NEIGHBORHOOD-BASED COLLABORATIVE FILTERING:

AARHUS BSS
DEPARTMENT OF ECONOMICS AND BUSINESS ECONOMICS
AARHUS UNIVERSITY

29 APRIL 2024

MORTEN BERG JENSEN
ASSOCIATE PROFESSOR

OUTLINE

Collaborative filtering

- User-based collaborative filtering
- Item-based collaborative filtering

How to use the R-package Recommenderlab?

- How to formulate a recommender model?
- How to evaluate a recommender model?
- How to compare recommender models?
- How to make recommendations?

BACKGROUND AND OVERVIEW

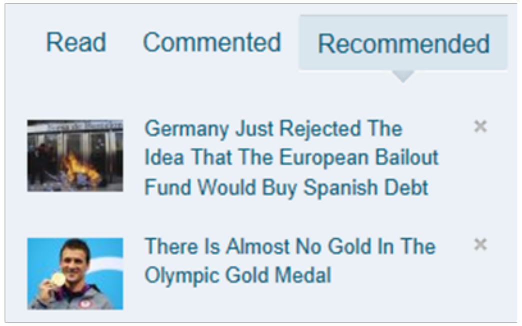
Raison d'être

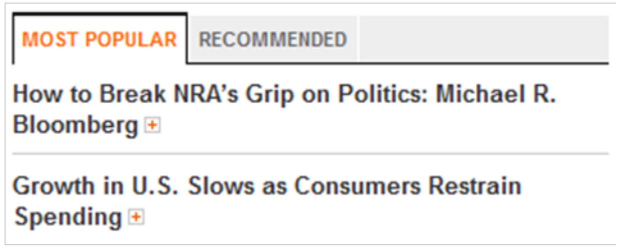
- The Web is an increasingly important medium for electronic and business transactions
- The Web provides ease in data collection
- The Web provides a user interface that can be employed to recommend items in a non-intrusive way

Recommender systems

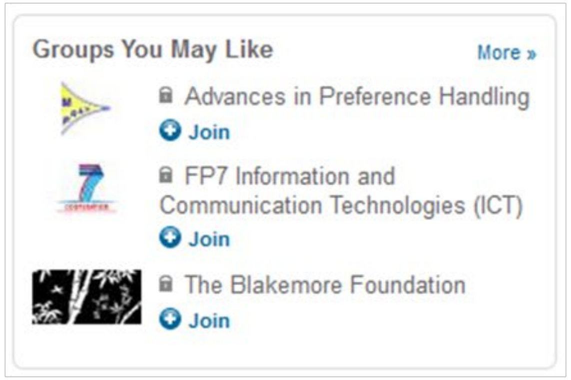
- Collaborative filtering methods (this lecture and the next)
- Content-based methods (not part of the course)
- Knowledge-based methods (not part of the course)
- Specialized methods exist for various data domains and contexts as well as for various application domains

E-COMMERCE AND NEWS

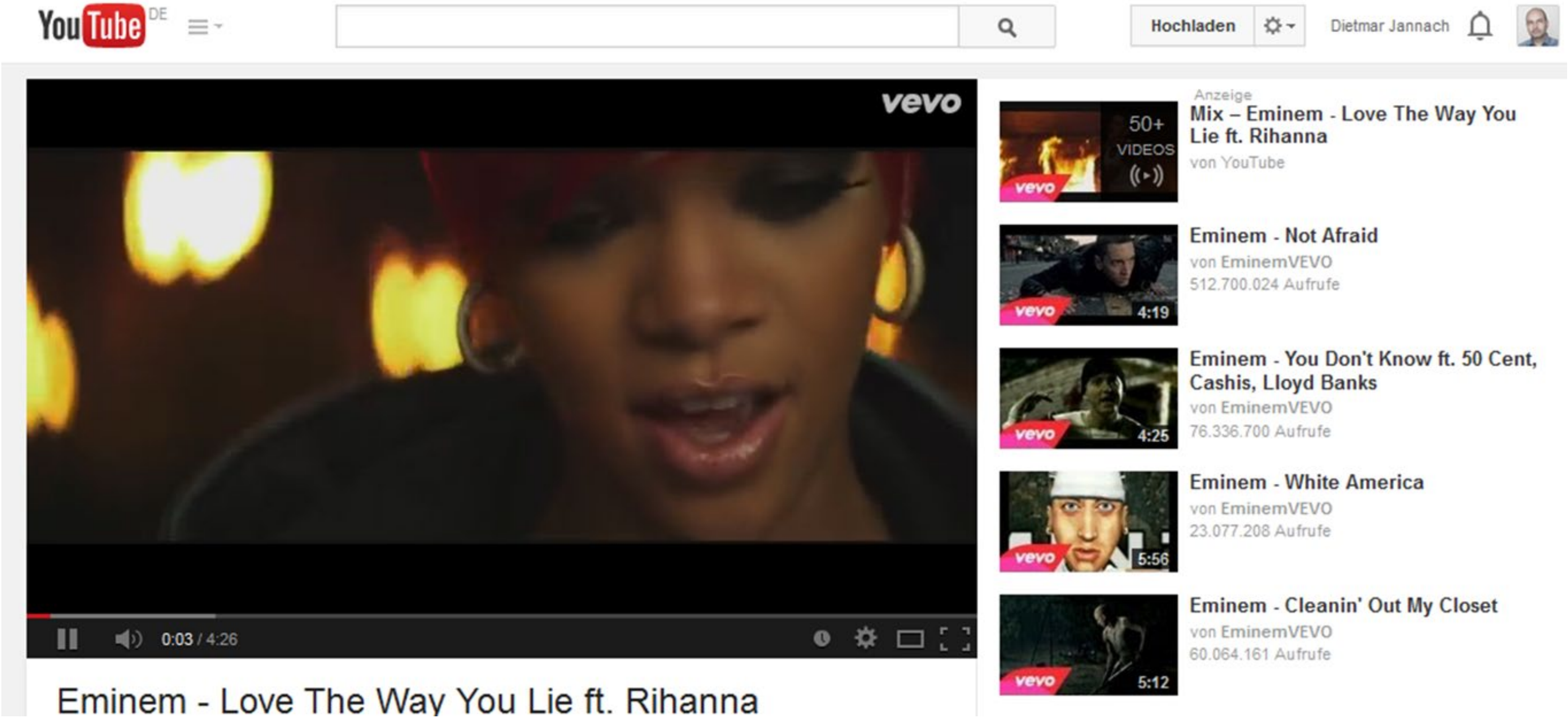




SOCIAL MEDIA



ENTERTAINMENT



WHY USE A RECOMMENDER SYSTEM?

Value for the customer

- Find things that are interesting
- Narrow down the set of choices
- Help me explore the space of options
- Discover new things
- Entertainment
- …

Value for the provider – increase in sales via

- Additional and probably unique personalized service for the customer
- Increase trust and customer loyalty
- Increase click trough rates, conversion etc.
- Opportunities for promotion, persuasion
- Obtain more knowledge about customers
- …

OPERATIONAL AND TECHNICAL GOALS

The business-centric goals of increased revenue can be achieved via

- Relevance – recommend items that are relevant to the user
- Novelty – recommend items that the user has not seen in the past
- Serendipity – recommend items that are unexpected
- Diversity – recommend different types of items

VALUE INDICATORS

Myths from industry

- Amazon.com generates X percent of their sales through the recommendation lists (30 < X < 70)
- Netflix generates X percent of their sales through the recommendation lists (30 < X < 70)

There must be some value in it

- News recommendation at Forbes.com (plus 37% CTR)

In academia

- A few studies exist that show the effect
  - increased sales, changes in sales behavior

