
##  Max.   :2.0000   Max.   :1.00000
##
```

The data set contains the following variables from 3 branches of CTF with id 16, 26, 222.

- individual: the index of each purchase. It is aligned with baseDate. If the baseDate is earlier, the individual has a smaller index number.
- baseDate: the date of the purchase happened.
- branchID_16: binary variable. 1-purchase happened in branch 16.
- branchID_26: binary variable. 1-purchase happened in branch 26.
- branchID_222: binary variable. 1-purchase happened in branch 222.
- ct: customer arrival count of the branch on that day.
- ProductID: the ID of available products. Each individual has same choices with the same order.
- r_info1_11: binary variable. 1 – the value of r_info1 is 11. r_info1 is a categorical variable describing a undisclosed feature of a product.
- r_info1_113_276: binary variable. 1 – the value of r_info1 is 113 or 276.
- r_info1_111_126: binary variable. 1 – the value of r_info1 is 111 or 126.
- r_info3_1: binary variable. 1 – the value of r_info3 is 1. r_info3 is a categorical variable describing a undisclosed feature of a product.
- r_info3_23: binary variable. 1 – the value of r_info3 is 2 or 3.
- r_info3_4567: binary variable. 1 – the value of r_info3 is 4 or 5 or 6 or 7.
- common_info1_0: binary variable. 1 – the value of common_info1 is 0. common_info1 is a categorical variable describing a undisclosed feature of a product.
- mode: binary variable. 1 – individual n chooses to buy that product. Each individual only can choose one product.
- noStock: the beginning inventory level of the product on that day.
- noDisplay: the number of units of the product displayed in the show room on that day.
- noSold: The number of units sold on that day. The branch can borrow the product from other branches, so the sales can be positive even if the stock level is zero.

2. Multinomial regression model

The model is to predict customers' choice.

2.1 Data preparation

In our analysis, we use the sales records in last 60 days as the test data and the previous records as the training data. In the training data, there are only 35 products with positive sales. Hence, we focus on these 35 products and assume that they form the choice set of each customer. The total sales number of these 35 products in the entire data set is 316, and we assume these are 316 independent purchases.

```
# Training/Testing set split
training <- my_data %>% filter(baseDate < "2006-11-04")
nrow(training)
```

```
## [1] 7980
```

```
testing <- my_data %>% filter(baseDate >= "2006-11-04")
nrow(testing)
```

```
## [1] 3080
```

2.2 Model Fitting using multinom

The function "multinom" from package {nnet} assumes all features are individual specific, so the final model will contain the effects of all included variables for the log odds of 35 classes.

```
training_selected <- training[training$mode == TRUE,]

training_multinom <- training
training_multinom$purchaseID <- rep(training_selected$productID, each=35)

fit.multinom <- multinom(purchaseID ~ r_info1_11 + r_info1_111_126 +
                  r_info3_1 + r_info3_23 + r_info3_4567 + common_info1_0 +
                  ct + branchID_16 + branchID_26, data=training_multinom)
```

```
## # weights:  385 (340 variable)
## initial  value 28371.677531
## iter  10 value 26327.937296
## iter  20 value 24223.314680
## iter  30 value 23681.891177
## iter  40 value 23205.003490
## iter  50 value 21936.175107
## iter  60 value 21241.418061
## iter  70 value 21053.479404
## iter  80 value 20914.762393
## iter  90 value 20794.634413
## iter 100 value 20727.189492
## final  value 20727.189492
## stopped after 100 iterations
```

```
fit.multinom
```

```
## Call:
## multinom(formula = purchaseID ~ r_info1_11 + r_info1_111_126 +
##     r_info3_1 + r_info3_23 + r_info3_4567 + common_info1_0 +
##     ct + branchID_16 + branchID_26, data = training_multinom)
##
```

