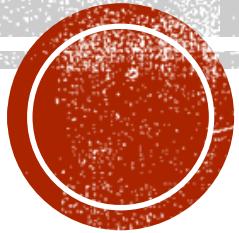
RECOMMENDER SYSTEMS

By Christopher Brossman

2017-05-16



CONTENTS

- Recommender Overview
- Recommender lab in R
- Example using behavioral data instead of rating data
  - Recommenders were designed for rating data, but most data isn't rated.
  - It can be extended to behavioral data – views, transactions, etc.

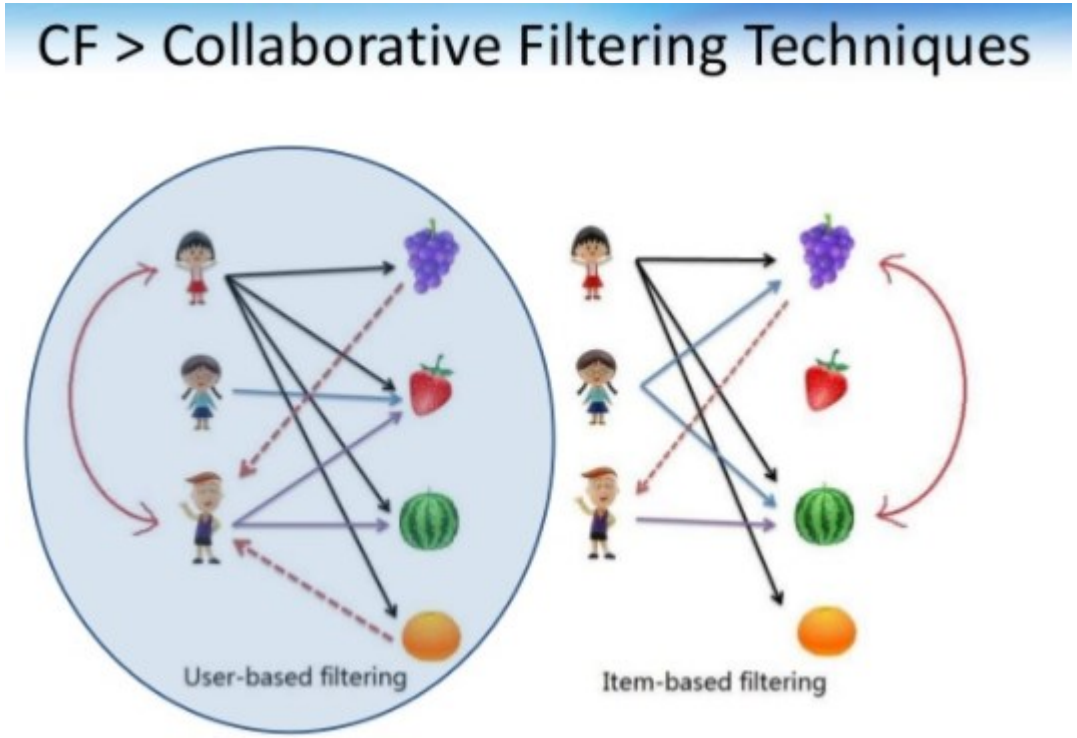
WHAT IS A RECOMMENDER SYSTEM

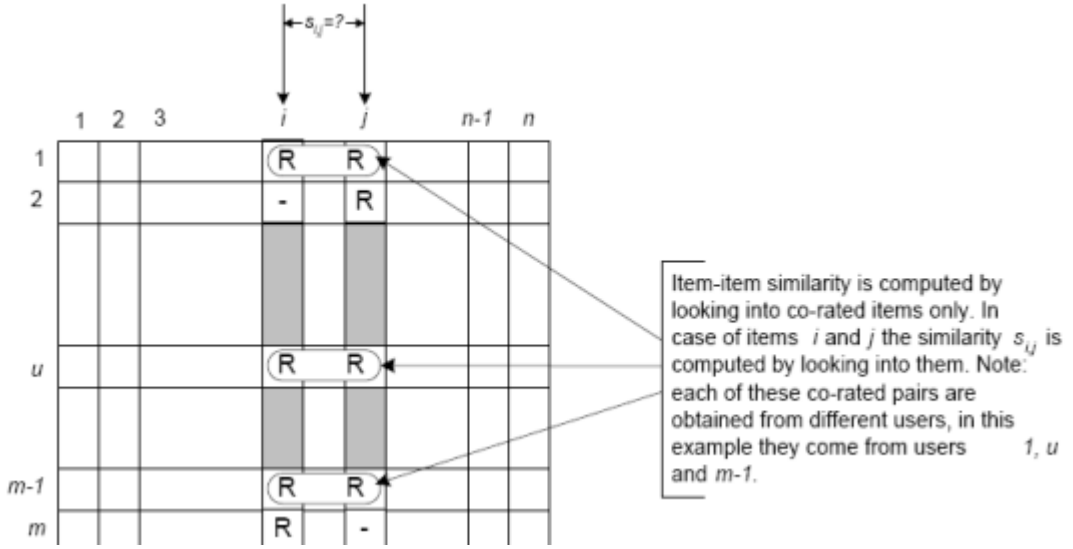
- Recommendation engines, a branch of information retrieval and artificial intelligence, are powerful tools and techniques to analyze huge volumes of data, especially product information and user information, and then provide relevant suggestions based on data mining approaches
- Types
  - Collaborative filtering
  - Content based
  - Hybrid
  - Context-aware
  - Probably more – new advances every day
- Some useful tools for recommender systems – some examples of each in repo!
  - R
  - Python
  - Spark
  - Neo4j

