title: “HarvardX: PH125.9x Data Science–My Own Jokes Recommendation System Project Submission–CYO” author: “Joaquin Emilio Jaime Coronel” date: “May 25 2021” output: html_document

Introduction

Recommendation systems are very important on many enterprises and social spheres around the world.They are used to help sell or offer products to users that have not seen them or have bought them.

In this ocassion we are recommending jokes to read, using the Jester5k data that contains the whole info to do this.The variables affected by the code are connected to those learned on the course series, that take us to know the best joke to recommend to others to read it and observe other characteristics.

The goal of this project is to highlight the importance of recommending systems that are taking an interesting place on our daily life and businesses.

The key steps to follow on this project were to have crystal clear the R package for recommendation, choose the data (Jester5k), prepare it, explore it, show the models used, visualize the most of the data to make clearer the understanding of each section,show results, comment it and conclude with an interesting paragraph telling to others the importance of the Recommending Systems on this modern life.

Installing packages.

if(!“gplots” %in% rownames(installed.packages())){ install.packages(“gplots”)} if(!“qplot” %in% rownames(installed.packages())){ install.packages(“qplot”)}

```
if(!require("ggplot2")) install.packages("ggplot2", repos =
"http://cran.us.r-project.org")

## Loading required package: ggplot2

if(!require("data.table")) install.packages("data.table", repos =
"http://cran.us.r-project.org")

## Loading required package: data.table

if(!"recommenderlab" %in% rownames(installed.packages())){
  install.packages("recommenderlab")}
library("recommenderlab")

## Loading required package: Matrix
```

```
## Loading required package: arules

##
## Attaching package: 'arules'

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

## Loading required package: proxy

##
## Attaching package: 'proxy'

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

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

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

## Loading required package: registry

## Registered S3 methods overwritten by 'registry':
##   method               from
##   print.registry_field proxy
##   print.registry_entry proxy

library("ggplot2")
data(Jester5k)
Jester5k

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

library("data.table")

data_to_use <- Jester5k
```

Line code to reproduce random components of recommenderlab.

```
set.seed(1)
```

Datasets that can be used to play with Recommenderlab functions.

```
data_package <- data(package = "recommenderlab")
data_package$results[, "Item"]
```

```
## [1] "Jester5k"                     "JesterJokes (Jester5k)"
## [3] "MSWeb"                        "MovieLense"
## [5] "MovieLenseMeta (MovieLense)" "MovieLenseUser (MovieLense)"
```

Look the methods we use with this kind of objects

```
methods(class = class(data_to_use))
```

```
##  [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
```

Making a comparison of the size of data_to_use with R matrix:

```
object.size(data_to_use)
```

```
## 4674488 bytes
```

Compute to know the times the recommenderlab matrix is more compact.

```
object.size(as(data_to_use, "matrix"))
```

```
## 4326888 bytes
```

Collaborative filtering algorithms use measuring the similarity between users and items. For this purpose, we use the similarity function.To do this we compute this using the cosine distance

Now compute the matrix of similarity.

```
similarity_users <- similarity(data_to_use[1:4, ], method =
                                 "cosine", which = "users")

similarity_users
```

```
##              u2841       u15547      u15221
## u15547 0.500128118
## u15221 0.158718154 0.089446933
## u15573 0.197523538 0.003513988 0.133177034
```

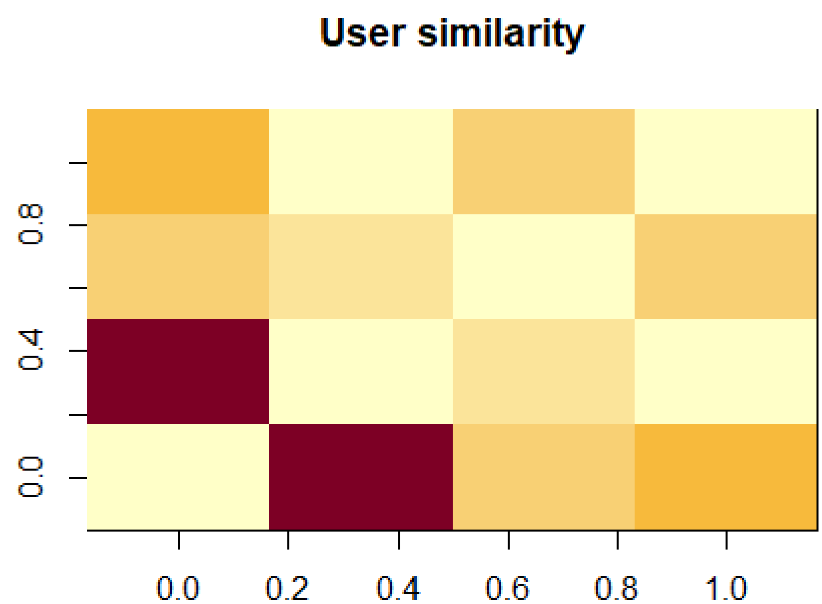
Let´s explore the dissimilarities.

Let´s convert similarity_users into a matrix.

```
as.matrix(similarity_users)

##             u2841      u15547     u15221      u15573
## u2841  0.0000000 0.500128118 0.15871815 0.197523538
## u15547 0.5001281 0.000000000 0.08944693 0.003513988
## u15221 0.1587182 0.089446933 0.00000000 0.133177034
## u15573 0.1975235 0.003513988 0.13317703 0.000000000
```

Use image to visualize the matrix. Rows and columns corresponds to a user, and cells corresponds to similarity between two users.

```
image(as.matrix(similarity_users), main = "User similarity")
```

