Capstone

# 1. Introduction

As stated in the CITI data science challenge[[1]](#footnote-22)

“One of the core assets of a Credit Card industry is the information rich data that is available for millions of its customers! And one such data source is transaction data. Each time a customer uses their card for purchase, a new transaction record is created. On average approximately 120 annual records are created for each active customer. If mined right, this data has the potential to yield incredible insights into customer preferences and lifestyle choices. However, we are faced with one fundamental challenge while using this data, namely the sheer size of the data and the processing power required, in order to mine value from it. This data was not optimally leveraged in the past; however, with better platforms and techniques becoming available, card issuers are increasingly turning towards mining this dataset.”

### 1.1 Objective

This data science challenge sought participants to perform the following:

* Identify the Top 10 merchants that customers have not transacted with in the past 12 months and are likely to transact with, in the next 3 months
* Recommendations must be personalized at a customer level
* Using historical data for the particular customer and the rest of the population, identify merchants the customer is likely to interact with

# 2 Approach

This data challenge is primarily a recommendation problem, where individual and group data is leveraged to make user-specific recommendations. One of the approaches to this type of problem is collaborative filtering[[2]](#footnote-26), where sparse data from many users can help inform recommendations. While there are many ways to approach this problem, it is useful to compare the performance of various algorithms and models to see which are the best for the dataset.

Recommenderlab[[3]](#footnote-28) is an R software package which allows for the creation, implementation and evaluation of multiple topN recommender algorithms. This package has an in-built registry of some of the most popular algorithms for solving recommender problems including:

* Alternate Least Squares (ALS);
* Item Based Collaborative Filtering (IBCF);
* Popularity;
* Random;
* Re-recommend;
* Single Value Decomposition (SVD); and
* User Based Collaborative Filtering (UBCF)

By creating a scheme in which all of these different approaches are used and compared, it is possible to find the best in-built algorithm for making recommendations with the given data.

# 3. Implementation

### 3.1 Importing and formatting data

The first step is to read in the data and check its structure.

#### Import CSV as “WorkData”

WorkData <- read.csv("C:/Users/M/Desktop/R/Capstone/Capstone Credit Card/DataScienceChallenge\_Data/DataScienceChallenge\_Training.csv")  
str(WorkData)

## 'data.frame': 18532355 obs. of 5 variables:  
## $ Cust\_map : num 1 1 1 1 1 1 1 1 1 1 ...  
## $ Merch\_Map\_final: int 1 1 1 1 1 1 1 1 1 1 ...  
## $ TXN\_MTH : int 201309 201310 201311 201312 201401 201402 201403 201404 201405 201406 ...  
## $ SPND\_CATGY : Factor w/ 15 levels "BUSINESS SERVICES",..: 6 6 6 6 6 6 6 6 6 6 ...  
## $ NumTrans : int 2 2 3 2 2 3 2 3 3 2 ...

The file consists of 18,532,355 observations of five variables. These variables include a customer id, a merchant id, a month of transaction, a category of the merchant, and the number of transactions occurring between a customer and merchant in the month.

Next, it is necessary to load the libraries to analyze the data.

#### Open required libraries and set seed

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(zoo)

## Warning: package 'zoo' was built under R version 3.4.4

##   
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':  
##   
## as.Date, as.Date.numeric

library(reshape2)

## Warning: package 'reshape2' was built under R version 3.4.4

library(arules)

## Warning: package 'arules' was built under R version 3.4.4

## Loading required package: Matrix

##   
## Attaching package: 'arules'

## The following object is masked from 'package:dplyr':  
##   
## recode

## The following objects are masked from 'package:base':  
##   
## abbreviate, write

library(arulesViz)

## Warning: package 'arulesViz' was built under R version 3.4.4

## Loading required package: grid

library(recommenderlab)

## Warning: package 'recommenderlab' was built under R version 3.4.4

## Loading required package: proxy

## Warning: package 'proxy' was built under R version 3.4.4

##   
## Attaching package: 'proxy'

## The following object is masked from 'package:Matrix':  
##   
## as.matrix

## The following objects are masked from 'package:stats':  
##   
## as.dist, dist

## The following object is masked from 'package:base':  
##   
## as.matrix

## Loading required package: registry

set.seed(1988)

The date infomration is not in the date class, and the columns could be better named.

