COLUMBIA UNIVERSITY
School of Professional Studies

Recommender Systems
Applied Analytics: Frameworks and Methods 2

Outline

- Discuss need for recommendation systems
- Discuss applications of recommender systems
- Explain how recommender systems work
  - Compare and contrast types of recommendation systems
- Utilize recommender systems to make product recommendations using R
  - Small simulated dataset
  - Jester dataset (5k sample)

Need

- Consumers are drawn to stores that offer them variety but when faced with a large number of options, they get overwhelmed and postpone their purchase.
- Recommender systems solve the problem by offering relevant suggestions that makes choosing easier.

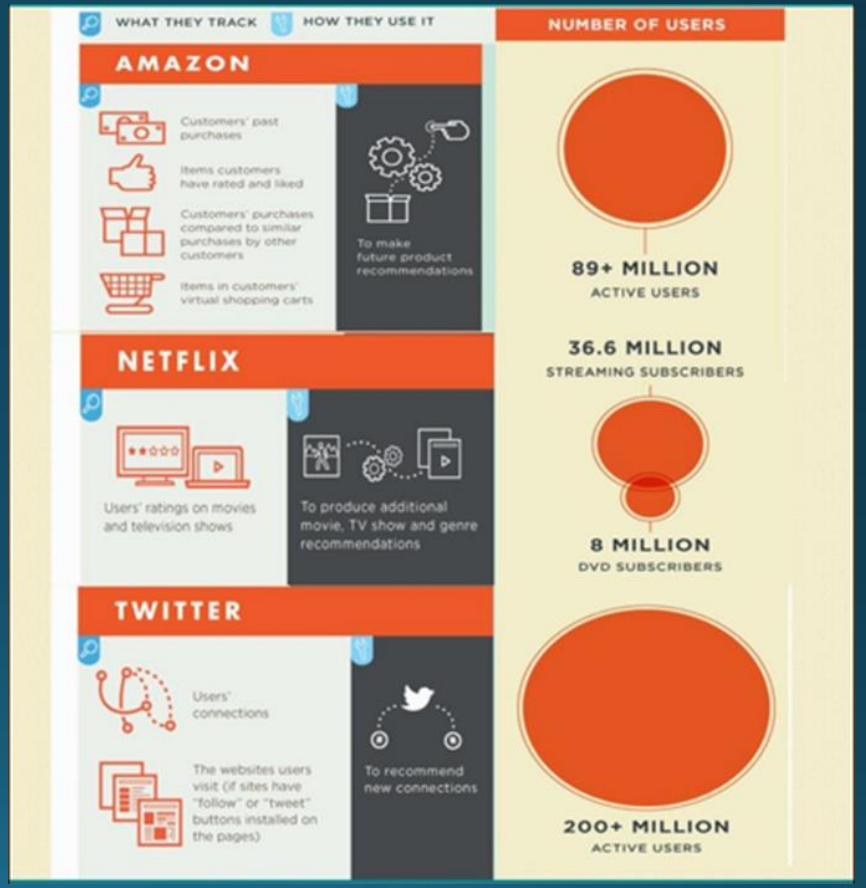
Who uses them

- Websites (e.g., Amazon, Walmart)
- Movie and Music systems (e.g., Netflix, Spotify, Pandora)
- Social Networks (e.g., Facebook, LinkedIn, Twitter)

- Consider the following
  - Two-thirds of movies watched by Netflix customers are recommended movies
  - 38% of click-through rates on Google News are recommended links
  - 35% of sales at Amazon arise from recommended products
  - ChoiceStream claims that 28% of people would like to buy more music, if they find what they like

Different Types of Recommender Systems

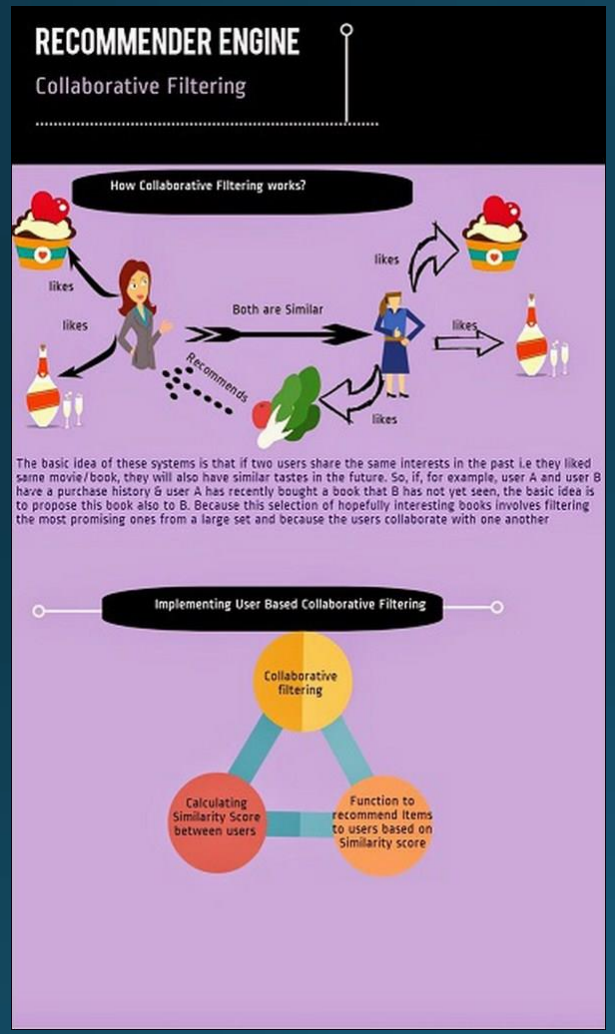
How does it Work



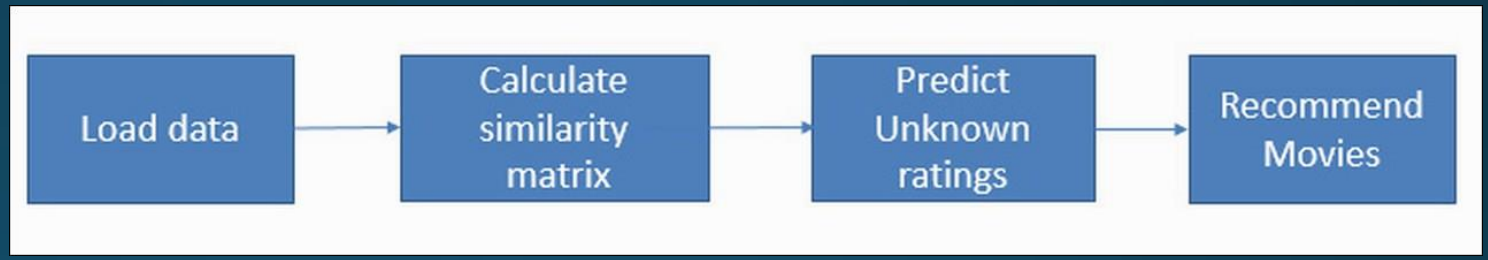
Types of Recommendation Systems

- Collaborative Filtering
  - User based
  - Item based
- Content-based
- Hybrid
- Context-aware

Collaborative filtering



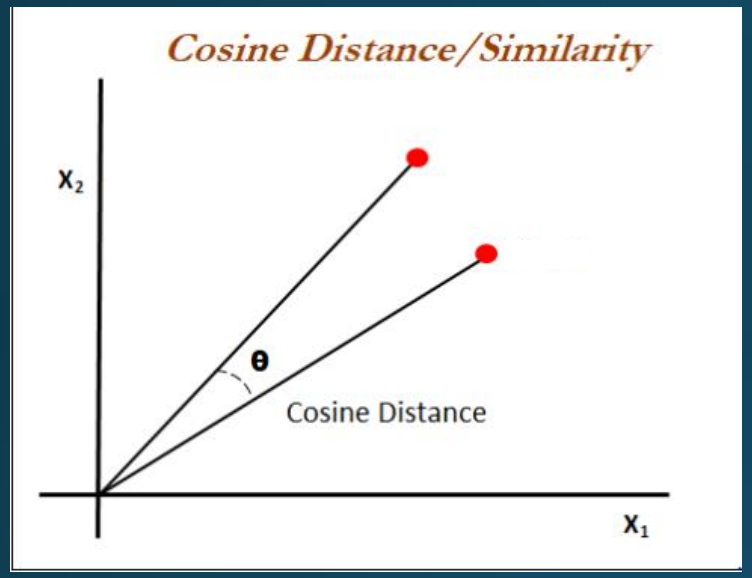
Basic idea is to identify users similar to the target user and then make product recommendations for unseen products.



Collaborative Filtering Process

- Load and prepare data:
  - Load data. Create user-item matrix. Standardize ratings.
- Calculate similarity matrix:
  - Similarity may be measured using Euclidean distance, Cosine similarity, Pearson coefficient, Jaccard distance
- Predict unknown ratings:
  - For an unknown item, obtain ratings of n nearest neighbors who have rated it. Computed a score weighted by similarity to target user.
- Recommend:
  - Suggest top k items.

Similarity Measure



with the same orientation have a cosine similarity of 1
oriented at 90° relative to each other have a similarity of 0

Source: Statistics for Machine Learning

Illustration

- A dataset containing 31 observations
- Three variables such as critic, title, and rating.
- Six critics
- Six movies
- Ratings are between 0 and 5.
- Each critic does not necessary rate all movies

```
critic,title,rating
Jack Matthews,Lady in the Water,3.0
Jack Matthews,Snakes on a Plane,4.0
Jack Matthews,You Me and Dupree,3.5
Jack Matthews,Superman Returns,5.0
Jack Matthews,The Night Listener,3.0
Mick LaSalle,Lady in the Water,3.0
Mick LaSalle,Snakes on a Plane,4.0
Mick LaSalle,Just My Luck,2.0
Mick LaSalle,Superman Returns,3.0
Mick LaSalle,You Me and Dupree,2.0
Mick LaSalle,The Night Listener,3.0
Claudia Puig,Snakes on a Plane,3.5
Claudia Puig,Just My Luck,3.0
Claudia Puig,You Me and Dupree,2.5
Claudia Puig,Superman Returns,4.0
Claudia Puig,The Night Listener,4.5
Lisa Rose,Lady in the Water,2.5
Lisa Rose,Snakes on a Plane,3.5
Lisa Rose,Just My Luck,3.0
Lisa Rose,Superman Returns,3.5
Lisa Rose,The Night Listener,3.0
Lisa Rose,You Me and Dupree,2.5
Toby,Snakes on a Plane,4.5
Toby,Superman Returns,4.0
Toby,You Me and Dupree,1.0
Gene Seymour,Lady in the Water,3.0
Gene Seymour,Snakes on a Plane,3.5
Gene Seymour,Just My Luck,1.5
Gene Seymour,Superman Returns,5.0
Gene Seymour,You Me and Dupree,3.5
Gene Seymour,The Night Listener,3.0
```

Collaborative filtering: User-based

Load Data

	Claudia Puig	Gene Seymour	Jack Matthews	Lisa Rose	Mick LaSalle	Toby
Just My Luck	3.0	1.5	NA	3.0	2	NA
Lady in the Water	NA	3.0	3.0	2.5	3	NA
Snakes on a Plane	3.5	3.5	4.0	3.5	4	4.5
Superman Returns	4.0	5.0	5.0	3.5	3	4.0
The Night Listener	4.5	3.0	3.0	3.0	3	NA
You Me and Dupree	2.5	3.5	3.5	2.5	2	1.0

- Toby has rated three movies.
- Gene Seymour, Lisa Rose, and Mick LaSalle have rated all the movies.
- Claudia Puig, and Jack Matthews have not rated one movie each.
- Objective:

  Recommend to critics movies that they have not rated, based on similar critics. For example, we shall recommend movies to Toby based on the ratings provided by other critics similar to Toby.

Collaborative filtering: User-based

Similarity Matrix

- Toby is very similar to
  - Lisa Rose (0.99), and
  - Mick LaSalle (0.92)

	Claudia Puig	Gene Seymour	Jack Matthews	Lisa Rose	Mick LaSalle	Toby
Claudia Puig	1.0000000	0.7559289	0.9285714	0.9449112	0.6546537	0.8934051
Gene Seymour	0.7559289	1.0000000	0.9449112	0.5000000	0.0000000	0.3812464
Jack Matthews	0.9285714	0.9449112	1.0000000	0.7559289	0.3273268	0.6628490
Lisa Rose	0.9449112	0.5000000	0.7559289	1.0000000	0.8660254	0.9912407
Mick LaSalle	0.6546537	0.0000000	0.3273268	0.8660254	1.0000000	0.9244735
Toby	0.8934051	0.3812464	0.6628490	0.9912407	0.9244735	1.0000000

Collaborative filtering: User-based

Predict the Unrated Movies for Toby

Predict the unrated movies of Toby using the ratings given by similar users.

The following are the steps to achieve this:

1. Extract the titles which Toby has not rated.
2. For these titles, separate all the ratings given by other critics.
3. Multiply the ratings given for these movies by all critics other than Toby with the similarity values of critics with Toby.
4. Sum up the ratings for each movie calculated in step 3, and divide this summed up value with the sum of similarity critic values.

Collaborative filtering: User-based

Identify missing ratings for Toby

1. Extract the titles which Toby has not rated.

Filter

	title	rating
1	Just My Luck	NA
2	Lady in the Water	NA
3	Snakes on a Plane	4.5
4	Superman Returns	4.0
5	The Night Listener	NA
6	You Me and Dupree	1.0

Collaborative filtering: User-based

Ratings by Others for Movies Toby has Not Watched

2. For these titles, separate all the ratings given by other critics.

critic	title	rating
Jack Matthews	Lady in the Water	3.0
Jack Matthews	The Night Listener	3.0
Mick LaSalle	Lady in the Water	3.0
Mick LaSalle	Just My Luck	2.0
Mick LaSalle	The Night Listener	3.0
Claudia Puig	Just My Luck	3.0
Claudia Puig	The Night Listener	4.5
Lisa Rose	Lady in the Water	2.5
Lisa Rose	Just My Luck	3.0
Lisa Rose	The Night Listener	3.0
Gene Seymour	Lady in the Water	3.0
Gene Seymour	Just My Luck	1.5
Gene Seymour	The Night Listener	3.0

Collaborative filtering: User-based

Append Similarity Scores

3. Multiply the ratings given for these movies by all critics other than Toby with the similarity values of critics with Toby.

critic	title	rating	similarity
Claudia Puig	Just My Luck	3.0	0.8934051
Claudia Puig	The Night Listener	4.5	0.8934051
Gene Seymour	Lady in the Water	3.0	0.3812464
Gene Seymour	Just My Luck	1.5	0.3812464
Gene Seymour	The Night Listener	3.0	0.3812464
Jack Matthews	Lady in the Water	3.0	0.6628490
Jack Matthews	The Night Listener	3.0	0.6628490
Lisa Rose	Lady in the Water	2.5	0.9912407
Lisa Rose	Just My Luck	3.0	0.9912407
Lisa Rose	The Night Listener	3.0	0.9912407
Mick LaSalle	Lady in the Water	3.0	0.9244735
Mick LaSalle	Just My Luck	2.0	0.9244735
Mick LaSalle	The Night Listener	3.0	0.9244735

Collaborative filtering: User-based

Combine Rating with Similarity

4. Sum up the "sim_rating" for each movie, and divide this summed up value with the sum of similarity critic values.

critic	title	rating	similarity	sim_rating
Claudia Puig	Just My Luck	3.0	0.8934051	2.6802154
Claudia Puig	The Night Listener	4.5	0.8934051	4.0203232
Gene Seymour	Lady in the Water	3.0	0.3812464	1.1437393
Gene Seymour	Just My Luck	1.5	0.3812464	0.5718696
Gene Seymour	The Night Listener	3.0	0.3812464	1.1437393
Jack Matthews	Lady in the Water	3.0	0.6628490	1.9885469
Jack Matthews	The Night Listener	3.0	0.6628490	1.9885469
Lisa Rose	Lady in the Water	2.5	0.9912407	2.4781018
Lisa Rose	Just My Luck	3.0	0.9912407	2.9737221
Lisa Rose	The Night Listener	3.0	0.9912407	2.9737221
Mick LaSalle	Lady in the Water	3.0	0.9244735	2.7734204
Mick LaSalle	Just My Luck	2.0	0.9244735	1.8489469
Mick LaSalle	The Night Listener	3.0	0.9244735	2.7734204

- for the Just My Luck title, the rating for Toby is calculated by summing up all the sim_rating values for Just My Luck divided by the sum of similarity values of all the critics who have rated the Just My Luck title:

- (2.6802154+0.5718696+2.9737221+1.8489469)/(0.8934051+0.3812464+0.9912407+0.9244735) = 2.530981

Collaborative filtering: User-based

Prediction is the Similarity weighted average

	title (fctr)	sum(sim_rating)/sum(similarity) (dbl)
1	Just My Luck	2.530981
2	Lady in the Water	2.832550
3	The Night Listener	3.347790