```
## Coefficients:
##       (Intercept)   r_info1_11 r_info1_111_126   r_info3_1  r_info3_23
## 290    -1.1425334  0.091501592    -0.034654951 -0.28672660 -0.38330047
## 6204   -0.1758391 -0.001591649    -0.053993904  0.01761025 -0.04499636
## 6605   -3.9133915 -0.058369667    -0.076313811 -1.36800800 -1.27651756
## 6713  -10.1810414  0.024511748    -0.005498796 -3.33603733 -3.37001200
## 70565  -3.9990628  0.068515766     0.058366701 -1.24226396 -1.30061968
## 70567  -2.1909141  0.107120439     0.043460689 -0.65116094 -0.72409623
## 70620   3.9190915  0.083633943    -0.176786323  1.32137500  1.34201705
## 70624   3.3455981  0.137920426    -0.321257792  1.29197351  1.12400819
## 70626   3.3649714 -0.035678555    -0.325021547  1.12559229  1.11325126
## 70643   0.6154041  0.032686831    -0.116095947  0.23687574  0.25300544
## 70645   4.7893194 -0.199555422    -0.049203873  1.77159168  1.78310452
## 70647   2.6693077  0.117778752    -0.106803859  0.80379037  0.79545642
## 73053   0.3256670  0.050779694    -0.148576596  0.13893301  0.10758491
## 73798  -3.0896226  0.080993314     0.002943067 -0.93964919 -1.00927366
## 74020   3.1076182 -0.036368435    -0.040409910  0.94533789  1.01006912
## 74021   4.3869497  0.019611302    -0.088469687  1.48501924  1.46842441
## 74022  -4.3321626  0.075378308     0.081591012 -1.38598838 -1.40346726
## 74023  -1.1295574  0.024335044    -0.031884712 -0.30610797 -0.47144251
## 74024   1.1739106 -0.105773930     0.059614580  0.22264558  0.36684743
## 74026   3.9241170 -0.023418894    -0.019668043  1.25924021  1.29629909
## 74028   4.2457435 -0.060254950    -0.123850492  1.38607611  1.42899388
## 74029   3.1841970 -0.013673121    -0.110235461  1.03830718  1.02895115
## 74430  -1.3204499  0.027407258    -0.410452584 -0.32099054 -0.34984878
## 74493  -0.7324368  0.184810404    -0.024164146 -0.03708147 -0.27217622
## 74495  -0.9868850  0.113263583    -0.050726694 -0.28324882 -0.25506378
## 74497  -1.3936003 -0.023872026    -0.167736829 -0.34950315 -0.51002092
## 75393  -2.3697889 -0.029151798    -0.007933390 -0.67004459 -0.81615969
## 75601  -2.6081842 -0.036218822    -0.068955585 -0.95797188 -0.92415639
## 76531   0.2137055 -0.003389108    -0.042660937  0.15012102  0.09936667
## 76538   0.0410076  0.048656653    -0.055884772  0.07259917  0.01131580
## 77980  -2.2125497  0.067098146    -0.014648725 -0.69375481 -0.74430136
## 78897  -7.6538332  0.006697315    -0.039585514 -2.56457314 -2.56500582
## 79691 -10.7236817  0.083854373    -0.144259463 -3.54689374 -3.61239823
## 79698   6.5946082  0.059612182     0.038062952  2.25741997  2.23230180
##       r_info3_4567 common_info1_0            ct branchID_16 branchID_26
## 290    -0.47250630   -0.075485866  0.0096915076  -6.9696778  2.89105112
## 6204   -0.14845302   -0.177749470  0.0079964248  -2.1417103 -2.37297051
## 6605   -1.26886599   -0.009477853  0.0087966444   1.7463049  7.85965551
## 6713   -3.47499205   -0.149173452  0.0207797461   5.2612399  7.19101315
## 70565  -1.45617918   -0.176571717  0.0145528069  -5.3838054  4.76876438
## 70567  -0.81565697   -0.021030481  0.0054951567   1.9672916  1.40122518
## 70620   1.25569943   -0.130567688  0.0016858538  -5.7325180 -0.04052626
## 70624   0.92961639   -0.380277398  0.0013800653  -4.6275086  0.56837720
## 70626   1.12612785   -0.234662502 -0.0002727803  -4.7935013 -5.69959292
## 70643   0.12552293   -0.107756498  0.0062865955  -3.3214315  1.61221885
## 70645   1.23462316   -0.434785040 -0.0054733985  -4.2393796 -0.69694217
## 70647   1.07006092    0.179706222  0.0023984466  -9.9768353 -5.47217806
## 73053   0.07914907   -0.095006834  0.0068264215  -3.0885563 -3.55566294
## 73798  -1.14069972   -0.194375529  0.0092986711   0.5728029  2.40877579
## 74020   1.15221119    0.197148376  0.0029110967  -4.3430213  1.04570113
## 74021   1.43350605   -0.061832301  0.0019922651  -3.8203483  0.21018230
## 74022  -1.54270696   -0.162254074  0.0101886172   2.5501989  7.64576132
```