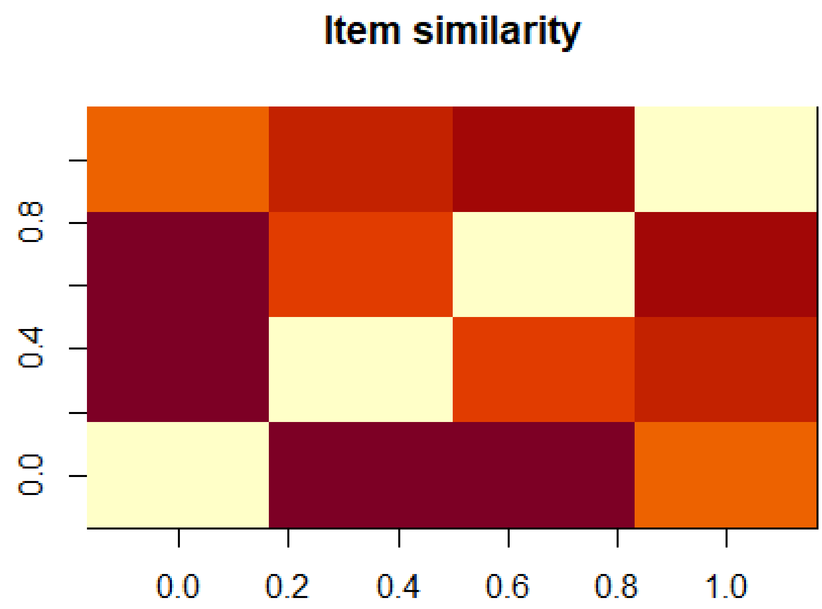


We can compute and visualize the similarity between the first four item.

```
similarity_items <- similarity(Jester5k[, 1:4], method =
                                 "cosine", which = "items")

as.matrix(similarity_items)

##           j1        j2        j3        j4
## j1 0.0000000 0.3839318 0.3916563 0.2357251
## j2 0.3839318 0.0000000 0.2764639 0.3020971
## j3 0.3916563 0.2764639 0.0000000 0.3532990
## j4 0.2357251 0.3020971 0.3532990 0.0000000
```

Now, we can visualize the matrix using this image.

```
image(as.matrix(similarity_items), main = "Item similarity")
```



Exploring the data

Extracting the size of the data, show us 5000 users and 100 jokes.

```
dim(data_to_use)

## [1] 5000  100
```

Exploring values of rating.

```
##   [1]  7.91 -3.20 -1.70 -7.38  0.10  0.83  2.91 -2.77 -3.35 -1.99 -
0.68  9.17
##  [13] -9.71 -3.16  5.58  9.08  0.00 -6.70  1.02 -3.01  5.87 -7.33
7.48  6.55
##  [25]  0.78 -0.10 -6.65  2.28 -8.35  5.53  4.47  7.28 -1.94  3.25
5.63  8.50
##  [37]  8.30  2.82 -7.96  4.27  9.27  8.01  1.70 -7.91  1.65 -8.93
0.87  3.64
##  [49] -3.69  6.50  3.06  2.09 -5.97  2.72  3.01 -9.51  6.41  1.31
5.15 -4.90
##  [61] -7.04 -4.37  7.86  2.04 -2.48 -0.49  0.49  8.11 -1.75  1.50
2.77 -1.02
##  [73]  1.80  0.29 -9.17  5.39  3.69  3.74 -6.89 -2.18  7.23  9.37
```

```
4.81  2.23
##  [85] -6.36 -3.54  6.89 -9.56  2.18 -5.24 -6.31 -9.66 -8.59  1.99
1.84 -4.13
##  [97] -4.17 -2.23  3.83  3.45  1.55 -6.02 -0.05 -0.53 -0.44  5.24
1.75 -7.43
## [109] -7.23 -9.27  7.52  3.98  2.43 -9.03  3.59  1.26  3.54 -7.86
2.67  3.79
## [121]  2.86  4.71 -7.77  6.94  2.62 -1.55  4.32  4.51  4.13 -1.80
6.02  1.21
## [133]  4.37 -4.71  5.97  8.40  7.67  6.07  6.26  3.88  7.43  1.36
2.38 -1.26
## [145]  8.69  2.48  7.57  6.60 -0.39 -9.85 -0.97  2.96  7.04  0.39
1.17 -4.85
## [157]  6.65  4.03 -1.41  1.89 -6.60  0.05  3.11  5.19 -0.19  5.34
6.70  1.46
## [169] -1.12  6.17 -9.81 -4.81  6.84  2.52  1.60 -8.50  0.53 -8.64 -
3.59  3.40
## [181]  9.13  8.20 -7.67 -4.47 -5.10  7.82 -3.50  8.88  6.99  4.42 -
7.57 -6.21
## [193]  0.58  1.41  0.34 -3.64  8.64 -0.87 -1.31 -9.76  7.33 -1.60 -
0.92  4.95
## [205]  3.30  5.05 -9.95 -2.09 -4.61  9.03 -5.78 -8.83  3.16  2.14 -
8.20 -6.17
## [217] -7.82 -6.46  3.93 -0.58  4.90  5.44 -2.82  4.08  3.20  4.66 -
5.05 -4.03
## [229]  8.59  8.79  2.57 -5.00  4.85 -3.45  7.38  5.78 -9.47  5.73
0.73  4.17
## [241] -2.04 -0.78 -2.57 -8.54  0.44 -9.08  7.62  8.74 -3.93 -2.96
5.68 -2.38
## [253] -3.40 -6.07  5.49 -5.39  9.32 -1.46 -3.98 -1.89 -0.29 -6.50
4.56  5.92
## [265] -7.28  1.12 -6.12 -6.75  7.96 -8.88 -2.67  2.33 -6.55 -9.61 -
7.09  5.83
## [277] -3.79 -4.56 -9.90 -7.62 -4.66 -5.68 -5.53 -5.63  6.31  6.36
5.29  6.21
## [289] -5.19 -5.92 -2.52 -4.42  8.16 -5.49 -2.14 -8.98 -5.87 -2.72 -
2.91  8.54
## [301]  7.14 -5.34  8.35  6.80 -6.94  0.24  6.75  8.25  3.35 -3.25 -
1.07 -8.06
## [313]  0.68 -0.15 -0.63 -6.26 -9.13  7.09  5.00 -8.30  0.97  7.72
9.22 -2.28
## [325] -9.32  6.12 -9.42  1.94 -2.43 -6.41  0.63  8.93 -5.58  4.76 -
8.45 -6.84
## [337] -4.32 -0.73 -7.18 -3.11 -0.34 -0.24  0.92 -7.14  0.15 -5.73
3.50 -8.16
## [349] -7.48 -6.80 -1.21 -2.33  8.98 -3.74 -2.62 -8.01  0.19 -6.99 -
3.83 -4.27
## [361] -1.84 -0.83  4.22  1.07 -4.08 -4.51 -2.86 -4.22  8.83 -8.11 -
8.69  6.46
## [373] -1.17 -7.52  8.45 -9.37 -8.79 -8.25  7.77 -1.65 -5.44 -1.50 -
```

```
3.88 -4.76
## [385]  8.06 -1.36  5.10  7.18 -8.40  4.61 -3.06 -5.83 -3.30 -5.29 -
5.15 -4.95
## [397] -9.22 -7.72 -8.74  9.61  9.90  9.81  9.42
```