Collaborative Filtering

- Basic idea is to identify users similar to the target user and then make product recommendations for unseen products.
- User-based
  - Recommendations are generated by considering the preferences in the user's neighborhood.
    1. Identify similar users based on similar user preferences
    2. 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
  - Recommendations are generated using the neighborhood of items. Unlike user-based collaborative filtering, we first find similarities between items and then recommend non-rated items which are similar to the items the active user has rated in past.
    1. Calculate the item similarity based on the item preferences
    2. Find the top non-rated items similar to the rated items by active user and recommend them

Collaborative filtering: Item-based

Movie/User	Claudia Puig	Gene Seymour	Jack Matthews	Lisa Rose	Mick LaSalle	Toby
Just My Luck	3	1.5		3	2	
Lady in the Water		3	3	2.5	3	
Snakes on a Plane	3.5	3.5	4	3.5	4	4.5
Superman Returns	4	5	5	3.5	3	4
The Night Listener	4.5	3	3	3	3	
You Me and Dupree	2.5	3.5	3.5	2.5	2	1

	Just My Luck	Lady in the Water	Snakes on a Plane	Superman Returns	The Night Listener	You Me and Dupree
Just My Luck	1.0000000	0.6339001	0.7372414	0.7194516	0.8935046	0.7598559
Lady in the Water	0.6339001	1.0000000	0.7950515	0.8149529	0.7977412	0.8897565
Snakes on a Plane	0.7372414	0.7950515	1.0000000	0.9779829	0.8585983	0.9200319
Superman Returns	0.7194516	0.8149529	0.9779829	1.0000000	0.8857221	0.9680784
The Night Listener	0.8935046	0.7977412	0.8585983	0.8857221	1.0000000	0.9412504
You Me and Dupree	0.7598559	0.8897565	0.9200319	0.9680784	0.9412504	1.0000000

- Predict Toby's rating for Lady in the Water

take the similarity score of *Lady in the Water* for each movie rated by Toby, multiply it by the corresponding rating, and sum up all the scores for all the rated movies. This final sum is divided by the total sum of similarity scores of *Lady in the Water* given as follows:

(0.795*4.5 + 0.814*4 + 0.889*1)/(0.795+0.814+0.889) = 3.09

Collaborative Filtering

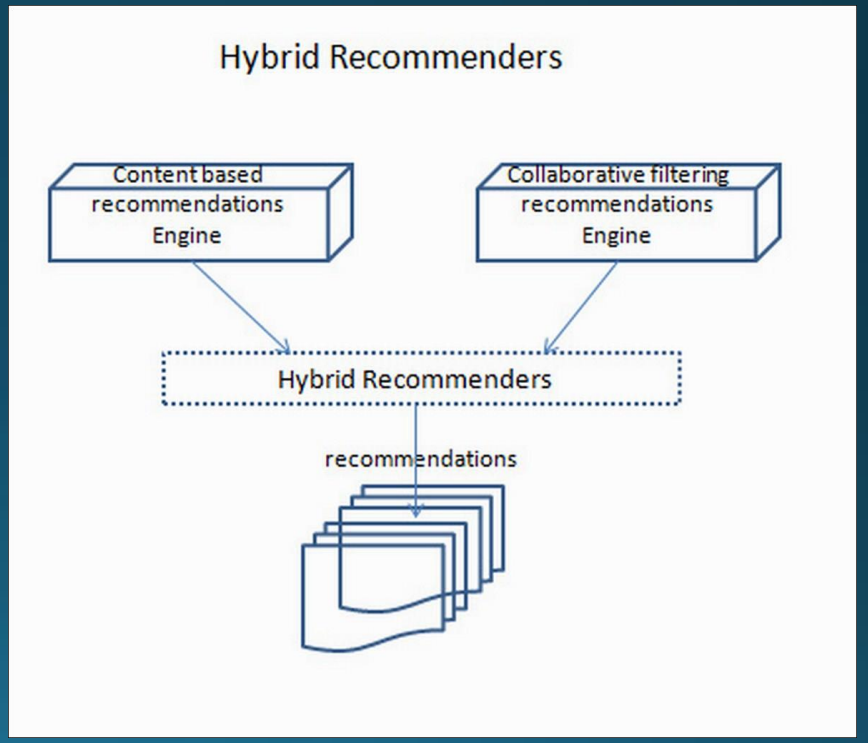
- These methods are easy to implement and very accurate. However, they suffer from the following problems
  - Sparsity problem: Users have not rated/used most items
  - Cold Start Problem: An extreme case of sparsity problem.
    - New users do not have any ratings
    - New products do not have any ratings

Content-Based Filtering

- Uses item properties and user preferences to the item properties while building content-based recommendation engines.
- A content-based recommender system uses the content information of the items for building the recommendation model. Contains
  - a user-profile-generation step,
  - item-profile-generation step- and
  - model-building step to generate recommendations for an active user.
- Though content-based recommenders have solved some shortcomings of collaborative filtering, these have their own inherent shortcomings such as not being able to recommend new items outside the user's preference scope, which collaborative filtering can do.

Hybrid Recommender System

- Works by combining various recommender systems to build a more robust system
- Addresses shortcomings of individual methods



Context Aware Recommendation

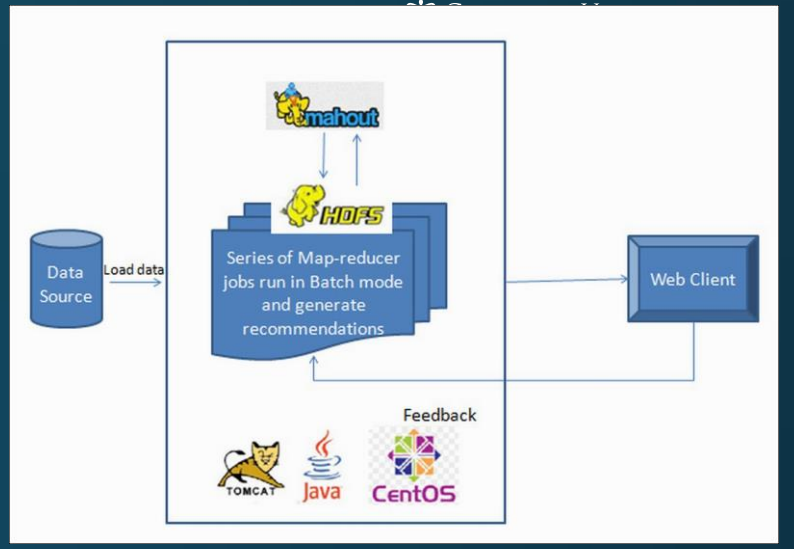
- A context-aware recommender system takes the context into account before computing or serving recommendations

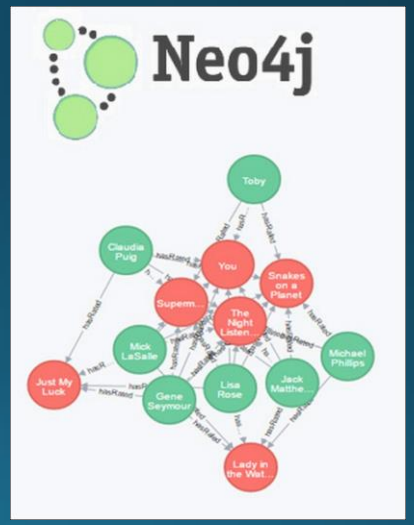
- User preferences may differ with the context, such as
  - time of day,
  - season,
  - mood,
  - place,
  - location,
  - options offered by the system, and so on.

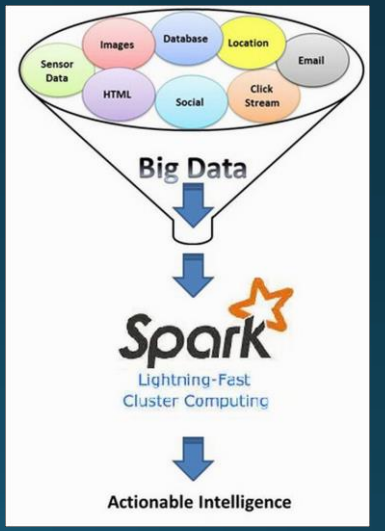
- A person at a different location at a different time with different people may need different things.

Advanced Recommendation Systems

- Scalable Recommenders:
  - Mahout, a machine-learning library built on the Hadoop platform provides infrastructure to build, evaluate, and tune the different types of recommendation-engine algorithms. Since Hadoop is designed for offline batch processing, we can build offline recommender systems, which are scalable.
- Scalable real-time Recommenders
  - "You may also like feature" requires generating personalized recommendations in real-time
  - Apache Spark Streaming leverages scalability of big data and generates recommendations in real time, and processes data in-memory.
- Graph-based recommender systems
  - graph databases allow us to store user and product information in graphs as nodes and edges (relations).
  - Neo4j, a NoSQL database







Building Recommender Systems in R

Small Simulated Dataset

Recommender - Small Data

1 Data

Here is some simulated data on movie ratings: 4 users x 6 movies

```
viewers = c('ViewerA','ViewerB','ViewerC','ViewerD')
movies = c('Beautiful_Mind','Goodwill_Hunting',
           'Remember_the_Titans','The_Rookie',
           'Jersey_Girl','Pursuit_of_Happyness')

data=data.frame()
set.seed(1)
for(viewer in viewers){
  random_number = round(runif(1,min = 1,max = 5))
  Viewer = rep(viewer,random_number)
  Movie = sample(movies,size = random_number,replace = F)
  Movie_rating = round(runif(random_number,min = 1,max = 5))
  data = rbind(data,data.frame(Viewer,Movie,Movie_rating))
}
data
```

Tall format

Viewer <fctr>	Movie <fctr>	Movie_rating <dbl>
ViewerA	The_Rookie	2
ViewerA	Beautiful_Mind	5
ViewerB	Remember_the_Titans	3
ViewerB	Goodwill_Hunting	4
ViewerB	Pursuit_of_Happyness	3
ViewerB	The_Rookie	4
ViewerB	Beautiful_Mind	5
ViewerC	Beautiful_Mind	2
ViewerC	Jersey_Girl	3
ViewerC	Pursuit_of_Happyness	1
ViewerD	Jersey_Girl	3
ViewerD	Goodwill_Hunting	3
ViewerD	Pursuit_of_Happyness	2

1.1 Restructure Data

Recast the data as realRatingMatrix

```
library(recommenderlab)
data_matrix = as(data,Class = 'realRatingMatrix')
as(data_matrix,'matrix')
```

```
##          Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerA               5                NA          NA                    NA
## ViewerB               5                 4          NA                     3
## ViewerC               2                NA           3                     1
## ViewerD              NA                 3           3                     2
##          Remember_the_Titans The_Rookie
## ViewerA                   NA          2
## ViewerB                    3          4
## ViewerC                   NA         NA
## ViewerD                   NA         NA
```

```
> data_matrix
4 x 6 rating matrix of class 'realRatingMatrix' with 13 ratings.
```

2 Explore Data

What rating did viewer B (ViewerB) given to The Beautiful Mind (Beautiful_Mind)

```
as(data_matrix,'matrix')['ViewerB','Beautiful_Mind']
```

```
## [1] 5
```

How many movies did viewer B (ViewerB) rate?

```
Row_counts = rowCounts(data_matrix['ViewerB',])
Row_counts
```

```
## ViewerB
##       5
```

```
> Row_counts_all = rowCounts(data_matrix)
> Row_counts_all
ViewerA ViewerB ViewerC ViewerD
      2       5       3       3
```

How many viewer ratings did The Beautiful Mind (Beautiful_Mind) receive?

```
colCounts(data_matrix[,'Beautiful_Mind'])
```

```
## Beautiful_Mind
##              3
```

```
> Col_counts_all = colCounts(data_matrix)
> Col_counts_all
      Beautiful_Mind     Goodwill_Hunting          Jersey_Girl Pursuit_of_Happyness
                   3                    2                    2                    3
 Remember_the_Titans          The_Rookie
                   1                    2
```

```
> ?`rowCounts,ratingMatrix-method`
>
```

Files | Plots | Packages | Help | Viewer

R: Class "ratingMatrix": Virtual Class for Rating Data — Find in Topic

colCounts

signature(x = "ratingMatrix"): number of ratings per column.

rowCounts

signature(x = "ratingMatrix"): number of ratings per row.

colMeans

signature(x = "ratingMatrix"): column-wise rating means.

rowMeans

signature(x = "ratingMatrix"): row-wise rating means.

What is the most common rating in the dataset?

```
table(getRatings(data_matrix))
```

```
##
## 1 2 3 4 5
## 1 3 5 2 2
```

What is the average rating of each movie?

```
colMeans(data_matrix)
```

```
##       Beautiful_Mind     Goodwill_Hunting          Jersey_Girl
##                  4.0                  3.5                  3.0
## Pursuit_of_Happyness  Remember_the_Titans          The_Rookie
##                  2.0                  3.0                  3.0
```

For each movie, What is the sum of ratings?

```
colSums(data_matrix)
```

```
##       Beautiful_Mind     Goodwill_Hunting          Jersey_Girl
##                   12                    7                    6
## Pursuit_of_Happyness  Remember_the_Titans          The_Rookie
##                    6                    3                    6
```

colMeans

`signature(x = "ratingMatrix")`: column-wise rating means.

rowMeans

`signature(x = "ratingMatrix")`: row-wise rating means.

Average rating for the Beautiful Mind

```
> colMeans(data_matrix['Beautiful_Mind'])
Beautiful_Mind
             4
```

colMeans

`signature(x = "ratingMatrix")`: column-wise rating means.

rowMeans

`signature(x = "ratingMatrix")`: row-wise rating means.

Rating patterns of viewer. What is average rating for each viewer? Do they vary?

```
rowMeans(data_matrix)
```

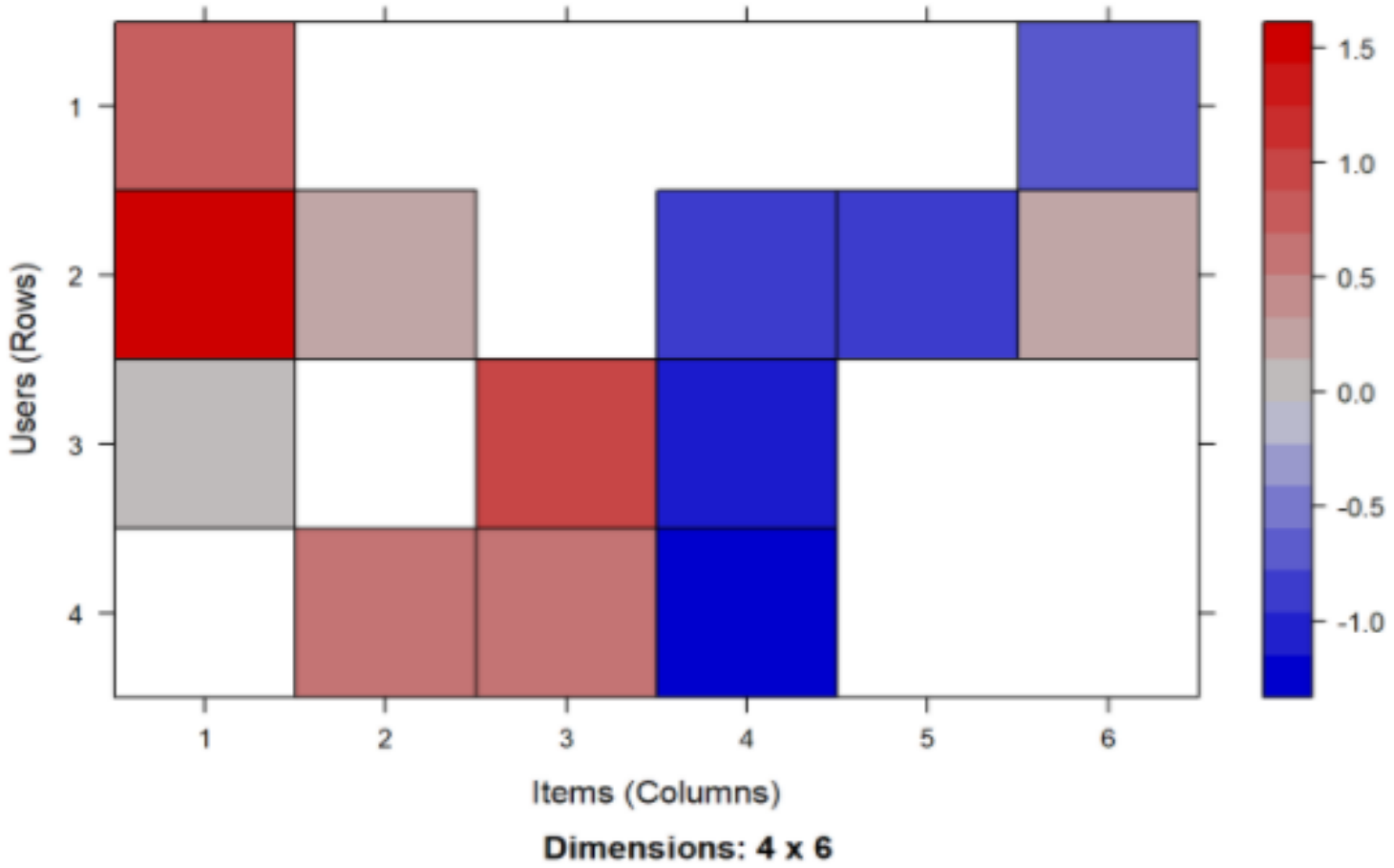
```
##  ViewerA  ViewerB  ViewerC  ViewerD
## 3.500000 3.800000 2.000000 2.666667
```

```
rowSds(data_matrix)
```

```
##   ViewerA   ViewerB   ViewerC   ViewerD
## 2.1213203 0.8366600 1.0000000 0.5773503
```

Since, users tend to vary in their pattern of evaluation (positivity bias, negativity bias, consistent ratings), it is a good practice to center (method = 'center') or standardize data by computing z score (method = 'z-score'). White space in the image indicating NAs

```
data_matrix2 = normalize(data_matrix,method='Z-score')
image(data_matrix2)
```



Dimensions: 4 x 6

```
normalize(x, ...)
## S4 method for signature 'realRatingMatrix'
normalize(x, method="center", row=TRUE)
```

Using the normalized user ratings generated above, assess the similarity between the first three viewers (ViewerA, ViewerB, ViewerC) over the first three items in the dataset using cosine similarity.

