Gradient Boosting Machine

Shivangi Vashi

6/25/2020

Association Rules Mining using Gradient Boosting

Loading data

```
source_code_filepath<-"readandsampledata.R"
cat("Loading file ", source_code_filepath, "\n")
```

```
## Loading file  readandsampledata.R
```

```
source(file=source_code_filepath)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.3.2     ✓ purrr   0.3.4
## ✓ tibble  3.0.1     ✓ dplyr   1.0.0
## ✓ tidyr   1.1.0     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.5.0
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::arrange()   masks plyr::arrange()
## x dplyr::between()   masks data.table::between()
## x purrr::compact()   masks plyr::compact()
## x dplyr::count()     masks plyr::count()
## x dplyr::failwith()  masks plyr::failwith()
## x dplyr::filter()    masks stats::filter()
## x dplyr::first()     masks data.table::first()
## x dplyr::id()        masks plyr::id()
## x dplyr::lag()       masks stats::lag()
## x dplyr::last()      masks data.table::last()
## x dplyr::mutate()    masks plyr::mutate()
## x dplyr::rename()    masks plyr::rename()
## x dplyr::summarise() masks plyr::summarise()
## x dplyr::summarize() masks plyr::summarize()
## x purrr::transpose() masks data.table::transpose()
```

```
## Number of orders (before):  3421083Number of orders (after):  346739
```

```
## Joining, by = "order_id"
```

```
## Joining, by = "order_id"
## Joining, by = "order_id"
## Joining, by = "order_id"
```

```
# aisles<-fread("instacart-market-basket-analysis/aisles.csv")
# departments<-fread("instacart-market-basket-analysis/departments.csv")
# order_products_prior<-fread("instacart-market-basket-analysis/order_products__prior
.csv")
# order_products_train<-fread("instacart-market-basket-analysis/order_products__train
.csv")
# orders<-fread("instacart-market-basket-analysis/orders.csv")
# products<-fread("instacart-market-basket-analysis/products.csv")
```

We recode variables as factors for later use in feature engineering.

```
orders<-orders %>%mutate(order_hour_of_day=as.numeric(order_hour_of_day))
orders$eval_set <-as.factor(orders$eval_set)
aisles$aisle <- as.factor(aisles$aisle)
departments$department <- as.factor(departments$department)
products$product_name <- as.factor(products$product_name)
```

Feature engineering

We perform feature engineering to create features for the xgboost model. The Xgboost is a package that allows you to create gradient boosting model- a number of weak learning decision trees that improve with each iteration.

The following are the features we created:
- Product reorder probability
- Average Cart Position
- User order probability
- user orders, which is the total number of orders a user makes
- user product orders, ie how many products does a user order

```
# xgboost- Gradient Boosting
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
# Creating a product level by combing product, department, aisles
product_level <- merge(x = products, y = aisles, by = "aisle_id")
product_level <- merge(x = product_level, y = departments, by = "department_id")
product_level$department_id <- NULL
product_level$aisle_id <- NULL
product_level <- arrange(product_level, product_id)



# combining order data and prior data
ordered_products <- merge(x = orders, y = order_products_prior, by = "order_id")



# product reorder probability and avg_cart_position of each cart
product_prob<-ordered_products %>%
        arrange(user_id, order_number, product_id) %>%
        group_by(product_id) %>%
        summarise(product_orders = n(), product_reorders = sum(reordered), avg_cart_p
os = mean(add_to_cart_order))
```

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

```
product_prob$reorder_prob <- product_prob$product_reorders/product_prob$product_order
s
product_prob$product_reorders <- NULL

# calculating user buy prob

# calculating user order probability by looking at prior orders
users_prob<-orders %>%
          filter(eval_set == "prior") %>%
          group_by(user_id) %>%
          summarise(user_orders = max(order_number), user_period = sum(days_since_pri
or_order, na.rm = TRUE), avg_days_since_prior = mean(days_since_prior_order, na.rm =
TRUE))
```