COLLABORATIVE FILTERING

- Basic form of recommender systems, by filtering items from a large set of alternatives is done collaboratively by users' preferences
- Assumes…
  - If two users share the same interests in past, this will continue in future
- Types
  - User based
    - Calculate similarity matrix between all users. Recommend new items to an active user based on the rating given by similar users on the items not rated by the active user
  - Item based
    - Calculate similarity matrix between all items. Find the top similar items to the non-rated items by active user and recommend them.
  - ALS (alternating least squares)
    - Used in distributed systems. Estimate matrix factorization of user-item matrix into a user matrix and item matrix with latent factors

COLLABORATIVE FILTERING – USER/ITEM





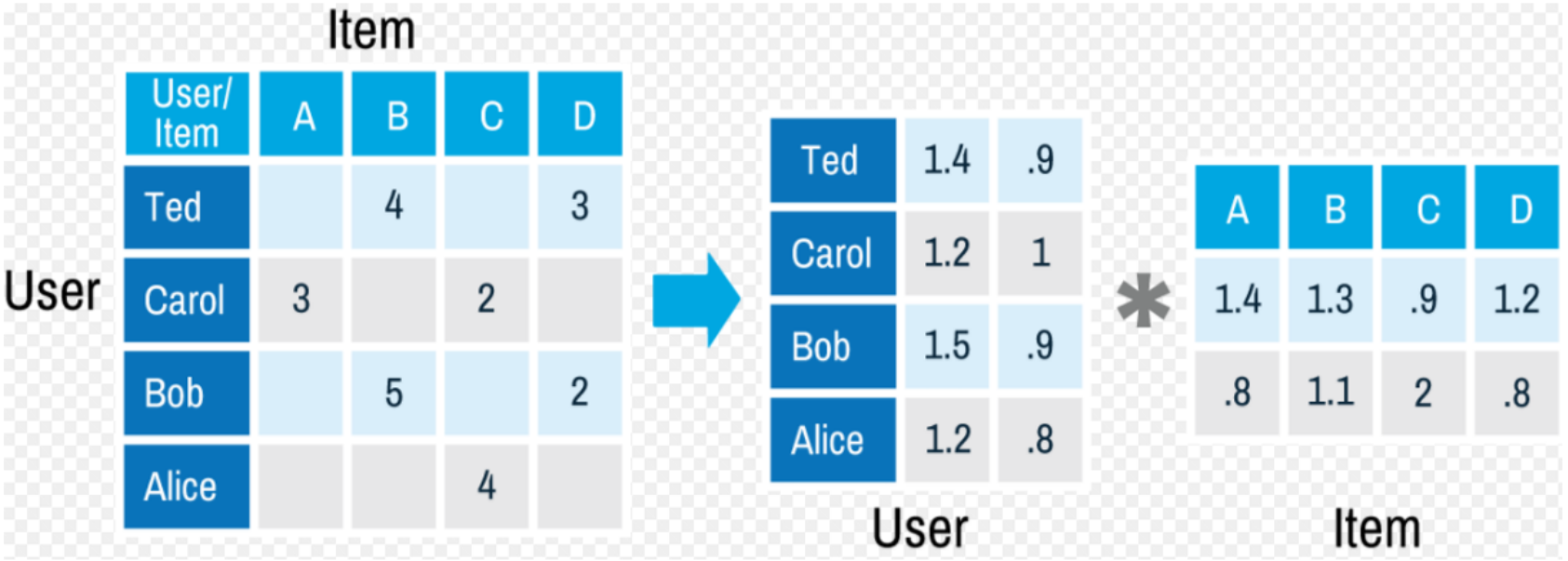
Cosine-based similarity

Also known as **vector-based similarity**, this formulation views two items and their ratings **as vectors**, and defines the similarity between them as the angle between these vectors:

$$sim(i,j) = \cos(\vec{i},\vec{j}) = \frac{\vec{i}\cdot\vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

$$\mathrm{sim}_{\mathrm{Jaccard}}(\mathcal{X},\mathcal{Y}) = \frac{|\mathcal{X}\cap\mathcal{Y}|}{|\mathcal{X}\cup\mathcal{Y}|},$$

COLLABORATIVE FILTERING - ALS




```
u <- c(1.4,1.2,1.5,1.2,.9,1,.9,.8)
i <- c(1.4,.8,1.3,1.1,.9,2,1.2,.8)
um <- matrix(u,nrow=4,ncol=2)
im <- matrix(i,nrow=2,ncol=4)
uim <- um%*%im
colnames(uim) <- c('A','B','C','D')
rownames(uim) <- c('Ted','Carol','Bob','Alice')
uim
```

```
          A    B    C    D
Ted   2.68 2.81 3.06 2.40
Carol 2.48 2.66 3.08 2.24
Bob   2.82 2.94 3.15 2.52
Alice 2.32 2.44 2.68 2.08
```

RECOMMENDER IN R

- recommenderlab: A Framework for Developing and Testing Recommendation Algorithms
- recommenderlab is implemented using formal classes in the S4 class system.
  - R has three object oriented (OO) systems: [[S3]], [[S4]] and [[R5]]. This page describes S4.
  - Compared to S3, the S4 object system is much stricter, and much closer to other OO systems. There are two major differences from S3:
    - formal class definitions: unlike S3, S4 formally defines the representation and inheritance for each class
    - multiple dispatch: the generic function can be dispatched to a method based on the class of any number of argument, not just one
  - They use "slots" accessed via "@" symbol. Similar to named lists accessed via "$"