As you can see a rating equal to 0 represents a missing value, so remove them from vector_ratings:

```
vector_ratings <- vector_ratings[vector_ratings != 0]
```

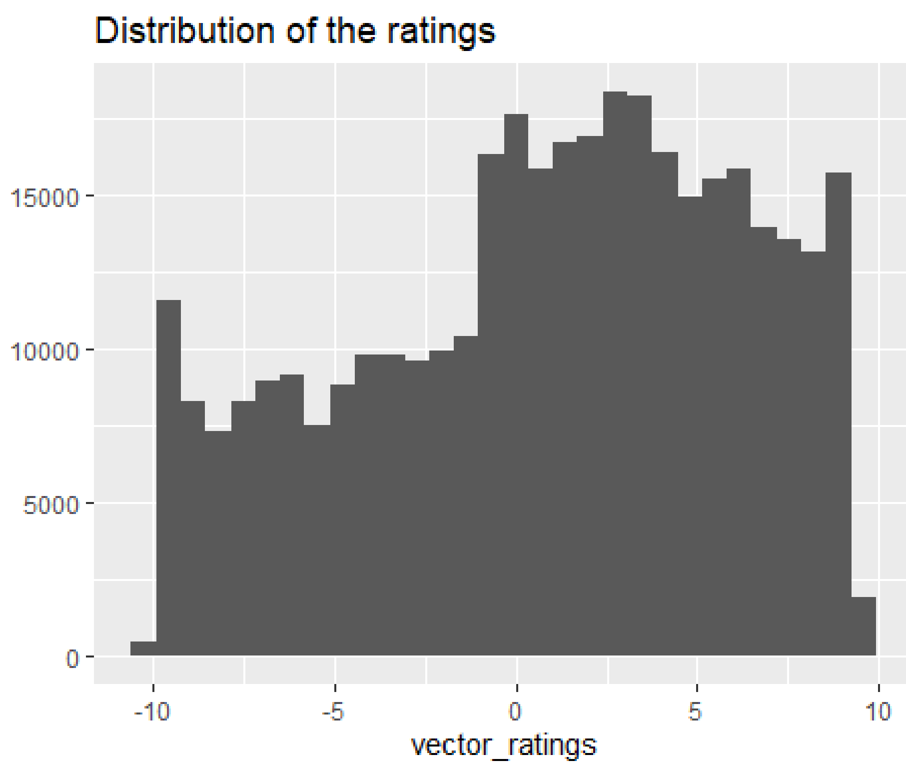
Now, we can plot the ratings. In order to visualize a bar plot Let's convert them into categories using factors and to see chart:

vector_ratings <- factor(vector_ratings)

vector_ratings

Now visualize their distribution with qplot:

```
qplot(vector_ratings) + ggtitle("Distribution of the ratings")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
views_per_joke <- colCounts(data_to_use)

views_per_joke

##    j1    j2    j3    j4    j5    j6    j7    j8    j9   j10   j11   j12   j13   j14
j15   j16
```

```
## 3314 3648 3338 3142 4998 4073 4999 5000 3173 4057 4353 4478 5000 4494
5000 4999
##  j17  j18  j19  j20  j21  j22  j23  j24  j25  j26  j27  j28  j29  j30
j31  j32
## 5000 5000 5000 5000 4983 4282 4025 3194 4172 4764 4981 4781 4992 3616
4937 4993
##  j33  j34  j35  j36  j37  j38  j39  j40  j41  j42  j43  j44  j45  j46
j47  j48
## 3366 4334 4994 4998 3379 4575 4611 4442 3764 4910 3515 3248 4276 4722
4463 4964
##  j49  j50  j51  j52  j53  j54  j55  j56  j57  j58  j59  j60  j61  j62
j63  j64
## 4995 4999 3803 4017 4997 4933 3972 4954 3223 3135 3651 3597 4964 4992
4047 3474
##  j65  j66  j67  j68  j69  j70  j71  j72  j73  j74  j75  j76  j77  j78
j79  j80
## 4951 4989 3532 4989 4987 4066 1706 1740 1699 1726 1751 1745 1758 1750
1799 1760
##  j81  j82  j83  j84  j85  j86  j87  j88  j89  j90  j91  j92  j93  j94
j95  j96
## 1819 1784 1835 1854 1864 1860 1872 1917 1901 1946 1931 1958 2002 2026
2047 2076
##  j97  j98  j99 j100
## 2088 2131 2179 1968
```