Which of the following pairs is most similar?

```
similarity(data_matrix2[1:3,1:3],method='cosine')
```

```
##         ViewerA ViewerB
## ViewerB       1
## ViewerC       0       0
```

Using the normalized user ratings generated above, assess the similarity between the first three viewers (ViewerA, ViewerB, ViewerC) over all items in the dataset using cosine similarity.

Which of the following pairs is most similar?

```
similarity(data_matrix2[1:3],method='cosine')
```

```
##           ViewerA   ViewerB
## ViewerB 0.5812382
## ViewerC 0.0000000 0.5547002
```

```
> recommenderRegistry$get_entry_names()
 [1] "ALS_realRatingMatrix"             "ALS_implicit_realRatingMatrix"
 [3] "ALS_implicit_binaryRatingMatrix" "AR_binaryRatingMatrix"
 [5] "IBCF_binaryRatingMatrix"          "IBCF_realRatingMatrix"
 [7] "LIBMF_realRatingMatrix"           "POPULAR_binaryRatingMatrix"
 [9] "POPULAR_realRatingMatrix"         "RANDOM_realRatingMatrix"
[11] "RANDOM_binaryRatingMatrix"        "RERECOMMEND_realRatingMatrix"
[13] "RERECOMMEND_binaryRatingMatrix"  "SVD_realRatingMatrix"
[15] "SVDF_realRatingMatrix"            "UBCF_binaryRatingMatrix"
[17] "UBCF_realRatingMatrix"
```

3 Non-Personalized Recommendation Systems

```
recommenderRegistry$get_entry("POPULAR", type ="realRatingMatrix")
```

```
## Recommender method: POPULAR for realRatingMatrix
## Description: Recommender based on item popularity.
## Reference: NA
## Parameters:
##   normalize
## 1  "center"
##                                                  aggregationRatings
## 1 new("standardGeneric", .Data = function (x, na.rm = FALSE, dims = 1,
##                                               aggregationPopularity
## 1 new("standardGeneric", .Data = function (x, na.rm = FALSE, dims = 1,
```

```
recom = Recommender(data_matrix,method='POPULAR',
                    parameter = list(normalize=NULL))
pred = predict(recom,data_matrix)
as(pred,'matrix')
```

```
##         Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerA             NA               3.5           3                     2
## ViewerB             NA                NA           3                    NA
## ViewerC             NA               3.5          NA                    NA
## ViewerD              4                NA          NA                    NA
##         Remember_the_Titans The_Rookie
## ViewerA                   3         NA
## ViewerB                  NA         NA
## ViewerC                   3          3
## ViewerD                   3          3
```

You will note that the imputed ratings above are simply average ratings for each movie.

```
colMeans(data_matrix)
```

```
##       Beautiful_Mind     Goodwill_Hunting         Jersey_Girl
##                  4.0                 3.5                 3.0
## Pursuit_of_Happyness  Remember_the_Titans          The_Rookie
##                  2.0                 3.0                 3.0
```

So, what is our top movie recommendation for each viewer? Obviously, the one with the highest rating!

```
pred = predict(recom,data_matrix,n=1)
getList(pred)
```

```
## $ViewerA
## [1] "Goodwill_Hunting"
##
## $ViewerB
## [1] "Jersey_Girl"
##
## $ViewerC
## [1] "Goodwill_Hunting"
##
## $ViewerD
## [1] "Beautiful_Mind"
```

Recall:

```
as(data_matrix,'matrix')
```

```
##         Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerA              5                NA          NA                    NA
## ViewerB              5                 4          NA                     3
## ViewerC              2                NA           3                     1
## ViewerD             NA                 3           3                     2
##         Remember_the_Titans The_Rookie
## ViewerA                  NA          2
## ViewerB                   3          4
## ViewerC                  NA         NA
## ViewerD                  NA         NA
```

```
recom = Recommender(data_matrix,method='POPULAR',
                    parameter = list(normalize=NULL))
pred = predict(recom,data_matrix)
as(pred,'matrix')
```

```
##         Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerA             NA               3.5           3                     2
## ViewerB             NA                NA           3                    NA
## ViewerC             NA               3.5          NA                    NA
## ViewerD              4                NA          NA                    NA
##         Remember_the_Titans The_Rookie
## ViewerA                   3         NA
## ViewerB                  NA         NA
## ViewerC                   3          3
## ViewerD                   3          3
```

Can we recommend top 2 movies? (Note that if it may not be possible to recommend top 2 if there is only one unwatched movie)

```
pred = predict(recom,data_matrix,n=2)
getList(pred)
```

```
## $ViewerA
## [1] "Goodwill_Hunting" "Jersey_Girl"
##
## $ViewerB
## [1] "Jersey_Girl"
##
## $ViewerC
## [1] "Goodwill_Hunting" "The_Rookie"
##
## $ViewerD
## [1] "Beautiful_Mind" "The_Rookie"
```

Recall:


```
as(data_matrix,'matrix')
```

```
##         Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerA              5                NA          NA                    NA
## ViewerB              5                 4          NA                     3
## ViewerC              2                NA           3                     1
## ViewerD             NA                 3           3                     2
##         Remember_the_Titans The_Rookie
## ViewerA                  NA          2
## ViewerB                   3          4
## ViewerC                  NA         NA
## ViewerD                  NA         NA
```

```
recom = Recommender(data_matrix,method='POPULAR',
                    parameter = list(normalize=NULL))
pred = predict(recom,data_matrix)
as(pred,'matrix')
```

```
##         Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerA             NA               3.5           3                     2
## ViewerB             NA                NA           3                    NA
## ViewerC             NA               3.5          NA                    NA
## ViewerD              4                NA          NA                    NA
##         Remember_the_Titans The_Rookie
## ViewerA                   3         NA
## ViewerB                  NA         NA
## ViewerC                   3          3
## ViewerD                   3          3
```

Which of the following is in the list of top 2 movies for viewer A?

```
as(pred,'list')['ViewerA']
```

```
## $ViewerA
## [1] "Goodwill_Hunting" "Jersey_Girl"
```

What is the predicted rating for The Beautiful Mind by viewer A?

```
pred = predict(recom,data_matrix,type='ratings')
as(pred,'matrix')['ViewerA','Beautiful_Mind']
```

```
## [1] NA
```

```
> pred = predict(recom,data_matrix,type='ratings')
> as(pred,'matrix')['ViewerD','Beautiful_Mind']
[1] 4
```

4 User-based Collaborative Filtering

Let's use cosine similarity measure, set nearest neighbors to 3, and set normalize to NULL

```
recommenderRegistry$get_entry(method='UBCF', type ='realRatingMatrix')
```

```
## Recommender method: UBCF for realRatingMatrix
## Description: Recommender based on user-based collaborative filtering.
## Reference: NA
## Parameters:
##     method nn sample weighted normalize min_matching_items min_predictive_items
## 1 "cosine" 25  FALSE     TRUE  "center"                   0                   0
```

If using defaults for parameters in Recommender function:

Recommender(data_matrix ,method='UBCF')

```
recom = Recommender(data_matrix,method='UBCF',
                    parameter=list(method='cosine',nn=3,normalize=NULL))
pred = predict(recom,data_matrix,type = 'ratings')
as(pred,'matrix')
```

```
##         Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerA             NA                4           3                1.97804
## ViewerB             NA               NA           3                     NA
## ViewerC             NA                4          NA                     NA
## ViewerD       3.525746               NA          NA                     NA
##         Remember_the_Titans The_Rookie
## ViewerA                   3         NA
## ViewerB                  NA         NA
## ViewerC                   3   2.998525
## ViewerD                   3   4.000000
```

Which movie to watch? Here is the top movie recommendation for each user.

```
pred = predict(recom, data_matrix, n = 1)
getList(pred)
```

```
## $ViewerA
## [1] "Goodwill_Hunting"
##
## $ViewerB
## [1] "Jersey_Girl"
##
## $ViewerC
## [1] "Goodwill_Hunting"
##
## $ViewerD
## [1] "The_Rookie"
```

Now, let us examine how much each user will like the recommended movie.

```
getRatings(pred)
```

```
## $ViewerA
## [1] 4
##
## $ViewerB
## [1] 3
##
## $ViewerC
## [1] 4
##
## $ViewerD
## [1] 4
```

Recall:


```
as(data_matrix,'matrix')
```

```
##         Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerA              5                NA          NA                    NA
## ViewerB              5                 4          NA                     3
## ViewerC              2                NA           3                     1
## ViewerD             NA                 3           3                     2
##         Remember_the_Titans The_Rookie
## ViewerA                  NA          2
## ViewerB                   3          4
## ViewerC                  NA         NA
## ViewerD                  NA         NA
```

```
recom = Recommender(data_matrix,method='UBCF',
                    parameter=list(method='cosine',nn=3,normalize=NULL))
pred = predict(recom,data_matrix,type = 'ratings')
as(pred,'matrix')
```

```
##         Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerA             NA                4           3                1.97804
## ViewerB             NA               NA           3                     NA
## ViewerC             NA                4          NA                     NA
## ViewerD       3.525746               NA          NA                     NA
##         Remember_the_Titans The_Rookie
## ViewerA                   3         NA
## ViewerB                  NA         NA
## ViewerC                   3   2.998525
## ViewerD                   3   4.000000
```

A recommendation is only good if it is better than what the user would normally have watched. So, let us examine the average ratings for each movie watched by the four users.

```
rowMeans(data_matrix)
```

```
##  ViewerA  ViewerB  ViewerC  ViewerD
## 3.500000 3.800000 2.000000 2.666667
```

Now, let us put it all together

```
cbind(Recommended_Movie = getList(pred),
      UBCF_Predicted_Rating = getRatings(pred),
      Avg_User_Rating = rowMeans(data_matrix))
```

```
##         Recommended_Movie  UBCF_Predicted_Rating Avg_User_Rating
## ViewerA "Goodwill_Hunting" 4                     3.5
## ViewerB "Jersey_Girl"      3                     3.8
## ViewerC "Goodwill_Hunting" 4                     2
## ViewerD "The_Rookie"       4                     2.666667
```

5 Item-based Collaborative Filtering


```
recommenderRegistry$get_entry(method='IBCF', type ='realRatingMatrix')
```

```
## Recommender method: IBCF for realRatingMatrix
## Description: Recommender based on item-based collaborative filtering.
## Reference: NA
## Parameters:
##    k   method normalize normalize_sim_matrix alpha na_as_zero
## 1 30 "Cosine"  "center"                FALSE   0.5      FALSE
```

```
recom = Recommender(data_matrix,method='IBCF',
                    parameter=list(method='cosine',k=3,normalize=NULL))
pred = predict(recom,data_matrix,type = 'ratings')
as(pred,'matrix')
```

```
##         Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerA             NA                2    5.000000                     2
## ViewerB             NA               NA    4.017403                    NA
## ViewerC             NA                3          NA                    NA
## ViewerD              3               NA          NA                    NA
##         Remember_the_Titans The_Rookie
## ViewerA                 2.0         NA
## ViewerB                  NA         NA
## ViewerC                 1.0        1.0
## ViewerD                 2.5        2.5
```

Which movie to watch? Here is the top movie recommendation for each user.

```
pred = predict(recom, data_matrix, n = 1)
getList(pred)
```

```
## $ViewerA
## [1] "Jersey_Girl"
##
## $ViewerB
## [1] "Jersey_Girl"
##
## $ViewerC
## [1] "Goodwill_Hunting"
##
## $ViewerD
## [1] "Beautiful_Mind"
```

Now, let us examine how much each user will like the recommended movie.

```
getRatings(pred)
```

```
## $ViewerA
## [1] 5
##
## $ViewerB
## [1] 4.017403
##
## $ViewerC
## [1] 3
##
## $ViewerD
## [1] 3
```

Finally, as noted above, a recommendation is only good if it is better than what the user would normally have watched. So, let us examine the average ratings for each movie watched by the four users.

```
cbind(Recommended_Movie = getList(pred),
      IBCF_Predicted_Rating = getRatings(pred),
      Avg_User_Rating = rowMeans(data_matrix))
```

```
##         Recommended_Movie  IBCF_Predicted_Rating Avg_User_Rating
## ViewerA "Jersey_Girl"      5                     3.5
## ViewerB "Jersey_Girl"      4.017403              3.8
## ViewerC "Goodwill_Hunting" 3                     2
## ViewerD "Beautiful_Mind"   3                     2.666667
```

6 Hybrid Recommender

Setup the recommenders

```
rec_pop = Recommender(data_matrix, "POPULAR",
                      parameter = list(normalize=NULL))
rec_ub = Recommender(data_matrix,'UBCF',
                     parameter=list(method='Cosine',nn=3,normalize=NULL))
rec_ib =  Recommender(data_matrix, "IBCF",
                      parameter = list(method='Cosine',k=3,normalize=NULL))
#Save recommender models for later so we do not have to rebuild them
save(rec_pop, rec_ub, rec_ib, file = "models.rda")
# Load models
load("models.rda")
```

Create predictions. Now, let us examine predicted ratings using each recommender but only for the fourth user, Viewer D

```
pred_pop = predict(rec_pop, data_matrix[4,], type = "ratings")
as(pred_pop,'list')
```

```
## $ViewerD
##      Beautiful_Mind Remember_the_Titans          The_Rookie
##                   4                    3                   3
```

```
pred_ub  = predict(rec_ub, data_matrix[4,], type = "ratings")
as(pred_ub,'list')
```

```
## $ViewerD
##      Beautiful_Mind Remember_the_Titans          The_Rookie
##            3.525746             3.000000            4.000000
```

```
pred_ib  = predict(rec_ib, data_matrix[4,], type = "ratings")
as(pred_ib, 'list')
```

```
## $ViewerD
##      Beautiful_Mind Remember_the_Titans          The_Rookie
##                 3.0                  2.5                 2.5
```

Aggregate predictions

```
pred_hybrid = rbind(
  as(pred_pop, 'matrix'),
  as(pred_ub,'matrix'),
  as(pred_ib, 'matrix'))
pred_hybrid
```

```
##         Beautiful_Mind Goodwill_Hunting Jersey_Girl Pursuit_of_Happyness
## ViewerD       4.000000                NA          NA                    NA
## ViewerD       3.525746                NA          NA                    NA
## ViewerD       3.000000                NA          NA                    NA
##         Remember_the_Titans The_Rookie
## ViewerD                 3.0        3.0
## ViewerD                 3.0        4.0
## ViewerD                 2.5        2.5
```

```
pred_hybrid = colMeans(pred_hybrid)
pred_hybrid
```

```
##       Beautiful_Mind     Goodwill_Hunting          Jersey_Girl
##             3.508582                   NA                   NA
## Pursuit_of_Happyness  Remember_the_Titans          The_Rookie
##                   NA             2.833333            3.166667
```

Create a top-N list from the aggregated ratings

```
pred <- getTopNLists(as(rbind(pred_hybrid), "realRatingMatrix"))
as(pred, 'list')
```

```
## $pred_hybrid
## [1] "Beautiful_Mind"      "The_Rookie"          "Remember_the_Titans"
```