EXAMPLE USING STORE TRANSACTIONS

ConsumerID	TransactionDateTime	Age	ConsumerGender	TransactionItems_ItemBarcode	TransactionItems_ItemQuantity
203777794	2016-10-23 22:46:00 UTC	0	M	0012000811319	1
203899651	2016-10-23 09:46:00 UTC	0	U	0028200003577	1
202717196	2016-11-04 08:08:00 UTC	0	M	0028200003843	1
202717196	2016-11-08 08:20:00 UTC	0	M	0999991218955	1
202712078	2016-12-24 19:19:00 UTC	0	U	0073100008825	1
203897615	2016-10-23 13:25:00 UTC	0	U	0028200173461	2
203991055	2016-12-20 06:57:00 UTC	0	U	0674776180114	1
203826960	2016-11-15 09:27:00 UTC	0	F	0999995303220	1
203803410	2016-09-21 22:20:00 UTC	0	F	0026100005738	1
204010514	2016-11-26 15:27:00 UTC	0	U	0786162150004	1

Transactions: 464,047 rows

Attributes of products: 28,528 rows

Product UPC	Product	NACS CATEGORY	NACS SUBCATEGORY	BRAND	COMPANY	NACS Category Code	NACS Subcategory Code
0000150000035	SIERRA NEVADA SEASONAL SHORT NECK BOTTLE	BEER	MICRO	SIERRA NEVADA SEASONAL	SIERRA NEVADA BREWING CO	04-00-00	04-06-00
0000340100095	FLAVORED BREAD BANANA 6.6 OUNCE	PACKAGED BREAD	PACKAGED BREADS	NO BRAND LISTED	NO COMPANY LISTED	14-00-00	14-00-00
0000450204430	ZYRTEC ALRGY COUGH COLD SINUS AND ALLRG	HEALTH BEAUTY CARE	COUGH COLD REMEDIES	ZYRTEC	MCNEIL CONSUMER HEALTHCARE	21-00-00	21-02-00
0000450449053	TYLNL CPLT XTRA STREN BTTL IN BOX PAIN R	HEALTH BEAUTY CARE	ANALGESICS	TYLENOL	MCNEIL CONSUMER HEALTHCARE	21-00-00	21-01-00
0000610007048	ARIVA ANTSM PRDCT 10 COUNT	HEALTH BEAUTY CARE	SMOKING CESSATION	ARIVA	STAR TOBACCO INC	21-00-00	21-14-00
0000630135615	TOO TARTS SUCK UPS BLU BEWWY LIQ NNCHC L	CANDY	NOVELTIES SEASONAL	TOO TARTS SUCK UPS	INNOVATIVE CANDY CONCEPTS	08-00-00	08-06-00
0000630135899	TOO TARTS BLU BEWWY LIQ NNCHC LQD CANDY	CANDY	NOVELTIES SEASONAL	TOO TARTS	INNOVATIVE CANDY CONCEPTS	08-00-00	08-06-00
0000630137343	TOO TARTS ITS GOO 4 YOU NNCHC LQD CANDY	CANDY	NOVELTIES SEASONAL	TOO TARTS	INNOVATIVE CANDY CONCEPTS	08-00-00	08-06-00
0000710100021	BLYS CGRTT 20 COUNT	CIGARETTES	FOURTH TIER	BAILEY S	S M BRANDS INC.	02-00-00	02-05-00
0000710100038	BLYS CGRTT 20 COUNT	CIGARETTES	FOURTH TIER	BAILEY S	S M BRANDS INC.	02-00-00	02-05-00

PREP DATA – CREATE BINARY MATRIX

```
require('tidyverse')
library(recommenderlab)
library(graphAM)

#########################################################################################################################################################
# read in data and do some light transformations
#########################################################################################################################################################

#read in data
trans_small <- read_csv(paste0(getwd(),'/data/CK_LoyaltyTrans_2016Ohio.csv'))
upc_attr <- read_csv(paste0(getwd(),'/data/UPC_attributes.csv'))

#creating a new product id -- just depends on what one wants to recommend. Here is another alternative
product_cat<- upc_attr %>% mutate(prod_val = paste0(`NACS CATEGORY`,'-',`NACS SUBCATEGORY`,'-',BRAND)) %>% distinct(prod_val) %>% mutate(prod_id = 1:NROW(prod_val))

#combined
combined_dat <- inner_join(trans_small, upc_attr, by=c('TransactionItems_ItemBarcode'='Product UPC')) %>%
  mutate(prod_val = paste0(`NACS CATEGORY`,'-',`NACS SUBCATEGORY`,'-',BRAND)) %>% inner_join(product_cat,by='prod_val')
```

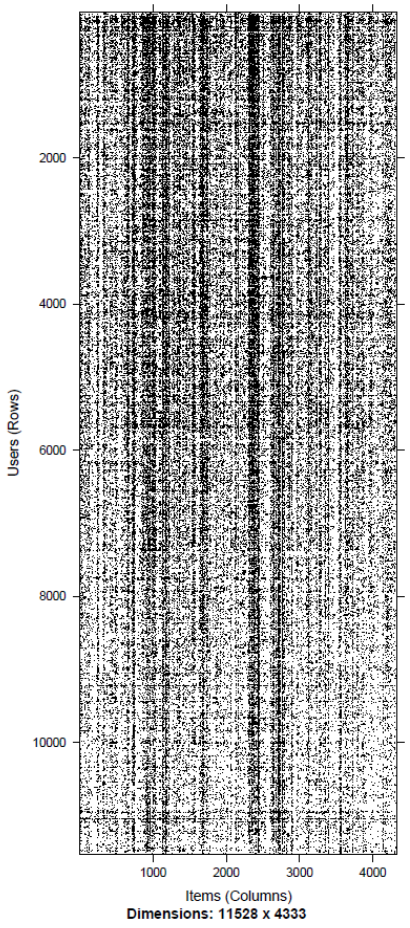
```
#user item matrix using product after JOINED with UPC info. Making this a binary matrix
user_item_mat <- combined_dat %>% mutate(purch = 1) %>% select(consumer = ConsumerID, item = Product, purch) %>%
  distinct() %>% spread(key = item, value = purch, fill=0) %>% as.matrix()
rownames(user_item_mat) <- user_item_mat[,1]
user_item_mat <- user_item_mat[,2:NCOL(user_item_mat)]
dim(user_item_mat)
#11528X4333

## coerce it into a binaryRatingMatrix, then remove user_item_matrix for memory
#382.2 MB vs 1.7MB, much more efficient storage
b <- as(user_item_mat, "binaryRatingMatrix"); rm(user_item_mat)
```