OVERALL PROBLEM

Given

- The profile of the “target" **user**

The profile …

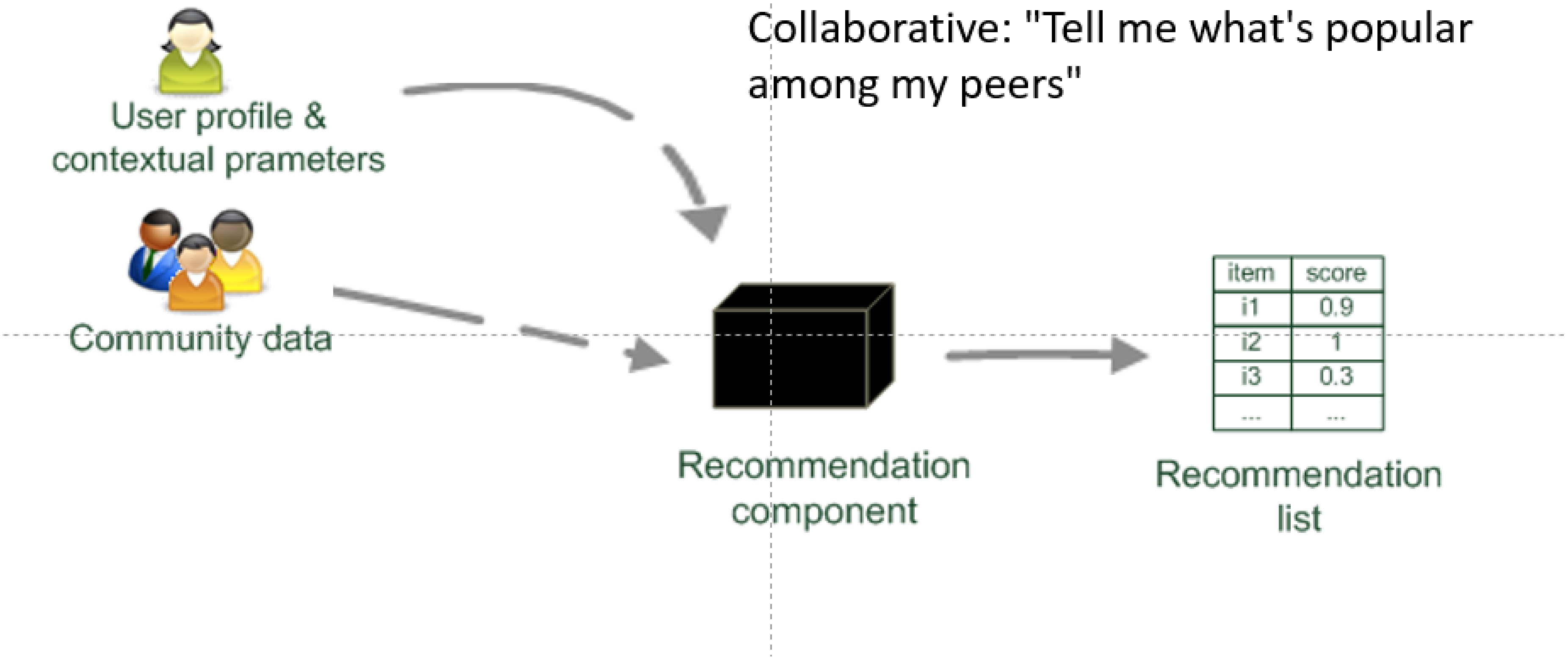
- … can include past user ratings, demographics and interest scores for item features (i.e.. clicks)

The problem …

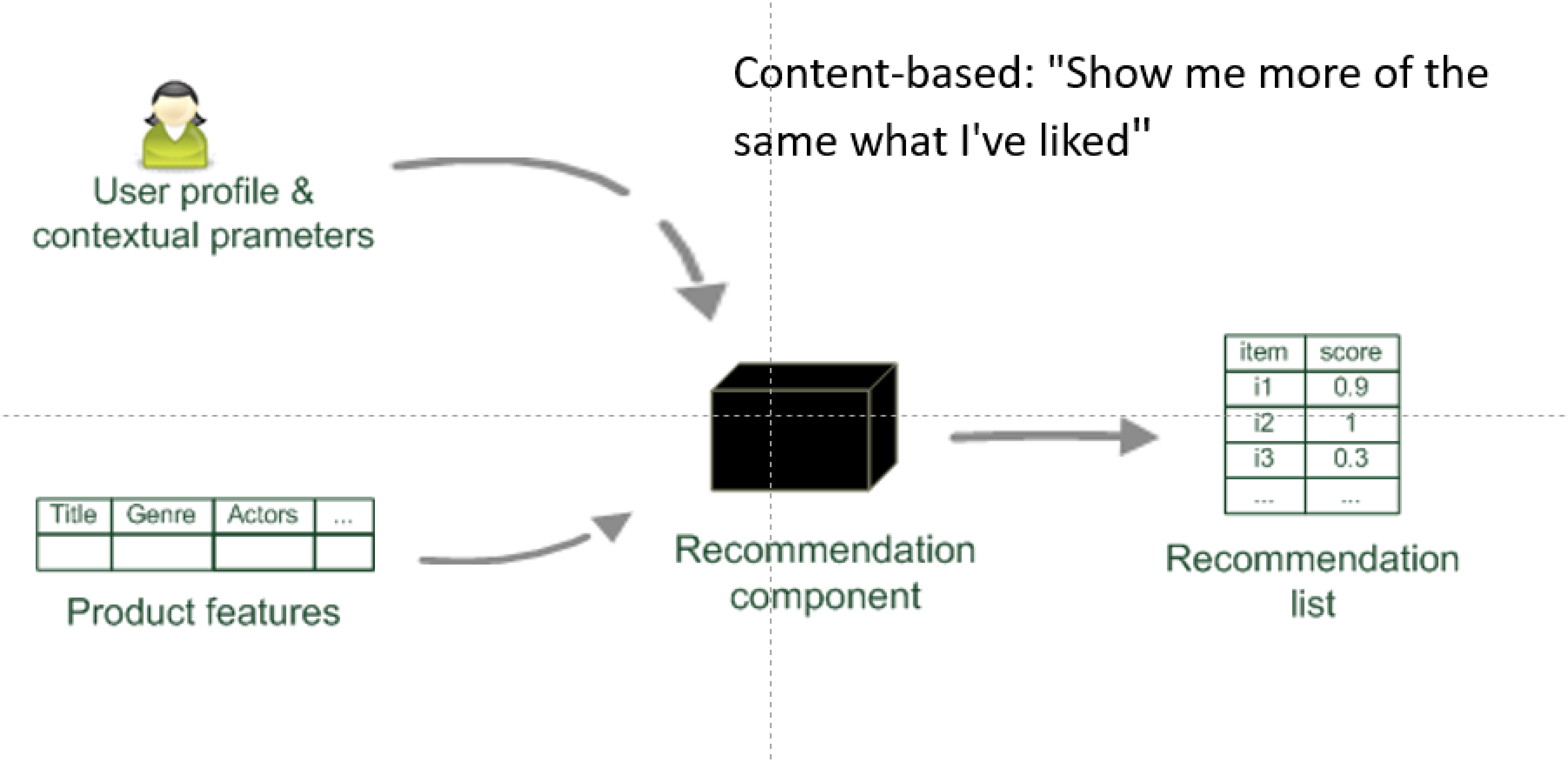
- … is to learn a function that predicts the relevance score for a given (typically unseen) **item** – matrix completion problem
- … is to learn the top-k most relevant **items** for a particular user – top-k recommendation problem

PARADIGMS OF RECOMMENDER SYSTEMS

Collaborative: "Tell me what's popular among my peers"



PARADIGMS OF RECOMMENDER SYSTEMS



COLLABORATIVE FILTERING

The most prominent approach to generate recommendations

- used by large, commercial e-commerce sites
- well-understood, various algorithms and variations exist
- applicable in many domains (book, movies, ..)

Approach

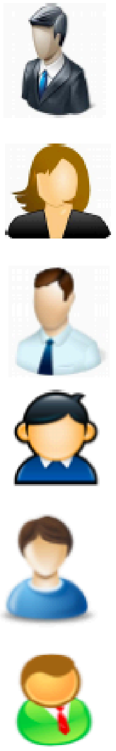
- use the preferences of a community to recommend items

Basic assumption and idea

- Users give ratings to catalog items (implicitly or explicitly)
- Patterns in the data help me predict the ratings of individuals, i.e., fill the missing entries in the rating matrix, e.g.,
    - there are customers with similar preference structures,

USER BASED CF

R_{mxn}



2			4	5	
5		4			1
		5		2	
	1		5		4
		4		2	
4	5		1		

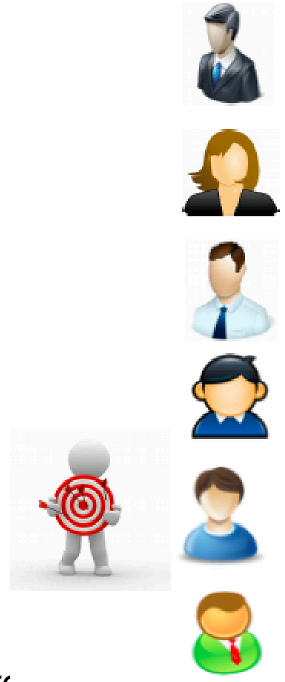
Each user has expressed

an opinion for some items

Explicit opinion:

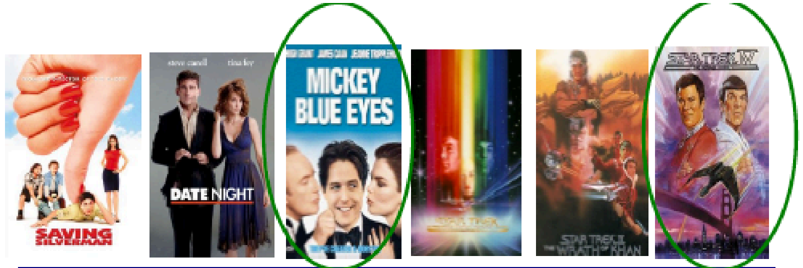
rating score

THE STEPS IN USER-BASED CF



2			4	5	
5		4			1
		5		2	
	1		5		4
		4			2
4	5		1		

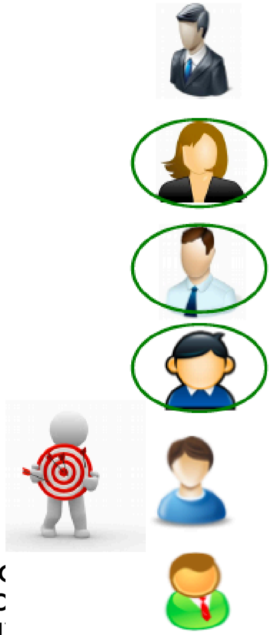
Target (or Active) user for whom the CF recommendation task is performed



2			4	5	
5		4			1
		5		2	
	1		5		4
		4			2
4	5		1		

1. Identify set of items rated by the target user

Example: User-based CF

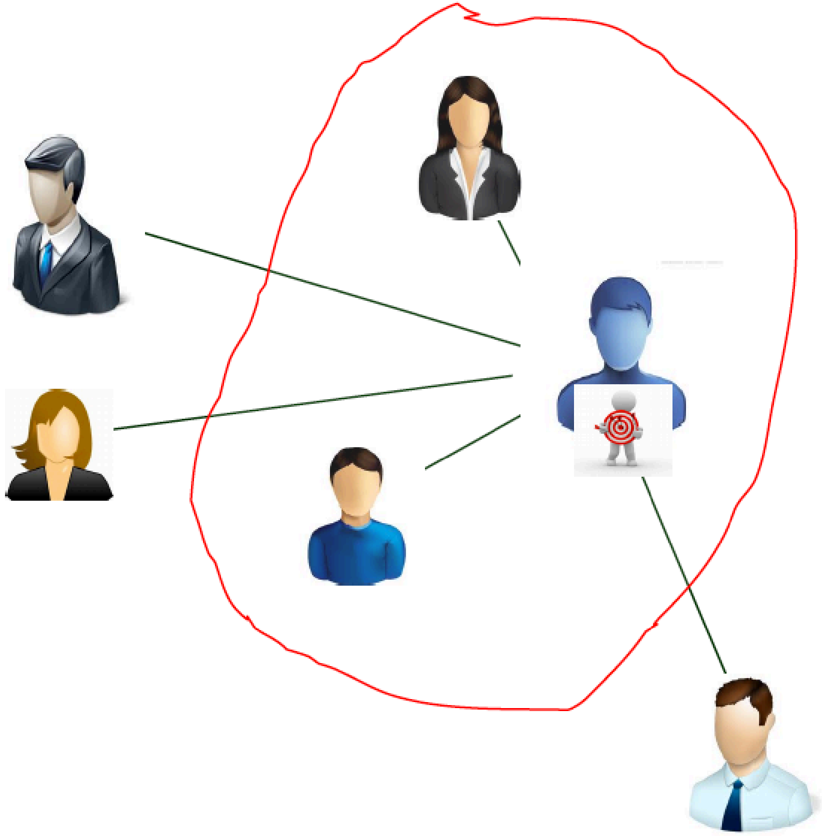


2			4	5	
5		4			1
		5		2	
	1		5		4
		4			2
4	5		1		

1. Identify set of items rated by the target user

2. Identify which other users rated 1+ items in this set (neighborhood formation)

User-based Similarity



3. Compute how similar each neighbor is to the target user (similarity function)

4. In case, select k most similar neighbors

CALCULATING SIMILARITY

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|} \quad \forall u \in \{1 \ldots m\} \tag{2.1}$$

Then, the Pearson correlation coefficient between the rows (users) u and v is defined as follows:

$$\text{Sim}(u,v) = \text{Pearson}(u,v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}} \tag{2.2}$$

I_u All items evaluated by u

$I_u \cap I_v$ All items evaluated both by u and v

User-based

Pearson correlation

R_{mxn}

						sim(u,v)
2			4	5		NA
5		4			1	0.87
		5		2		
	1		5		4	
		4			2	
4	5		1			NA

	μ
u	(4+2)/2=3
v	(5+4+1)/3=3,33

$$\frac{(4-3)(4-3,33)+(2-3)(1-3,33)}{\sqrt{(4-3)^2+(2-3)^2}\sqrt{(4-3,33)^2+(1-3,33)^2}}=0,87$$

CALCULATING PREDICTION

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \mathrm{Sim}(u,v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |\mathrm{Sim}(u,v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} \mathrm{Sim}(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\mathrm{Sim}(u,v)|} \qquad (2.4)$$

- Calculate, whether the neighbors' ratings for the unseen item *i* are higher or lower than their average
- Combine the rating differences – use the similarity with as a weight
- Add/subtract the neighbors' bias from the active user's average and use this as a prediction

- How many neighbors?
  - Only consider positively correlated neighbors (or higher threshold)
  - Can be optimized based on data set
  - Often, between 50 and 200

						sim(u,v)
2			4	5		NA
5		4			1	0.87
		5		2		1
	1		5		4	-1
		4			2	
4	5		1			NA

CALCULATION OF RATING VALUE FOR ITEM 1 AND 4 FOR TARGET USER

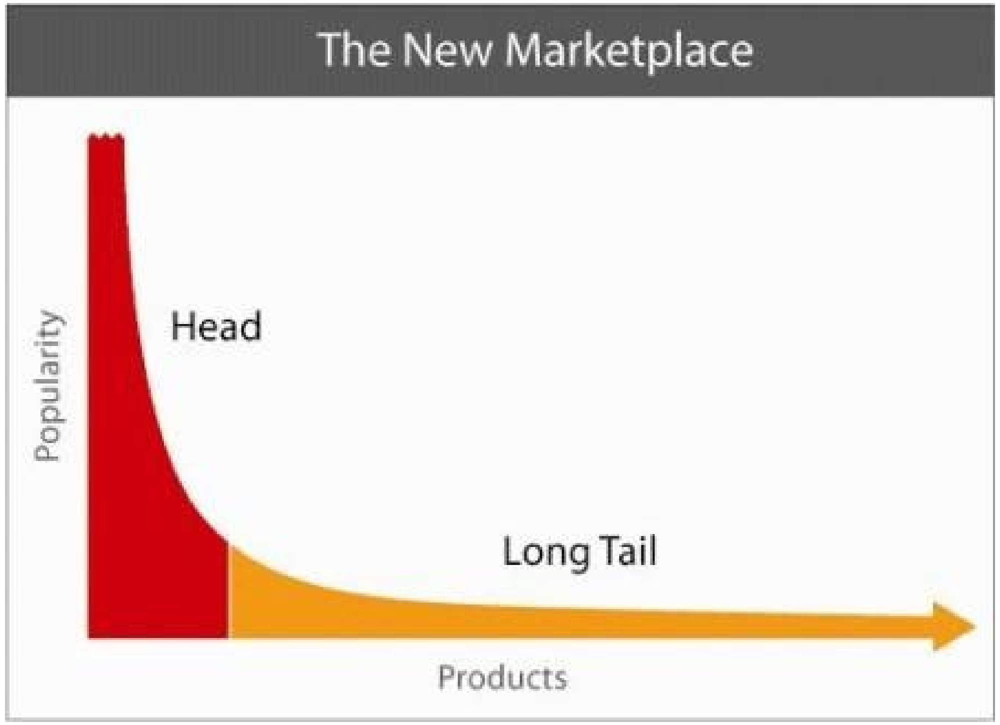
Users	μ	Sim(u,v)
1	3,67	NA
2	3,33	0,87
3	3,5	1
4	3,33	-1
Target	3	
6	3,33	NA

$$3+\frac{0,87*(5-3,33)}{0,87}=4,67$$

Item 4

$$3+\frac{-1*(5-3,33)}{1}=1,33$$

THE LONG TAIL



IMPACT OF LONG TAIL

information retrieval literature. Just as the notion of *Inverse Document Frequency* (idf) exists in the information retrieval literature [400], one can use the notion of *Inverse User Frequency* in this case. If m_j is the number of ratings of item j, and m is the total number of users, then the weight w_j of the item j is set to the following:

$$w_j = \log\left(\frac{m}{m_j}\right) \quad \forall j \in \{1 \dots n\} \tag{2.12}$$

Each item j is weighted by w_j both during the similarity computation and during the recommendation process. For example, the Pearson correlation coefficient can be modified to include the weights as follows:

$$\text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} w_k \cdot (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} w_k \cdot (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} w_k \cdot (r_{vk} - \mu_v)^2}} \tag{2.13}$$

ATTEMPTS TO IMPROVE THE STANDARD SOLUTION

Not all neighbor ratings might be equally "valuable"

- Agreement on commonly liked items is not so informative as agreement on controversial items
- **Possible solution**: Give more weight to items with large variance in ratings

Case amplification

- Intuition: Give additional weight to "very similar" neighbors, i.e., where the similarity value is close to 1.