Compare with the results by the individual recommenders

```
as(getTopNLists(pred_pop), 'list')
```

```
## $ViewerD
## [1] "Beautiful_Mind"      "Remember_the_Titans" "The_Rookie"
```

```
as(getTopNLists(pred_ib), 'list')
```

```
## $ViewerD
## [1] "Beautiful_Mind"      "Remember_the_Titans" "The_Rookie"
```

```
as(getTopNLists(pred_ub), 'list')
```

```
## $ViewerD
## [1] "The_Rookie"          "Beautiful_Mind"      "Remember_the_Titans"
```

Building Recommender Systems in R

Jetster Dataset (5k Sample)

1 Read Data

2 Get Familiar with the Data and Structure

3 Split Data

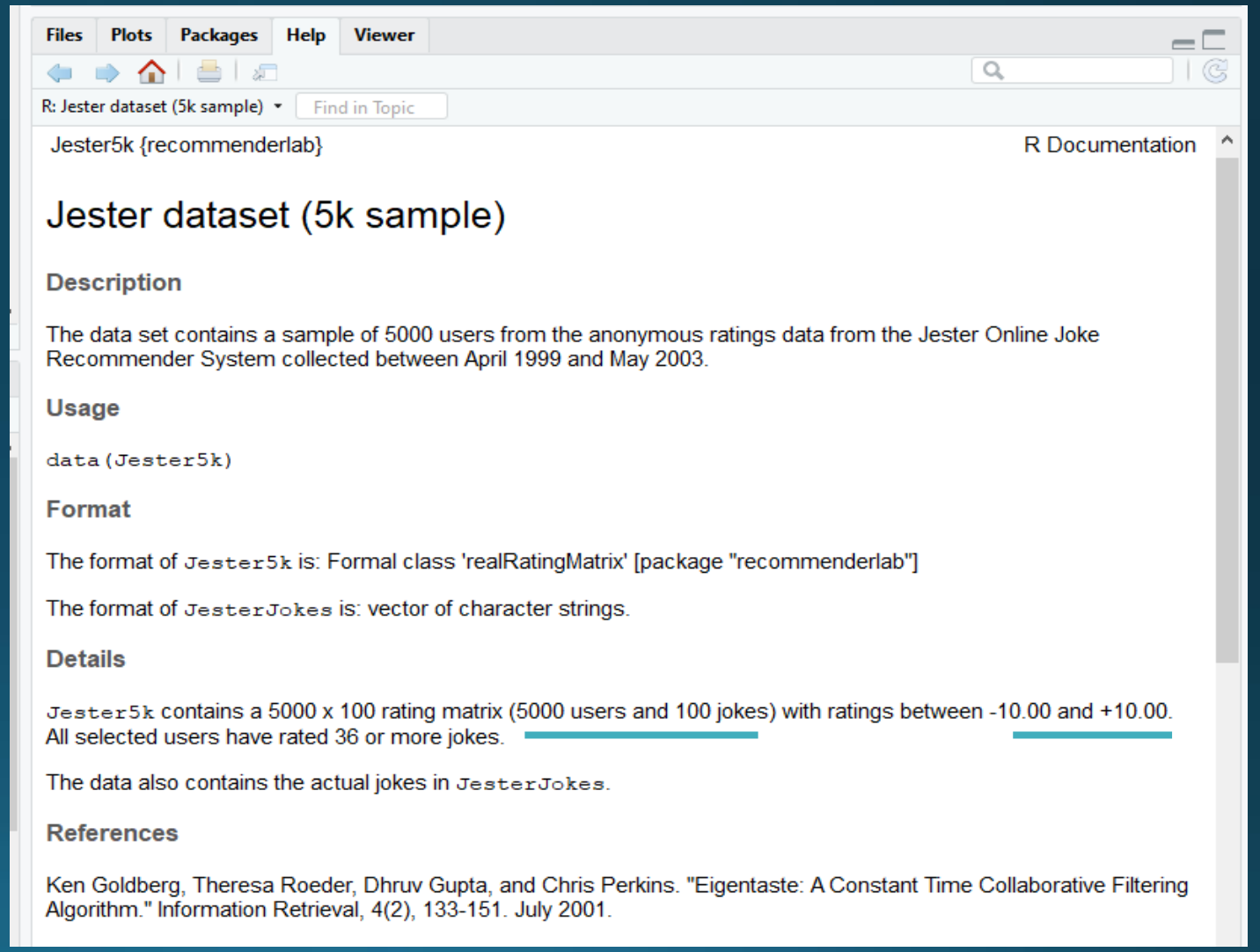
4 Explore Train data

5 Prepare Data

6 Similarity

7 Construct Recommendations

8 Evaluation Scheme



Recommender System

This note will review recommender systems that are categorized as collaborative filtering. The recommenderlab package offers an efficient set of functions to implement these recommenders. recommenderlab is implemented using formal classes in the S4 class system. For details, see Hahsler (2018) https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf. This package was updated recently, so even if you have installed it before, it is best to reinstall it.

```
# install.packages('recommenderlab')
library(recommenderlab)
```

1 Read Data

Use a dataset of jokes that comes with recommenderlab package. The data set contains a sample of 5000 users from the anonymous ratings data from the Jester Online Joke Recommender System collected between April 1999 and May 2003.

```
data(Jester5k)
```

Here is one of the jokes.

```
JesterJokes[[48]]
```

```
## [1] "The graduate with a Science degree asks, \"Why does it work?\" The graduate with an Engineering degree asks, \"How does it work?\" The graduate with an Accounting degree Asks, \"How much will it cost?\" The graduate with a Liberal Arts degree asks, \"Do you want fries with that?\""
```

Data is stored as a realRatingMatrix

How many ratings?

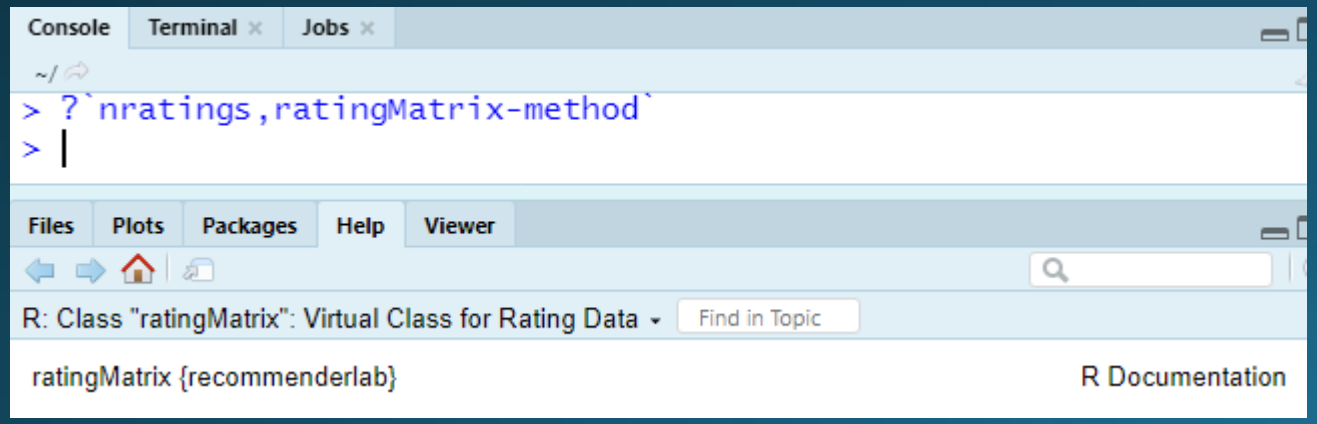
```
Jester5k
```

```
## 5000 x 100 rating matrix of class 'realRatingMatrix' with 362106 ratings.
```

User-item matrices are often sparse. Some users rate more than others and some items get rated more than others but few or no items are rated by all and few or no users rate everything. Thus, the user-item matrix contains a number of missing values. Jester5k is a sparse matrix with only 72% elements rated.

```
nratings(Jester5k)/(ncol(Jester5k)*nrow(Jester5k))
```

```
## [1] 0.724212
```



nratings

`signature(x = "ratingMatrix")`: number of ratings in the matrix.

Using a realRatingMatrix makes a number of recommenderlab functions accessible.

```
methods(class = 'realRatingMatrix')
```

```
##  [1] [                      [<-                    binarize
##  [4] calcPredictionAccuracy coerce                 colCounts
##  [7] colMeans               colSds                 colSums
## [10] denormalize            dim                    dimnames
## [13] dimnames<-             dissimilarity          evaluationScheme
## [16] getData.frame          getList                getNormalize
## [19] getRatingMatrix        getRatings             getTopNLists
## [22] hasRating              image                  normalize
## [25] nratings               Recommender            removeKnownRatings
## [28] rowCounts              rowMeans               rowSds
## [31] rowSums                sample                 show
## [34] similarity
## see '?methods' for accessing help and source code
```

and also a number of recommender models

```
names(recommenderRegistry$get_entries(dataType='realRatingMatrix'))
```

```
##  [1] "HYBRID_realRatingMatrix"       "ALS_realRatingMatrix"
##  [3] "ALS_implicit_realRatingMatrix" "IBCF_realRatingMatrix"
##  [5] "LIBMF_realRatingMatrix"        "POPULAR_realRatingMatrix"
##  [7] "RANDOM_realRatingMatrix"       "RERECOMMEND_realRatingMatrix"
##  [9] "SVD_realRatingMatrix"          "SVDF_realRatingMatrix"
## [11] "UBCF_realRatingMatrix"
```

2 Get Familiar with the Data and Structure


```
Jester5k
```

```
## 5000 x 100 rating matrix of class 'realRatingMatrix' with 362106 ratings.
```

Dimensions of this realRatingMatrix

```
dim(Jester5k)
```

```
## [1] 5000  100
```

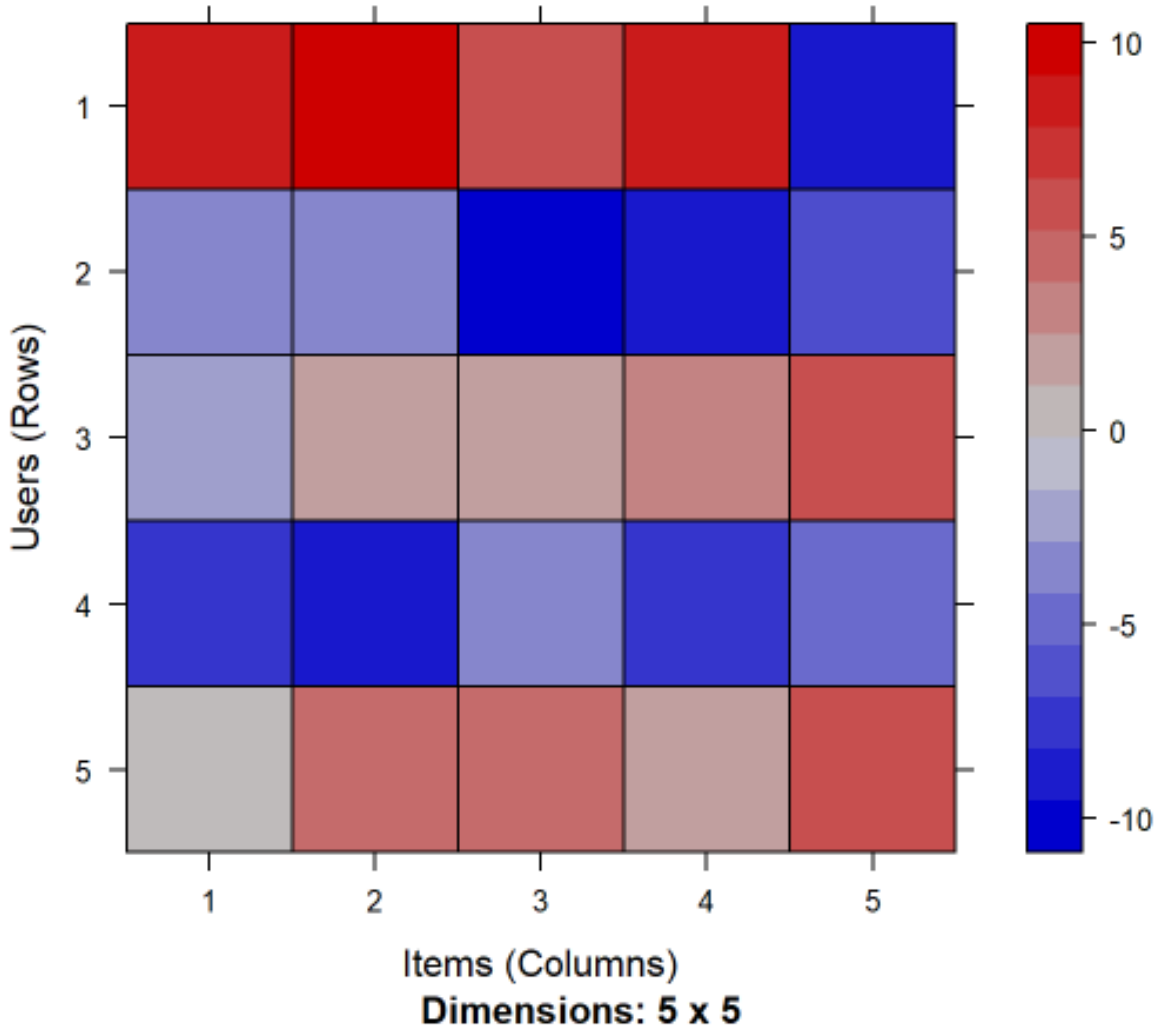
structure of the object

```
str(Jester5k)
```

```
## Formal class 'realRatingMatrix' [package "recommenderlab"] with 2 slots
##   ..@ data     :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
##   .. .. ..@ i       : int [1:362106] 0 1 2 3 4 5 6 7 8 9 ...
##   .. .. ..@ p       : int [1:101] 0 3314 6962 10300 13442 18440 22513 27512 32512 35685 ...
##   .. .. ..@ Dim     : int [1:2] 5000 100
##   .. .. ..@ Dimnames:List of 2
##   .. .. .. ..$ : chr [1:5000] "u2841" "u15547" "u15221" "u15573" ...
##   .. .. .. ..$ : chr [1:100] "j1" "j2" "j3" "j4" ...
##   .. .. ..@ x       : num [1:362106] 7.91 -3.2 -1.7 -7.38 0.1 0.83 2.91 -2.77 -3.35 -1.99 ...
##   .. .. ..@ factors : list()
##   ..@ normalize: NULL
```

Implementation of image by recommenderlab generates a heatmap.

```
image(Jester5k[1:5,1:5])
```



To view the raw data, we transform it to a matrix form. You will note that a realRatingMatrix places users along rows and items along columns.

```
as(Jester5k,'matrix')[1:5, 1:5]
```

```
##           j1    j2    j3    j4    j5
## u2841   7.91  9.17  5.34  8.16 -8.74
## u15547 -3.20 -3.50 -9.56 -8.74 -6.36
## u15221 -1.70  1.21  1.55  2.77  5.58
## u15573 -7.38 -8.93 -3.88 -7.23 -4.90
## u21505  0.10  4.17  4.90  1.55  5.53
```

Another alternative for viewing raw data is to use the getRatingMatrix from recommenderlab

```
getRatingMatrix(Jester5k)[1:5,1:5]
```

```
## 5 x 5 sparse Matrix of class "dgCMatrix"
##           j1    j2    j3    j4    j5
## u2841   7.91  9.17  5.34  8.16 -8.74
## u15547 -3.20 -3.50 -9.56 -8.74 -6.36
## u15221 -1.70  1.21  1.55  2.77  5.58
## u15573 -7.38 -8.93 -3.88 -7.23 -4.90
## u21505  0.10  4.17  4.90  1.55  5.53
```

Let us extract rating of Jokes 56 to 60 by u7676. Once data is converted to a matrix format, traditional methods for subsetting a matrix can be employed.

```
as(Jester5k,'matrix')['u7676',c('j56','j57','j58','j59','j60')]
```

```
##  j56  j57  j58  j59  j60
## 5.34   NA   NA   NA 4.22
```

From the foregoing lines of code, it must be obvious that we can extract data by applying recommenderlab functions to a realRatingMatrix or convert the latter to matrix and then apply traditional functions. Let us answer the following questions using each method.

Q: How many jokes did u7676 rate?

A: Using recommenderlab functions

```
nratings(Jester5k['u7676'])
```

```
## [1] 60
```

A: Using traditional ways

```
sum(!is.na(as(Jester5k,'matrix')['u7676',]))
```

```
## [1] 60
```

Q: What was the mean rating of jokes by u7676?

