DMC@ISU: Iowa State University Data Mining Cup Team 2015

Initial Exploration

Spring 2015, Iowa State

Due Date: April 14 2015

I am using the following packages:

```
library(ggplot2)
library(lubridate)
library(xtable)
library(foreach)
library(rCharts)
library(plyr)
library(dplyr)
options(dplyr.width = Inf)
library(reshape2)
library(gtools)
library(sqldf)
```

and my working directory is set to `dmc2015/ian`.

0.1 Reading the Data

Read the data into R:

```
# training set ('historical data')
trn <- read.delim("../data/raw_data/DMC_2015_orders_train.txt",
    stringsAsFactors = FALSE, sep = "|", quote = "")
trn.raw <- trn

# test set ('future data')
tst <- read.delim("../data/raw_data/DMC_2015_orders_class.txt",
    stringsAsFactors = FALSE, sep = "|", quote = "")
tst.raw <- tst
```

0.2 Stacking Coupons

The data is formatted in a poor way to examine coupon level behavior (the coupon1, coupon2, coupon3 structure). We can alter this fairly simply:

```
source("~/dmc2015/ian/R/stackCoupons.R")
d.stack <- stackCoupons(trn, tst)

## using the following as id:
## orderID,
## orderTime,
## userID,
## couponsReceived,
## basketValue
```

```
##
## using the following as measure columns:
## couponID1,
## price1,
## basePrice1,
## reward1,
## premiumProduct1,
## brand1,
## productGroup1,
## categoryIDs1,
## coupon1Used,
## couponID2,
## price2,
## basePrice2,
## reward2,
## premiumProduct2,
## brand2,
## productGroup2,
## categoryIDs2,
## coupon2Used,
## couponID3,
## price3,
## basePrice3,
## reward3,
## premiumProduct3,
## brand3,
## productGroup3,
## categoryIDs3,
## coupon3Used
```

0.2.1 Relabeling coupons

The following function renames the ID variables in a data set.

- It does so in a way that provides a 1-1 map between labels.
- Preserves the order of the labels either as they appear in the dataset **or** with respect to another column in the dataset.
- Can return both the mapping **and** the the relabeled dataset.
- Allows you to choose whether to replace the old ID labels or not.

```
smartLabel <- function(dsn, varn, abrv = "smart.",
    orderby = NULL, replace = FALSE, listout = TRUE) {
    # dsn is the dataset with a 'anonymous' ID variable
    # varn is the label to improve orderby is a
    # variable to sort the new ids by replace = TRUE
    # removes old label = FALSE keeps old label and
    # names new label 'smart.varn' list.out = TRUE
    # returns a list with the data and the map varn to
    # smart.varn = FALSE returns the dataset only
```

```
    if (!is.null(orderby)) {
        orderby_query <- gsub("varn", varn, gsub("orderby",
            paste(orderby, collapse = ", "), "select varn, orderby from dsn order by orderby"))
        dsn_labels <- data.frame(unique(sqldf(orderby_query)[,
            varn]))
        names(dsn_labels) <- varn
        dsn_labels$smart_label <- paste0(abrv, 1:nrow(dsn_labels))
        dsn <- sqldf(gsub("varn", varn, gsub("orderby",
            paste(orderby, collapse = ", "), "select a.*, b.smart_label from dsn as a left join dsn_lab
    } else {
        dsn_labels <- data.frame(unique(dsn[, varn]))
        names(dsn_labels) <- varn
        dsn_labels$smart_label <- paste0(abrv, 1:nrow(dsn_labels))
        dsn <- sqldf(gsub("varn", varn, "select a.*, b.smart_label from dsn as a left join dsn_labels a
    }

    if (replace) {
        dsn[, varn] <- dsn$smart_label
        dsn <- dsn[, -which(names(dsn) == "smart_label")]
    } else {
        names(dsn)[which(names(dsn) == "smart_label")] <- paste0("smart_",
            varn)
    }

    if (listout) {
        message("returning a list")
        ret <- list(data = dsn, mapping = dsn_labels)
    } else {
        ret <- dsn
    }

    return(ret)
}
```