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

```
# calculating total_products, reorder probability and num of products
users_reorder_prob <- ordered_products %>%
              group_by(user_id) %>%
              summarise(user_total_products = n(), user_reorder_prob = sum(reordered
== 1) / sum(order_number > 1),num_products = n_distinct(product_id))
```

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

```
# merging above two to get user level
users_prob<- merge(x = users_prob, y = users_reorder_prob, by = "user_id", all.x = TR
UE)
```

Dividing the data into train and test

The orders table has a flag eval_set that indicates whether the data is for training or testing. So we divide the data using this flag into train and test. The test data has no information about whether the product was reordered or not, so we use this to test our data on.

```
# filtering training and testing data from orders
train_test <- orders %>% filter(eval_set != "prior") %>% select(user_id, order_id, ev
al_set, days_since_prior_order)

# left join users_prob with train_test data
users_prob <- merge(x = users_prob, y = train_test, all.x = TRUE)




# calculating average cart position and total orders of each product user purchased
user_product_cart <- ordered_products %>% group_by(user_id, product_id) %>%
  summarise( user_product_orders = n(), avg_user_product_pos = mean(add_to_cart_order
))
```

```
## `summarise()` regrouping output by 'user_id' (override with `.groups` argument)
```

```
# now combining all the user_level and product _level info
user_product_cart <- merge(user_product_cart,product_prob, by = "product_id", all.x =
TRUE)
user_product_cart <- merge(user_product_cart,users_prob, by = "user_id", all.x = TRUE
)


#taking only user_ids that are common in orders
order_products_train$user_id <- orders$user_id[match(order_products_train$order_id, o
rders$order_id)]
# combining training data and infered data by product id and user id
order_products_train <- order_products_train %>% select(user_id, product_id, reordere
d)
user_product_cart <- merge(user_product_cart, order_products_train, by = c("user_id",
"product_id"), all.x = TRUE)



# training data
train <- as.data.frame(user_product_cart[user_product_cart$eval_set == "train",])
# removing char - xgboost
train$eval_set <- NULL
#no need
train$user_id <- NULL
train$product_id <- NULL
train$order_id <- NULL
train$reordered[is.na(train$reordered)] <- 0
```

```
# testing data
test <- as.data.frame(user_product_cart[user_product_cart$eval_set == "test",])
test$eval_set <- NULL
test$user_id <- NULL
test$reordered <- NULL



#we got our training and testing data
# we have taken only numeric data because we will be using xgboost



# Parameters for the xgboost model
params <- list(
  # logistic model
  "objective"           = "reg:logistic",

  # logless for cross-validation
  "eval_metric"         = "logloss",

  #learning rate
  "eta"                 = 0.1,

  #depth of tree
  "max_depth"           = 6,

  # min sum of weights
  # should be high enough to prevent over fitting
  # but not too high for over fitting
  "min_child_weight"    = 10,

  # the min loss value require to split
  "gamma"               = 0.70,

  # fraction of observations to be included in each tree
  # generally varies from 0.5-1
  "subsample"           = 0.75,

  # fraction of column to be randomly sample in each tree
  "colsample_bytree"    = 0.95,

  # regularization coefficients
  "alpha"               = 2e-05,
  "lambda"              = 10
)
```

XGBoost Model

We create the xgboost model using xgboost(), iterated 80 times. The data is restructured a xgb.DMatrix and the parameters are given as a list.

We test our model and get an error of 0.1185, which is very low.

Finally, this model can be used for predicting product reorders or associations, which are found by:
- filtering by only those products that were reordered
- grouping them by order id
- summarising all products associated with that order id.