A: Using recommenderlab functions

```
mean(getRatings(Jester5k['u7676']))
```

```
## [1] 3.476
```

A: Using traditional ways

```
mean(as(Jester5k,'matrix')['u7676',],na.rm=T)
```

```
## [1] 3.476
```

You will note that the realRatingMatrix is not a tidy format. Here is what the data would look like in a tidy format. Shown below is the head of a tidy dataset.

```
library(tidyr); library(tibble)
data.frame(as(Jester5k,'matrix'))%>%
  rownames_to_column(var = 'user')%>%
  gather(key = 'joke',value = rating,2:101)%>%
  head()
```

	user <chr>	joke <chr>	rating <dbl>
1	u2841	j1	7.91
2	u15547	j1	-3.20
3	u15221	j1	-1.70
4	u15573	j1	-7.38
5	u21505	j1	0.10
6	u15994	j1	0.83

6 rows

3 Split Data

Split the dataset, retaining 80% in the train sample

```
set.seed(617)
split = sample(x = nrow(Jester5k),size = 0.8*nrow(Jester5k))
train = Jester5k[split,]
test = Jester5k[-split,]
```

4 Explore Train data

Let us examine the number of jokes rated by users. What can you say about the number of jokes rated by each user and the number of jokes rated by most users?

```
summary(rowCounts(train))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   36.00   53.75   72.00   72.48  100.00  100.00
```

```
> ?`rowCounts,ratingMatrix-method`
>
```

Files | Plots | Packages | Help | Viewer

R: Class "ratingMatrix": Virtual Class for Rating Data — Find in Topic

colCounts

`signature(x = "ratingMatrix")`: number of ratings per column.

rowCounts

`signature(x = "ratingMatrix")`: number of ratings per row.

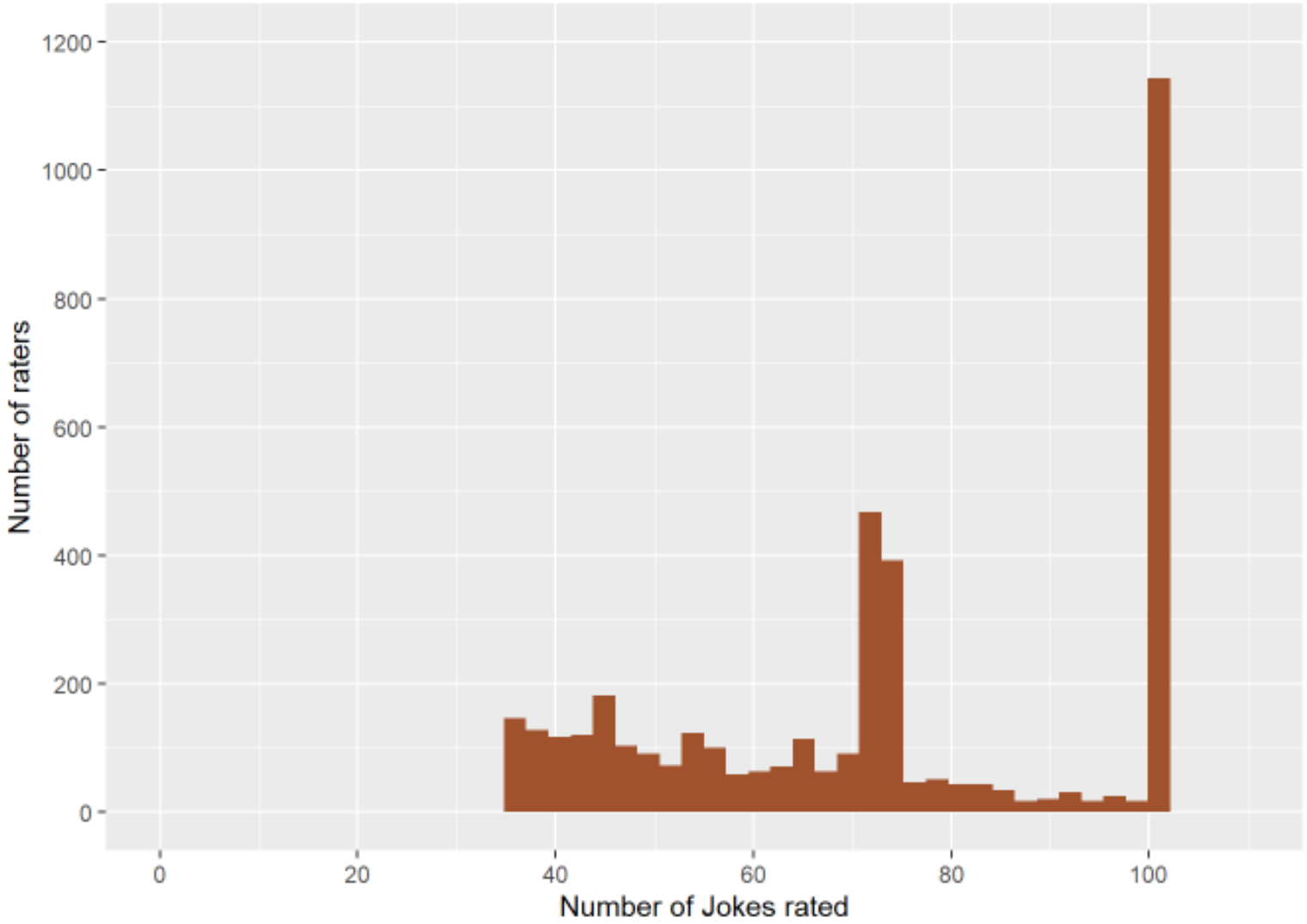
colMeans

`signature(x = "ratingMatrix")`: column-wise rating means.

rowMeans

`signature(x = "ratingMatrix")`: row-wise rating means.

```
library(ggplot2)
ggplot(data=data.frame(no_of_jokes_rated = rowCounts(train)),aes(x=no_of_jokes_rated))+
  geom_histogram(bins=50,fill='sienna')+ylab('Number of raters')+xlab('Number of Jokes rated')+
  scale_x_continuous(breaks = seq(0,100,20),limits=c(0,110))+
  scale_y_continuous(breaks=seq(0,1200,200),limits=c(0,1200))
```

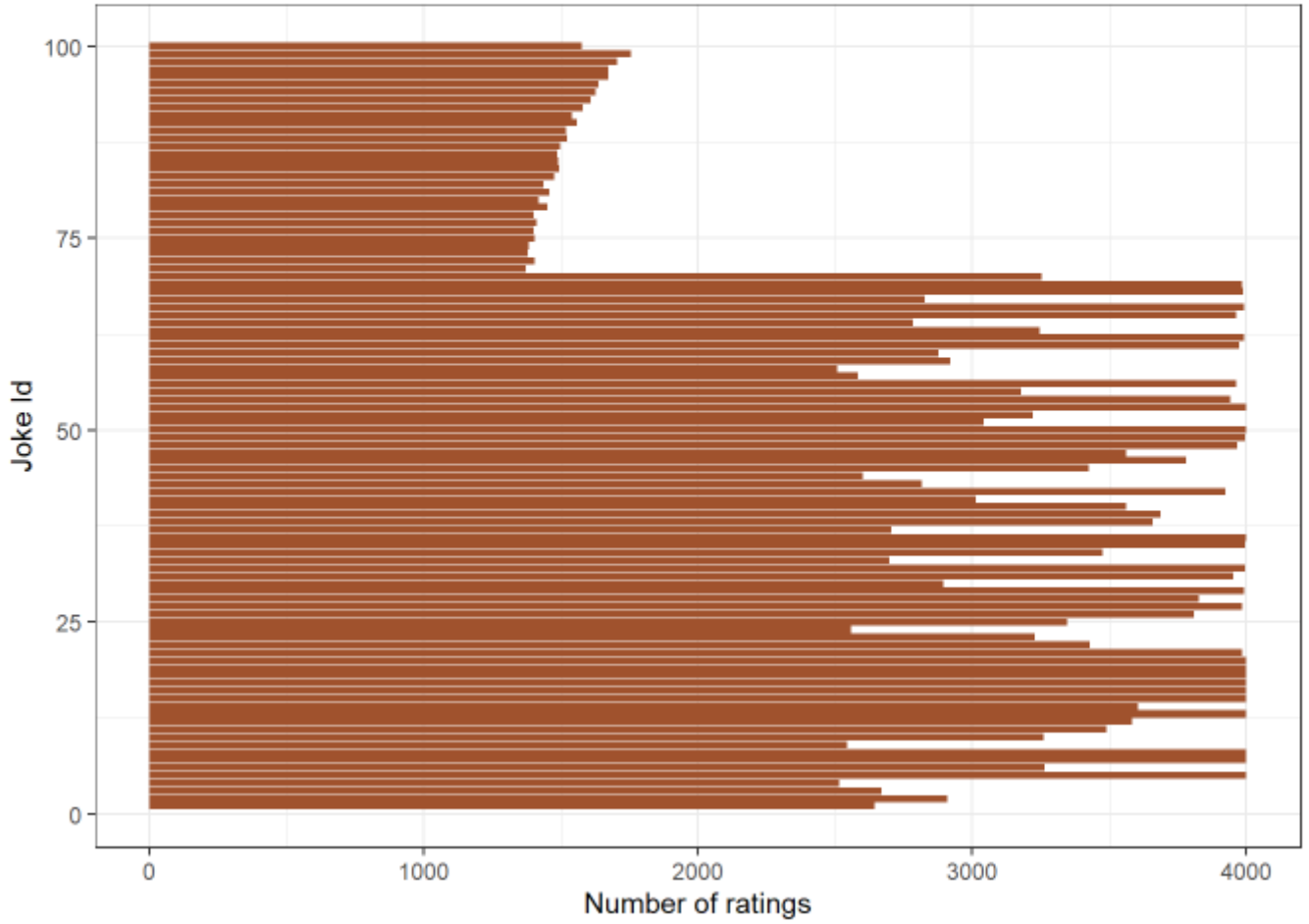


Now, let us examine the number of ratings jokes received. What can you say about the number of ratings each jokes received? Did all jokes get rated the same number of times?

```
summary(colCounts(train))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1373    1634    3199    2899    3964    4000
```

```
library(tidyr);library(tibble); library(ggthemes)
data.frame(no_of_ratings=colCounts(train))%>%
  rownames_to_column()%>%
  ggplot(aes(x=as.numeric(gsub(rowname,pattern = '[a-z]',replacement = '')), y=no_of_ratings))+
  geom_col(fill='sienna')+xlab('Joke Id')+ylab('Number of ratings')+coord_flip()+ theme_bw()
```



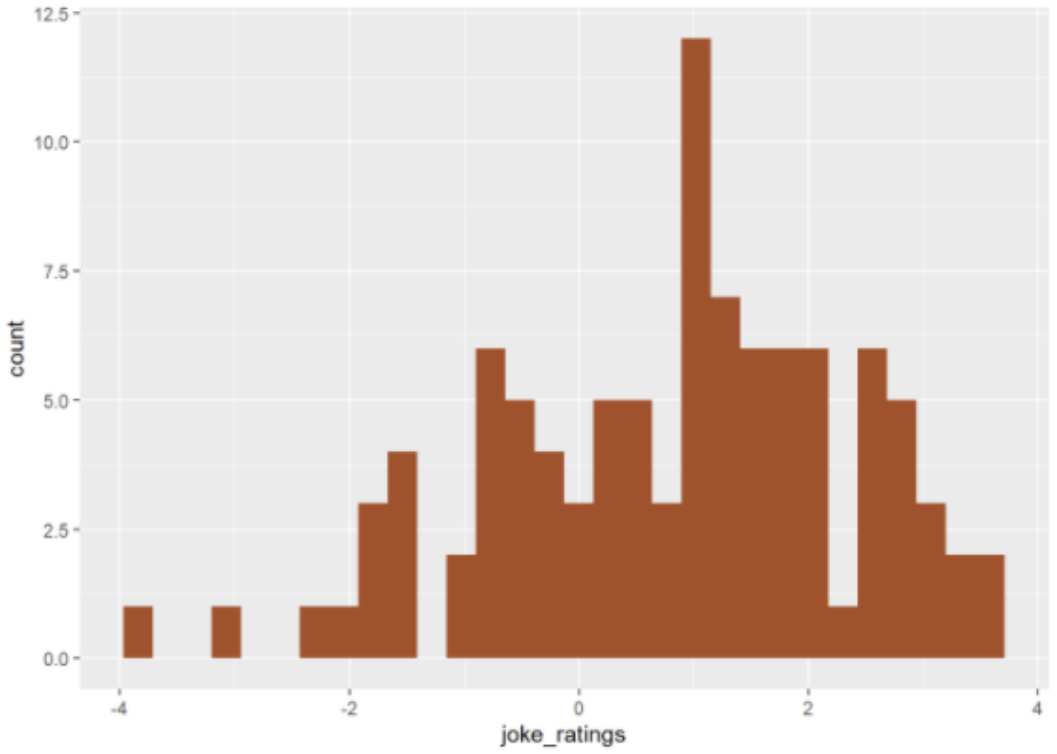
Let us examine the average rating of the first six Jokes

```
head(colMeans(train))
```

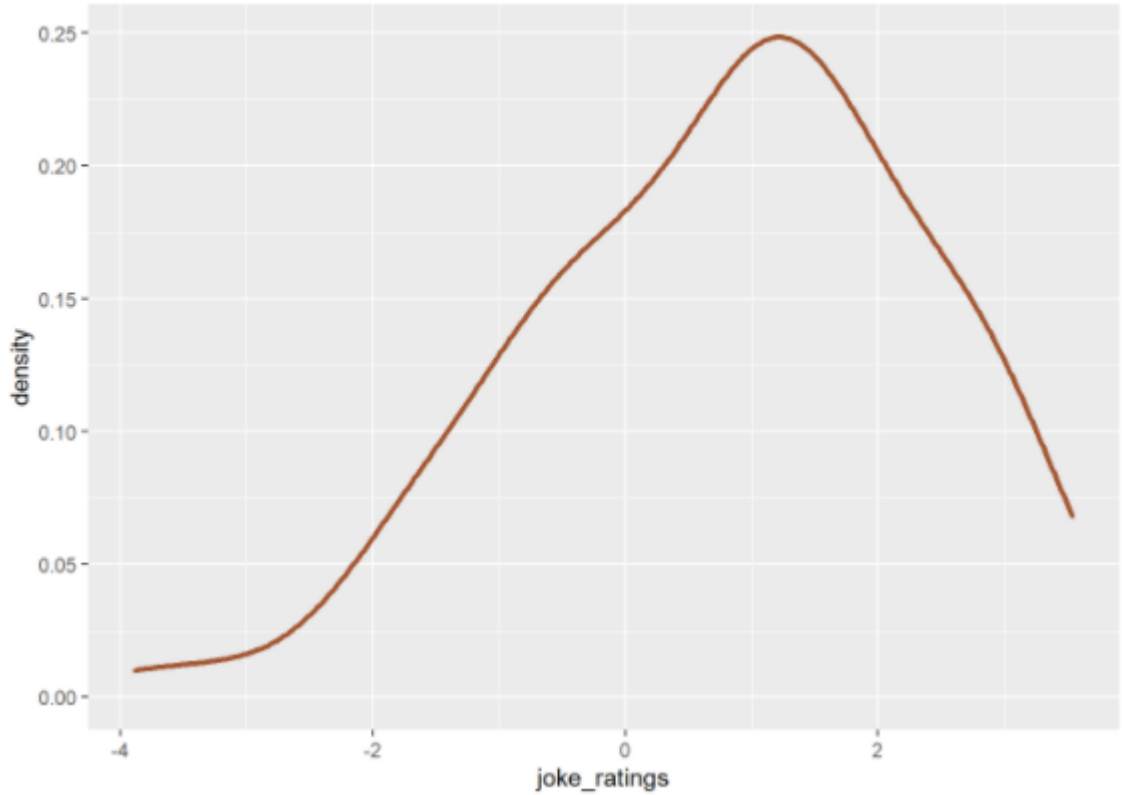
```
##         j1         j2         j3         j4         j5         j6
##  0.9440469  0.2282680  0.2021386 -1.5022554  0.3858890  1.6987688
```

Let us examine the distribution of average ratings of the 100 Jokes. Are some jokes rated better than others? Are ratings of jokes skewed to the high end of the scale?

```
library(ggplot2)
ggplot(data=data.frame(joke_ratings = colMeans(train)),aes(x=joke_ratings))+
  geom_histogram(fill='sienna')
```




```
ggplot(data=data.frame(joke_ratings = colMeans(train)),aes(x=joke_ratings))+
  geom_density(color='sienna', size=1.2)
```