Neighborhood selection

- Use similarity threshold or fixed number of neighbors

CONSIDERATIONS

Very simple scheme leading to quite accurate recommendations

- Still today often used as a baseline scheme

Possible issues

- Scalability
  - Thinking of millions of users and thousands of items
  - Pre-computation of similarities possible but potentially unstable
- Coverage
  - Problem of finding enough neighbors
  - Users with preferences for niche products

EVALUATION

Single split in train/test

- 80/20 as a possibility

k-fold , cross-validation approach

Evaluating the fit

- Model evaluation based upon a comparison between observed and predicted ratings
- Calculation of MAE, MSE, RMSE for each user and as an average over all users

Evaluating the recommendations if top-N recommendations

- Calculation of precision, recall, accuracy etc

ITEM-BASED CF

THE STEPS IN ITEM-BASED CF

Find the ratings across items for the target user

Find the similarity matrix between items

Select k most similar neighbor items to the target item

Predict ratings for the target item (prediction function) for the ”target” user

OVERALL IDEA IN ITEM-BASED COLLABORATIVE FILTERING

S	i_1	i_2	i_3	i_4	i_5	i_6	i_7	i_8	$\hat{r}_a$	k=3
i_1	-	0.1	0	**0.3**	**0.2**	**0.4**	0	0.1	-	
i_2	0.1	-	**0.8**	**0.9**	0	**0.2**	0.1	0	0.0	
i_3	0	**0.8**	-	0	**0.4**	0.1	0.3	**0.5**	4.6	
i_4	**0.3**	**0.9**	0	-	0	0.1	0	**0.2**	3.2	
i_5	**0.2**	0	**0.4**	0	-	0.1	**0.2**	0.1	-	
i_6	**0.4**	**0.2**	0.1	**0.3**	0.1	-	0	0.1	2.0	
i_7	0	**0.1**	**0.3**	0	**0.2**	0	-	0	4.0	
i_8	**0.1**	0	**0.5**	**0.2**	0.1	0.1	0	-	-	
u_a	2	?	?	?	4	?	?	5		

$$4,6 = \frac{4*0,4+5*0,5}{0,4+0,5}$$

Figure 2: Item-based collaborative filtering

From vignette to recommenderlab

PRE-PROCESSING ITEM TO ITEM

Pre-processing approach by Amazon.com (in 2003)

- Calculate all pair-wise item similarities in advance
- The neighborhood to be used at run-time is typically rather small, because only items are taken into account which the user has rated
- Item similarities are supposed to be more stable than user similarities

Memory requirements

- Up to N^2 pair-wise similarities to be memorized (N = number of items) in theory
- In practice, this is significantly lower (items with no co-ratings)
- Further reductions possible
  - Minimum threshold for co-ratings
  - Limit the neighborhood size (might affect recommendation accuracy)

PERFORMANCE IMPLICATIONS

User-based similarity is more dynamic and may lead to serendipity

Item-based similarity

- May lead to obvious recommendations and maybe also more relevant recommendations because user´s own ratings are used to produce recommendations
- We can precompute item neighbourhood.
- Online computation of the predicted ratings.
- Item-item more efficient because it requires much less memory to store item-item similarities compared to user-user similarities
- May give concrete reason for recommendations
  - Because you watched ....

USER- AND ITEM-BASED COLLABORATIVE FILTERING IN R

Recommender lab

- Works on both standard and binary rating matrices (both explicit and implicit feedback)
- A wide range of recommender algorithms
  - UBCF , IBCF and many other algorithms

Rrecsys

- Works on a standard user-item rating matrix (only explicit feedback)

RECOMMENDERLAB

AN EXAMPLE BASED UPON A DATA SET CALLED MOVIELENSE

Step 1

- Format data as a realRatingMatrix for efficient storage

```
R> r <- as(m, "realRatingMatrix")
R> r
```

- MovieLense is a preloaded dataset that comes with Recommenderlab

```
data(MovieLense)
help(MovieLense)
class(MovieLense)
dim(MovieLense)
```

```
> help(MovieLense)
> class(MovieLense)
[1] "realRatingMatrix"
attr(,"package")
[1] "recommenderlab"
> dim(MovieLense)
[1]  943 1664
```

Description

The 100k MovieLense ratings data set. The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. The data set contains about 100,000 ratings (1-5) from 943 users on 1664 movies. Movie metadata is also provided in `MovieLenseMeta`.

```r
### Step 1 - storage
data(MovieLense)
help(MovieLense)
class(MovieLense)
dim(MovieLense)
# Data is given in realRatingMatrix format  ; Optimized to store sparse matrices
str(MovieLense,vec.len=2) #not as we normally reference list elements by \\$ but \\@
methods(class=class(MovieLense)) # methods applicable to this class
```

```
> class(MovieLense)
[1] "realRatingMatrix"
attr(,"package")
[1] "recommenderlab"
> dim(MovieLense)
[1]  943 1664
> # Data is given in realRatingMatrix format  ; Optimized to store sparse matrices
> str(MovieLense,vec.len=2) #not as we normally reference list elements by \\$ but \\@
Formal class 'realRatingMatrix' [package "recommenderlab"] with 2 slots
  ..@ data     :Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
  .. .. ..@ i       : int [1:99392] 0 1 4 5 9 ...
  .. .. ..@ p       : int [1:1665] 0 452 583 673 882 ...
  .. .. ..@ Dim     : int [1:2] 943 1664
  .. .. ..@ Dimnames:List of 2
  .. .. .. ..$ : chr [1:943] "1" "2" ...
  .. .. .. ..$ : chr [1:1664] "Toy Story (1995)" "GoldenEye (1995)" ...
  .. .. ..@ x       : num [1:99392] 5 4 4 4 4 ...
  .. .. ..@ factors : list()
  ..@ normalize: NULL
> methods(class=class(MovieLense)) # methods applicable to this class
 [1] [                      [<-                    binarize               calcPredictionAccuracy
 [5] coerce                 colCounts              colMeans               colSds
 [9] colSums                denormalize            dim                    dimnames
[13] dimnames<-             dissimilarity          evaluationScheme       getData.frame
[17] getList                getNormalize           getRatingMatrix        getRatings
[21] getTopNLists           hasRating              image                  normalize
[25] nratings               Recommender            removeKnownRatings     rowCounts
[29] rowMeans               rowSds                 rowSums                sample
[33] show                   similarity
see '?methods' for accessing help and source code
> 
```