We need consistent labeling between the test set and the training set - this means that we need to run this function on the combined set. Fortunately, we have used the variable `dsn` to specify which set a row in the combined output from `stackCoupons` originated from. Once we have relabeled everything, we can split the sets apart again using:

```
splitByAndDrop <- function(combined.dsn, splitvar = "dsn") {
    train <- combined.dsn[which(combined.dsn[, splitvar] ==
        "train"), ]
    test <- combined.dsn[which(combined.dsn[, splitvar] ==
        "test"), ]
    return(list(train = train, test = test))
}
```

First, lets handle these coupons names:

```
couponIDmap <- smartLabel(d.stack$combined, "couponID",
    abrv = "cpn", orderby = c("orderID", "couponCol",
        "dsn"), listout = TRUE, replace = TRUE)

## Loading required package:  tcltk
## returning a list
```

```
d.stack$combined <- couponIDmap$data
```

And while we're at it, let's fix the other "scrambled" IDs:

```
brandIDmap <- smartLabel(d.stack$combined, "brand",
    abrv = "brand", orderby = c("orderID", "couponCol",
        "dsn"), listout = TRUE, replace = TRUE)

## returning a list

d.stack$combined <- brandIDmap$data

groupIDmap <- smartLabel(d.stack$combined, "productGroup",
    abrv = "prod", orderby = c("orderID", "couponCol",
        "dsn"), listout = TRUE, replace = TRUE)

## returning a list

d.stack$combined <- groupIDmap$data

userIDmap <- smartLabel(d.stack$combined, "userID",
    abrv = "user", orderby = c("orderID", "couponCol",
        "dsn"), listout = TRUE, replace = TRUE)

## returning a list

d.stack$combined <- userIDmap$data
```

Getting the category IDs to be more simple is not insignificant. The following function takes a data set and splits a single column into multiple columns on a given `splitby` value. If columns have different numbers of splits, the additional columns are filled with `NA`s:

```
splitColumn <- function(dsn, varn, orderby, splitby = ",") {
    res_d <- dsn[, c(orderby, varn)] %>% mutate(parts = strsplit(dsn[,
        varn], splitby)) %>% group_by_(varn) %>% do(data.frame({
        idx <- 1:length(.$parts[[1]])
        lst <- lapply(idx, function(x) .$parts[[1]][x])
        names(lst) <- lapply(idx, function(x) paste0(varn,
            x))
        (lst)
    }, stringsAsFactors = FALSE))
    res_d <- sqldf(gsub("varn", varn, gsub("orderby",
        paste(orderby, collapse = ", "), "select a.*, b.* from dsn as a left join res_d as b on a.varn =
    res_d <- res_d[, -(ncol(dsn) + 1)]
    return(res_d)
}
```

```
catID.d <- splitColumn(d.stack$combined, "categoryIDs",
    "orderID")
catID.m <- melt(catID.d, id = 1:ncol(d.stack$combined),
    measure = (ncol(d.stack$combined) + 1):ncol(catID.d))
catID.m$variable <- as.numeric(as.character(gsub("categoryIDs",
    "", catID.m$variable)))
names(catID.m)[(ncol(catID.m) - 1)] <- "categoryEntry"
names(catID.m)[(ncol(catID.m))] <- "catIDs"
```

```
catID.m <- catID.m[which(!is.na(catID.m$catIDs)), ]
catID.m <- catID.m[with(catID.m, order(orderID, couponCol,
    categoryEntry, dsn)), ]

catIDmap <- smartLabel(catID.m, "catIDs", abrv = "cat",
    orderby = c("orderID", "couponCol", "categoryEntry",
        "dsn"), listout = TRUE, replace = TRUE)

## returning a list

rd <- catIDmap$data %>% group_by(orderID, couponID) %>%
    summarise(catIDs = paste(catIDs, collapse = ":"))

ld <- d.stack$combined
res_d <- sqldf("select a.*, b.catIDs from ld as a left join rd as b on a.orderID = b.orderId and a.coupo

res_d$categoryIDs <- res_d$catIDs
d.stack$combined <- res_d %>% select(-catIDs)
```