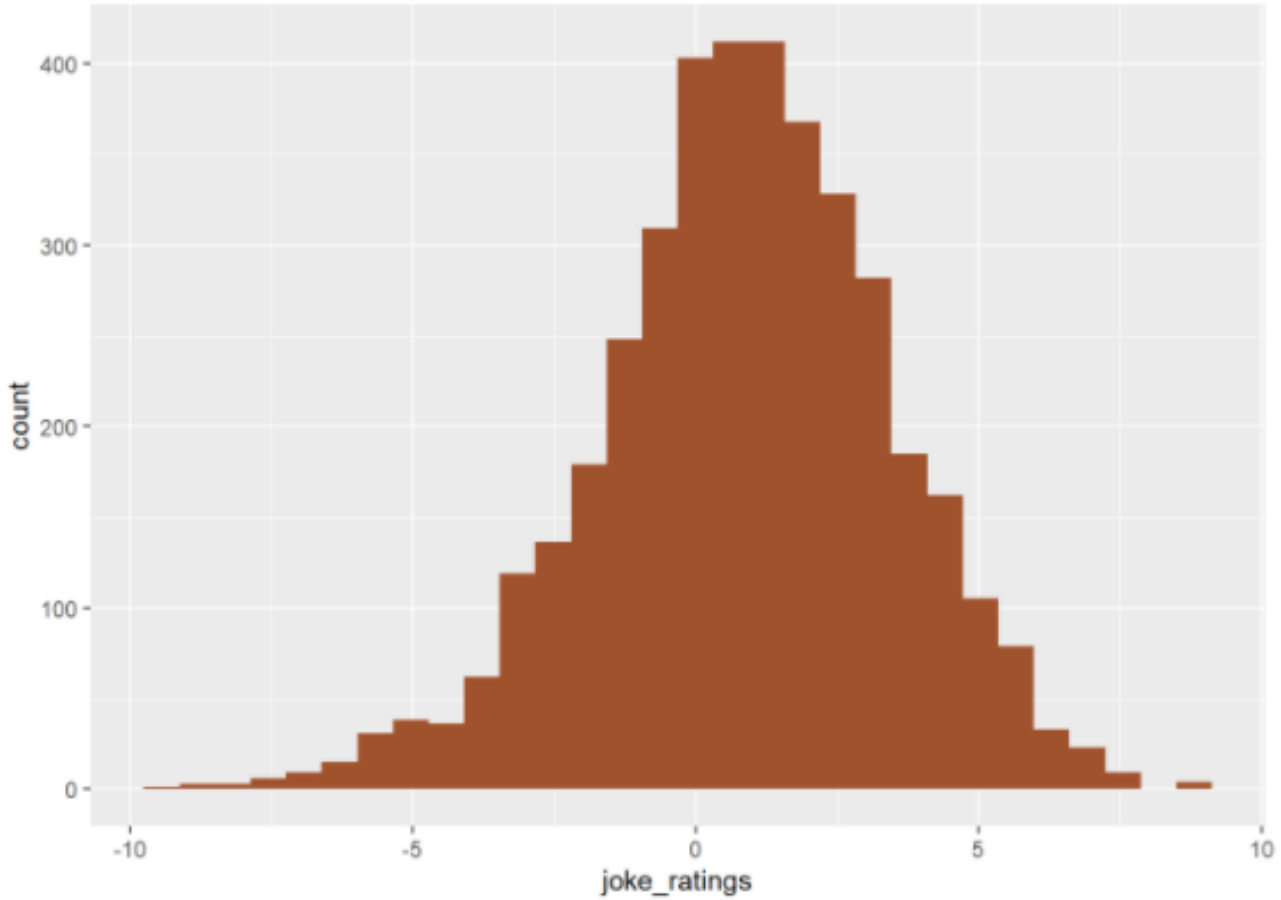
Let us examine the average rating of the first six users

```
head(rowMeans(train))
```

```
##      u5092     u14163     u22876       u195     u16903       u245
## -2.4134286  1.3239583  4.9240278 -2.7950000  3.4979167  0.5396491
```

Let us examine the distribution of jokes ratings by 4000 users in the train sample. Do users consistently rate jokes positively or negatively?

```
library(ggplot2)
ggplot(data=data.frame(joke_ratings = rowMeans(train)),aes(x=joke_ratings))+
  geom_histogram(fill='sienna')
```



5 Prepare Data

COLUMBIA UNIVERSITY
School of Professional Studies

-Normalization (centering)

People tend to differ in their ratings styles. For instance, some will rate few items (e.g., jokes) highly while others will rate most items highly. To control for individual differences, it is common to normalize the data by either centering (center) it or standardizing it (Z-score).

Let us examine the effect of centering on the ratings of u7676. Data before centering.

```
getRatings(train['u7676'])
```

```
##  [1]  4.66  5.05  4.95  1.31  4.61  2.57  0.63  4.42  5.29  5.44  0.19  2.52
## [13] -1.80  0.78 -1.26  2.62 -0.68  1.80  4.17  4.08  5.05  5.29  2.09  2.77
## [25]  3.11  4.76  6.26  6.46  4.76  2.43  2.72  3.06  0.92  2.28  6.26  8.69
## [37]  4.56  4.95  5.10  2.28  4.37  1.26  6.36  3.11  2.67  3.74  7.23  2.28
## [49]  5.34  4.22  2.14  1.12  3.98  5.05  6.50  4.47  5.83  2.72 -0.19  1.21
```

```
summary(getRatings(train['u7676']))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -1.800   2.245   3.860   3.476   5.050   8.690
```

Data after centering.

```
getRatings(normalize(train, method='center')['u7676'])
```

```
##  [1]  1.184  1.574  1.474 -2.166  1.134 -0.906 -2.846  0.944  1.814  1.964
## [11] -3.286 -0.956 -5.276 -2.696 -4.736 -0.856 -4.156 -1.676  0.694  0.604
## [21]  1.574  1.814 -1.386 -0.706 -0.366  1.284  2.784  2.984  1.284 -1.046
## [31] -0.756 -0.416 -2.556 -1.196  2.784  5.214  1.084  1.474  1.624 -1.196
## [41]  0.894 -2.216  2.884 -0.366 -0.806  0.264  3.754 -1.196  1.864  0.744
## [51] -1.336 -2.356  0.504  1.574  3.024  0.994  2.354 -0.756 -3.666 -2.266
```

```
summary(getRatings(normalize(train, method='center')['u7676']))
```

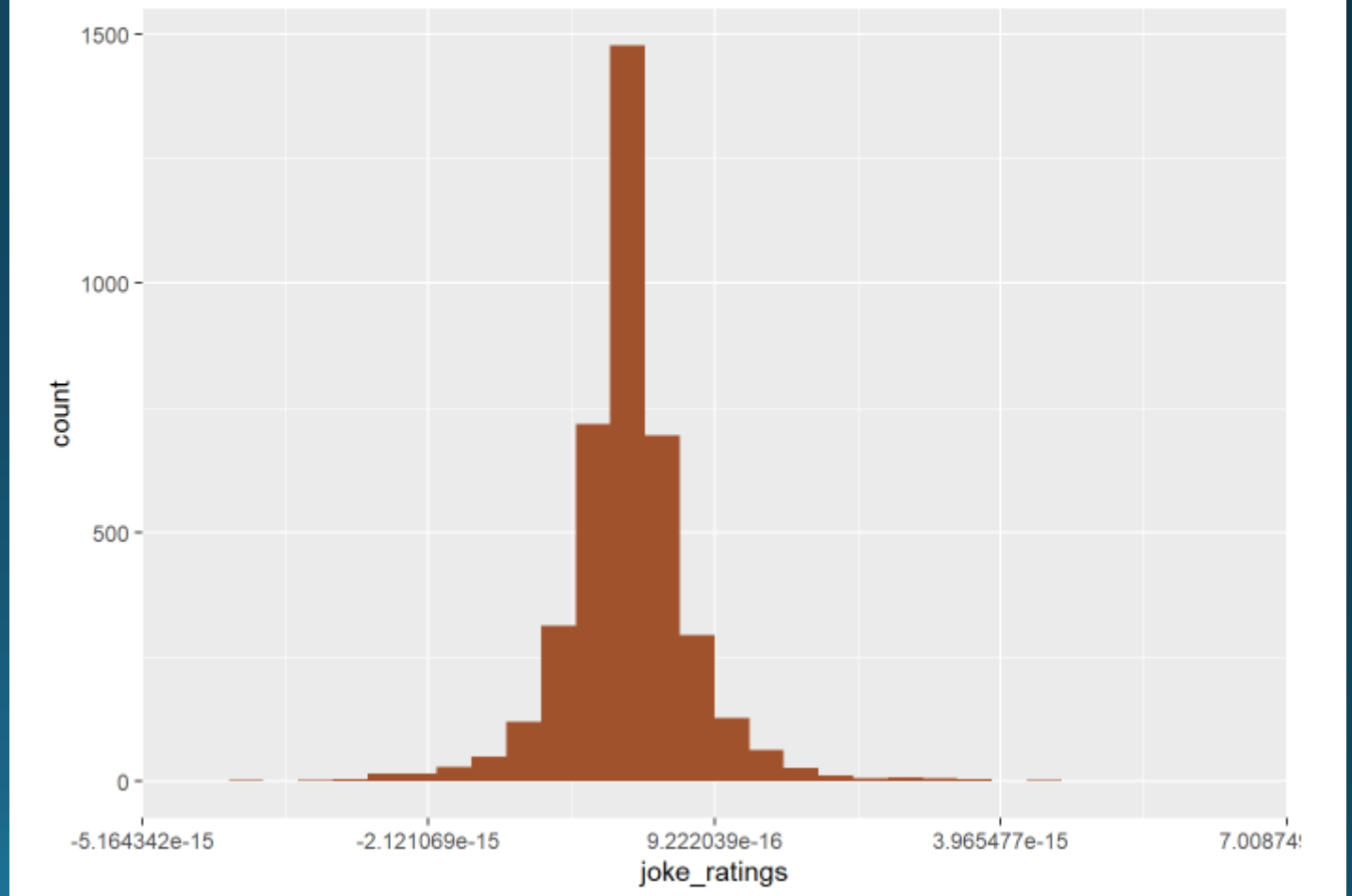
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -5.276  -1.231   0.384   0.000   1.574   5.214
```

Summary of ratings for all users after centering.

```
summary(rowMeans(normalize(train,method='center')))
```

```
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max. 
## -4.505e-15 -2.822e-16  0.000e+00  9.028e-18  2.822e-16  6.193e-15
```

```
ggplot(data=data.frame(joke_ratings = rowMeans(normalize(train))),aes(x=joke_ratings))+
  geom_histogram(fill='sienna')
```



Now, let us examine the effect of centering on ratings of j50. Rating of j50 for first 25 users before centering.

```
getRatings(train[1:25,'j50'])
```

```
##  [1]  2.96  6.41  9.22  1.80  7.28  0.73 -3.93  3.35  0.68  4.42  1.02  6.12
## [13]  4.42  4.76  6.50  9.13  7.04  4.71 -2.18  9.08  8.79  2.96  8.45  1.65
## [25]  5.44
```

Rating of j50 for first 25 users after centering.

```
getRatings(normalize(train)[1:25,'j50'])
```

```
##  [1]  5.3734286  5.0860417  4.2959722  4.5950000  3.7820833  0.1903509
##  [7] -6.0808451  4.1358000  3.3042857  0.7572973  1.0349000  1.1707000
## [13]  6.8935135  4.4711268  5.5832000  7.7457778  5.3176389  4.8777632
## [19] -6.5945833  9.8675000  9.7074468  5.7832432  8.0625352  1.9634524
## [25]  1.3727778
```

From the above, you may have noted that we generally only normalize user data not item data.

```
summary(getRatings(normalize(train)[,'j50']))
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -14.8441   0.7144   2.9783   2.6754   5.1323  14.2604
```

Finally, one could go one step ahead of centering by standardizing the data (i.e., center and express in standard deviation units). Data before standardizing

```
getRatings(train['u7676'])
```

```
##  [1]  4.66  5.05  4.95  1.31  4.61  2.57  0.63  4.42  5.29  5.44  0.19  2.52
## [13] -1.80  0.78 -1.26  2.62 -0.68  1.80  4.17  4.08  5.05  5.29  2.09  2.77
## [25]  3.11  4.76  6.26  6.46  4.76  2.43  2.72  3.06  0.92  2.28  6.26  8.69
## [37]  4.56  4.95  5.10  2.28  4.37  1.26  6.36  3.11  2.67  3.74  7.23  2.28
## [49]  5.34  4.22  2.14  1.12  3.98  5.05  6.50  4.47  5.83  2.72 -0.19  1.21
```

```
summary(getRatings(train['u7676']))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -1.800   2.245   3.860   3.476   5.050   8.690
```

Data after standardizing.

```
getRatings(normalize(train, method='Z-score')['u7676'])
```

```
##  [1]  0.5438467  0.7229854  0.6770524 -0.9949087  0.5208802 -0.4161529
##  [7] -1.3072531  0.4336075  0.8332246  0.9021240 -1.5093583 -0.4391194
## [13] -2.4234249 -1.2383536 -2.1753867 -0.3931864 -1.9089753 -0.7698370
## [19]  0.3187750  0.2774353  0.7229854  0.8332246 -0.6366313 -0.3242870
## [25] -0.1681148  0.5897797  1.2787746  1.3706406  0.5897797 -0.4804591
## [31] -0.3472535 -0.1910813 -1.1740474 -0.5493586  1.2787746  2.3949464
## [37]  0.4979137  0.6770524  0.7459519 -0.5493586  0.4106410 -1.0178752
## [43]  1.3247076 -0.1681148 -0.3702199  0.1212631  1.7243247 -0.5493586
## [49]  0.8561910  0.3417415 -0.6136648 -1.0821814  0.2315023  0.7229854
## [55]  1.3890138  0.4565740  1.0812627 -0.3472535 -1.6839036 -1.0408417
```

```
summary(getRatings(normalize(train, method='Z-score')['u7676']))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.4234 -0.5654  0.1764  0.0000  0.7230  2.3949
```

6 Similarity

Let us examine cosine similarity among the first five users. Library recommenderlab comes with a function, similarity() that can be used to compute euclidean distance, cosine similarity, and pearson correlation. The three methods vary in the extent to which they standardize similarity.

```
similarity(normalize(train)[1:5],method = 'euclidean')
```

```
##            u5092     u14163     u22876       u195
## u14163 0.02051023
## u22876 0.01830506 0.01804014
## u195   0.01944195 0.01667579 0.01357619
## u16903 0.01884840 0.01605939 0.01575788 0.01337091
```

```
similarity(normalize(train)[1:5],method = 'cosine')
```

```
##              u5092       u14163       u22876         u195
## u14163  0.004226426
## u22876  0.181413039  0.221160094
## u195    0.168586227  0.005961224 -0.172707678
## u16903  0.109864977 -0.045638098  0.216789387 -0.189342144
```

```
similarity(normalize(train)[1:5],method = 'pearson')
```

```
##           u5092    u14163    u22876      u195
## u14163 0.5010599
## u22876 0.5498844 0.5622133
## u195   0.5460327 0.5014963 0.5000000
## u16903 0.5290751 0.5000000 0.5606983 0.5000000
```

recommenderlab functions perform normalizing and compute similarity measures within the Recommender function, so in practice there is no need to perform these functions earlier.

7 Construct Recommendations

7.1 Non-Personalized Recommendations: Popular

One of the oldest approaches to recommendations involves recommending the most popular item. Although less widely used today, we still see this in the form of Top 50 Music Hits, Billboard Top Hits, and New York Times Bestsellers. The underlying assumption is that if most people like a movie, a book or a song, you will like it too.

Let us examine the POPULAR method that implements this technique

```
recommenderRegistry$get_entries(dataType='realRatingMatrix')$POPULAR_realRatingMatrix
```

```
## Recommender method: POPULAR for realRatingMatrix
## Description: Recommender based on item popularity.
## Reference: NA
## Parameters:
##   normalize
## 1  "center"
##                                                 aggregationRatings
## 1 new("standardGeneric", .Data = function (x, na.rm = FALSE, dims = 1,
##                                               aggregationPopularity
## 1 new("standardGeneric", .Data = function (x, na.rm = FALSE, dims = 1,
```

We implement the POPULAR method using the Recommend function. We pass it a parameter to normalize train data using the center method (although, center is the default)

```
recom_popular = Recommender(train,
                            method='POPULAR',
                            parameter=list(normalize='center'))
```

7.1.1 Top n recommendations

Use the Recommender object to generate Top 5 predictions for the test sample. Note, topNList is the default type.