EXAMINE DATA

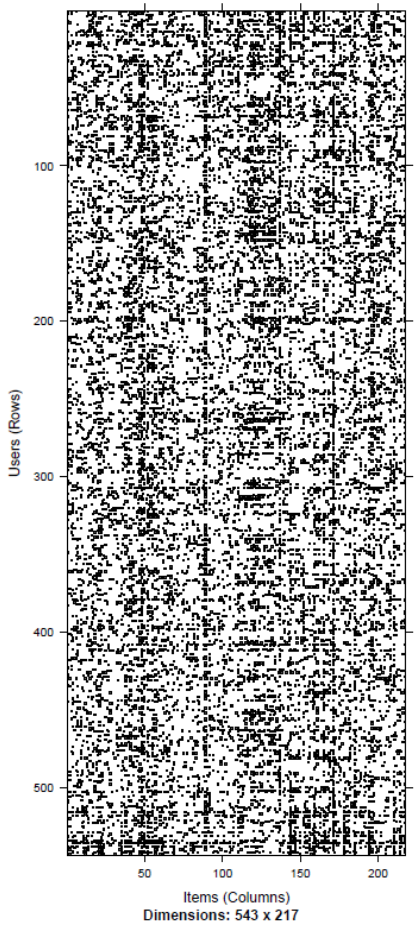
```r
#top 5% of customers and products mapped
image(b[rowCounts(b) > quantile(rowCounts(b),.95), colCounts(b) > quantile(colCounts(b),.95)])


## use some methods defined in ratingMatrix
class(b)
dim(b)
dimnames(b)
## counts
rowCounts(b)
colCounts(b)

## plot
image(b)

#methods available in recommenderlab
methods(class = class(b))

#get methods
recommenderRegistry$get_entries(dataType = "binaryRatingMatrix")
names(recommenderRegistry$get_entries(dataType = "binaryRatingMatrix"))

#data exploration
customerPurchase <- as.data.frame(table(rowCounts(b))) %>% select(NumPurchases = Var1, Freq) %>% mutate(pct = Freq/sum(Freq), cum = cumsum(Freq)/sum(Freq))
n_users <- rowCounts(b)
qplot(n_users) + stat_bin(binwidth=5) + ggtitle("distribution of the number of items purchased by customers")
qplot(n_users[n_users <= 50]) + stat_bin(binwidth=5) + ggtitle("distribution of the number of items purchased by customers, customers with <50 purchases")

itemPurchase <- as.data.frame(table(colCounts(b))) %>% select(NumPurchases = Var1, Freq) %>% mutate(pct = Freq/sum(Freq), cum = cumsum(Freq)/sum(Freq))
n_items <- colCounts(b)
qplot(n_items) + stat_bin(binwidth=5) + ggtitle("distribution of the products purchased")
qplot(n_items[n_items <= 50]) + stat_bin(binwidth=5) + ggtitle("distribution of the products purchased, items with <50 purchases")

bsamp <- b[rowCounts(b)>=5,colCounts(b)>=5]
#bsamp <- b[names(rowCounts(b)>=5),names(colCounts(b)>=5)]
image(bsamp)

#data exploration
customerPurchaseSamp <- as.data.frame(table(rowCounts(bsamp))) %>% select(NumPurchases = Var1, Freq) %>% mutate(pct = Freq/sum(Freq), cum = cumsum(Freq)/sum(Freq))
n_users <- rowCounts(bsamp)
qplot(n_users) + stat_bin(binwidth=5) + ggtitle("distribution of the number of items purchased by customers")
qplot(n_users[n_users <= 50]) + stat_bin(binwidth=5) + ggtitle("distribution of the number of items purchased by customers, customers with <50 purchases")

itemPurchaseSamp <- as.data.frame(table(colCounts(bsamp))) %>% select(NumPurchases = Var1, Freq) %>% mutate(pct = Freq/sum(Freq), cum = cumsum(Freq)/sum(Freq))
n_items <- colCounts(bsamp)
qplot(n_items) + stat_bin(binwidth=5) + ggtitle("distribution of the products purchased")
qplot(n_items[n_items <= 50]) + stat_bin(binwidth=5) + ggtitle("distribution of the products purchased, items with <50 purchases")
```