#### Rename columns and convert date to date format

names(WorkData) <- c("customer\_id", "merchant\_id", "date", "category", "number\_trans")  
WorkData$date <- as.Date(as.yearmon(as.character(WorkData$date), "%Y%m"))  
WorkData <- arrange(WorkData, customer\_id, merchant\_id, date)  
str(WorkData)

## 'data.frame': 18532355 obs. of 5 variables:  
## $ customer\_id : num 1 1 1 1 1 1 1 1 1 1 ...  
## $ merchant\_id : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ date : Date, format: "2013-09-01" "2013-10-01" ...  
## $ category : Factor w/ 15 levels "BUSINESS SERVICES",..: 6 6 6 6 6 6 6 6 6 6 ...  
## $ number\_trans: int 2 2 3 2 2 3 2 3 3 2 ...

summary(WorkData)

## customer\_id merchant\_id date   
## Min. : 1 Min. : 1 Min. :2013-09-01   
## 1st Qu.: 89529 1st Qu.: 24 1st Qu.:2013-12-01   
## Median :178934 Median : 174 Median :2014-03-01   
## Mean :181226 Mean :1104 Mean :2014-02-18   
## 3rd Qu.:274220 3rd Qu.:1191 3rd Qu.:2014-06-01   
## Max. :374334 Max. :9822 Max. :2014-08-01   
##   
## category number\_trans   
## F&B :5176100 Min. : 2.000   
## HOUSEHOLD & UTILITIES :3029456 1st Qu.: 2.000   
## CLOTHING & ACCESSORIES:2268674 Median : 3.000   
## RETAIL :1558180 Mean : 3.435   
## GROCERY STORES :1347928 3rd Qu.: 3.000   
## RECREATION : 992189 Max. :4344.000   
## (Other) :4159828

sum(WorkData$number\_trans)

## [1] 63659708

### 3.2 Casting the data for recommender lab

The first step in using recommenderlab is converting the data into a rating matrix, of n customers by m merchants, and the number of transactions as the data intersections. In the case of the dataset involved in the challenge, this would entail creating a 374,334 x 9,822 matrix, or a total of 3,676,708,548 cells. This casting exceeded the capabilites of the present system.

Even attempting to sample 1% of the data exceeded the system’s abilities.

### 3.3 Casting the data for recommender lab using a subset of data

In order to continue with the project and explore how recommenderlab might be implemented, it was necessary to proceed with a much smaller subset of data, in this case the first 100 customer IDs. Customer IDs were used as the criteria to capture all the interactions a given customer had with merchants, as opposed to other stratification methods, which may have omitted some of this data.

#### Create working data set of first 100 customer IDs “n”

n <- filter(WorkData, customer\_id <= 100)  
str(n)

## 'data.frame': 5287 obs. of 5 variables:  
## $ customer\_id : num 1 1 1 1 1 1 1 1 1 1 ...  
## $ merchant\_id : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ date : Date, format: "2013-09-01" "2013-10-01" ...  
## $ category : Factor w/ 15 levels "BUSINESS SERVICES",..: 6 6 6 6 6 6 6 6 6 6 ...  
## $ number\_trans: int 2 2 3 2 2 3 2 3 3 2 ...

summary(n)

## customer\_id merchant\_id date   
## Min. : 1.0 Min. : 1 Min. :2013-09-01   
## 1st Qu.: 19.0 1st Qu.: 24 1st Qu.:2013-11-01   
## Median : 45.0 Median : 165 Median :2014-02-01   
## Mean : 45.9 Mean :1039 Mean :2014-02-13   
## 3rd Qu.: 70.0 3rd Qu.:1096 3rd Qu.:2014-06-01   
## Max. :100.0 Max. :9822 Max. :2014-08-01   
##   
## category number\_trans   
## F&B :1343 Min. : 2.000   
## HOUSEHOLD & UTILITIES : 898 1st Qu.: 2.000   
## CLOTHING & ACCESSORIES: 669 Median : 3.000   
## RETAIL : 419 Mean : 3.241   
## GROCERY STORES : 395 3rd Qu.: 3.000   
## RECREATION : 344 Max. :44.000   
## (Other) :1219

length(unique(n$merchant\_id))

## [1] 1549

sum(n$number\_trans)

## [1] 17135

Filtering the data to the first 100 customer IDs resulted in 100 customers, 1,549 merchants, and 17,135 interactions. This results in a much more managable 154,900 cells, though at the loss of including much of the data.