```
pred_popular_topN = predict(recom_popular,newdata=test,type='topNList',n=5)
```

Top 5 joke recommendations for the first ten users in the test sample

```
getList(pred_popular_topN)[1:10]
```

```
## $u238
## [1] "j53" "j49" "j61" "j42" "j11"
##
## $u6662
## character(0)
##
## $u7556
## character(0)
##
## $u6617
## character(0)
##
## $u7602
## [1] "j89" "j72" "j93" "j76" "j91"
##
## $u4519
## [1] "j89" "j72" "j91" "j83" "j88"
##
## $u13802
## [1] "j89" "j72" "j93" "j76" "j91"
##
## $u7778
## [1] "j89" "j11" "j6"  "j72" "j93"
##
## $u12519
## [1] "j89" "j72" "j93" "j76" "j91"
##
## $u6083
## character(0)
```

Let's see the top 5 joke recommendations for u7602

```
getList(pred_popular_topN)['u7602']
```

```
## $u7602
## [1] "j89" "j72" "j93" "j76" "j91"
```

```
> nratings(test['u6662'])
[1] 100
```

7.1.2 Ratings

Now, instead of generating the Top n recommendations for each user, we can also generate ratings for each joke. To do this, we use the type, 'ratings' instead of 'topNList'

```
pred_popular = predict(recom_popular,newdata=test,type='ratings')
```

Jokes rated by u7602

```
as(test,'matrix')['u7602',]
```

```
##    j1    j2    j3    j4    j5    j6    j7    j8    j9   j10   j11   j12   j13
##    NA  9.22 -9.56    NA  7.28 -9.22  6.60 -7.77    NA  9.03 -2.86 -1.36 -7.77
##   j14   j15   j16   j17   j18   j19   j20   j21   j22   j23   j24   j25   j26
##  7.96  7.77 -8.83 -8.01 -8.59 -7.77  7.04  7.57 -9.27 -9.61    NA  8.11 -0.68
##   j27   j28   j29   j30   j31   j32   j33   j34   j35   j36   j37   j38   j39
## -5.29  9.17  8.16    NA -5.68  6.21    NA -0.68  6.84 -7.62    NA -4.08 -9.61
##   j40   j41   j42   j43   j44   j45   j46   j47   j48   j49   j50   j51   j52
##  2.72 -9.66 -9.22    NA    NA -9.56 -9.22  8.16  3.79 -4.61  6.75 -9.66  1.70
##   j53   j54   j55   j56   j57   j58   j59   j60   j61   j62   j63   j64   j65
## -6.94 -0.78    NA  0.24    NA    NA -6.99 -9.32 -9.47  3.64 -9.66    NA  0.10
##   j66   j67   j68   j69   j70   j71   j72   j73   j74   j75   j76   j77   j78
##  9.13    NA  9.03 -7.52 -9.71    NA    NA    NA    NA -6.60    NA    NA    NA
##   j79   j80   j81   j82   j83   j84   j85   j86   j87   j88   j89   j90   j91
##    NA    NA    NA    NA    NA    NA -6.80    NA    NA    NA    NA    NA    NA
##   j92   j93   j94   j95   j96   j97   j98   j99  j100
##    NA    NA    NA -8.40    NA    NA    NA    NA    NA
```

```
> as(pred_popular,'matrix')['u7602','j71']
[1] -3.443148
```

Predicted ratings for jokes not rated by u7602

```
as(pred_popular,'matrix')['u7602',]
```

```
##          j1          j2          j3          j4          j5          j6
## -1.64723478          NA          NA -4.18169918          NA          NA
##          j7          j8          j9         j10         j11         j12
##          NA          NA -3.38636173          NA          NA          NA
##         j13         j14         j15         j16         j17         j18
##          NA          NA          NA          NA          NA          NA
##         j19         j20         j21         j22         j23         j24
##          NA          NA          NA          NA          NA -4.32852149
##         j25         j26         j27         j28         j29         j30
##          NA          NA          NA          NA          NA -3.21832398
##         j31         j32         j33         j34         j35         j36
##          NA          NA -4.16497024          NA          NA          NA
##         j37         j38         j39         j40         j41         j42
## -4.10681634          NA          NA          NA          NA          NA
##         j43         j44         j45         j46         j47         j48
## -3.54201000 -4.93186911          NA          NA          NA          NA
##         j49         j50         j51         j52         j53         j54
##          NA          NA          NA          NA          NA          NA
##         j55         j56         j57         j58         j59         j60
## -2.25327191          NA -4.81129545 -6.57366700          NA          NA
##         j61         j62         j63         j64         j65         j66
##          NA          NA          NA -3.36220570          NA          NA
##         j67         j68         j69         j70         j71         j72
## -3.54204936          NA          NA          NA -3.44314836 -0.02190592
##         j73         j74         j75         j76         j77         j78
## -1.78579297 -4.42473991          NA -0.38115821 -2.19652733 -1.28500649
##         j79         j80         j81         j82         j83         j84
## -2.85167662 -1.88882197 -1.13534402 -1.86123518 -0.85544917 -2.16243200
##         j85         j86         j87         j88         j89         j90
##          NA -2.67536436 -1.14943969 -0.91539768  0.60549796 -2.38261891
##         j91         j92         j93         j94         j95         j96
## -0.85502632 -1.63362176 -0.31209917 -1.97021469          NA -1.47030541
##         j97         j98         j99        j100
## -1.22914858 -2.02297399 -2.98789643 -1.57957444
```

```
sort(as(pred_popular,'matrix')['u7602',],decreasing = T)
```

```
##         j89         j72         j93         j76         j91         j83
##  0.60549796 -0.02190592 -0.31209917 -0.38115821 -0.85502632 -0.85544917
##         j88         j81         j87         j97         j78         j96
## -0.91539768 -1.13534402 -1.14943969 -1.22914858 -1.28500649 -1.47030541
##        j100         j92          j1         j73         j82         j80
## -1.57957444 -1.63362176 -1.64723478 -1.78579297 -1.86123518 -1.88882197
##         j94         j98         j84         j77         j55         j90
## -1.97021469 -2.02297399 -2.16243200 -2.19652733 -2.25327191 -2.38261891
##         j86         j79         j99         j30         j64          j9
## -2.67536436 -2.85167662 -2.98789643 -3.21832398 -3.36220570 -3.38636173
##         j71         j43         j67         j37         j33          j4
## -3.44314836 -3.54201000 -3.54204936 -4.10681634 -4.16497024 -4.18169918
##         j24         j74         j57         j44         j58
## -4.32852149 -4.42473991 -4.81129545 -4.93186911 -6.57366700
```

```
summary(as(test,'matrix')['u7602',])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##  -9.710  -8.710  -4.610  -1.901   6.675   9.220      41
```

Ratings for jokes 51-60 by users 55-60 in the test

```
as(test,'matrix')[55:60,51:60]
```

```
##          j51   j52   j53   j54   j55   j56   j57   j58   j59   j60
## u19374    NA    NA  5.44  4.27  5.92  7.77    NA    NA    NA  8.06
## u18224 -9.85 -5.19  2.48  3.25 -2.23  0.24 -9.85  1.41 -6.70 -3.16
## u17195 -3.20 -2.62  5.10  8.88 -0.10  3.54 -8.40 -8.64 -1.50 -7.18
## u15028  1.31  6.17 -7.04  6.46 -2.28 -0.92 -5.78  0.97  3.20  2.96
## u7628     NA    NA  0.83  2.09    NA  2.96    NA -5.78    NA    NA
## u23430  3.74 -6.84  6.02 -9.08  7.38  3.11  4.95 -4.61 -7.57 -7.14
```

Predicted Ratings for jokes 51-60 by users 55-60 in the test

```
as(pred_popular,'matrix')[55:60,51:60]
```

```
##              j51       j52 j53 j54        j55 j56        j57       j58       j59
## u19374  1.850405  2.551580  NA  NA         NA  NA  0.4542113 -1.30816  1.983546
## u18224        NA        NA  NA  NA         NA  NA         NA       NA        NA
## u17195        NA        NA  NA  NA         NA  NA         NA       NA        NA
## u15028        NA        NA  NA  NA         NA  NA         NA       NA        NA
## u7628  -1.904918 -1.203743  NA  NA -0.7430883  NA -3.3011118       NA -1.771777
## u23430        NA        NA  NA  NA         NA  NA         NA       NA        NA
##              j60
## u19374        NA
## u18224        NA
## u17195        NA
## u15028        NA
## u7628  -1.500422
## u23430        NA
```

7.2 User-based collaborative filtering

Examine parameters for UBCF

```
recommenderRegistry$get_entries(data='realRatingMatrix')$UBCF_realRatingMatrix
```

```
## Recommender method: UBCF for realRatingMatrix
## Description: Recommender based on user-based collaborative filtering.
## Reference: NA
## Parameters:
##     method nn sample weighted normalize min_matching_items min_predictive_items
## 1 "cosine" 25  FALSE     TRUE  "center"                  0                   0
```

```
recom_ubcf = Recommender(train, method='UBCF', parameter=list(method='cosine',
                                                              nn=25, normalize='center'))
```

7.2.1 Top n recommendations

```
pred_ubcf_topN = predict(recom_ubcf,newdata=test,method='topNList',n=5)
```

```
getList(pred_ubcf_topN)[1:5]
```

```
## $u238
## [1] "j61"  "j12"  "j100" "j11"  "j34"
##
## $u6662
## character(0)
##
## $u7556
## character(0)
##
## $u6617
## character(0)
##
## $u7602
## [1] "j72" "j89" "j83" "j98" "j93"
```

```
getList(pred_ubcf_topN)['u7602']
```

```
## $u7602
## [1] "j72" "j89" "j83" "j98" "j93"
```

```
slotNames(pred_ubcf_topN)
```

```
## [1] "items"      "ratings"    "itemLabels" "n"
```

```
pred_ubcf_topN@items['u7602']
```

```
## $u7602
## [1] 72 89 83 98 93
```

7.2.2 Ratings

Now, instead of generating the Top n recommendations for each user, we can also generate ratings for each joke. To do this, we use the type, 'ratings' instead of 'topNList'

```
pred_ubcf = predict(recom_ubcf,newdata=test,type='ratings')
```

Jokes rated by u7602

```
as(test,'matrix')['u7602',]
```

```
##    j1    j2    j3    j4    j5    j6    j7    j8    j9   j10   j11   j12   j13
##    NA  9.22 -9.56    NA  7.28 -9.22  6.60 -7.77    NA  9.03 -2.86 -1.36 -7.77
##   j14   j15   j16   j17   j18   j19   j20   j21   j22   j23   j24   j25   j26
##  7.96  7.77 -8.83 -8.01 -8.59 -7.77  7.04  7.57 -9.27 -9.61    NA  8.11 -0.68
##   j27   j28   j29   j30   j31   j32   j33   j34   j35   j36   j37   j38   j39
## -5.29  9.17  8.16    NA -5.68  6.21    NA -0.68  6.84 -7.62    NA -4.08 -9.61
##   j40   j41   j42   j43   j44   j45   j46   j47   j48   j49   j50   j51   j52
##  2.72 -9.66 -9.22    NA    NA -9.56 -9.22  8.16  3.79 -4.61  6.75 -9.66  1.70
##   j53   j54   j55   j56   j57   j58   j59   j60   j61   j62   j63   j64   j65
## -6.94 -0.78    NA  0.24    NA    NA -6.99 -9.32 -9.47  3.64 -9.66    NA  0.10
##   j66   j67   j68   j69   j70   j71   j72   j73   j74   j75   j76   j77   j78
##  9.13    NA  9.03 -7.52 -9.71    NA    NA    NA    NA -6.60    NA    NA    NA
##   j79   j80   j81   j82   j83   j84   j85   j86   j87   j88   j89   j90   j91
##    NA    NA    NA    NA    NA    NA -6.80    NA    NA    NA    NA    NA    NA
##   j92   j93   j94   j95   j96   j97   j98   j99  j100
##    NA    NA    NA -8.40    NA    NA    NA    NA    NA
```

Recommendations for Jokes not rated

```
as(pred_ubcf,'matrix')['u7602',]
```

```
##         j1         j2         j3         j4         j5         j6         j7
##  0.4252532         NA         NA -4.0240540         NA         NA         NA
##         j8         j9        j10        j11        j12        j13        j14
##         NA -6.0664725         NA         NA         NA         NA         NA
##        j15        j16        j17        j18        j19        j20        j21
##         NA         NA         NA         NA         NA         NA         NA
##        j22        j23        j24        j25        j26        j27        j28
##         NA         NA -7.1110200         NA         NA         NA         NA
##        j29        j30        j31        j32        j33        j34        j35
##         NA -7.0806337         NA         NA -2.2160850         NA         NA
##        j36        j37        j38        j39        j40        j41        j42
##         NA -5.9338766         NA         NA         NA         NA         NA
##        j43        j44        j45        j46        j47        j48        j49
## -5.0117518 -7.0550270         NA         NA         NA         NA         NA
##        j50        j51        j52        j53        j54        j55        j56
##         NA         NA         NA         NA         NA -4.6272718         NA
##        j57        j58        j59        j60        j61        j62        j63
## -7.3055620 -7.7070706         NA         NA         NA         NA         NA
##        j64        j65        j66        j67        j68        j69        j70
## -4.7577581         NA         NA -6.0812369         NA         NA         NA
##        j71        j72        j73        j74        j75        j76        j77
## -1.8074904  2.1899084 -1.9344307 -5.4937182         NA -2.5589974 -3.0194354
##        j78        j79        j80        j81        j82        j83        j84
## -1.1072481 -4.1423077 -1.3538005  0.8341537 -0.6767704  1.4713462 -5.1126313
##        j85        j86        j87        j88        j89        j90        j91
##         NA -4.5480534 -1.4192413 -0.9956362  1.6231380 -2.4259397 -1.5079380
##        j92        j93        j94        j95        j96        j97        j98
## -1.7055887  1.1689603 -3.2716326         NA -4.0462800 -0.3282858  1.2396412
##        j99       j100
## -1.7344293  0.1999128
```

Ratings for jokes 51-60 by users 55-60 in the test

```
as(test,'matrix')[55:60,51:60]
```

```
##         j51   j52   j53   j54   j55   j56   j57   j58   j59   j60
## u19374    NA    NA  5.44  4.27  5.92  7.77    NA    NA    NA  8.06
## u18224 -9.85 -5.19  2.48  3.25 -2.23  0.24 -9.85  1.41 -6.70 -3.16
## u17195 -3.20 -2.62  5.10  8.88 -0.10  3.54 -8.40 -8.64 -1.50 -7.18
## u15028  1.31  6.17 -7.04  6.46 -2.28 -0.92 -5.78  0.97  3.20  2.96
## u7628     NA    NA  0.83  2.09    NA  2.96    NA -5.78    NA    NA
## u23430  3.74 -6.84  6.02 -9.08  7.38  3.11  4.95 -4.61 -7.57 -7.14
```

Predicted Ratings for jokes 51-60 by users 55-60 in the test

```
as(pred_ubcf,'matrix')[55:60,51:60]
```

```
##              j51       j52 j53 j54       j55 j56       j57      j58       j59
## u19374  2.5387996 4.4196047  NA  NA        NA  NA  3.313522 1.168495  3.835591
## u18224         NA        NA  NA  NA        NA  NA        NA       NA        NA
## u17195         NA        NA  NA  NA        NA  NA        NA       NA        NA
## u15028         NA        NA  NA  NA        NA  NA        NA       NA        NA
## u7628  -0.6526787 0.7085059  NA  NA -1.04053  NA -1.287465       NA -3.081738
## u23430         NA        NA  NA  NA        NA  NA        NA       NA        NA
##              j60
## u19374        NA
## u18224        NA
## u17195        NA
## u15028        NA
## u7628  -2.853737
## u23430        NA
```

7.3 Item-based collaborative filtering

```
recommenderRegistry$get_entries(data='realRatingMatrix')$IBCF_realRatingMatrix # see parameters for IBCF
```

```
## Recommender method: IBCF for realRatingMatrix
## Description: Recommender based on item-based collaborative filtering.
## Reference: NA
## Parameters:
##    k   method normalize normalize_sim_matrix alpha na_as_zero
## 1 30 "Cosine"  "center"                FALSE   0.5      FALSE
```

```
recom_ibcf = Recommender(train, method='IBCF', parameter=list(k=30, method='cosine',
                                                              normalize='center'))
recom_ibcf
```

```
## Recommender of type 'IBCF' for 'realRatingMatrix'
## learned using 4000 users.
```

7.3.1 Top n recommendations

```
pred_ibcf_topN = predict(recom_ibcf,newdata=test,type='topNList',n=5)
```

```
getList(pred_ibcf_topN)[1:5]
```

```
## $u238
## [1] "j25"  "j52"  "j100" "j45"  "j11"
##
## $u6662
## character(0)
##
## $u7556
## character(0)
##
## $u6617
## character(0)
##
## $u7602
## [1] "j97" "j91" "j89" "j88" "j93"
```