Knowing which jokes have been viewed.

```
table_views <- data.frame(
  jokes = names(views_per_joke),
  views = views_per_joke
)
table_views <- table_views[order(table_views$views, decreasing =
                                   TRUE), ]
table_views
```

```
##      jokes views
## j8      j8  5000
## j13    j13  5000
## j15    j15  5000
## j17    j17  5000
## j18    j18  5000
## j19    j19  5000
## j20    j20  5000
## j7      j7  4999
## j16    j16  4999
## j50    j50  4999
## j5      j5  4998
## j36    j36  4998
## j53    j53  4997
## j49    j49  4995
## j35    j35  4994
```

```
## j32    j32  4993
## j29    j29  4992
## j62    j62  4992
## j66    j66  4989
## j68    j68  4989
## j69    j69  4987
## j21    j21  4983
## j27    j27  4981
## j48    j48  4964
## j61    j61  4964
## j56    j56  4954
## j65    j65  4951
## j31    j31  4937
## j54    j54  4933
## j42    j42  4910
## j28    j28  4781
## j26    j26  4764
## j46    j46  4722
## j39    j39  4611
## j38    j38  4575
## j14    j14  4494
## j12    j12  4478
## j47    j47  4463
## j40    j40  4442
## j11    j11  4353
## j34    j34  4334
## j22    j22  4282
## j45    j45  4276
## j25    j25  4172
## j6      j6  4073
## j70    j70  4066
## j10    j10  4057
## j63    j63  4047
## j23    j23  4025
## j52    j52  4017
## j55    j55  3972
## j51    j51  3803
## j41    j41  3764
## j59    j59  3651
## j2      j2  3648
## j30    j30  3616
## j60    j60  3597
## j67    j67  3532
## j43    j43  3515
## j64    j64  3474
## j37    j37  3379
## j33    j33  3366
## j3      j3  3338
## j1      j1  3314
## j44    j44  3248
```

```
## j57    j57  3223
## j24    j24  3194
## j9      j9  3173
## j4      j4  3142
## j58    j58  3135
## j99    j99  2179
## j98    j98  2131
## j97    j97  2088
## j96    j96  2076
## j95    j95  2047
## j94    j94  2026
## j93    j93  2002
## j100  j100  1968
## j92    j92  1958
## j90    j90  1946
## j91    j91  1931
## j88    j88  1917
## j89    j89  1901
## j87    j87  1872
## j85    j85  1864
## j86    j86  1860
## j84    j84  1854
## j83    j83  1835
## j81    j81  1819
## j79    j79  1799
## j82    j82  1784
## j80    j80  1760
## j77    j77  1758
## j75    j75  1751
## j78    j78  1750
## j76    j76  1745
## j72    j72  1740
## j74    j74  1726
## j71    j71  1706
## j73    j73  1699
```

We can classify by number of views.

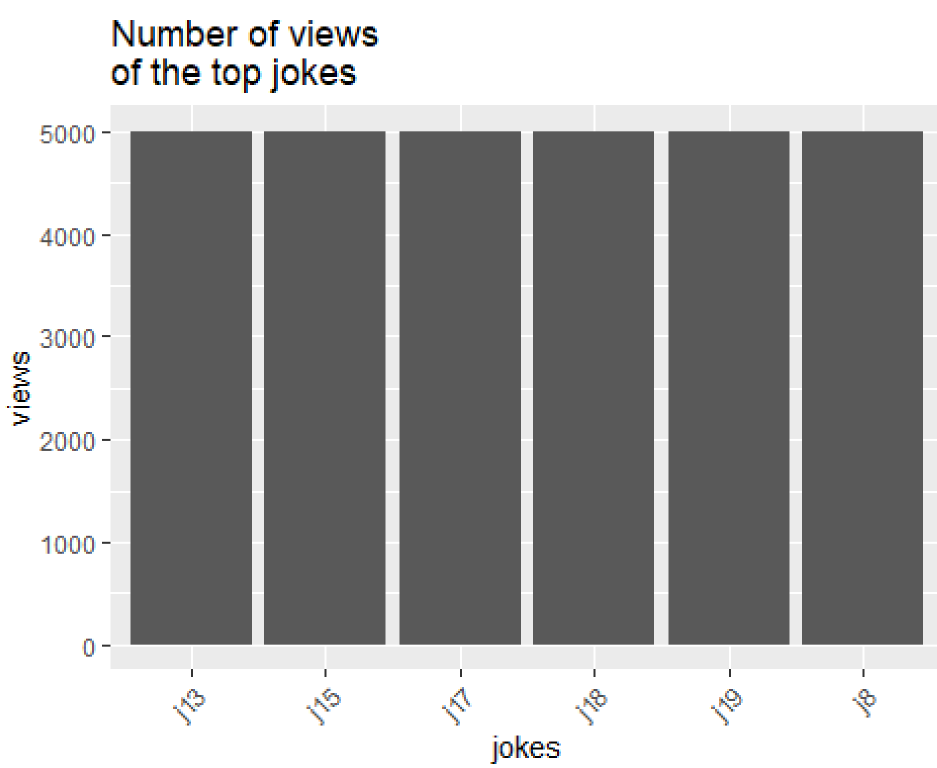
Which are the jokes most viewed?

```
views_per_joke <- colCounts(data_to_use)
views_per_jokeviews_per_joke <- colCounts(data_to_use)
views_per_joke

##   j1   j2   j3   j4   j5   j6   j7   j8   j9  j10  j11  j12  j13  j14 
j15  j16 
## 3314 3648 3338 3142 4998 4073 4999 5000 3173 4057 4353 4478 5000 4494 
5000 4999 
##  j17  j18  j19  j20  j21  j22  j23  j24  j25  j26  j27  j28  j29  j30 
j31  j32 
## 5000 5000 5000 5000 4983 4282 4025 3194 4172 4764 4981 4781 4992 3616
```

```
4937 4993
##   j33   j34   j35   j36   j37   j38   j39   j40   j41   j42   j43   j44   j45   j46
j47  j48
## 3366 4334 4994 4998 3379 4575 4611 4442 3764 4910 3515 3248 4276 4722
4463 4964
##   j49   j50   j51   j52   j53   j54   j55   j56   j57   j58   j59   j60   j61   j62
j63  j64
## 4995 4999 3803 4017 4997 4933 3972 4954 3223 3135 3651 3597 4964 4992
4047 3474
##   j65   j66   j67   j68   j69   j70   j71   j72   j73   j74   j75   j76   j77   j78
j79  j80
## 4951 4989 3532 4989 4987 4066 1706 1740 1699 1726 1751 1745 1758 1750
1799 1760
##   j81   j82   j83   j84   j85   j86   j87   j88   j89   j90   j91   j92   j93   j94
j95  j96
## 1819 1784 1835 1854 1864 1860 1872 1917 1901 1946 1931 1958 2002 2026
2047 2076
##   j97   j98   j99 j100
## 2088 2131 2179 1968
```

