Market Basket Demo

Jeffrey Sumner

February 18, 2019

Introduction

Data

This data comes from a kaggle competition in 2017. All files are open to the public, particularly those interested in Kaggle competitions. The data competition overview can be found at the following link: https://www.kaggle.com/c/instacart-market-basket-analysis.

Data specs of note:

aisles.csv - 134 x 2
departments.csv - 21 x 2
order_products_prior.csv - 32.4m x 4
order_products_train.csv - 1.38m x 4
orders.csv - 3.42m x 7
products.csv 49.7k x 4

Objective

To find potential product pairings based on order_products_prior and order_products_train.

Methodology

Data collection: Kaggle created .csv's
Data manipulation: Extensive use of R's tidyverse package, particularly dplyr to create end-user and dashboard friendly datasets.
Graphics: ggplot can be used to create beautiful visuals to further showcase the final pairings information; dashboarding tools such as PowerBI and Tableau can be used to create end-user friendly UI

R Scripting

Required Libraries

As mentioned above, tidyverse will be a pivotal package for this analysis as well as multidplyr. multidplyr is a lesser-known package that acts similarly to **parallel**. multidplyr uses multiple cores to make efficient use of the CPU and increase computational speeds.

```
# libraries required to perform analysis
library(tidyverse)
library(multidplyr)
library(data.table)
library(writexl)

# initialize clusters for multidplyr functionality
cluster <- get_default_cluster()
# add required packages to each cluster that are used in conjunction with multidplyr
cluster_library(cluster,"tidyverse")
options(scipen = 999)
```

Reading and Cleaning the Data

Next up is reading in the files from Kaggle. The files are stored on my local machine, in this case, inside of my project folder. I prefer to duplicate as little code as possible, so I loop through each file and assign them to data.frames

```
# read in data
# fread is located in the data.table package
# it is an extremely fast way to read in data
for(i in list.files(pattern = ".csv",full.names = FALSE)){
  assign(gsub(".csv","",i),
         fread(i,
               data.table = FALSE)
         )
}


# Here I combine the prior and train data to complete the full dataset
order_products <- rbind(order_products__prior,order_products__train)
rm(order_products__prior)
rm(order_products__train)
gc()
```

```
##            used  (Mb) gc trigger (Mb)  max used (Mb)
## Ncells   902790  48.3    1665137   89   1665137   89
## Vcells 98435806 751.1  264761633 2020 264761633 2020
```

```
# For this demo I wanted to focus only on Yogurt data.
# I look for any products that contain "Yogurt"" in the products file
products_yogurt <- products %>%
  mutate(contains_yogurt = str_detect(product_name,"Yogurt")) %>%
  filter(contains_yogurt)

# filter non-yogurt products from the data
order_products_yogurt <- order_products %>%
  filter(product_id %in% products_yogurt$product_id)
```

Create Pairing Data

The last major portion of this analysis is to create the pairing data. To do this, we will perform very simple, yet powerful data maniuplation. This manipulation will allow us to create pair combinations which can then be used to create visuals.

```
# Create pairs dataset via data manipulations
yogurt_pairs <- order_products_yogurt %>%
  select(-add_to_cart_order,-reordered) %>%
  # Partition replaces generic group_by
  # This significantly increases computation speed
  partition(order_id) %>%
  # str_c in combination with partition creates
  # A new column with all possible combinations of a particular order
  mutate(product_id_c = str_c(product_id,collapse = ","),
         counts = length(unique(product_id))) %>%
  # Collect is always used after partition to
  # "Collect" the data off the cores used
  collect() %>%
```

```
  ungroup() %>%
  # Maxlen is created to determine the maximum number of new
  # Columns needed to finalize the manipulation
  mutate(maxlen = max(counts)) %>%
  # Separate spreads our product_id_c into multiple columns based
  # on the ","
  # Some products may pair only once or twice so missing values will fill
  # Any particular pairings that do not need the full maximum
  # Number of columns
  separate(product_id_c,
           into = paste("V",
                        1:unique(.$maxlen)
                        ),
           sep = ",") %>%
  # finally we gather the newly created V columns
  # then filter out the unwanted NA's
  gather(pair_num,pair_id,-order_id,-product_id,-counts,-maxlen) %>%
  mutate(pair_id = as.numeric(pair_id)) %>%
  filter(!pair_id %in% NA)
```

```
## Warning: group_indices_.grouped_df ignores extra arguments
```

```
## Warning: Expected 40 pieces. Missing pieces filled with `NA` in 1406741
## rows [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
## 20, ...].
```

```
glimpse(yogurt_pairs)
```

```
## Observations: 3,803,709
## Variables: 6
## $ order_id   <int> 14, 27, 32, 46, 46, 55, 70, 80, 99, 114, 120, 130, ...
## $ product_id <int> 39475, 30442, 15991, 34519, 11983, 17872, 44008, 10...
## $ counts     <int> 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, ...
## $ maxlen     <dbl> 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,...
## $ pair_num   <chr> "V 1", "V 1", "V 1", "V 1", "V 1", "V 1", "V 1", "V...
## $ pair_id    <dbl> 39475, 30442, 15991, 34519, 34519, 17872, 44008, 10...
```

```
head(yogurt_pairs,25)
```

```
## # A tibble: 25 x 6
##    order_id product_id counts maxlen pair_num pair_id
##       <int>      <int>  <int>  <dbl> <chr>      <dbl>
##  1       14      39475      1     40 V 1        39475
##  2       27      30442      1     40 V 1        30442
##  3       32      15991      1     40 V 1        15991
##  4       46      34519      2     40 V 1        34519
##  5       46      11983      2     40 V 1        34519
##  6       55      17872      1     40 V 1        17872
##  7       70      44008      1     40 V 1        44008
##  8       80      10761      1     40 V 1        10761
##  9       99      46584      1     40 V 1        46584
## 10      114      24954      1     40 V 1        24954
## # ... with 15 more rows
```