```
## 74023  -0.35200688    0.063462080  0.0095003975  -0.8770451 -1.72819908
## 74024   0.58441759    0.380323570  0.0068997687  -3.2031467  1.73769622
## 74026   1.36857771    0.139260046  0.0008892916  -4.0417979  0.51283917
## 74028   1.43067354   -0.005116841  0.0010547709  -4.9780119  0.78719046
## 74029   1.11693863   -0.043163484  0.0033711582  -4.6795973  0.16758256
## 74430  -0.64961056   -0.346647399  0.0024358089  -4.7244050  5.44800013
## 74493  -0.42317915   -0.388052750  0.0107680432  -1.5796553  2.82469396
## 74495  -0.44857240   -0.038292065  0.0018427056   0.7989832  4.90960441
## 74497  -0.53407619   -0.008184440  0.0111401582  -7.0609398  2.63040561
## 75393  -0.88358467   -0.195945201  0.0068721910  -3.1323042  6.13433410
## 75601  -0.72605592    0.006843223  0.0070422808   0.6977630  6.30581660
## 76531  -0.03578224   -0.073622620  0.0071172589  -7.2293931 -2.71999488
## 76538  -0.04290736   -0.090132183  0.0074039313  -7.6431055  2.03887587
## 77980  -0.77449349   -0.159222414  0.0118577319  -1.6034589 -2.14784348
## 78897  -2.52425426   -0.095568703  0.0161234919   3.9640610  5.34917063
## 79691  -3.56438973   -0.157745524  0.0210288497   3.1560671 10.56683687
## 79698   2.10488643   -0.115491249 -0.0187933066  -4.4920726 -2.83892509
##
## Residual Deviance: 41454.38
## AIC: 42066.38
```

2.2.1 Training result

```r
#Predicted training result
train_result <- predict(fit.multinom, newdata=training_selected)
#train_result

#Actual result of training set
train_actual <- training_selected$productID
#train_actual
```

Training accuracy of the predicted 1st choice

```r
train_correct_cnt <- 0
for (i in 1:length(train_actual)){
  if (train_actual[i] == train_result[i]){
    train_correct_cnt = train_correct_cnt + 1
  }
}
train_correct_cnt
```

```
## [1] 54
```

```r
length(train_actual)
```

```
## [1] 228
```

```r
train_correct_cnt/length(train_actual)
```

```
## [1] 0.2368421
```

Among 228 training data, 54 actual choice (23.7%) is indeed the top choice.

```r
train_result_df <- predict(fit.multinom, newdata=training_selected,
                           type="probs")
#train_result_df
```

Training accuracy of the predicted first 3 choices

```r
train_correct_cnt_1 <- 0
for (i in 1:length(train_actual)){
  array_1 <- sort(desc(train_result_df[i,]))[1:3]
  name_1 <- names(array_1)
  if (train_actual[i] %in% name_1){
    train_correct_cnt_1 = train_correct_cnt_1 + 1
  }
}
train_correct_cnt_1
```

```
## [1] 99
```

```r
length(train_actual)
```

```
## [1] 228
```

```r
train_correct_cnt_1/length(train_actual)
```

```
## [1] 0.4342105
```

Among 228 training data, 99 actual choice (43.4%) is in the first 3 choices with highest predicted probability.

Training accuracy of the predicted first 5 choices

```r
train_correct_cnt_2 <- 0
for (i in 1:length(train_actual)){
  array_1 <- sort(desc(train_result_df[i,]))[1:5]
  name_1 <- names(array_1)
  if (train_actual[i] %in% name_1){
    train_correct_cnt_2 = train_correct_cnt_2 + 1
  }
}
train_correct_cnt_2
```

```
## [1] 137
```

```r
train_correct_cnt_2/length(train_actual)
```

```
## [1] 0.6008772
```

2.2.2 Testing result

```r
#TEST
testing_selected <- testing[testing$mode == TRUE,]

test_result <- predict(fit.multinom, newdata=testing_selected)
#test_result

# Actual result of test set
test_actual <- testing_selected$productID
#test_actual
```

Test accuracy of predicted 1st choice

```r
test_correct_cnt <- 0
for (i in 1:length(test_actual)){
  if (test_actual[i] == test_result[i]){
    test_correct_cnt = test_correct_cnt + 1
```