Let´s see the first six rows through a histogram.

```
ggplot(table_views[1:6, ], aes(x = jokes, y = views)) +
  geom_bar(stat="identity") + theme(axis.text.x =
                                        element_text(angle = 45, hjust =
1)) + ggtitle("Number of views
of the top jokes")
```



We can visualize the top-rated jokes by computing the average rating of each of them. For this we can use colMeans that automatically ignores the 0s. Now, let´s see the average ratings.

```
average_ratings <- colMeans(data_to_use)
average_ratings

##          j1          j2          j3          j4          j5
j6
##  0.91863005  0.19124726  0.24371480 -1.45172502  0.32256303
1.68529094
##          j7          j8          j9         j10         j11
j12
## -0.54855371 -0.56191200 -0.70693980  1.26791472  1.68047094
1.45719741
##         j13         j14         j15         j16         j17
j18
## -1.75118800  1.51045394 -1.78701400 -3.13562312 -1.03938000 -
0.77061600
##         j19         j20         j21         j22         j23
j24
##  0.12378600 -0.99176200  2.15730283  0.89803129  0.08989068 -
1.70114903
##         j25         j26         j27         j28         j29
j30
##  0.38720278  1.27950252  3.08581409  1.54059611  2.93675681 -
0.45462113
##         j31         j32         j33         j34         j35
j36
##  2.11708730  3.20562187 -1.46514260  0.92079834  3.00082899
3.30338936
##         j37         j38         j39         j40         j41
j42
## -1.45658479  1.22668415  1.06476036  1.01203062 -0.23986982
1.93844399
##         j43         j44         j45         j46         j47
j48
## -0.86555334 -2.28771244  1.03434051  1.40048920  1.54403988
1.82856567
##         j49         j50         j51         j52         j53
j54
##  2.78352152  3.56603921 -0.70165133 -0.04984068  2.98078647
2.74732414
##         j55         j56         j57         j58         j59
j60
##  0.40220292  1.76746468 -2.17755507 -3.89709091 -0.56381539 -
0.41973867
##         j61         j62         j63         j64         j65
j66
##  2.42350725  2.87906851  0.27151717 -0.65654001  2.27239144
```

```
2.47685709
##         j67         j68         j69         j70         j71
j72
## -0.91405719  2.58261175  2.53107880  0.43737088 -0.65388042
2.91218966
##         j73         j74         j75         j76         j77
j78
##  1.12183637 -1.51619351 -0.24462022  2.43083668  0.65608077
1.70238857
##         j79         j80         j81         j82         j83
j84
##  0.02400222  1.00899432  1.79955470  1.01404709  2.04449591
0.71366235
##         j85         j86         j87         j88         j89
j90
##  0.94548820  0.24382796  1.79840278  2.06774126  3.51381378
0.42097636
##         j91         j92         j93         j94         j95
j96
##  1.99332988  1.23293156  2.45494505  0.93815400  0.88979482
1.39264933
##         j97         j98         j99        j100
##  1.55030172  0.78936180 -0.13144562  1.25246443
```

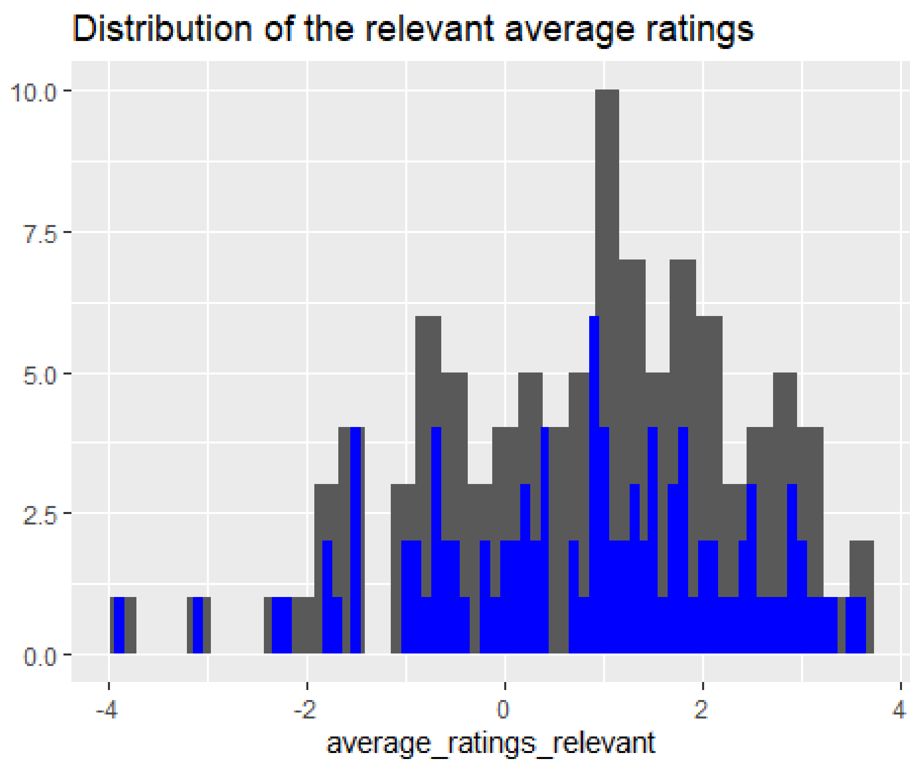
As we see the highest value is 3, and there are a few movies whose rated 1 or 5 Maybe, these jokes received were rated from a few people, so we don´t take them into account. We can remove the jokes whose number of views is below 100.

```
average_ratings_relevant <- average_ratings [views_per_joke > 100]

average_ratings_relevant

##          j1          j2          j3          j4          j5
j6
##  0.91863005  0.19124726  0.24371480 -1.45172502  0.32256303
1.68529094
##          j7          j8          j9         j10         j11
j12
## -0.54855371 -0.56191200 -0.70693980  1.26791472  1.68047094
1.45719741
##         j13         j14         j15         j16         j17
j18
## -1.75118800  1.51045394 -1.78701400 -3.13562312 -1.03938000 -
0.77061600
##         j19         j20         j21         j22         j23
j24
##  0.12378600 -0.99176200  2.15730283  0.89803129  0.08989068 -
1.70114903
##         j25         j26         j27         j28         j29
j30
```

```
##  0.38720278  1.27950252  3.08581409  1.54059611  2.93675681 -
0.45462113
##         j31         j32         j33         j34         j35
j36
##  2.11708730  3.20562187 -1.46514260  0.92079834  3.00082899
3.30338936
##         j37         j38         j39         j40         j41
j42
## -1.45658479  1.22668415  1.06476036  1.01203062 -0.23986982
1.93844399
##         j43         j44         j45         j46         j47
j48
## -0.86555334 -2.28771244  1.03434051  1.40048920  1.54403988
1.82856567
##         j49         j50         j51         j52         j53
j54
##  2.78352152  3.56603921 -0.70165133 -0.04984068  2.98078647
2.74732414
##         j55         j56         j57         j58         j59
j60
##  0.40220292  1.76746468 -2.17755507 -3.89709091 -0.56381539 -
0.41973867
##         j61         j62         j63         j64         j65
j66
##  2.42350725  2.87906851  0.27151717 -0.65654001  2.27239144
2.47685709
##         j67         j68         j69         j70         j71
j72
## -0.91405719  2.58261175  2.53107880  0.43737088 -0.65388042
2.91218966
##         j73         j74         j75         j76         j77
j78
##  1.12183637 -1.51619351 -0.24462022  2.43083668  0.65608077
1.70238857
##         j79         j80         j81         j82         j83
j84
##  0.02400222  1.00899432  1.79955470  1.01404709  2.04449591
0.71366235
##         j85         j86         j87         j88         j89
j90
##  0.94548820  0.24382796  1.79840278  2.06774126  3.51381378
0.42097636
##         j91         j92         j93         j94         j95
j96
##  1.99332988  1.23293156  2.45494505  0.93815400  0.88979482
1.39264933
##         j97         j98         j99        j100
##  1.55030172  0.78936180 -0.13144562  1.25246443
```