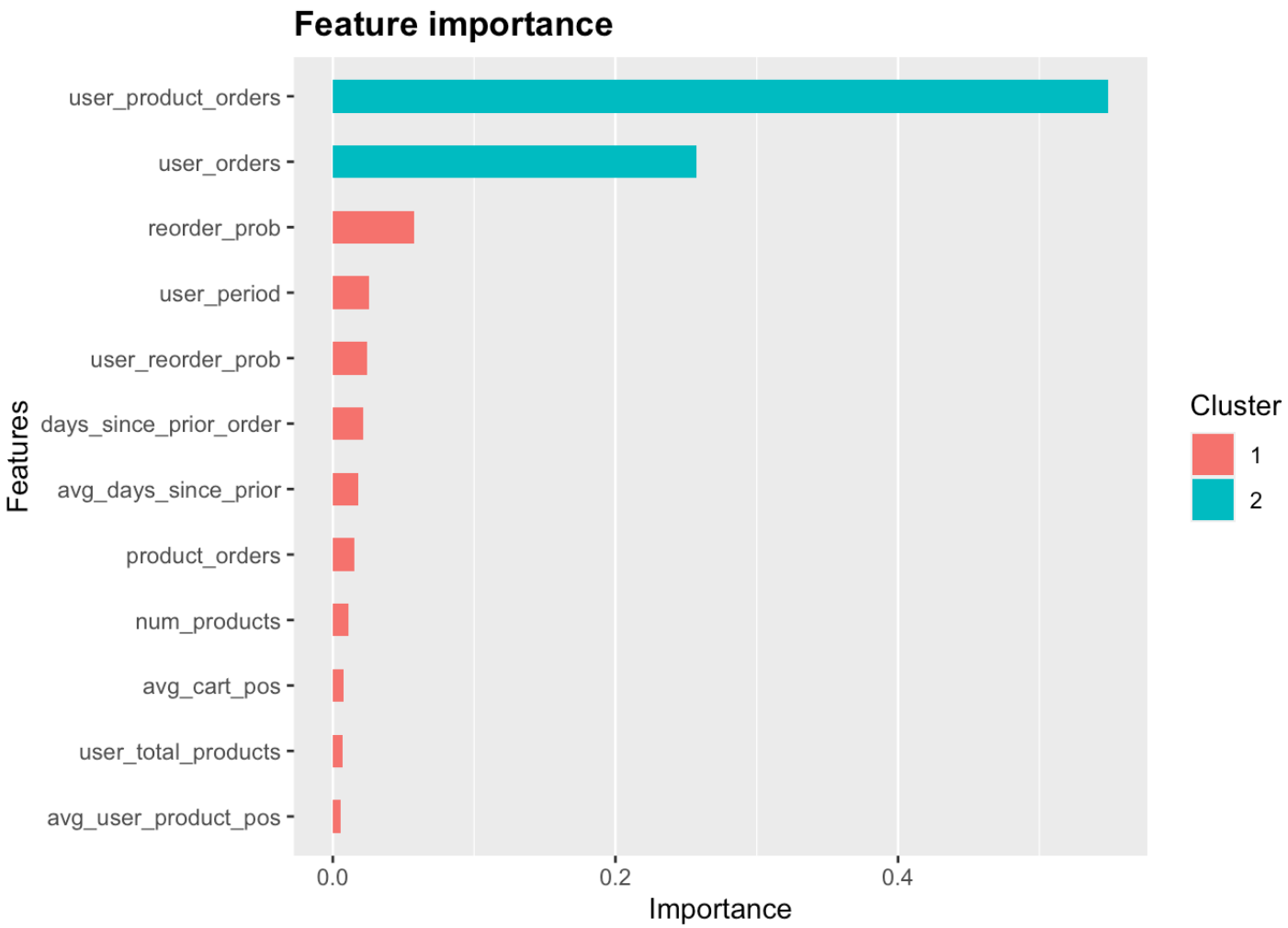
```
X <- xgb.DMatrix(as.matrix(train %>% select(-reordered)), label = train$reordered)
model <- xgboost(data = X, params = params, nrounds = 80)
```

```
## [1]  train-logloss:0.625153
## [2]  train-logloss:0.572386
## [3]  train-logloss:0.528003
## [4]  train-logloss:0.489470
## [5]  train-logloss:0.458636
## [6]  train-logloss:0.430934
## [7]  train-logloss:0.407880
## [8]  train-logloss:0.387902
## [9]  train-logloss:0.370704
## [10] train-logloss:0.356107
## [11] train-logloss:0.343744
## [12] train-logloss:0.332836
## [13] train-logloss:0.323072
## [14] train-logloss:0.314583
## [15] train-logloss:0.307188
## [16] train-logloss:0.301143
## [17] train-logloss:0.295876
## [18] train-logloss:0.290702
## [19] train-logloss:0.287032
## [20] train-logloss:0.282995
## [21] train-logloss:0.280224
## [22] train-logloss:0.277313
## [23] train-logloss:0.274970
## [24] train-logloss:0.273138
## [25] train-logloss:0.271178
## [26] train-logloss:0.269576
## [27] train-logloss:0.268391
## [28] train-logloss:0.267344
## [29] train-logloss:0.266278
## [30] train-logloss:0.265219
## [31] train-logloss:0.264434
```

```
## [32] train-logloss:0.263813
## [33] train-logloss:0.263279
## [34] train-logloss:0.262831
## [35] train-logloss:0.262406
## [36] train-logloss:0.262020
## [37] train-logloss:0.261637
## [38] train-logloss:0.261259
## [39] train-logloss:0.260949
## [40] train-logloss:0.260664
## [41] train-logloss:0.260423
## [42] train-logloss:0.260209
## [43] train-logloss:0.260058
## [44] train-logloss:0.259913
## [45] train-logloss:0.259787
## [46] train-logloss:0.259645