RECOMMENDERLAB

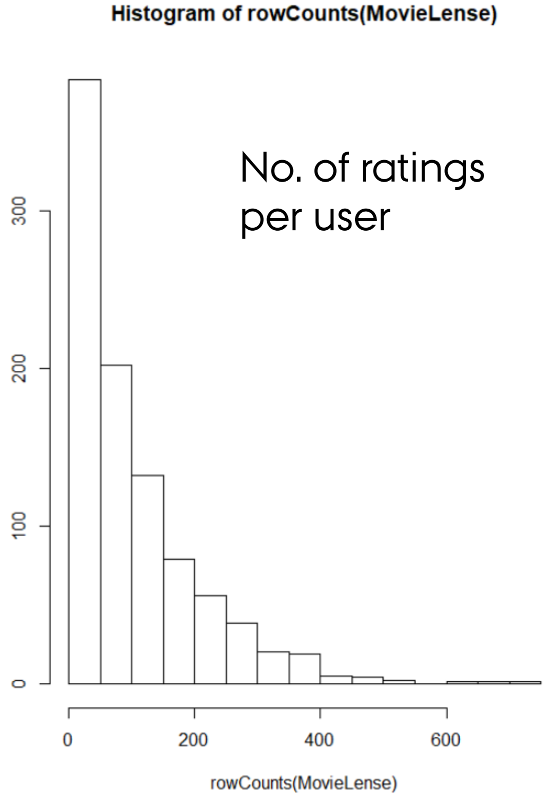
AN EXAMPLE BASED UPON A DATA SET CALLED MOVIELENSE

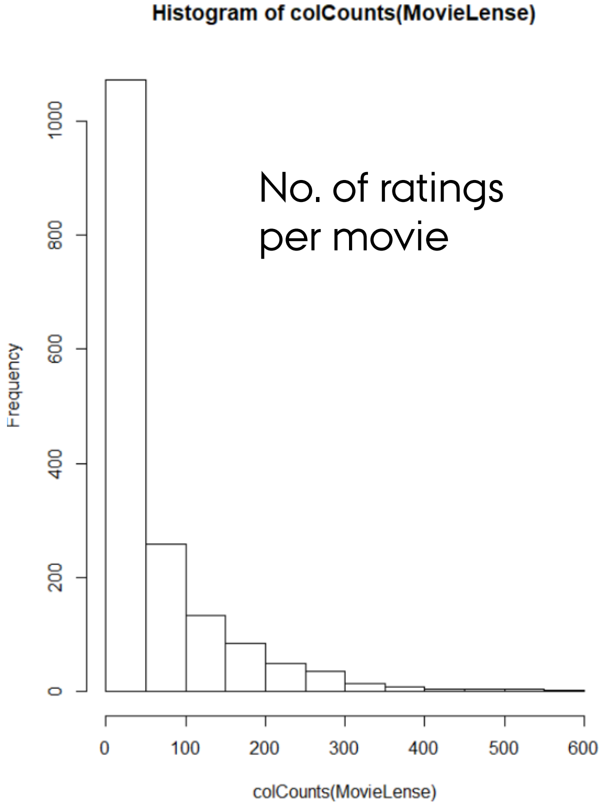
Step 2

Explore data

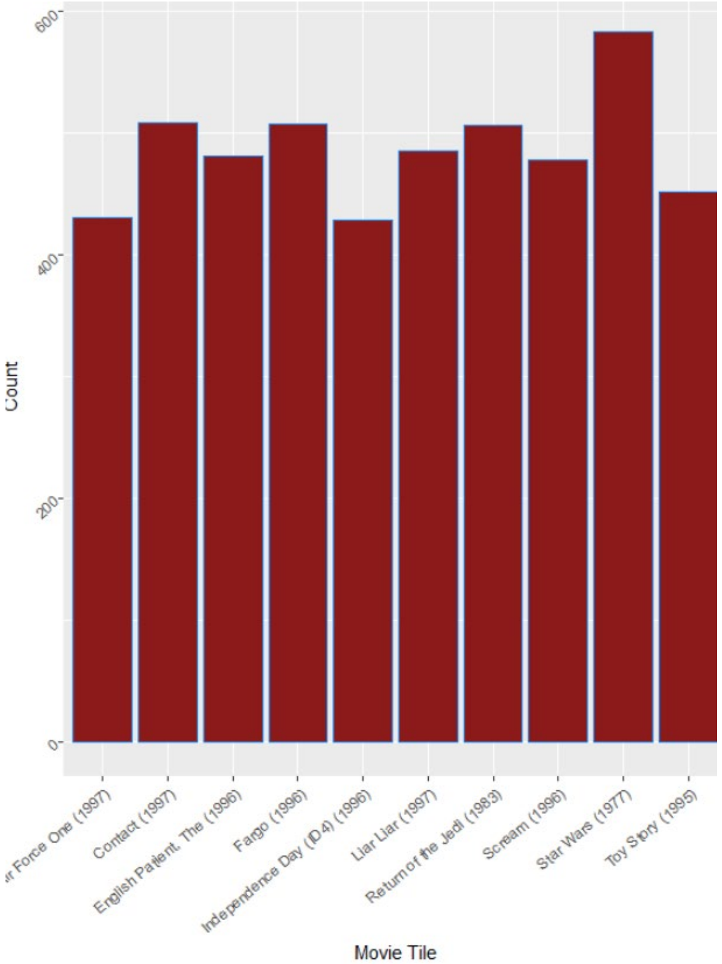
```r
### Step 2 - explore data
## Loading the metadata that gets loaded with main dataset
moviemeta <- MovieLenseMeta
class(moviemeta)
colnames(moviemeta)

## What do we know about the films?
library(pander)
pander(head(moviemeta,2),caption = "First few Rows within Movie Meta Data ")
# Look at the first few ratings of the first user
head(as(MovieLense[1,], "list")[[1]])
# Number of ratings per user
hist(rowCounts(MovieLense))
# Number of ratings per movie
hist(colCounts(MovieLense))
# Top 10 movies
movie_watched <- data.frame(
  movie_name = names(colCounts(MovieLense)),
  watched_times = colCounts(MovieLense)
)
top_ten_movies <- movie_watched[order(movie_watched$watched_times, decreasing = TRUE), ][1:10, ]
# Plot top 10
ggplot(top_ten_movies) + aes(x=movie_name, y=watched_times) +
  geom_bar(stat = "identity",fill = "firebrick4", color = "dodgerblue2") + xlab("Movie Tile") + ylab("Count") +
  theme(axis.text = element_text(angle = 40, hjust = 1))
```





The 10 most rated movies



Step 3

Split in training and test

```
# Training and test set: At least 30 items evaluated or at least 100 users for each item
rates <- MovieLense[rowCounts(MovieLense) > 30, colCounts(MovieLense) > 100]
rates1 <- rates[rowCounts(rates) > 30,]
# We randomly define the which_train vector that is True for users in the training set and FALSE for the others.
# We will set the probability in the training set as 80%
set.seed(1234)
which_train <- sample(x = c(TRUE, FALSE), size = nrow(rates1), replace = TRUE, prob = c(0.8, 0.2))
# Define the training and the test sets
recc_data_train <- rates1[which_train, ]
recc_data_test <- rates1[!which_train, ]
```

STEP 4
LIST OF RECOMMENDER MODELS

```
recommender_models <- recommenderRegistry$get_entries(dataType="realRatingMatrix")
names(recommender_models)
lapply(recommender_models,"[[","description")
recommender_models$IBCF_realRatingMatrix$parameters
```

```
> recommender_models <- recommenderRegistry$get_entries(dataType="realRatingMatrix")
> names(recommender_models)
 [1] "HYBRID_realRatingMatrix"       "ALS_realRatingMatrix"         "ALS_implicit_realRatingMatrix"
 [4] "IBCF_realRatingMatrix"         "LIBMF_realRatingMatrix"       "POPULAR_realRatingMatrix"
 [7] "RANDOM_realRatingMatrix"       "RERECOMMEND_realRatingMatrix" "SVD_realRatingMatrix"
[10] "SVDF_realRatingMatrix"         "UBCF_realRatingMatrix"
```

UBCF	User-based	2.3.1
IBCF	Item-based	2.3.2
SVD	Singular value decomposition	2.5
ALS	Alternating least squares	3.6.4.4
SVDF	Steepest descent	3.6.4

```
$ALS_realRatingMatrix
Recommender method: ALS for realRatingMatrix Description: Recommender fo
  explicit ratings based on latent factors, calculated by alternating le
  squares algorithm. Reference: Yunhong Zhou, Dennis Wilkinson, Robert
  Schreiber, Rong Pan (2008). Large-Scale Parallel Collaborative Filteri
  the Netflix Prize, 4th Int'l Conf. Algorithmic Aspects in Information
  Management, LNCS 5034.
Parameters:
  normalize lambda n_factors n_iterations min_item_nr seed
1      NULL    0.1        10           10           1 NULL
```

```
$IBCF_realRatingMatrix
Recommender method: IBCF for realRatingMatrix Description: Recommender
  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
```

```
$SVD_realRatingMatrix
Recommender method: SVD for realRatingMatrix Description: Recommender b
  SVD approximation with column-mean imputation. Reference: NA
Parameters:
   k maxiter normalize
1 10     100  "center"
```