#### Cast n into customer by merchant matrix with sum of transactions for recommenderlab

Recommenderlab provides:

“[…] two concrete implementations realRatingMatrix and binaryRatingMatrix to represent different types of rating matrices R. realRatingMatrix implements a rating matrix with real valued ratings stored in sparse format defined in package Matrix. Sparse matrices in Matrix typically do not store 0s explicitly, however for realRatingMatrix we use these sparse matrices such that instead of 0s, NAs are not explicitly stored.”

Therefore, it was neccessary to cast the data in the proper format, and also convert all 0s to NAs.

n$number\_trans <- as.numeric(n$number\_trans)  
n\_cast <- (dcast(n, customer\_id ~ merchant\_id, sum))

## Using number\_trans as value column: use value.var to override.

n\_cast <- as.data.frame(n\_cast)   
  
  
row.names(n\_cast) <- paste("c",n\_cast[,1], sep = "")  
n\_cast[,1] <- NULL  
colnames(n\_cast) <- paste("m", colnames(n\_cast), sep = "")  
n\_cast[n\_cast == 0] <- NA  
print(n\_cast[1:10, 1:10])

## m1 m2 m3 m4 m5 m6 m7 m8 m9 m10  
## c1 29 85 33 NA 42 NA NA NA 138 NA  
## c2 NA NA NA NA NA 31 NA NA 3 NA  
## c3 NA NA NA NA NA NA NA NA NA NA  
## c4 27 NA 31 NA NA NA NA NA 20 NA  
## c5 11 NA NA 5 NA 27 NA NA NA NA  
## c6 NA NA NA NA NA NA NA 65 NA NA  
## c7 53 28 NA NA 79 NA NA NA NA 3  
## c8 NA NA NA 5 NA NA NA NA NA NA  
## c9 NA NA NA NA NA NA NA NA NA NA  
## c10 33 5 NA 5 NA NA NA NA NA NA

#### Convert n\_cast into a matrix of class matrix

It was found to be essential to convert n\_cast into class matrix before proceeding with the next steps.

n\_cast <- as.matrix(n\_cast)  
str(n\_cast)

## num [1:100, 1:1549] 29 NA NA 27 11 NA 53 NA NA 33 ...  
## - attr(\*, "dimnames")=List of 2  
## ..$ : chr [1:100] "c1" "c2" "c3" "c4" ...  
## ..$ : chr [1:1549] "m1" "m2" "m3" "m4" ...

Once the data is in the proper format, it can be converted to a sparse “realRatingMatrix” matrix for recommender lab.

#### Convert n\_cast into a sparse matrix for recommenderlab

n\_sparse <- as(n\_cast, "realRatingMatrix")  
str(n\_sparse)

## Formal class 'realRatingMatrix' [package "recommenderlab"] with 2 slots  
## ..@ data :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots  
## .. .. ..@ i : int [1:2806] 0 3 4 6 9 10 15 17 18 23 ...  
## .. .. ..@ p : int [1:1550] 0 22 50 62 80 103 116 124 129 145 ...  
## .. .. ..@ Dim : int [1:2] 100 1549  
## .. .. ..@ Dimnames:List of 2  
## .. .. .. ..$ : chr [1:100] "c1" "c2" "c3" "c4" ...  
## .. .. .. ..$ : chr [1:1549] "m1" "m2" "m3" "m4" ...  
## .. .. ..@ x : num [1:2806] 29 27 11 53 33 29 58 28 40 50 ...  
## .. .. ..@ factors : list()  
## ..@ normalize: NULL

#### Prove n\_cast and n\_sparse are identical

It is possible to prove there was not any loss of data converting into a sparse matrix by proving n\_sparse can be converted back into a matrix identical to n\_cast

identical(as(n\_sparse, "matrix"), n\_cast)

## [1] TRUE

It is now possible to use recommenderlabs Recommender function do develop recommendation models using the in-built registry functions.Recommenderlab will normalize each row when normalization is required for the algorithms.

#### Create recommenders in recommenderlab

ALS <- Recommender(n\_sparse, method = "ALS")  
ALS\_implicit <- Recommender(n\_sparse, method = "ALS\_implicit")  
IBCF <- Recommender(n\_sparse, method = "IBCF")  
popular <- Recommender(n\_sparse, method = "POPULAR")  
random <- Recommender(n\_sparse, method = "RANDOM")  
rerecommend <- Recommender(n\_sparse, method = "RERECOMMEND")  
SVD <- Recommender(n\_sparse, method = "SVD")  
SVDF <- Recommender(n\_sparse, method = "SVDF")  
UBCF <- Recommender(n\_sparse, method = "UBCF")