## [47] train-logloss:0.259523
## [48] train-logloss:0.259416
## [49] train-logloss:0.259334
## [50] train-logloss:0.259216
## [51] train-logloss:0.259113
## [52] train-logloss:0.259032
## [53] train-logloss:0.258964
## [54] train-logloss:0.258893
## [55] train-logloss:0.258803
## [56] train-logloss:0.258741
## [57] train-logloss:0.258680
## [58] train-logloss:0.258609
## [59] train-logloss:0.258533
## [60] train-logloss:0.258483
## [61] train-logloss:0.258402
## [62] train-logloss:0.258313
## [63] train-logloss:0.258234
## [64] train-logloss:0.258166
## [65] train-logloss:0.258103
## [66] train-logloss:0.258056
## [67] train-logloss:0.258004
## [68] train-logloss:0.257967
## [69] train-logloss:0.257920
## [70] train-logloss:0.257877
## [71] train-logloss:0.257817
## [72] train-logloss:0.257770
## [73] train-logloss:0.257735
## [74] train-logloss:0.257669
## [75] train-logloss:0.257625
## [76] train-logloss:0.257586
## [77] train-logloss:0.257521
## [78] train-logloss:0.257488
```

```
## [79] train-logloss:0.257463
## [80] train-logloss:0.257432
```

```
importance <- xgb.importance(colnames(X), model = model)
xgb.ggplot.importance(importance)
```




```
X2 <- xgb.DMatrix(as.matrix(test %>% select(-order_id, -product_id)))
# predicting reordered values from test dataset
test$reordered <- predict(model, X2)



#Test error
p<-test$reordered
err <- mean(as.numeric(p > 0.5) != train$reordered)
```

```
## Warning in as.numeric(p > 0.5) != train$reordered: longer object length is not a
## multiple of shorter object length
```

```
print(paste("test-error=", err))
```

```
## [1] "test-error= 0.113578285102201"
```