Now, let´s visualize it.

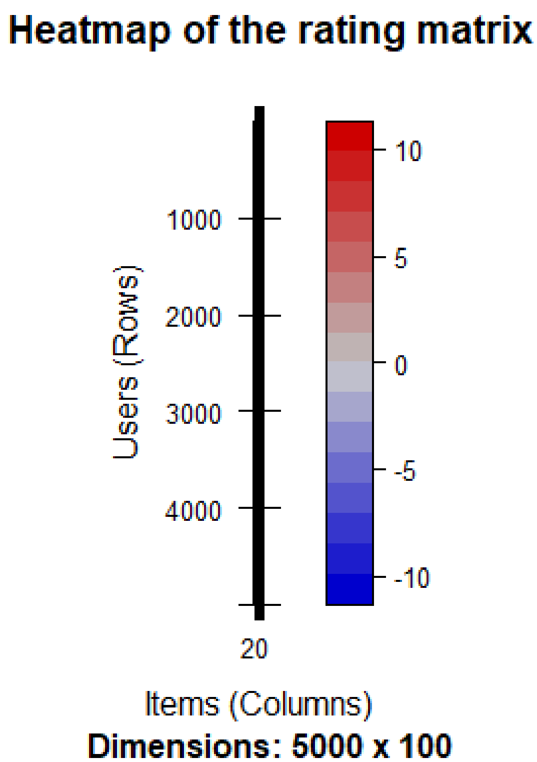
```
qplot(average_ratings_relevant) + stat_bin(fill ="blue", binwidth = 0.1)
+
  ggtitle(paste("Distribution of the relevant average ratings"))

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



It is possible visualize the matrix through a heat map using colors that represent the ratings. Rows correspond to a user, columns to a joke, and cells to its rating..

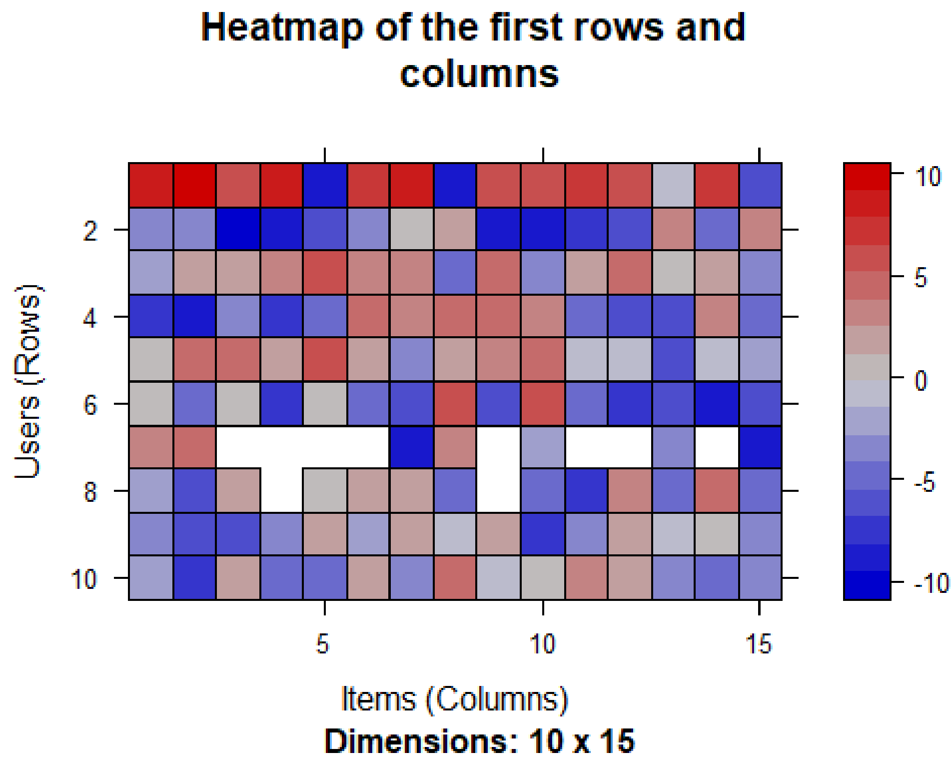
Let´s do visualize the matrix.