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

```
$RANDOM_realRatingMatrix
Recommender method: RANDOM for realRatingMatrix Description: Produce random
  recommendations (real ratings). Reference: NA
Parameters: None
```

```
$SVDF_realRatingMatrix
Recommender method: SVDF for realRatingMatrix Description: Recommender based on
  Funk SVD with gradient descend
  (https://sifter.org/~simon/journal/20061211.html). Reference: NA
Parameters:
   k gamma lambda min_epochs max_epochs min_improvement normalize verbose
1 10 0.015  0.001         50        200           1e-06  "center"   FALSE
```

```
$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
```

AN IBCF MODEL AND SOME PREDICTIONS

```r
# Let's build the recommender IBCF - cosine:
recc_model <- Recommender(data = recc_data_train, method = "IBCF", parameter = list(k = 30))
# We have now created a IBCF Recommender Model
# We will define n_recommended that defines the number of items to recommend to
# each user and with the predict function, create prediction(recommendations) for the test set.
n_recommended <- 5
recc_predicted <- predict(object = recc_model, newdata = recc_data_test, n = n_recommended)
# This is the recommendation for the first user
recc_predicted@items[[1]]
# Now let's define a list with the recommendations for each user
recc_matrix <- lapply(recc_predicted@items, function(x){
  colnames(rates)[x]
})
# Let's take a look the recommendations for the first four users:
recc_matrix[1:4]
```

A FEW PREDICTIONS FROM IBCF

```
> # We will define n_recommended that defines the number of items to recommend to
> # each user and with the predict function, create prediction(recommendations) for the test set.
> n_recommended <- 5
> recc_predicted <- predict(object = recc_model, newdata = recc_data_test, n = n_recommended)
> # This is the recommendation for the first user
> recc_predicted@items[[1]]
[1] 217 318 198  70 173
> # Now let's define a list with the recommendations for each user
> recc_matrix <- lapply(recc_predicted@items, function(x){
+   colnames(rates)[x]
+ })
> # Let's take a look the recommendations for the first four users:
> recc_matrix[1:4]
$`0`
[1] "Wag the Dog (1997)"        "Amistad (1997)"            "Apt Pupil (1998)"
[4] "Bound (1996)"              "Good Will Hunting (1997)"

$`1`
[1] "Contact (1997)"            "Crimson Tide (1995)"       "Shine (1996)"
[4] "L.A. Confidential (1997)"  "Good Will Hunting (1997)"

$`2`
[1] "Taxi Driver (1976)"                      "Die Hard (1988)"
[3] "Brazil (1985)"                           "Good, The Bad and The Ugly, The (1966)"
[5] "Clockwork Orange, A (1971)"

$`3`
[1] "Babe (1995)"                "Swingers (1996)"            "Brazil (1985)"
[4] "Unforgiven (1992)"          "This Is Spinal Tap (1984)"

> |
```

A UBCF MODEL WITH STANDARD SETTINGS

```r
# UBCF = User-based collaborative filtering
# The method computes the similarity between users with cosine
# Let's build a recommender model leaving the parameters to their defaults.
recc_model <- Recommender(data = recc_data_train, method = "UBCF")
# A UBCF recommender has now been created
recc_predicted <- predict(object = recc_model, newdata = recc_data_test, n = n_recommended)
# Let's define a list with the recommendations to the test set users.
recc_matrix <- sapply(recc_predicted@items, function(x) {
  colnames(rates)[x]
})
# Again, let's look at the first four users
recc_matrix[,1:4]
```

IBCF Evaluation on a comparison between observed and predicted ratings on the items to keep ("unknown")

Cross validation

How to

What model and data

The criteria

```
# Cross validation
# We can split the data into some chunks, take a chunk out as the test set, and evaluate the
# accuracy. Then we can do the same with each other chunk and compute the average accuracy.
# Here we construct the evaluation model
n_fold <- 4
rating_threshold <- 4 # threshold at which we consider the item to be good
items_to_keep <- 20 # given=20 means that while testing the model use only 20 randomly picked
# ratings from every user to predict the unknown ratings in the test set the known data set has
# the ratings specified by given and the unknown data set the remaining ratings used for validation
eval_sets <- evaluationScheme(data = rates1, method = "cross-validation", k = n_fold,
                              given = items_to_keep, goodRating = rating_threshold)
size_sets <-sapply(eval_sets@runsTrain, length)
size_sets
#IBCF
model_to_evaluate <- "IBCF"
model_parameters <- NULL  #   we use the standard settings
eval_recommender <-Recommender(data = getData(eval_sets, "train"), method = model_to_evaluate, para
# The IBCF can recommend new items and predict their ratings. In order to build
# the model, we need to specify how many items we want to recommend, for example, 5.
items_to_recommend <- 5
# We can build the matrix with the predicted ratings using the predict function:
eval_prediction <- predict(object = eval_recommender, newdata = getData(eval_sets, "known"), n = it
# By using the calcPredictionAccuracy, we can calculate the Root mean square
# error (RMSE), Mean squared error (MSE), and the Mean absolute error (MAE).
eval_accuracy <- calcPredictionAccuracy(
  x = eval_prediction, data = getData(eval_sets, "unknown"), byUser = TRUE
  )
# This is a small sample of the results for the Prediction and Accuracy
head(eval_accuracy)
# Now, let's take a look at the RMSE by each user
ggplot(data=as.data.frame(eval_accuracy),aes(x=RMSE)) + geom_histogram(binwidth = 0.1) +
  ggtitle("Distribution of the RMSE by user")
# However, we need to evaluate the model as a whole, so we will set the byUser to False
eval_accuracy <- calcPredictionAccuracy(
  x = eval_prediction, data = getData(eval_sets, "unknown"), byUser = FALSE
)
eval_accuracy #for IBCF
```

EVALUATION
THE PREDICTION ERROR FOR THE UNUSED TEST USER *RATINGS*

```
> head(eval_accuracy)
        RMSE       MSE       MAE
1  1.0928010 1.1942140 0.8040594
5  1.6479916 2.7158764 1.3158936
7  0.9336741 0.8717473 0.7407975
8  1.0446759 1.0913477 0.7618064
11 1.1349793 1.2881781 0.8764519
16 1.0074408 1.0149370 0.5234495

> eval_accuracy #for IBCF
     RMSE       MSE       MAE
1.0902557 1.1886574 0.8100222
```

A typical way to evaluate a prediction is to compute the deviation of the prediction from the true value. This is the basis for the *Mean Average Error (MAE)*

$$\mathrm{MAE} = \frac{1}{|\mathcal{K}|} \sum_{(i,j)\in\mathcal{K}} |r_{ij} - \hat{r}_{ij}|, \tag{8}$$

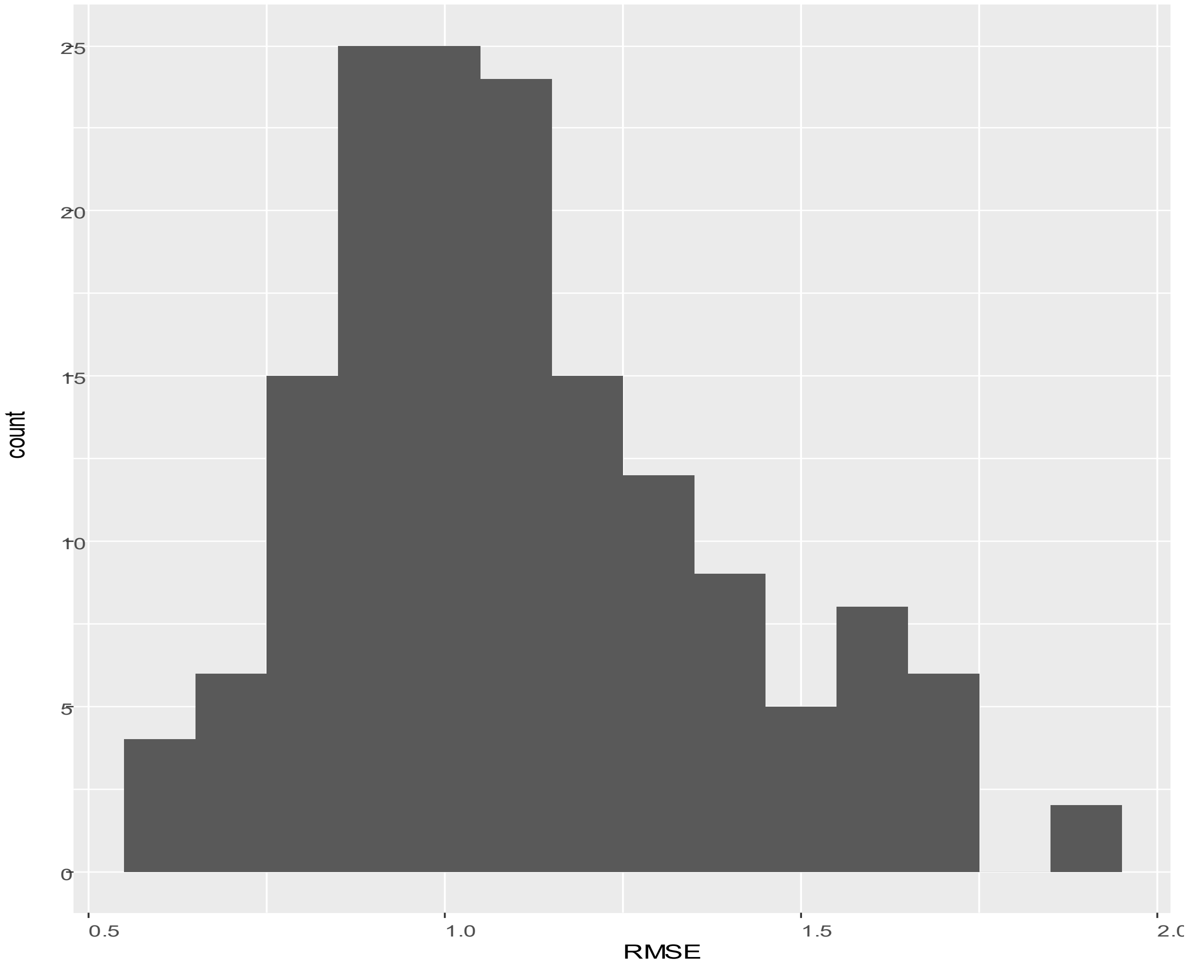
where $\mathcal{K}$ is the set of all user-item pairings (i, j) for which we have a predicted rating $\hat{r}_{ij}$ and a known rating r_{ij} which was not used to learn the recommendation model.