Now we can split this into the original test and training sets:

```
train.test <- splitByAndDrop(d.stack$combined)

train <- train.test$train[, -which(names(train.test$train) ==
    "dsn")]
test <- train.test$test[, -which(names(train.test$test) ==
    "dsn")]

head(train)

##   orderID           orderTime userID
## 1       1 2015-01-06 09:38:35  user1
## 2       1 2015-01-06 09:38:35  user1
## 3       1 2015-01-06 09:38:35  user1
## 4       2 2015-01-06 10:03:19  user2
## 5       2 2015-01-06 10:03:19  user2
## 6       2 2015-01-06 10:03:19  user2
##       couponsReceived basketValue couponID price
## 1 2015-01-06 09:34:53      187.60     cpn1  3.24
## 2 2015-01-06 09:34:53      187.60     cpn2  5.19
## 3 2015-01-06 09:34:53      187.60     cpn3 12.92
## 4 2015-01-06 10:00:44      185.93     cpn4  2.32
## 5 2015-01-06 10:00:44      185.93     cpn5  3.70
## 6 2015-01-06 10:00:44      185.93     cpn6  3.89
##   basePrice reward premiumProduct  brand
## 1      5.40   1.57              0 brand1
## 2      0.57   1.57              0 brand2
## 3     12.92   2.20              0 brand3
## 4      1.59   1.57              0 brand2
## 5      1.85   0.94              0 brand3
## 6      0.06   2.20              0 brand2
##   productGroup    categoryIDs couponUsed
## 1        prod1      cat1:cat2          0
## 2        prod2      cat3:cat4          1
```

```
## 3        prod3 cat5:cat4:cat2          0
## 4        prod4           cat5          1
## 5        prod5           cat6          0
## 6        prod6           cat6          1
##   couponCol
## 1         1
## 2         2
## 3         3
## 4         1
## 5         2
## 6         3
```

```
head(test)
```

```
##       orderID           orderTime  userID
## 18160    6054 2015-03-10 08:09:55   user8
## 18161    6054 2015-03-10 08:09:55   user8
## 18162    6054 2015-03-10 08:09:55   user8
## 18163    6055 2015-03-10 10:03:15 user769
## 18164    6055 2015-03-10 10:03:15 user769
## 18165    6055 2015-03-10 10:03:15 user769
##           couponsReceived basketValue couponID
## 18160 2015-03-10 08:03:09          NA    cpn22
## 18161 2015-03-10 08:03:09          NA    cpn23
## 18162 2015-03-10 08:03:09          NA    cpn24
## 18163 2015-03-10 09:07:00          NA   cpn158
## 18164 2015-03-10 09:07:00          NA    cpn22
## 18165 2015-03-10 09:07:00          NA   cpn585
##       price basePrice reward premiumProduct
## 18160  9.17      2.04   0.94              1
## 18161  4.82      0.60   1.57              0
## 18162  6.21      1.24   1.57              1
## 18163  2.69      2.69   1.57              1
## 18164  9.17      2.04   0.94              1
## 18165  9.22      2.56   1.57              0
##        brand productGroup    categoryIDs
## 18160 brand4        prod7      cat1:cat7
## 18161 brand3       prod11           cat3
## 18162 brand4       prod19           cat5
## 18163 brand4       prod75 cat5:cat7:cat4
## 18164 brand4        prod7      cat1:cat7
## 18165 brand3       prod28           cat3
##       couponUsed couponCol
## 18160         NA         1
## 18161         NA         2
## 18162         NA         3
## 18163         NA         3
## 18164         NA         1
## 18165         NA         2
```

We can save these melted version of the training and test sets:

```
write.csv(train, file = "~/dmc2015/data/melted_train_simple_name.csv",
    row.names = FALSE, na = "", quote = FALSE)
```

```
write.csv(test, file = "~/dmc2015/data/melted_test_simple_name.csv",
    row.names = FALSE, na = "", quote = FALSE)
```