```
image(data_to_use, main = "Heatmap of the rating matrix")
```



Due to there are too many users and items,We can build another chart just showings rows and columns.

```
image(data_to_use[1:10, 1:15], main = "Heatmap of the first rows and columns")
```



Data preparation

Let´s see how to prepare the data to be used in the recommending system.models. For doing this, we have to select the most important data and normalized it.Exploring the data we find that jokes have been seen and rated few times.Let´s determine the number of users per joke.Now, we define ratings_jokes that are contained the matrix that we will use.

```
ratings_jokes <- data_to_use[rowCounts(data_to_use) > 50,
                             colCounts(data_to_use) > 100]
ratings_jokes

## 3875 x 100 rating matrix of class 'realRatingMatrix' with 314302
ratings.
```

Let´s explore the most important data.

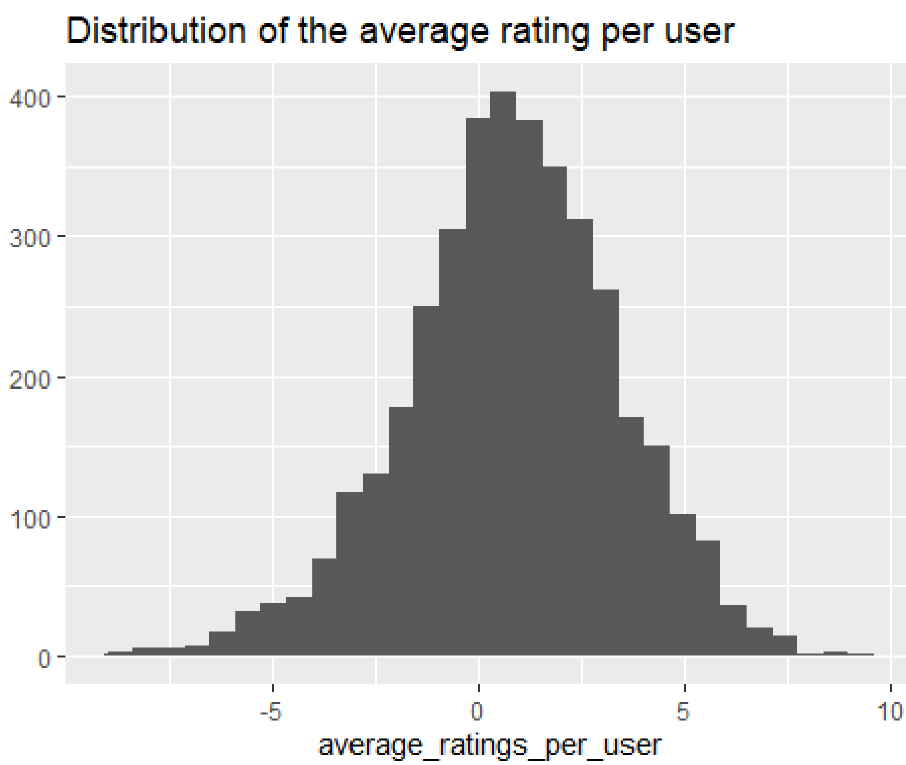
let's visualize the top 2 percent of users and jokes in a new matrix:

```
min_jokes <- quantile(rowCounts(ratings_jokes), 0.98)
min_jokes
```

```
## 98%
## 100

min_users <- quantile(colCounts(ratings_jokes), 0.98)
min_users

##  98%
## 3875
```

Now, let´s see the distribution.

```
average_ratings_per_user <- rowMeans(ratings_jokes)
qplot(average_ratings_per_user) + stat_bin(binwidth = 0.1) +
  ggtitle("Distribution of the average rating per user")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



As you can see the last plot shows that the average rating varies between different users.

Now let´s normalize the data.Taking into account that some users have given high (or low) ratings to all their jokes could change the results. Let´s remove this effect to normalize the average rating of each user being 0.

```
ratings_jokes_norm <- normalize(ratings_jokes)
ratings_jokes_norm
```

```
## 3875 x 100 rating matrix of class 'realRatingMatrix' with 314302
ratings.
## Normalized using center on rows.
```

Let's see now the average rating done by users:

```
sum(rowMeans(ratings_jokes_norm) > 0.00001)

## [1] 0
```

Let´s see a heatmap with this info.

Models

Model I—Item-based collaborative filtering model

Collaborative filtering takes into account of the information about different users. It refers to the fact that users collaborate with each other to recommend items. A mean element is a rating matrix in which rows belongs to users and columns to items.The men algorithm is based on: measure how similar they are in terms of having received similar ratings by similar users in two items, identify the k-most similar items in each one and the items that are most similar to the user's preferences

Defining the training and test sets.

We will be using a part of the data_to_us dataset (the training set) and apply it on the other part (the test set).With this We will recommend jokes to the users in the test set. These sets are defined in this way: Training sets: include users from which the model learn. Test sets : include users to whom we recommends jokes. We will set the training set in 80 percent and 20 percent on the test set as this:

```
data_train <- sample(x = c(TRUE, FALSE), size = nrow(ratings_jokes),
                      replace = TRUE, prob = c(0.8, 0.2))
head(data_train)

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

Let's define the training and the test sets:

```
r_data_train <- ratings_jokes[data_train, ]

r_data_test <- ratings_jokes[!data_train, ]
```

AS we want to recommend items to each user, we will use the k-fold for doing this we have to split the users randomly into 5 groups, use a group as a test set and the other groups as training sets Repeat it in each group.

```
get_set <- sample(x = 1:5, size = nrow(ratings_jokes), replace =
                      TRUE)
for(i_model in 1:5) {
  get_train <- get_set == i_model
  r_data_train <- ratings_jokes[data_train, ]
  r_data_test <- ratings_jokes[!data_train, ]
}
```

Building “IBCF” recommendation model.

Let´s take into account the next info.The model is item-based collaborative filtering (IBCF).