```r
  }
}
test_correct_cnt
```

```
## [1] 15
```

```r
length(test_actual)
```

```
## [1] 88
```

```r
test_correct_cnt/length(test_actual)
```

```
## [1] 0.1704545
```

```r
test_result_df <- predict(fit.multinom, newdata=testing_selected,
                          type="probs")
#test_result_df
```

Test accuracy of predicted first 3 choice

```r
test_correct_cnt_1 <- 0
for (i in 1:length(test_actual)){
  array_1 <- sort(desc(test_result_df[i,]))[1:3]
  name_1 <- names(array_1)
  if (test_actual[i] %in% name_1){
    test_correct_cnt_1 = test_correct_cnt_1 + 1
  }
}
test_correct_cnt_1
```

```
## [1] 24
```

```r
test_correct_cnt_1/length(test_actual)
```

```
## [1] 0.2727273
```

Test accuracy of predicted first 5 choice

```r
test_correct_cnt_2 <- 0
for (i in 1:length(test_actual)){
  array_1 <- sort(desc(test_result_df[i,]))[1:5]
  name_1 <- names(array_1)
  if (test_actual[i] %in% name_1){
    test_correct_cnt_2 = test_correct_cnt_2 + 1
  }
}
test_correct_cnt_2
```

```
## [1] 31
```

```r
test_correct_cnt_2/length(test_actual)
```

```
## [1] 0.3522727
```

2.3 Model Fitting using mnlogit

The function "mnlogit" from package {mnlogit} can be more flexible in modeling alternative specific variables. We would like to reduce the number of parameters as in the previous section ((35-1) * 10 = 340 coefficients in the previous model since the first class is our baseline) and assume the effects of "r_info1", "r_info3", "common_info1" on log odds are shared for each product. We can do this using "mnlogit".

```
# Convert dataframe into mlogit format with shape "long" to fit mnlogit model
training_mlogit <- mlogit.data(training, choice = "mode", shape = "long",
                               alt.var = "productID", id.var = "individual")
testing_mlogit <- mlogit.data(testing, choice = "mode", shape = "long",
                              alt.var = "productID", id.var = "individual")
# choice - a binary level variable indicating customer buy or not buy
# alt.var - the column indicating the choices
# id.var - the column indicating the choice-maker
```

```
# mnlogit model formula
fm_train <- formula(mode ~ r_info1_11 + r_info1_111_126 +  r_info3_1 +
                      r_info3_23 + r_info3_4567 + common_info1_0 + 1| ct +
                      branchID_16 + branchID_26 + 1| 1)
#training
fit_train <- mnlogit(fm_train, training_mlogit, choiceVar= "productID",ncores=2)
fit_train
```

```
##
## Call:
## mnlogit(formula = fm_train, data = training_mlogit, choiceVar = "productID",     ncores = 2)
##
## Coefficients:
##   (Intercept):290   (Intercept):6204   (Intercept):6605   (Intercept):6713
##        3.9827e+00        -4.1412e+01        -1.1844e+02        -1.5208e+03
## (Intercept):70565  (Intercept):70567  (Intercept):70620  (Intercept):70624
##        4.2478e+01         2.6652e+01        -4.8563e+01        -4.9726e+01
## (Intercept):70626  (Intercept):70643  (Intercept):70645  (Intercept):70647
##       -4.9303e+01        -5.3019e+01        -4.7781e+01        -5.0104e+01
## (Intercept):73053  (Intercept):73798  (Intercept):74020  (Intercept):74021
##       -4.0736e+01        -1.8411e+01        -4.9484e+01        -4.7952e+01
## (Intercept):74022  (Intercept):74023  (Intercept):74024  (Intercept):74026
##       -7.7439e+01        -5.5456e+01        -5.1949e+01        -4.8459e+01
## (Intercept):74028  (Intercept):74029  (Intercept):74430  (Intercept):74493
##       -4.7881e+01        -4.9576e+01         2.9175e+01         4.6379e+01
## (Intercept):74495  (Intercept):74497  (Intercept):75393  (Intercept):75601
##        2.8862e+01         4.5795e+01        -1.6040e+01        -1.1742e+02
## (Intercept):76531  (Intercept):76538  (Intercept):77980  (Intercept):78897
##       -5.3381e+01        -5.3760e+01        -3.1988e+01        -1.2655e+02
## (Intercept):79691  (Intercept):79698             ct:290            ct:6204
##       -1.3414e+02        -9.9519e+01         1.0431e-02         8.6827e-03
##           ct:6605            ct:6713           ct:70565           ct:70567
##        9.5756e-03         1.3952e+00         1.4808e-02         6.5356e-03
##          ct:70620           ct:70624           ct:70626           ct:70643
##        2.1819e-03         2.3102e-03        -1.3490e-06         7.0121e-03
##          ct:70645           ct:70647           ct:73053           ct:73798
##       -3.4622e-03         3.0507e-03         7.6325e-03         1.1142e-02
##          ct:74020           ct:74021           ct:74022           ct:74023
##        3.6171e-03         2.7838e-03         1.0921e-02         1.0668e-02
##          ct:74024           ct:74026           ct:74028           ct:74029
```