SPARSITY OF MATRIX – HIGHLY SPARSE

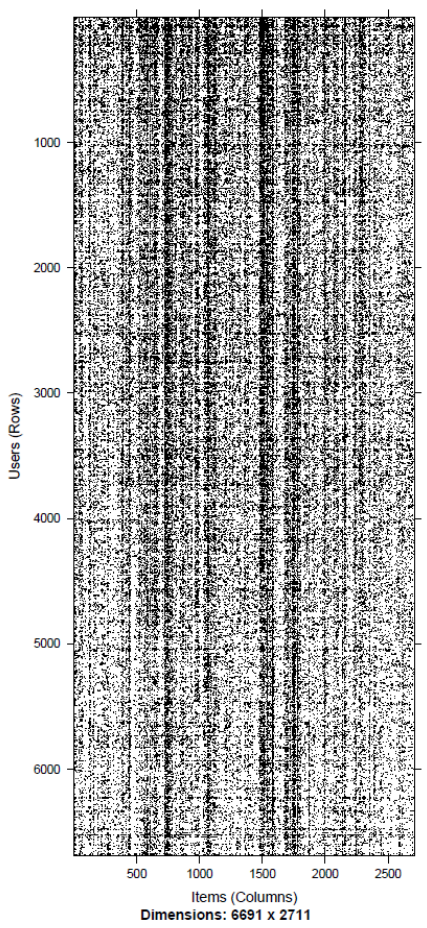
All Data



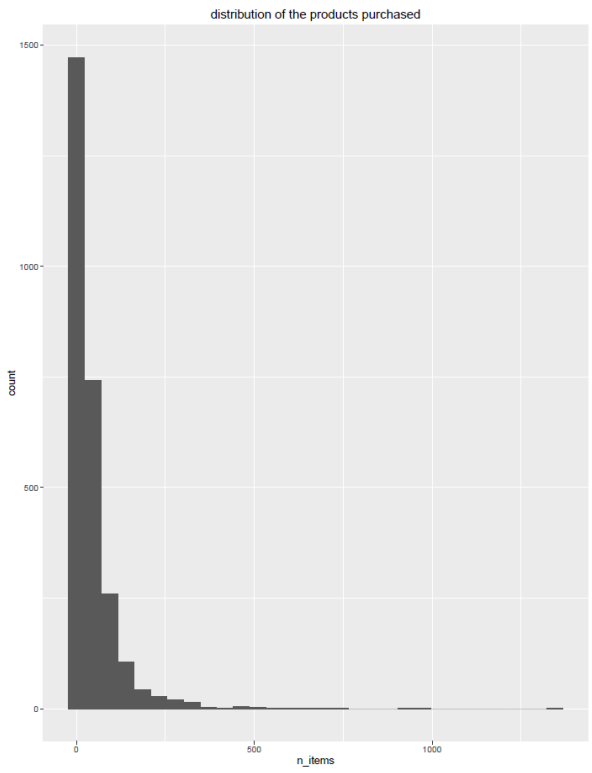
Top 5% Customer/Product

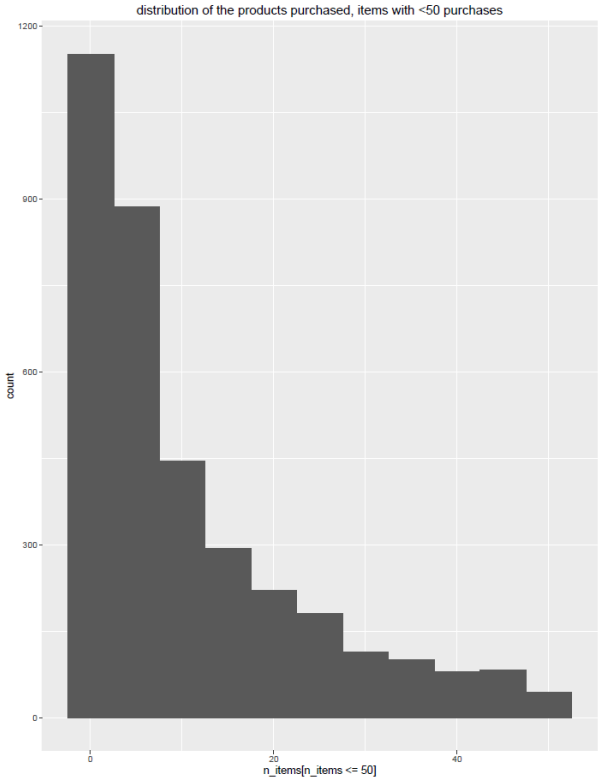


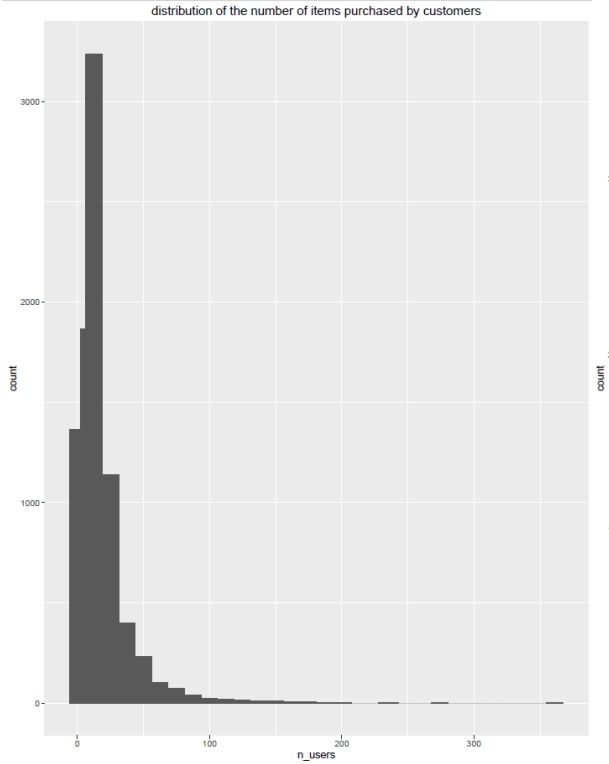
Customer/Product >=5 interactions

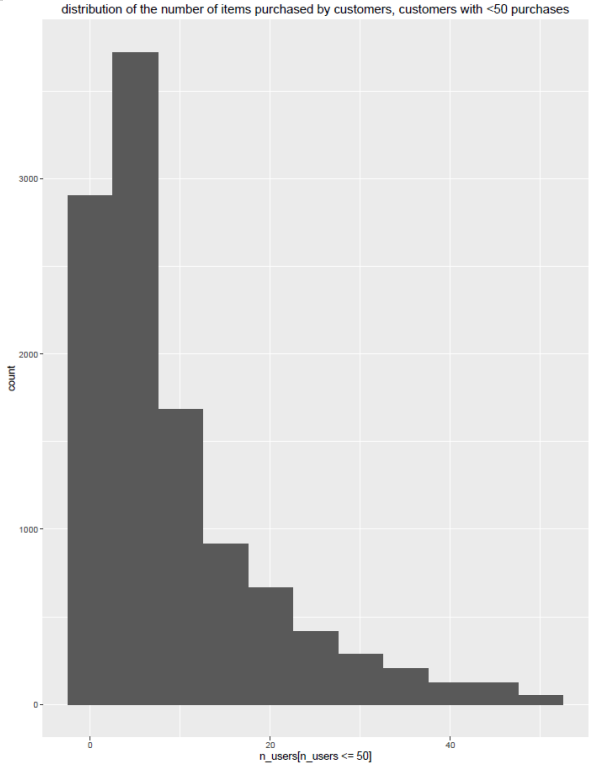


TRANSACTIONS PER CUSTOMER/PRODUCT









CREATE MODELS

```r
model = "POPULAR"
params = NULL

pop_r <- Recommender(recc_data_train, method = model, parameter = params)
#The model can be obtained from a recommender using getModel().
names(getModel(pop_r))

#predict using predict method in R, makes 5 predictions
pop_recom <- predict(pop_r, recc_data_test, n=5)
#prediction list
pop_list <- as(pop_recom, "list")
```

```r
model = "ALS"
params = list(lambda = .1, alpha = 10, n_factors = 10, n_iterations = 10, min_item_nr = 1, seed = NULL)

ALS_r <- Recommender(recc_data_train, method = model, parameter = params)

#The model can be obtained from a recommender using getModel().
names(getModel(ALS_r))

#predict using predict method in R, makes 5 predictions
ALS_recom <- predict(ALS_r, recc_data_test, n=5)
#prediction list
ALS_list <- as(ALS_recom, "list")
```

PREDICTION/HISTORY OF PURCHASES

Build UBCF model and make predictions

```
# create UBCF based recommendations
#######################################################################
model = "UBCF"
params = list(nn = 25, method="Jaccard", weighted = TRUE, sample = FALSE)
UBCF_r <- Recommender(recc_data_train, method = model, parameter = params)
#The model can be obtained from a recommender using getModel().
names(getModel(UBCF_r))

#predict using predict method in R, makes 5 predictions
UBCF_recom <- predict(UBCF_r, recc_data_test, n=5)
#prediction list
UBCF_list <- as(UBCF_recom, "list")
```

Top 5 product rec. for customer #200077422

```
> for(i in UBCF_list$`200077422`){print(i)}
[1] "CTL BR FRNCH ON THE ORGNL SR CRM 31 GRAM"
[1] "DORITOS NACHO CHS TRNGL TRTLL CHIP TORTI"
[1] "MTN DEW CTRS SCREW CAP LOW SDM SOFT DRNK"
[1] "PEPSI COLA TAB TOP LOW SDM SOFT DRNK 16"
[1] "LAYS SOUR CREAM AND ONION 1 OUNCE POT CH"
```

Historic purchases of customer #200077422

ConsumerID	TransactionDateTime	Age	ConsumerGender	TransactionItems_ItemBarcode	TransactionItems_ItemQuantity	Product	NACS CATEGORY