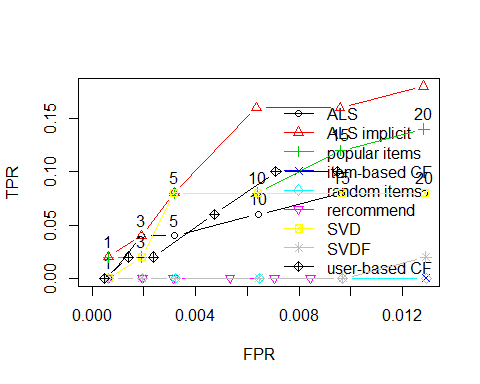
#### Evaluation of a topN recommnder with RoC and prec/rec curves

Recommenderlab allows for the evaluation and comparison of different recommender algorithms. The scheme belwo uses a 5-fold cross-validation, all-but-one rating (given=-1), and a goodRating = 2.

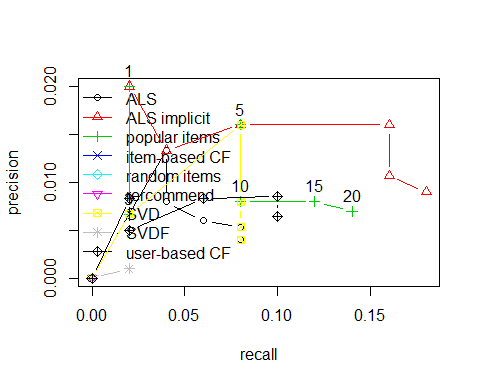
scheme <- evaluationScheme(n\_sparse, method="split", train = 0.9, k=5, given=-1, goodRating=2)  
  
algorithms <- list(  
 "ALS" = list(name="ALS"),  
 "ALS implicit" = list(name="ALS\_implicit"),  
 "popular items" = list(name="POPULAR", param=NULL),  
 "item-based CF" = list(name="IBCF"),  
 "random items" = list(name="RANDOM", param=NULL),  
 "rercommend" = list(name="RERECOMMEND"),  
 "SVD" = list(name = "SVD"),  
 "SVDF" = list(name = "SVDF"),  
 "user-based CF" = list(name="UBCF", param=(nn=150)))  
  
  
results <- evaluate(scheme, algorithms, type = "topNList",  
 n=c(1, 3, 5, 10, 15, 20))

## ALS run fold/sample [model time/prediction time]  
## 1 [0.29sec/38.58sec]   
## 2 [0.02sec/38.28sec]   
## 3 [0sec/35.84sec]   
## 4 [0sec/38.34sec]   
## 5 [0sec/37.11sec]   
## ALS\_implicit run fold/sample [model time/prediction time]  
## 1 [0.29sec/55.26sec]   
## 2 [0sec/56.3sec]   
## 3 [0sec/54.92sec]   
## 4 [0sec/55sec]   
## 5 [0sec/56.45sec]   
## POPULAR run fold/sample [model time/prediction time]  
## 1 [0.11sec/0.08sec]   
## 2 [0.01sec/0.08sec]   
## 3 [0.03sec/0.05sec]   
## 4 [0.01sec/0.05sec]   
## 5 [0.01sec/0.05sec]   
## IBCF run fold/sample [model time/prediction time]  
## 1 [32.99sec/0.06sec]   
## 2 [33.57sec/0.03sec]   
## 3 [34.34sec/0.03sec]   
## 4 [34.43sec/0.05sec]   
## 5 [36.15sec/0.03sec]   
## RANDOM run fold/sample [model time/prediction time]  
## 1 [0.06sec/0.08sec]   
## 2 [0sec/0.06sec]   
## 3 [0sec/0.06sec]   
## 4 [0sec/0.05sec]   
## 5 [0sec/0.07sec]   
## RERECOMMEND run fold/sample [model time/prediction time]  
## 1 [0.03sec/0sec]   
## 2 [0sec/0sec]   
## 3 [0sec/0sec]   
## 4 [0sec/0sec]   
## 5 [0sec/0sec]   
## SVD run fold/sample [model time/prediction time]  
## 1 [0.96sec/0.04sec]   
## 2 [0.02sec/0.04sec]   
## 3 [0.03sec/0.05sec]   
## 4 [0.03sec/0.05sec]   
## 5 [0.03sec/0.04sec]   
## SVDF run fold/sample [model time/prediction time]  
## 1 [32.37sec/61.07sec]   
## 2 [33.08sec/14.65sec]   
## 3 [34.03sec/20.37sec]   
## 4 [32.65sec/20.68sec]   
## 5 [33.75sec/20.6sec]   
## UBCF run fold/sample [model time/prediction time]  
## 1 [0.1sec/0.13sec]   
## 2 [0.01sec/0.09sec]   
## 3 [0sec/0.08sec]   
## 4 [0sec/0.12sec]   
## 5 [0sec/0.11sec]

plot(results, annotate=c(1,3), legend="bottomright")



plot(results, "prec/rec", annotate=3, legend="topleft")



Based on the best performing recommendation algorithm, as determined by best RoC performance, it is possible to make predictions and print them to an output file.