```
##        7.5260e-03          1.6028e-03          1.3822e-03          4.1292e-03
##          ct:74430            ct:74493            ct:74495            ct:74497
##        2.0932e-03          1.1435e-02          2.5829e-03          1.1669e-02
##          ct:75393            ct:75601            ct:76531            ct:76538
##        7.6013e-03          7.9162e-03          7.5982e-03          8.1946e-03
##          ct:77980            ct:78897            ct:79691            ct:79698
##        1.2052e-02          1.9502e-02          2.7293e-02         -3.4469e-02
##   branchID_16:290    branchID_16:6204    branchID_16:6605    branchID_16:6713
##       -4.3142e+01         -1.9735e+01          1.7215e+00          7.2564e+02
## branchID_16:70565   branchID_16:70567   branchID_16:70620   branchID_16:70624
##       -4.1064e+01          2.2493e+00         -2.3193e+01         -2.2068e+01
## branchID_16:70626   branchID_16:70643   branchID_16:70645   branchID_16:70647
##       -2.2510e+01         -2.0931e+01         -2.1963e+01         -4.5980e+01
## branchID_16:73053   branchID_16:73798   branchID_16:74020   branchID_16:74021
##       -2.0751e+01          2.3098e+00         -2.2010e+01         -2.1458e+01
## branchID_16:74022   branchID_16:74023   branchID_16:74024   branchID_16:74026
##        2.8440e+00         -1.8355e+01         -2.0782e+01         -2.1623e+01
## branchID_16:74028   branchID_16:74029   branchID_16:74430   branchID_16:74493
##       -2.2724e+01         -2.2363e+01         -2.3666e+01         -1.9159e+01
## branchID_16:74495   branchID_16:74497   branchID_16:75393   branchID_16:75601
##        5.7169e-01         -4.3231e+01         -2.2236e+01          1.2579e+00
## branchID_16:76531   branchID_16:76538   branchID_16:77980   branchID_16:78897
##       -4.4146e+01         -4.3951e+01         -1.9196e+01          5.8367e+00
## branchID_16:79691   branchID_16:79698     branchID_26:290    branchID_26:6204
##       -1.3359e+01         -5.9431e+00          2.5426e+00         -1.9588e+01
##  branchID_26:6605    branchID_26:6713   branchID_26:70565   branchID_26:70567
##        2.5185e+01          5.6522e+02          4.4080e+00          1.6712e+00
## branchID_26:70620   branchID_26:70624   branchID_26:70626   branchID_26:70643
##       -2.7467e-01          5.6593e-01         -2.1786e+01          1.4005e+00
## branchID_26:70645   branchID_26:70647   branchID_26:73053   branchID_26:73798
##       -1.0069e+00         -2.1850e+01         -1.9929e+01          3.1731e+00
## branchID_26:74020   branchID_26:74021   branchID_26:74022   branchID_26:74023
##        8.1063e-01         -3.8490e-02          2.5218e+01         -1.8848e+01
## branchID_26:74024   branchID_26:74026   branchID_26:74028   branchID_26:74029
##        1.5570e+00          3.3580e-01          4.3578e-01         -9.0021e-02
## branchID_26:74430   branchID_26:74493   branchID_26:74495   branchID_26:74497
##        2.1934e+01          2.5236e+00          2.2114e+01          2.3294e+00
## branchID_26:75393   branchID_26:75601   branchID_26:76531   branchID_26:76538
##        2.4198e+01          2.4296e+01         -1.9934e+01          1.7680e+00
## branchID_26:77980   branchID_26:78897   branchID_26:79691   branchID_26:79698
##       -1.8239e+01          6.7601e+00          3.1094e+01         -5.9745e+00
##        r_info1_11     r_info1_111_126           r_info3_1          r_info3_23
##       -1.4322e+02         -4.6601e+01         -2.2766e+01          1.9144e+01
##      r_info3_4567       common_info1_0
##        3.0997e+01          6.6429e+01
```