200077422	2016-03-07 13:05:00 UTC	50	F	0028200009654	1	MRLBR SMTH ORGNL FLVR BOX IN WRAP 72 MLL	CIGARETTES
200077422	2016-08-25 16:14:00 UTC	50	F	0028200009654	1	MRLBR SMTH ORGNL FLVR BOX IN WRAP 72 MLL	CIGARETTES
200077422	2016-04-07 20:40:00 UTC	50	F	0715637162006	1	CTL BR FRNCH ON SR CRM 31 GRAM TUB SPRD	OTHER DAIRY DELI PRODUCTS
200077422	2016-04-07 20:40:00 UTC	50	F	0028400598125	1	LAYS CLSSC 1 OUNCE FMLY SIZE 28 GRAM KSH	SALTY SNACKS
200077422	2016-12-11 12:37:00 UTC	50	F	0028400006200	1	FRITO LAY MNCHS SANDW CRCKR 1.38 OUNCE	SALTY SNACKS
200077422	2016-04-07 20:40:00 UTC	50	F	0012000027215	1	MTN DEW CTRS KSHR TAB TOP LOW SDM SOFT D	PACKAGED BEVERAGES
200077422	2016-04-07 20:40:00 UTC	50	F	0012000027215	1	MTN DEW CTRS KSHR TAB TOP LOW SDM SOFT D	PACKAGED BEVERAGES
200077422	2016-08-25 16:14:00 UTC	50	F	0715637162006	1	CTL BR FRNCH ON SR CRM 31 GRAM TUB SPRD	OTHER DAIRY DELI PRODUCTS

HYBRID MODEL - IBCF

```
model = "IBCF"
params = list(k = 30, method="Jaccard", normalize_sim_matrix = FALSE, alpha = .5)

IBCF_r <- Recommender(recc_data_train, method = model, parameter = params)

#get similarity matrix
dist_ratings <- as(IBCF_r@model$sim, 'matrix')
product_names <- rownames(dist_ratings)
dist_category <- dist_ratings
lookup <- combined_dat %>% select(Product, cat = `NACS SUBCATEGORY`) %>% distinct() %>% arrange(cat)
cnt = 1
for(p in product_names){
  print(paste0(cnt,' of ',NROW(product_names)))
  p_cat <- lookup %>% filter(Product == p) %>% select(cat)
  poss <- lookup %>% filter(cat == p_cat$cat) %>% select(Product)
  for(j in product_names){
    dist_category[p,j] <- ifelse(j %in% poss$Product,1,0)
  }
  cnt = cnt + 1
}

#image(dist_category); image(dist_ratings)
#check for same product vals
identical(dim(dist_category), dim(dist_ratings))
identical(rownames(dist_category), rownames(dist_ratings))
identical(colnames(dist_category),colnames(dist_ratings))
w <- .25
dist_tot <- dist_category*w + dist_ratings*(1-w)

IBCF_r@model$sim <- as(dist_tot, 'dgCMatrix')

#The model can be obtained from a recommender using getModel().
names(getModel(IBCF_r))

#predict using predict method in R, makes 5 predictions
IBCF_recom <- predict(IBCF_r, recc_data_test, n=5)
#prediction list
IBCF_list <- as(IBCF_recom, "list")
```