The Final Touches

The hard part is over! The data has been cleaned and transformed to fit our needs. Now all that remains is to create product references for the original product as well as the newly created pairing product.

```
# reference for original product
product_id_ref <- products_yogurt %>%
  select(product_id, product_name_1 = product_name,
         aisle_id_1 = aisle_id, department_id_1 = department_id)
# reference for paired product
pair_id_ref <- products_yogurt %>%
  select(product_id, product_name_2 = product_name,
         aisle_id_2 = aisle_id, department_id_2 = department_id)
# join the references back to the yogurt_pairs
# remove any pairs that are same item pairs
yogurt_pairs_clean <- yogurt_pairs %>%
  left_join(product_id_ref, by = c("product_id" = "product_id")) %>%
  left_join(pair_id_ref, by = c("pair_id" = "product_id")) %>%
  mutate(same_id = ifelse(product_id == pair_id,TRUE,FALSE)) %>%
  filter(!same_id)
```

Conclusions

How to use the Data

Now that we have finished all of the heavy lifting, how can this data be used? Below we created a table of counts for each pairing. These can also be created in PowerBI, Tableau, or any other BI software as needed.

```
table_counts <- yogurt_pairs_clean %>%
  group_by(product_id,product_name_1,pair_id,product_name_2) %>%
  summarize(orders_together = n_distinct(order_id)) %>%
  arrange(desc(orders_together)) %>%
  ungroup()

head(table_counts,10)
```

```
## # A tibble: 10 x 5
##    product_id product_name_1     pair_id product_name_2     orders_together
##         <int> <chr>                <dbl> <chr>                       <int>
##  1       4957 Total 2% Lowfat G~   33754 Total 2% with Str~            9565
##  2      33754 Total 2% with Str~    4957 Total 2% Lowfat G~            9565
##  3      33754 Total 2% with Str~   33787 Total 2% Lowfat G~            8241
##  4      33787 Total 2% Lowfat G~   33754 Total 2% with Str~            8241
##  5      28465 Icelandic Style S~   36865 Non Fat Raspberry~            7444
##  6      36865 Non Fat Raspberry~   28465 Icelandic Style S~            7444
##  7      24799 Vanilla Skyr Nonf~   28465 Icelandic Style S~            6844
##  8      28465 Icelandic Style S~   24799 Vanilla Skyr Nonf~            6844
##  9       4957 Total 2% Lowfat G~   33787 Total 2% Lowfat G~            6461
## 10      33787 Total 2% Lowfat G~    4957 Total 2% Lowfat G~            6461
```

Additional Analysis

We can extend the work in R by adding in order values for each product to determine probabilites of a pair occurring. To do this, we will first create a count for each order.

```
orders_by_product <- order_products_yogurt %>%
  group_by(product_id) %>%
  summarize(order_counts = n_distinct(order_id,na.rm = TRUE)) %>%
  ungroup()
```