```r
# just for data formatting
attr(testing_mlogit,"index") <- idx(testing_mlogit)
attr(training_mlogit,"index") <- idx(training_mlogit)
```

The model has (35-1) * 4 + 6 = 142 coefficients, where the 6 coefficients are shared across products.

2.3.1 Training result

```
#Predicted training result
train_result <- predict(fit_train,newdata=training_mlogit, probability = F,
                        choiceVar = "productID")
#train_result

#Actual result of training set
train_actual <- (training %>% filter(mode == 1))$productID
#train_actual
```

Training accuracy of the predicted 1st choice

```
train_correct_cnt <- 0
for (i in 1:length(train_actual)){
  if (train_actual[i] == train_result[i]){
    train_correct_cnt = train_correct_cnt + 1
  }
}
train_correct_cnt
```

```
## [1] 54
```

```
length(train_actual)
```

```
## [1] 228
```

```
train_correct_cnt/length(train_actual)
```

```
## [1] 0.2368421
```

```
train_result_df <- predict(fit_train,newdata=training_mlogit, probability = T,
                           choiceVar = "productID")
#train_result_df
```

Training accuracy of the predicted first 3 choices

```
train_correct_cnt_1 <- 0
for (i in 1:length(train_actual)){
  array_1 <- sort(desc(train_result_df[i,]))[1:3]
  name_1 <- names(array_1)
  if (train_actual[i] %in% name_1){
    train_correct_cnt_1 = train_correct_cnt_1 + 1
  }
}
train_correct_cnt_1
```

```
## [1] 99
```

```
train_correct_cnt_1/length(train_actual)
```

```
## [1] 0.4342105
```

Training accuracy of the predicted first 5 choices

```
train_correct_cnt_2 <- 0
for (i in 1:length(train_actual)){
  array_1 <- sort(desc(train_result_df[i,]))[1:5]
  name_1 <- names(array_1)
  if (train_actual[i] %in% name_1){
```

```
    train_correct_cnt_2 = train_correct_cnt_2 + 1
  }
}
train_correct_cnt_2
```

```
## [1] 139
```

```
train_correct_cnt_2/length(train_actual)
```

```
## [1] 0.6096491
```

2.3.2 Testing result

```
#TEST
test_result <- predict(fit_train,newdata=testing_mlogit, probability = F,
                       choiceVar = "productID")
#test_result

# Actual result of test set
test_actual <- (testing %>% filter(mode == 1))$productID
#test_actual
```

Test accuracy of predicted 1st choice

```
test_correct_cnt <- 0
for (i in 1:length(test_actual)){
  if (test_actual[i] == test_result[i]){
    test_correct_cnt = test_correct_cnt + 1
  }
}
test_correct_cnt
```

```
## [1] 15
```

```
length(test_actual)
```

```
## [1] 88
```

```
test_correct_cnt/length(test_actual)
```

```
## [1] 0.1704545
```

```
test_result_df <- predict(fit_train,newdata=testing_mlogit, probability = T,
                          choiceVar = "productID")
#test_result_df
```

Test accuracy of predicted first 3 choice

```
test_correct_cnt_1 <- 0
for (i in 1:length(test_actual)){
  array_1 <- sort(desc(test_result_df[i,]))[1:3]
  name_1 <- names(array_1)
  if (test_actual[i] %in% name_1){
    test_correct_cnt_1 = test_correct_cnt_1 + 1
  }
}
test_correct_cnt_1
```

```
## [1] 24
```

```
test_correct_cnt_1/length(test_actual)
```

```
## [1] 0.2727273
```

Test accuracy of predicted first 5 choice

```
test_correct_cnt_2 <- 0
for (i in 1:length(test_actual)){
  array_1 <- sort(desc(test_result_df[i,]))[1:5]
  name_1 <- names(array_1)
  if (test_actual[i] %in% name_1){
    test_correct_cnt_2 = test_correct_cnt_2 + 1
  }
}
test_correct_cnt_2
```

```
## [1] 33
```

```
test_correct_cnt_2/length(test_actual)
```

```
## [1] 0.375
```

In terms of performance, the sparser model from mnlogit gives slight improvement over the dense model from multinom. So it may be slightly favored.