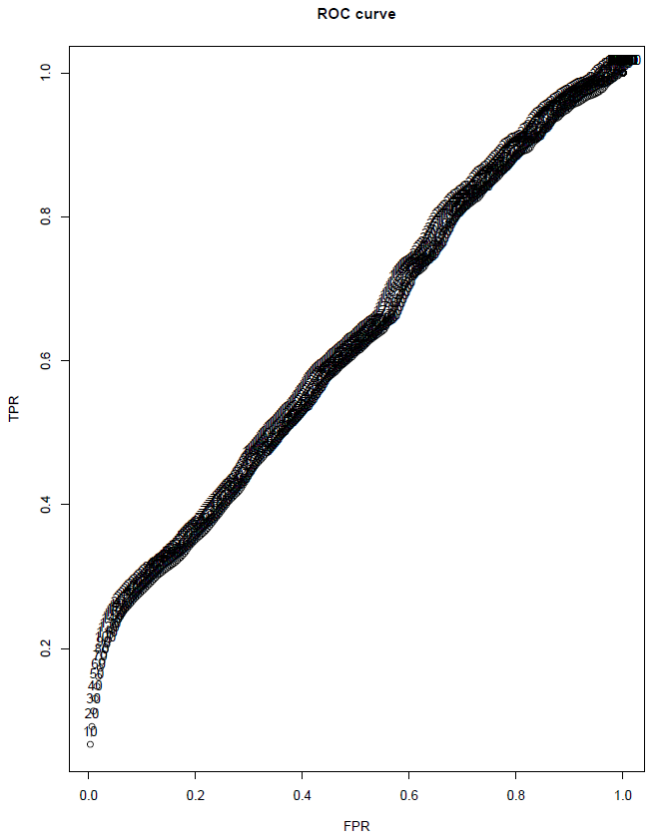
- IBCF creates similarity matrix between all products.
- We can integrate other dimensional information about products as long as they conform to the same matrix dimensions
- Here we create a binary matrix if two products are in the same product category
- Then integrate this information in the similarity matrix

EVALUATE MODEL

```
# Evaluation
#############################################################################################
#create evaluation method
#given is the number of items to give to recommender for prediction, rest are held for compu
# group we know everyone has purchased atleast 3 products, so we are given 2 to the recommen
e <- evaluationScheme(bsamp, method="split", train=0.8, k=2, given=2)
e <- evaluationScheme(bsamp, method="cross-validation", train=0.8, k=2, given=2)

## different models
ubcf_r <- Recommender(getData(e, "train"), "UBCF")
ibcf_r <- Recommender(getData(e, "train"), "IBCF")

## create predictions for the test data using known ratings (see given above)
ubcf_p <- predict(ubcf_r, getData(e, "known"), type="topNList", n=2)
ibcf_p <- predict(ibcf_r, getData(e, "known"), type="topNList", n=2)

ubcf_p_l <- as(ubcf_p, "list")
ibcf_p_l <- as(ibcf_p, "list")


error <- rbind(
  UBCF = calcPredictionAccuracy(ubcf_p, getData(e, "unknown"), given=2),
  IBCF = calcPredictionAccuracy(ibcf_p, getData(e, "unknown"), given=2)
  )

errorUserUBCF = calcPredictionAccuracy(ubcf_p, getData(e, "unknown"), given=2,byUser=TRUE)
errorUserIBCF = calcPredictionAccuracy(ibcf_p, getData(e, "unknown"), given=2,byUser=TRUE)

model_to_evaluate <- "UBCF"
results <- evaluate(x = e, method = model_to_evaluate, n = seq(10, 10000, 10))
head(getConfusionMatrix(results)[[1]])

columns_to_sum <- c("TP", "FP", "FN", "TN")
indices_summed <- Reduce("+", getConfusionMatrix(results))[, columns_to_sum]
head(indices_summed)
plot(results, annotate = TRUE, main = "ROC curve")
```

- Evaluation scheme handles split or cross validation
- New models are created here due to evaluation scheme, although earlier models could be modified to this format.
- Can create grid using "evaluate" to determine optimal params
- Can obtain confusion matrix using code below

EVALUATE MODEL




```
results <- evaluate(x = e, method = model_to_evaluate, n = seq(10, 10000, 10))
```