Now we have total orders for each product. We need to add this back to our counts table. This must be done twice to account for both the original product and the paired product. We will also go ahead and add in the total number of orders in the entire dataset to calculate additional metrics.

```
total_orders <- length(unique(order_products_yogurt$order_id))

table_counts_additions <- table_counts %>%
  left_join(orders_by_product %>%
              select(product_id,order_counts_1 = order_counts),by = "product_id") %>%
    left_join(orders_by_product %>%
              select(product_id,order_counts_2 = order_counts),by = c("pair_id"="product_id")) %>%
  mutate(total_orders = total_orders)
glimpse(table_counts_additions)
```

```
## Observations: 153,508
## Variables: 8
## $ product_id      <int> 4957, 33754, 33754, 33787, 28465, 36865, 24799...
## $ product_name_1  <chr> "Total 2% Lowfat Greek Strained Yogurt With Bl...
## $ pair_id         <dbl> 33754, 4957, 33787, 33754, 36865, 28465, 28465...
## $ product_name_2  <chr> "Total 2% with Strawberry Lowfat Greek Straine...
## $ orders_together <int> 9565, 9565, 8241, 8241, 7444, 7444, 6844, 6844...
## $ order_counts_1  <int> 21405, 30866, 30866, 20552, 19962, 16953, 1889...
## $ order_counts_2  <int> 30866, 21405, 20552, 30866, 16953, 19962, 1996...
## $ total_orders    <int> 837039, 837039, 837039, 837039, 837039, 837039...
```

Looking at the **total_orders** column we see that there were 837k unique orders of yogurt. When looking at **orders_together**, i.e the number of orders in which a given pair occurs, compared to **total_orders** we see that there is a very low chance that a given pair occurs. The first set of pairs that we see are the most ordered of all the pairs and this still equals 9565/837039 or roughly 1.1%. Looking at this value alone would be insufficient. We must dig a little deeper to fully understand the story. This is why we added in the product_id order counts as well as the pair_id order counts.

Next we will calculate the percent of orders for each product_id and pair_id.

```
table_counts_additions <- table_counts_additions %>%
  mutate(product_name_1_pct = 100*orders_together/order_counts_1,
         product_name_2_pct = 100*orders_together/order_counts_2)
head(data.frame(table_counts_additions %>%
       select(product_name_1,product_name_1_pct,product_name_2,product_name_2_pct)),
     10
)
```

```
##                                        product_name_1
## 1   Total 2% Lowfat Greek Strained Yogurt With Blueberry
## 2  Total 2% with Strawberry Lowfat Greek Strained Yogurt
## 3  Total 2% with Strawberry Lowfat Greek Strained Yogurt
## 4       Total 2% Lowfat Greek Strained Yogurt with Peach
## 5          Icelandic Style Skyr Blueberry Non-fat Yogurt
## 6                              Non Fat Raspberry Yogurt
## 7                            Vanilla Skyr Nonfat Yogurt
## 8          Icelandic Style Skyr Blueberry Non-fat Yogurt
```

```
## 9   Total 2% Lowfat Greek Strained Yogurt With Blueberry
## 10      Total 2% Lowfat Greek Strained Yogurt with Peach
##    product_name_1_pct
## 1            44.68582
## 2            30.98879
## 3            26.69928
## 4            40.09829
## 5            37.29085
## 6            43.90963
## 7            36.21356
## 8            34.28514
## 9            30.18454
## 10           31.43733
##                                          product_name_2
## 1  Total 2% with Strawberry Lowfat Greek Strained Yogurt
## 2   Total 2% Lowfat Greek Strained Yogurt With Blueberry
## 3       Total 2% Lowfat Greek Strained Yogurt with Peach
## 4  Total 2% with Strawberry Lowfat Greek Strained Yogurt
## 5                              Non Fat Raspberry Yogurt
## 6          Icelandic Style Skyr Blueberry Non-fat Yogurt
## 7          Icelandic Style Skyr Blueberry Non-fat Yogurt
## 8                            Vanilla Skyr Nonfat Yogurt
## 9       Total 2% Lowfat Greek Strained Yogurt with Peach
## 10  Total 2% Lowfat Greek Strained Yogurt With Blueberry
##    product_name_2_pct
## 1            30.98879
## 2            44.68582
## 3            40.09829
## 4            26.69928
## 5            43.90963
## 6            37.29085
## 7            34.28514
## 8            36.21356
## 9            31.43733
## 10           30.18454
```

The output above is a little messy, but extremely useful. Now we know that out of ALL the times **"Total 2% Lowfat Greek Strained Yogurt With Blueberry"** was bought (**21,405 times**) it was **paired with "Total 2% with Strawberry Lowfat Greek Strained Yogurt" 9,565 times**. This tells us that **the pairing occured 44.7% of the time given that the Blueberry was purchased**.

Likewise we can reverse this information. Out of ALL the times **"Total 2% with Strawberry Lowfat Greek Strained Yogurt"** was bought (**30,866 times** in all) it was **paired with "Total 2% Lowfat Greek Strained Yogurt With Blueberry" 9,565 times**. This tells us that **the pairing occurred 30.99% of the time given that the Strawberry was purchased**.

Taking this a step further was much more insightful than stopping at the pairing counts divided by total number of orders. Now we have decisions that can be made. Products can be placed in closer proximity or coupons/digital deals can be created to increase the customer's chance of purchasing a pair of items and therefore, in return, further increasing basket size.