Another popular measure is the *Root Mean Square Error (RMSE)*.

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{(i,j)\in\mathcal{K}}(r_{ij} - \hat{r}_{ij})^2}{|\mathcal{K}|}} \tag{9}$$

RMSE penalizes larger errors stronger than MAE and thus is suitable for situations where small prediction errors are not very important.

Distribution of the RMSE by user



IBCF EVALUATED ON TOP-N

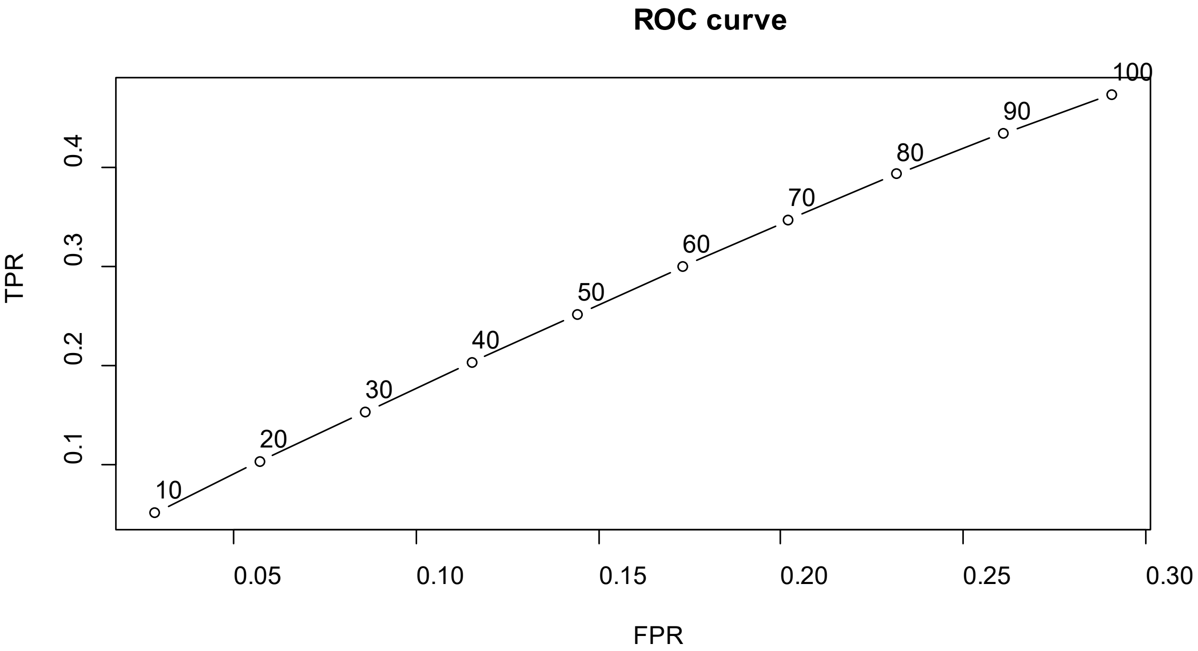
```r
# Confusion matrix good threshold =4
results <- evaluate(x = eval_sets, method = model_to_evaluate, n = seq(10, 100, 10)) #n number top-n recommendations
# results object is an evaluationResults object containing the results of the evaluation.
# Each element of the list corresponds to a different split of the k-fold.
# Let's look at the first element
head(getConfusionMatrix(results)[[1]])
# In this case, look at the first four columns
# True Positives (TP): These are recommended items that have been purchased.
# False Positives (FP): These are recommended items that haven't been purchased
# False Negatives (FN): These are not recommended items that have been purchased.
# True Negatives (TN): These are not recommended items that haven't been purchased.
# If we want to take account of all the splits at the same time, we can just sum up the indices:
columns_to_sum <- c("TP", "FP", "FN", "TN")
indices_summed <- Reduce("+", getConfusionMatrix(results))[, columns_to_sum]
head(indices_summed)

## Building an ROC curve. Will need these factors
# 1. True Positive Rate (TPR): Percentage of purchased items that have been recommended. TP/(TP + FN)
# 2. False Positive Rate (FPR): Percentage of not purchased items that have been recommended. FP/(FP + TN)
plot(results, annotate = TRUE, main = "ROC curve")

## We can also look at the accuracy metrics as well
# precision: Percentage of recommended items that have been purchased. FP/(TP + FP)
# recall: Percentage of purchased items that have been recommended. TP/(TP + FN) = True Positive Rate
plot(results, "prec/rec", annotate = TRUE, main = "Precision-Recall")
```

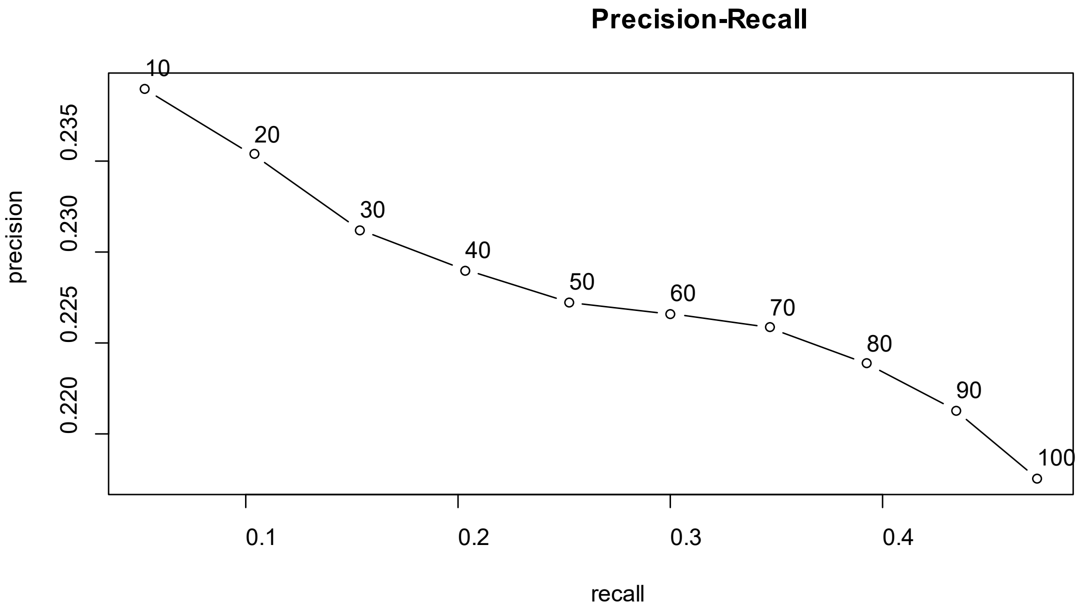
```
> head(getConfusionMatrix(results)[[1]])
            TP        FP       FN       TN        N precision     recall        TPR        FPR  n
[1,]  2.371795  7.628205 43.01282 259.2179 312.2308 0.2371795 0.05084429 0.05084429 0.02836862 10
[2,]  4.750000 15.250000 40.63462 251.5962 312.2308 0.2375000 0.10552058 0.10552058 0.05681358 20
[3,]  7.134615 22.865385 38.25000 243.9808 312.2308 0.2378205 0.15890568 0.15890568 0.08529809 30
[4,]  9.275641 30.724359 36.10897 236.1218 312.2308 0.2318910 0.20908467 0.20908467 0.11472110 40
[5,] 11.352564 38.647436 34.03205 228.1987 312.2308 0.2270513 0.25548075 0.25548075 0.14432946 50
[6,] 13.391026 46.608974 31.99359 220.2372 312.2308 0.2231838 0.29992798 0.29992798 0.17409775 60
> # In this case, look at the first four columns
> # True Positives (TP): These are recommended items that have been purchased.
> # False Positives (FP): These are recommended items that haven't been purchased
> # False Negatives (FN): These are not recommended items that have been purchased.
> # True Negatives (TN): These are not recommended items that haven't been purchased.
> # If we want to take account of all the splits at the same time, we can just sum up the indices:
> columns_to_sum <- c("TP", "FP", "FN", "TN")
> indices_summed <- Reduce("+", getConfusionMatrix(results))[, columns_to_sum]
> head(indices_summed)
            TP        FP       FN        TN
[1,]  9.557692  30.44231 172.7179 1035.3974
[2,] 18.826923  61.17308 163.4487 1004.6667
[3,] 27.743590  92.25641 154.5321  973.5833
[4,] 36.628205 123.32051 145.6474  942.5192
[5,] 45.423077 154.46154 136.8526  911.3782
[6,] 54.352564 185.46795 127.9231  880.3718
```