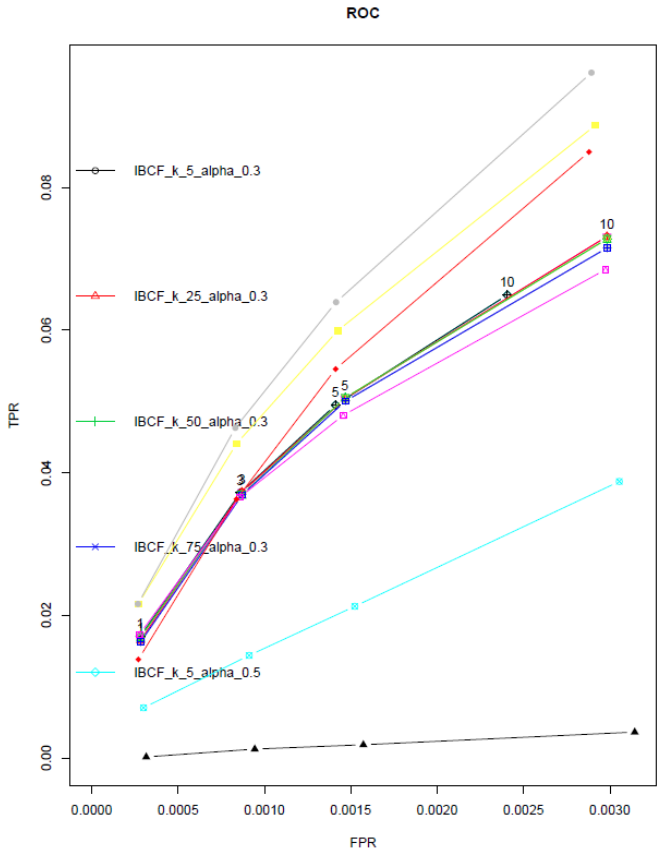
	TP	FP	FN	TN	precision	recall	TPR	FPR
10	0.5876623	9.412338	12.98516	3157.015	0.05876623	0.06619965	0.06619965	0.002971605
20	0.8620130	19.137987	12.71081	3147.289	0.04310065	0.09098530	0.09098530	0.006042555
30	1.0964750	28.903525	12.47635	3137.524	0.03654917	0.11382322	0.11382322	0.009126231
40	1.3063544	38.693646	12.26647	3127.734	0.03265886	0.13129677	0.13129677	0.012217617
50	1.4995362	48.500464	12.07328	3117.927	0.02999072	0.14915253	0.14915253	0.015314421
60	1.6688312	58.331169	11.90399	3108.096	0.02781385	0.16238089	0.16238089	0.018418712

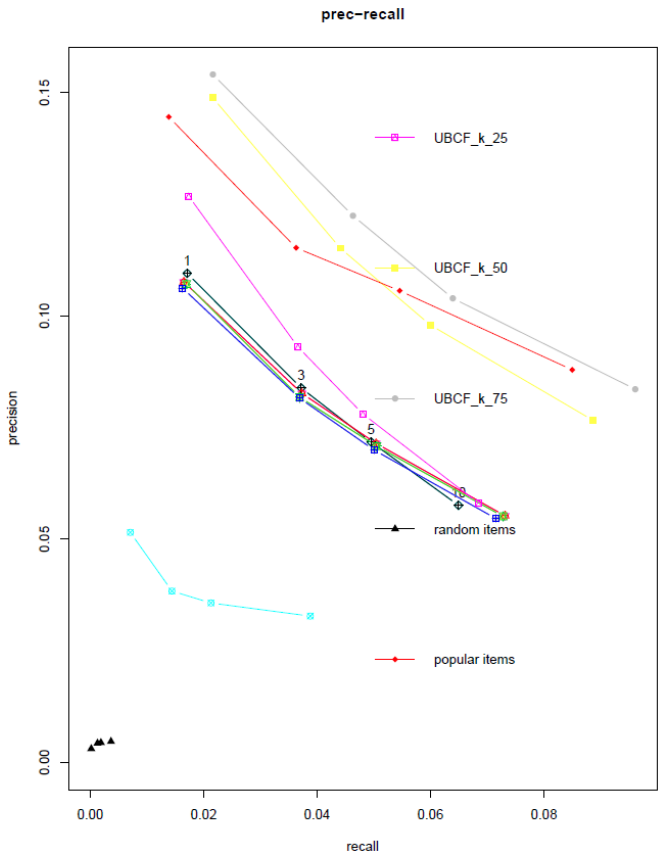
	TP	FP	FN	TN
10	1.161178	18.83882	25.75325	6314.247
20	1.700603	38.29940	25.21382	6294.786
30	2.161874	57.83813	24.75255	6275.247
40	2.571197	77.42880	24.34323	6255.657
50	2.951763	97.04824	23.96266	6236.037
60	3.288033	116.71197	23.62639	6216.374

GRID SEARCH TO OPTIMIZE PARAMETERS

```r
vector_k <- c(5, 25, 50, 75)
alpha <- seq(.3, .5, .7)
models <- list()

model1 <- lapply(vector_k, function(k,l){ list(name = "IBCF", param = list(method = "Jaccard", k = k, alpha = .3)) })
names(model1) <- paste0("IBCF_k_", vector_k,'_alpha_',.3)

model2 <- lapply(vector_k, function(k,l){ list(name = "IBCF", param = list(method = "Jaccard", k = k, alpha = .5)) })
names(model2) <- paste0("IBCF_k_", vector_k,'_alpha_',.5)

model3 <- lapply(vector_k, function(k,l){ list(name = "IBCF", param = list(method = "Jaccard", k = k, alpha = .7)) })
names(model3) <- paste0("IBCF_k_", vector_k,'_alpha_',.7)

model4 <- lapply(vector_k, function(k,l){   list(name = "UBCF", param = list(method = "Jaccard", nn = k))})
names(model4) <- paste0("UBCF_k_", vector_k)

model5 <- list("random items" = list(name="RANDOM", param=NULL),"popular items" = list(name="POPULAR", param=NULL))

for(i in 1:5){
  models <- append(models, get(paste0('model',i)))
}

n_recs <- c(1, 3, 5, 10)
list_results <- evaluate(x = e, method = models, n = n_recs)
plot(list_results, annotate = c(1,2), legend = 'topleft'); title("ROC")
plot(list_results, 'prec/rec', annotate = 1, legend = 'bottomright'); title('prec-recall')
```

GRID SEARCH TO OPTIMIZE PARAMETERS





BIG DATA

- In most real world scenarios recommenders will have data too large to analyze on a local machine.
- Way to get around…
  - Incremental – use small program and database to loop through and generates counts of all products uploading in database as you go…
  - More ad-hoc
  - Use big data tools!!!
    - Spark
    - Mahout
    - Neo4j
    - etc