#### Use best top10 recommenders based on RoC to make customer-specific recommendations

The objective of this project is to recommend merchants with whom the customer has not transacted with in the past. It should be noted the topNList are based on the predicted values and predicted values are not generated for items already rated by the active users [[4]](#footnote-47). Therefore, the following recommendations meet the objective’s requirements.

popularp <- predict(popular, n\_sparse, n=10)  
popularp\_list <- as(popularp, "list")  
  
UBCFp <- predict(UBCF, n\_sparse, n=10)  
UBCPp\_list <- as(UBCFp, "list")  
  
  
ALS\_implicitp <-predict(ALS\_implicit, n\_sparse, n=10)  
ALS\_implicitp\_list <- as(ALS\_implicitp, "list")  
  
write.csv(ALS\_implicitp\_list, file="Top 10 recommendations using the ALS\_implicit algorithm.csv")

# 4 Discussion

The above project shows how recommenderlab can be used to create, compare, and make predictions using various recommendation algorithms in recommenderlab in R.

### 4.1 Limitations

This project faced limitations based on system resources. This was most evident when trying to cast data into the appropriate form, which exceeded the available system RAM. Even stratifying and then casting <1% of the data resulted in an object of between 4-8 Gb, which could then not be manipulated or mined in a reasonable timeframe. While R does have the capacity to do parallel processing[[5]](#footnote-50), this was not implemented in the current project.

It was also difficult to gauge the objective success of the algorithms given the results are compared to eachother, but not to a true baseline. While the re-recommend and random algortihms did not seem to be successful strategies, it is difficult to assess whether the other alrogithms actually represented an improvement for the intended application over what CITI may current use. No additional data was provided to test the predictions against.

### 4.2 Assumptions

The above implementation of recommenderlab used the number of transactions as a proxy for rating. In addition, it was assumed a good rating was a number of transactions >2. The assumption being if the interaction was not good, a customer would not return to the merchant. It was also assumed there were no refund transactions or fraudulent transactions. These may not be valid assumptions, but it was beyond the scope of the given parameters to assess.

### 4.3 Next Steps

The implemenation of recommenderlab’s recommendation algorithms allow for various parameters to be set. Better performance could likely be achieved by optimizing the settings for the most successful algorithms.

Ideally, a system, such as a virtual machine, could be used to implement recommenderlab using all the available data. It would be useful to compare the results using recommenderlab’s binary matrix format as opposed to realRatingMatrix to see if the results are different. This implementation could use any interaction as a positive call, and a lack of interaction as a negative call.

Another interesting pursuit could include the use of context-sensitive filtering, which would use additional data (such as the date information) to better inform customer recommendations. The current implementation ignores this data, but it is possible to imagine a situation in which this data would be important. For instance, recommendations to merchants selling plants or garden items might be more relevant in spring than other times of the year.

# 5 Conclusion

In conslusion, the above implementation of recommenderlab used a subset of credit card transaction data to successfully make better than random predictions of merchant-customer interactions. This satisfied the objectives by providing recommendations at the individual customer level, for merchants whom the cusotmer has not transacted with in previous 12 months, and uses both customer-specific and group data.

1. <https://www.datascientistchallenge.com> This website has seen been taken down. [↑](#footnote-ref-22)
2. <https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf>, pages 4-10 [↑](#footnote-ref-26)
3. Michael Hahsler (2017). recommenderlab: Lab for Developing and Testing Recommender Algorithms. R package version 0.2-2.<http://lyle.smu.edu/IDA/recommenderlab/> [↑](#footnote-ref-28)
4. Michael Hahsler (2017). recommenderlab: Lab for Developing and Testing Recommender Algorithms. R package version 0.2-2.<http://lyle.smu.edu/IDA/recommenderlab/> [↑](#footnote-ref-47)
5. <https://stat.ethz.ch/R-manual/R-devel/library/parallel/doc/parallel.pdf> [↑](#footnote-ref-50)