Now we can reformat this data so that it resembles the original data:

```
train_backbone <- unique(train[, which(names(train) %in%
    names(trn.raw))])

coupon1cols <- train[which(train$couponCol == 1), ]
coupon1cols <- coupon1cols[, -which(names(coupon1cols) ==
    "couponCol")]
names(coupon1cols)[which(!(names(coupon1cols) %in%
    names(trn.raw)))] <- names(trn.raw)[grepl("1",
    names(trn.raw))]
train_backbone <- sqldf(gsub("bcols", paste("b.", names(trn.raw)[grepl("1",
    names(trn.raw))], collapse = ", ", sep = ""), "select a.*, bcols from train_backbone as a left join

coupon2cols <- train[which(train$couponCol == 2), ]
coupon2cols <- coupon2cols[, -which(names(coupon2cols) ==
    "couponCol")]
names(coupon2cols)[which(!(names(coupon2cols) %in%
    names(trn.raw)))] <- names(trn.raw)[grepl("2",
    names(trn.raw))]
train_backbone <- sqldf(gsub("bcols", paste("b.", names(trn.raw)[grepl("2",
    names(trn.raw))], collapse = ", ", sep = ""), "select a.*, bcols from train_backbone as a left join

coupon3cols <- train[which(train$couponCol == 3), ]
coupon3cols <- coupon3cols[, -which(names(coupon3cols) ==
    "couponCol")]
names(coupon3cols)[which(!(names(coupon3cols) %in%
    names(trn.raw)))] <- names(trn.raw)[grepl("3",
    names(trn.raw))]
train_backbone <- sqldf(gsub("bcols", paste("b.", names(trn.raw)[grepl("3",
    names(trn.raw))], collapse = ", ", sep = ""), "select a.*, bcols from train_backbone as a left join

test_backbone <- unique(test[, which(names(test) %in%
    names(trn.raw))])

coupon1cols <- test[which(test$couponCol == 1), ]
coupon1cols <- coupon1cols[, -which(names(coupon1cols) ==
    "couponCol")]
names(coupon1cols)[which(!(names(coupon1cols) %in%
    names(trn.raw)))] <- names(trn.raw)[grepl("1",
    names(trn.raw))]
test_backbone <- sqldf(gsub("bcols", paste("b.", names(trn.raw)[grepl("1",
    names(trn.raw))], collapse = ", ", sep = ""), "select a.*, bcols from test_backbone as a left join

coupon2cols <- test[which(test$couponCol == 2), ]
coupon2cols <- coupon2cols[, -which(names(coupon2cols) ==
    "couponCol")]
names(coupon2cols)[which(!(names(coupon2cols) %in%
    names(trn.raw)))] <- names(trn.raw)[grepl("2",
    names(trn.raw))]
test_backbone <- sqldf(gsub("bcols", paste("b.", names(trn.raw)[grepl("2",
```

```
    names(trn.raw))], collapse = ", ", sep = ""), "select a.*, bcols from test_backbone as a left join
coupon3cols <- test[which(test$couponCol == 3), ]
coupon3cols <- coupon3cols[, -which(names(coupon3cols) ==
    "couponCol")]
names(coupon3cols)[which(!(names(coupon3cols) %in%
    names(trn.raw)))] <- names(trn.raw)[grepl("3",
    names(trn.raw))]
test_backbone <- sqldf(gsub("bcols", paste("b.", names(trn.raw)[grepl("3",
    names(trn.raw))], collapse = ", ", sep = ""), "select a.*, bcols from test_backbone as a left join
```

and save them

```
write.csv(train_backbone, file = "~/dmc2015/data/train_simple_name.csv",
    row.names = FALSE, na = "", quote = FALSE)
write.csv(test_backbone, file = "~/dmc2015/data/test_simple_name.csv",
    row.names = FALSE, na = "", quote = FALSE)
```

We can also save each map:

```
write.csv(couponIDmap$mapping, file = "~/dmc2015/ian/written_data/couponIDmap.csv",
    row.names = FALSE, na = "", quote = FALSE)
write.csv(brandIDmap$mapping, file = "~/dmc2015/ian/written_data/brandIDmap.csv",
    row.names = FALSE, na = "", quote = FALSE)
write.csv(groupIDmap$mapping, file = "~/dmc2015/ian/written_data/groupIDmap.csv",
    row.names = FALSE, na = "", quote = FALSE)
write.csv(userIDmap$mapping, file = "~/dmc2015/ian/written_data/userIDmap.csv",
    row.names = FALSE, na = "", quote = FALSE)
write.csv(catIDmap$mapping, file = "~/dmc2015/ian/written_data/catIDmap.csv",
    row.names = FALSE, na = "", quote = FALSE)
```