```
getList(pred_ibcf_topN)['u7602']
```

```
## $u7602
## [1] "j97" "j91" "j89" "j88" "j93"
```

7.3.2 Ratings

```
pred_ibcf = predict(recom_ibcf,newdata=test,type='ratings')
```

Jokes rated

```
as(test,'matrix')['u7602',]
```

```
##    j1    j2    j3    j4    j5    j6    j7    j8    j9   j10   j11   j12   j13
##    NA  9.22 -9.56    NA  7.28 -9.22  6.60 -7.77    NA  9.03 -2.86 -1.36 -7.77
##   j14   j15   j16   j17   j18   j19   j20   j21   j22   j23   j24   j25   j26
##  7.96  7.77 -8.83 -8.01 -8.59 -7.77  7.04  7.57 -9.27 -9.61    NA  8.11 -0.68
##   j27   j28   j29   j30   j31   j32   j33   j34   j35   j36   j37   j38   j39
## -5.29  9.17  8.16    NA -5.68  6.21    NA -0.68  6.84 -7.62    NA -4.08 -9.61
##   j40   j41   j42   j43   j44   j45   j46   j47   j48   j49   j50   j51   j52
##  2.72 -9.66 -9.22    NA    NA -9.56 -9.22  8.16  3.79 -4.61  6.75 -9.66  1.70
##   j53   j54   j55   j56   j57   j58   j59   j60   j61   j62   j63   j64   j65
## -6.94 -0.78    NA  0.24    NA    NA -6.99 -9.32 -9.47  3.64 -9.66    NA  0.10
##   j66   j67   j68   j69   j70   j71   j72   j73   j74   j75   j76   j77   j78
##  9.13    NA  9.03 -7.52 -9.71    NA    NA    NA    NA -6.60    NA    NA    NA
##   j79   j80   j81   j82   j83   j84   j85   j86   j87   j88   j89   j90   j91
##    NA    NA    NA    NA    NA    NA -6.80    NA    NA    NA    NA    NA    NA
##   j92   j93   j94   j95   j96   j97   j98   j99  j100
##    NA    NA    NA -8.40    NA    NA    NA    NA    NA
```

Recommendations for Jokes not rated

```
as(pred_ibcf,'matrix')['u7602',]
```

```
##         j1         j2         j3         j4         j5         j6         j7
## -1.2018367         NA         NA -4.2837760         NA         NA         NA
##         j8         j9        j10        j11        j12        j13        j14
##         NA -6.4862906         NA         NA         NA         NA         NA
##        j15        j16        j17        j18        j19        j20        j21
##         NA         NA         NA         NA         NA         NA         NA
##        j22        j23        j24        j25        j26        j27        j28
##         NA         NA -6.2239619         NA         NA         NA         NA
##        j29        j30        j31        j32        j33        j34        j35
##         NA -6.8038739         NA         NA -5.6159458         NA         NA
##        j36        j37        j38        j39        j40        j41        j42
##         NA -4.4365071         NA         NA         NA         NA         NA
##        j43        j44        j45        j46        j47        j48        j49
## -7.1514112 -5.1258272         NA         NA         NA         NA         NA
##        j50        j51        j52        j53        j54        j55        j56
##         NA         NA         NA         NA         NA -5.6733279         NA
##        j57        j58        j59        j60        j61        j62        j63
## -5.4207885 -5.0225761         NA         NA         NA         NA         NA
##        j64        j65        j66        j67        j68        j69        j70
## -6.1754690         NA         NA -6.7985650         NA         NA         NA
##        j71        j72        j73        j74        j75        j76        j77
## -2.1703178  0.4463620 -6.9194944 -4.2379636         NA  0.9519787 -3.9375637
##        j78        j79        j80        j81        j82        j83        j84
## -2.3680282 -5.4473974 -5.0196057 -0.5256426 -1.9485224  0.6247610 -2.4777716
##        j85        j86        j87        j88        j89        j90        j91
##         NA -5.1215795 -0.9135127  1.0597379  1.6344292 -0.8523732  1.8335773
##        j92        j93        j94        j95        j96        j97        j98
## -1.8944481  0.9721270 -6.1305959         NA -2.6063404  2.4407542 -0.8066422
##        j99       j100
## -5.4506372 -3.1873168
```

```
as(test,'matrix')[55:60,51:60]
```

```
##         j51   j52   j53   j54   j55   j56   j57   j58   j59   j60
## u19374    NA    NA  5.44  4.27  5.92  7.77    NA    NA    NA  8.06
## u18224 -9.85 -5.19  2.48  3.25 -2.23  0.24 -9.85  1.41 -6.70 -3.16
## u17195 -3.20 -2.62  5.10  8.88 -0.10  3.54 -8.40 -8.64 -1.50 -7.18
## u15028  1.31  6.17 -7.04  6.46 -2.28 -0.92 -5.78  0.97  3.20  2.96
## u7628     NA    NA  0.83  2.09    NA  2.96    NA -5.78    NA    NA
## u23430  3.74 -6.84  6.02 -9.08  7.38  3.11  4.95 -4.61 -7.57 -7.14
```

```
as(pred_ubcf,'matrix')[55:60,51:60]
```

```
##             j51       j52 j53 j54      j55 j56       j57       j58       j59
## u19374  2.5387996 4.4196047  NA  NA       NA  NA 3.313522 1.168495  3.835591
## u18224         NA        NA  NA  NA       NA  NA        NA       NA        NA
## u17195         NA        NA  NA  NA       NA  NA        NA       NA        NA
## u15028         NA        NA  NA  NA       NA  NA        NA       NA        NA
## u7628  -0.6526787 0.7085059  NA  NA -1.04053  NA -1.287465       NA -3.081738
## u23430         NA        NA  NA  NA       NA  NA        NA       NA        NA
##             j60
## u19374       NA
## u18224       NA
## u17195       NA
## u15028       NA
## u7628  -2.853737
## u23430       NA
```

8 Evaluation Scheme

8.1 Split: Train - Test

Recommenderlab package has some handy built-in functions to evaluate different recommender models. Here, we are going to create an evaluation scheme from Jester5k, not the train dataset. evaluationScheme() handles splits, k-fold cross validation and bootstrapping under the hood.

Let us begin with a 80:20 split. We will give the recommender algorithm 30 items from the test set and hold out the other items for computing the error. The number of items ('given') must be less than the minimum items rated by any user. For this dataset, the least number of jokes rated by any user is 36

```
min(rowCounts(Jester5k))
```

```
## [1] 36
```

```
set.seed(1031)
es = evaluationScheme(Jester5k,method='split',train=0.8, given=30)
```

Train set

```
getData(es,'train')
```

```
## 4000 x 100 rating matrix of class 'realRatingMatrix' with 289668 ratings.
```

Test set with the items used to build the recommendations

```
getData(es,'known')
```

```
## 1000 x 100 rating matrix of class 'realRatingMatrix' with 30000 ratings.
```

Test set with the items used to evaluate the recommendations

```
getData(es,'unknown')
```

```
## 1000 x 100 rating matrix of class 'realRatingMatrix' with 42438 ratings.
```

Total number of ratings = train + known + unknown

```
nratings(Jester5k) == nratings(getData(es,'train')) + nratings(getData(es,'known')) +
                      nratings(getData(es,'unknown'))
```

```
## [1] TRUE
```

Number of items used to generate recommendations; value of 'given'

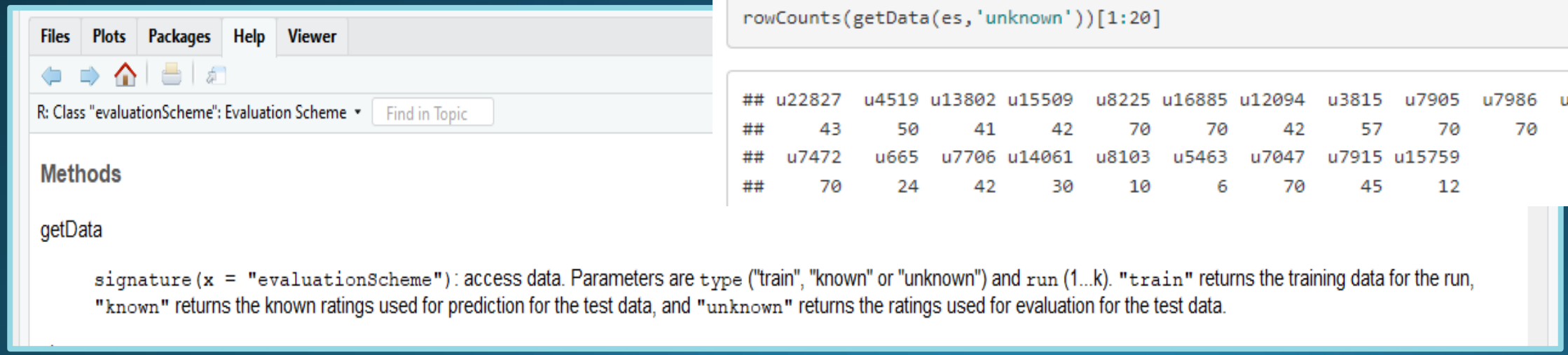
```
rowCounts(getData(es,'known'))[1:20]
```

```
## u22827  u4519 u13802 u15509  u8225 u16885 u12094  u3815  u7905  u7986  u3552
##     30     30     30     30     30     30     30     30     30     30     30
##  u7472   u665  u7706 u14061  u8103  u5463  u7047  u7915 u15759
##     30     30     30     30     30     30     30     30     30
```

Remaining items being held out for computing error.

```
rowCounts(getData(es,'unknown'))[1:20]
```

```
## u22827  u4519 u13802 u15509  u8225 u16885 u12094  u3815  u7905  u7986  u3552
##     43     50     41     42     70     70     42     57     70     70     63
##  u7472   u665  u7706 u14061  u8103  u5463  u7047  u7915 u15759
##     70     24     42     30     10      6     70     45     12
```



Build recommender

```
recom_ubcf = Recommender(data = getData(es,'train'),
                         method='UBCF',
                         parameter = list(method='cosine',nn=25,normalize='center'))
```

Generate Predictions

```
pred_ubcf = predict(recom_ubcf,newdata=getData(es,'known'), type='ratings')
```

Evaluate Predictions

```
calcPredictionAccuracy(pred_ubcf,data = getData(es,'unknown'))
```

```
##       RMSE        MSE        MAE
##   4.544705 20.654345   3.565705
```

User-based collaborative filtering recommender using defaults:

Recommender(data=getData(es,'train'), method='UBCF')

Next let us calculate prediction accuracy for an IBCF

```
recommenderRegistry$get_entries(data='realRatingMatrix')$IBCF_realRatingMatrix
```

```
## Recommender method: IBCF for realRatingMatrix
## Description: Recommender based on item-based collaborative filtering.
## Reference: NA
## Parameters:
##    k   method normalize normalize_sim_matrix alpha na_as_zero
## 1 30 "Cosine"  "center"                FALSE   0.5      FALSE
```

```
recom_ibcf = Recommender(data = getData(es,'train'),
                         method='IBCF',
                         parameter = list(method='cosine',k=30,normalize='center'))
pred_ibcf = predict(recom_ibcf,newdata=getData(es,'known'), type='ratings')
calcPredictionAccuracy(pred_ibcf,data = getData(es,'unknown'))
```

```
##      RMSE       MSE       MAE
##  4.341846 18.851624  3.342718
```

8.2 k-fold cross-validation

Next, let us use k-fold cross-validation, first, to evaluate just UBCF

```
set.seed(1031)
es = evaluationScheme(Jester5k,method='cross-validation',k=10,given=30)
ev = evaluate(x = es,method='UBCF',parameter=list(method='cosine',nn=25), type='ratings')
```

```
## UBCF run fold/sample [model time/prediction time]
##    1  [0.03sec/2.25sec]
##    2  [0.04sec/2.4sec]
##    3  [0.02sec/2.24sec]
##    4  [0.03sec/2.26sec]
##    5  [0.03sec/2.43sec]
##    6  [0.03sec/2.24sec]
##    7  [0.04sec/2.3sec]
##    8  [0.03sec/2.28sec]
##    9  [0.03sec/2.44sec]
##    10  [0.01sec/2.23sec]
```

```
avg(ev)
```

```
##          RMSE      MSE      MAE
## [1,] 4.561912 20.81313 3.573399
```

Use k-fold cross-validation to evaluate a set of recommenders

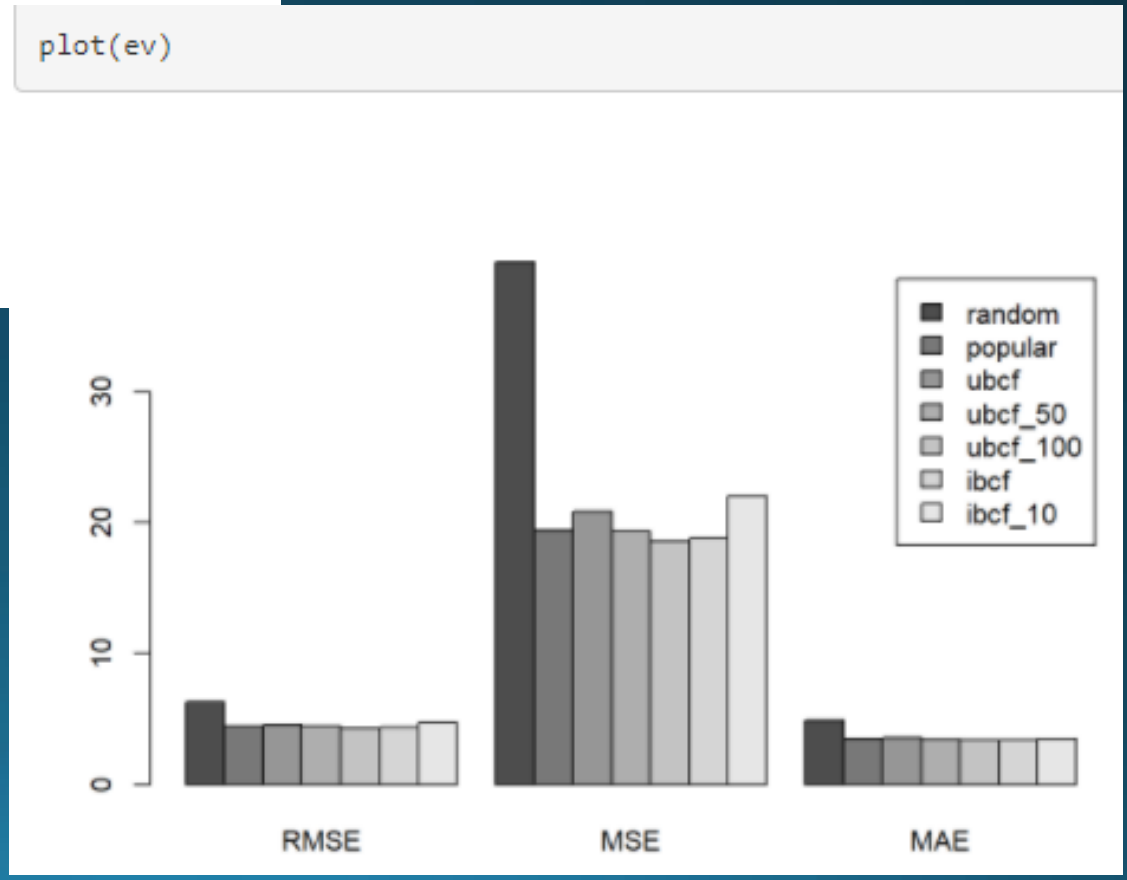
```
recommender_algorithms = list(random = list(name='RANDOM'),
                              popular = list(name='POPULAR'),
                              ubcf = list(name='UBCF'),
                              ubcf_50 = list(name='UBCF',parameters=list(nn=50)),
                              ubcf_100 = list(name='UBCF',parameters=list(nn=100)),
                              ibcf = list(name='IBCF'),
                              ibcf_10 = list(name='IBCF', parameters=list(k=10)))
ev = evaluate(x = es,method=recommender_algorithms, type='ratings')
```

```
## RANDOM run fold/sample [model time/prediction time]
##    1  [0sec/0.03sec]
##    2  [0sec/0.01sec]
##    3  [0sec/0.01sec]
##    4  [0.02sec/0.01sec]
##    5  [0sec/0.01sec]
##    6  [0sec/0.03sec]
##    7  [0.02sec/0.01sec]
##    8  [0sec/0.01sec]
##    9  [0sec/0.03sec]
##    10  [0.02sec/0.01sec]
## POPULAR run fold/sample [model time/prediction time]
##    1  [0.18sec/0sec]
##    2  [0.04sec/0sec]
```

```
results = matrix(unlist(avg(ev)),ncol=3,byrow=TRUE)
colnames(results) = c('RMSE','MSE','MAE')
rownames(results) = c('random','popular','ubcf','ubcf_50','ubcf_100','ibcf','ibcf_10')
results
```

```
##              RMSE      MSE      MAE
## random   6.315695 39.88997 4.897584
## popular  4.410310 19.45274 3.494418
## ubcf     4.561912 20.81313 3.573399
## ubcf_50  4.394851 19.31646 3.454142
## ubcf_100 4.310734 18.58408 3.390582
## ibcf     4.333407 18.78179 3.333610
## ibcf_10  4.692304 22.02181 3.493000
```

```
plot(ev)
```



Summary

In this module we

- Discuss need for recommendation systems
- Discuss applications of recommender systems
- Explain how recommender systems work
  - Compare and contrast types of recommendation systems
- Utilize recommender systems to make product recommendations using R
  - Small simulated dataset
  - Jester dataset (5k sample)