ROC AND PRECISION/RECALL, IBCF



Percentage Falsely predicted positive

How many selected items are relevant ?

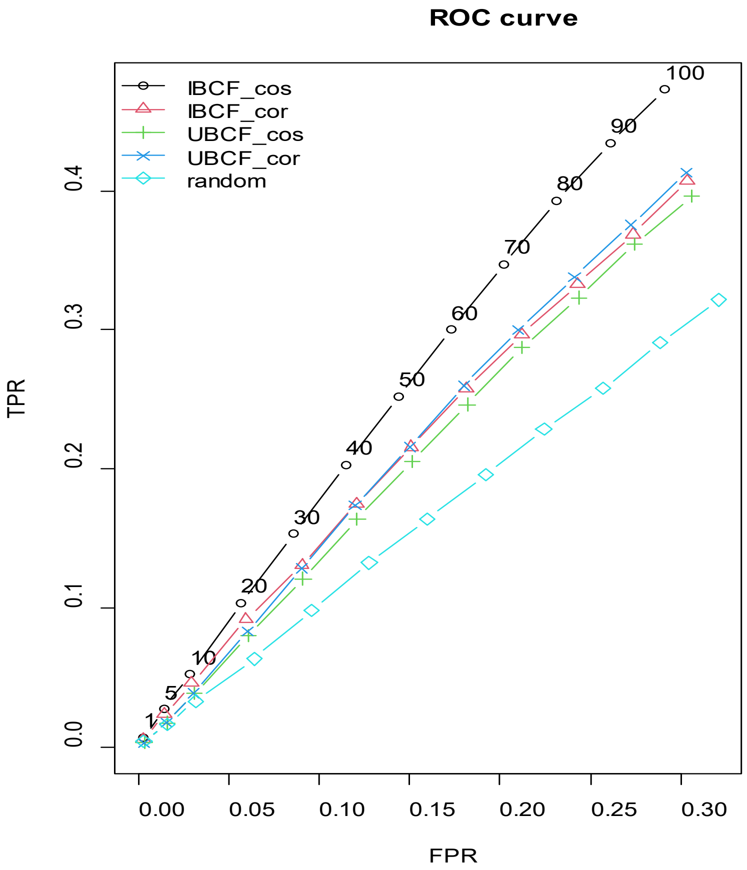


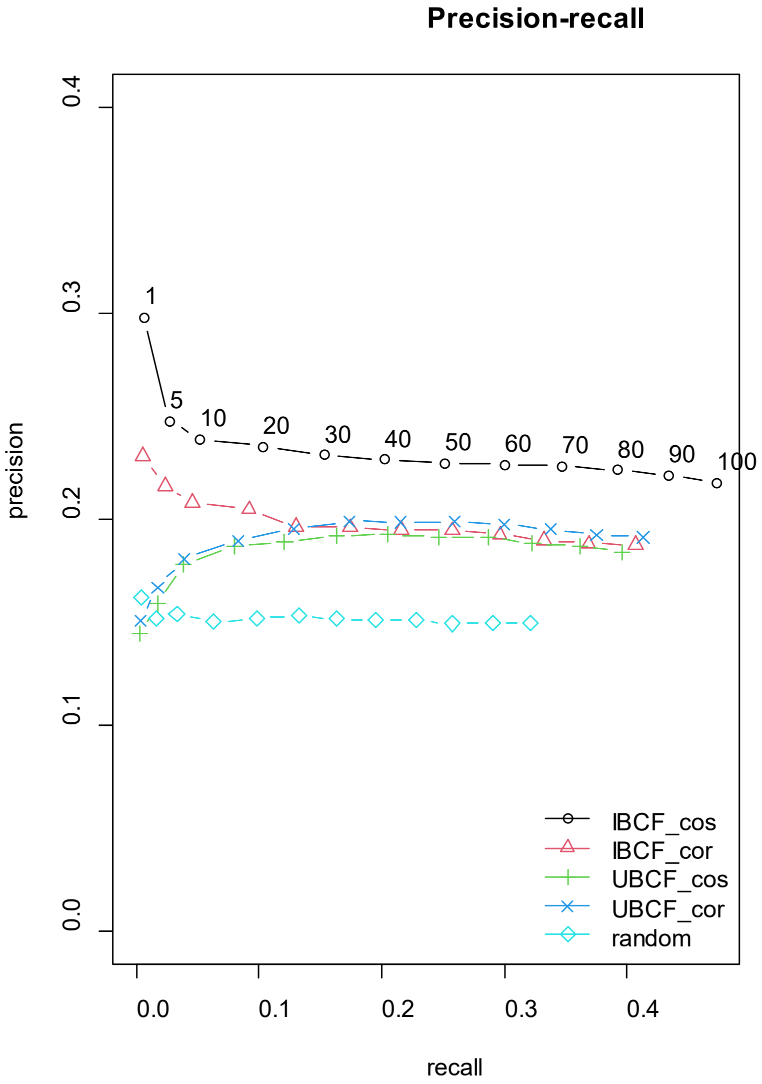
How many relevant items are selected ?

COMPARING MODELS

```r
## Comparing models
models_to_evaluate <- list(IBCF_cos = list(name = "IBCF", param = list(method = "cosine")),
                           IBCF_cor = list(name = "IBCF", param = list(method = "pearson")),
                           UBCF_cos = list(name = "UBCF", param = list(method = "cosine")),
                           UBCF_cor = list(name = "UBCF", param = list(method = "pearson")),
                           random = list(name = "RANDOM", param = NULL))
# In order to evaluate the models, we need to test them, varying the number of items.
n_recommendations <- c(1,5,seq(10,100,10))
# Now let's run and evaluate the models
list_results <- evaluate(x = eval_sets, method = models_to_evaluate, n = n_recommendations)
# Plot the ROC curve
plot(list_results, annotate = 1, legend = "topleft")
title("ROC curve")
# Plot precision-recall
plot(list_results, "prec/rec", annotate = 1, legend = "bottomright", ylim = c(0,0.4))
title("Precision-recall")
```

COMPARING UBCF AND IBCF ON TOP N , CROSS-VALIDATION





MORE EVALUATION CRITERIA

Coverage

- For how many users can we make recommendations?
- How many catalog items are ever recommended?

Diversity & Novelty

- Avoiding monotone lists, discover new (families of) items

Serendipity

- Unexpected and surprising items might be valuable

Familiarity

- Give the user the impression of understanding his/her needs

Biases

- Does the recommender only recommend popular items and blockbusters?

THURSDAY

Recommender systems part 2 Latent factor models

Read Aggarwal chapter 2 p.51-56 and section 3.6 p.90-109

UNIVERSITAS ARHUSIENSIS

AARHUS
BSS

DEPARTMENT OF ECONOMICS
AND BUSINESS ECONOMICS
AARHUS UNIVERSITY