As we can see from this plot, the user_product_orders and user_orders features are the most important in predicting whether a product will be reordered or not.

Generating Association Rules using Gradient Boosting Model

We finally use this gradient boosting model to generate association rules within the test dataset.

```
test$reordered <- (test$reordered > 0.21) * 1


# summarise as order_id and products
rules <- test %>% filter(reordered == 1) %>% group_by(order_id) %>%
        summarise(products = paste(product_id, collapse = " "))
```

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

```
# filling the missing values
missing <- data.frame(
  order_id = unique(test$order_id[!test$order_id %in% rules$order_id]), products = "N
one")
rules <- rules %>% bind_rows(missing) %>% arrange(order_id)

head(rules,20)
```

```
## # A tibble: 20 x 2
##    order_id products
##       <int> <chr>
##  1     1195 2707 13870 47626
##  2     2070 25 432 5535 9076 19986 24852 30391 39322 47144
##  3     2247 9076 11422 13176 14272 18234 19125 19677 31717 49235
##  4     2267 2054 4595 38569 46088
##  5     2721 21137 21903 24852 27086 28204 33129 47626
##  6     4458 7781 9076 14999 16611 16965 23029 25976 33065 36425 36459
##  7     4602 329 21903 27156 29307 29950 47766
##  8     4686 24852 35948 37803 47626
##  9     5349 15777 36431
## 10     5918 9839 18465 38383 40174 45066 49683
## 11     6100 196 5161 12427 29794 46149
## 12     6275 16797 16874 24852 34993 37592 38672
## 13     6506 47209
## 14     6718 12341 17461 24852 26096 31371 45763
## 15     7736 7644
## 16     8156 3243 3583 3765 4210 24852 26369 41787
## 17     8240 9036 12606 21137 25931 47209 48679
## 18     8418 28985
## 19    10840 2295 8424 11481 15592 21137 21938 25890 26497 29487 32747 39947 477…
## 20    11556 5428 26165
```

```
rulesgbm <- test%>%inner_join(products)%>%
            filter(reordered == 1) %>% group_by(order_id) %>%
            summarise(rules = paste(product_name, collapse = ","))
```

```
## Joining, by = "product_id"
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
head(rulesgbm,20)
```

```
## # A tibble: 20 x 2
##    order_id rules
##       <int> <chr>
##  1     1195 Complete™ ActionPacs™ Fresh Scent Dishwasher Detergent,Lightly Salt…
##  2     2070 Salted Caramel Lean Protein & Fiber Bar,Vanilla Almond Breeze Almon…
##  3     2247 Blueberries,Plain Greek Yogurt,Bag of Organic Bananas,Toasted Cocon…
##  4     2267 Belgium Beer,Italian Dry Salame,Old Vine Zinfandel Wine,Beer
##  5     2721 Organic Strawberries,Organic Baby Spinach,Banana,Half & Half,Organi…
##  6     4458 Organic Sticks Low Moisture Part Skim Mozzarella String Cheese,Blue…
##  7     4602 Organic Whole Grassmilk Milk,Organic Baby Spinach,Organic Black Bea…
##  8     4686 Banana,Tonic Water,Oyster Mushrooms,Large Lemon
##  9     5349 Prune Juice,Hardwood Smoked Bacon
## 10     5918 Organic Broccoli,Organic Grade A Free Range Large Brown Eggs,Organi…
## 11     6100 Soda,Dried Mango,Original Beef Jerky,Iced Coffee,Zero Calorie Cola
## 12     6275 Strawberries,Chocolate Fudge Brownie Ice Cream,Banana,Mango Yoghurt…
## 13     6506 Organic Hass Avocado
## 14     6718 Hass Avocados,Air Chilled Organic Boneless Skinless Chicken Breasts…
## 15     7736 Scoops! Tortilla Chips
## 16     8156 Red Onions,Unsweetened Coconut Milk Beverage,Original Whole Fat Lac…
## 17     8240 Unsweetened Vanilla Cashew Milk,100% Natural Spring Water,Organic S…
## 18     8418 Michigan Organic Kale
## 19    10840 Yellow Bell Pepper,Broccoli Crown,Organic Quinoa & Brown Rice With …
## 20    11556 1 Liter,Electrolyte Enhanced Water
```

Conclusion

- We explored the data to analyze buying patterns and popular products
- We performed logistic regression to predict whether a product will be reordered, with 0.53% accuracy.
- We implemented apriori algorithm for associations rule mining to generate strong rules that dictate what products are most likely to be bought together
- We use the gradient boosting model to create a highly accurate model that predicts whether a product will be reordered, and we use this model to further generate association rules in the test dataori algorithm for associations rule mining to generate strong rules that dictate what products are most likely to be bought together
- We use the gradient boosting model to create a highly accurate model that predicts whether a product will be reordered, and we use this model to further generate association rules in the test data