```
recom_models <- recommenderRegistry$get_entries(dataType =

"realRatingMatrix")

recom_models$IBCF_realRatingMatrix$parameters

## $k
## [1] 30
##
## $method
## [1] "Cosine"
##
## $normalize
## [1] "center"
##
## $normalize_sim_matrix
## [1] FALSE
##
## $alpha
## [1] 0.5
##
## $na_as_zero
## [1] FALSE
```

In order to show how to change parameters, we set k = 30, which is the default, this computes the similarities among each pair of items.

```
r_model <- Recommender(data = r_data_train, method = "IBCF",
                          parameter = list(k = 30))

r_model
```

```
## Recommender of type 'IBCF' for 'realRatingMatrix'
## learned using 3070 users.

class(r_model)

## [1] "Recommender"
## attr(,"package")
## [1] "recommenderlab"
```

Now, let´s the show the recommendation model.

We will use getModel to extract some details such as its description and parameters:

```
model_details <- getModel(r_model)
model_details$description

## [1] "IBCF: Reduced similarity matrix"
```

We will use The model_details$sim matrix component to find similarities.

Let´s see what this show us.

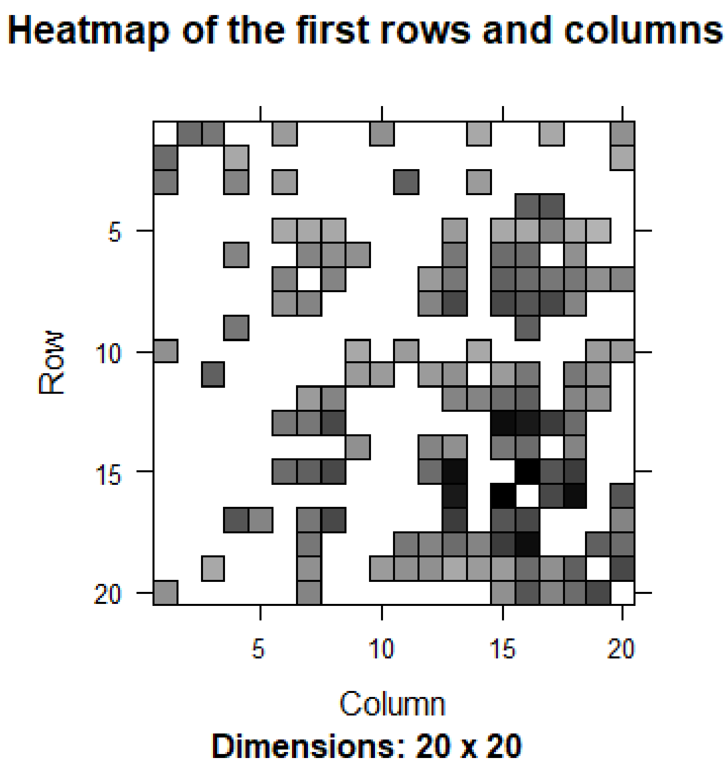
```
class(model_details$sim)

## [1] "dgCMatrix"
## attr(,"package")
## [1] "Matrix"

dim(model_details$sim)

## [1] 100 100
```

As you can see, model_details$sim is a square matrix whose size is equal to the number of items. We can explore a part of it using image:

```
n_items_top <- 20
```

Now let's see what the heat map shows us:

```
image(model_details$sim[1:n_items_top, 1:n_items_top],
      main = "Heatmap of the first rows and columns")
```



If we check the heatmap most of the values are equal to 0. The reason is that each row contains only k elements.

```
model_details$k

## [1] 30

row_sums <- rowSums(model_details$sim > 0)
table(row_sums)

## row_sums
##  30
## 100
```

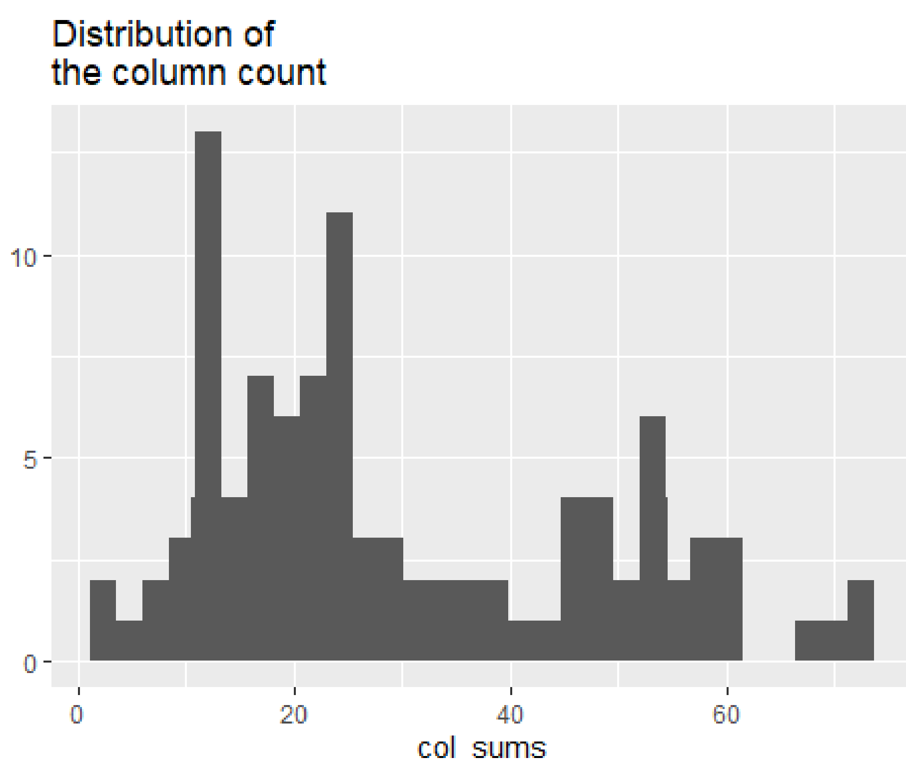
Now, let's check the distribution of elements by column.

```
col_sums <- colSums(model_details$sim > 0)

col_sums

##   j1   j2   j3   j4   j5   j6   j7   j8   j9  j10  j11  j12  j13  j14
j15  j16
##   12   15    9   53   11   16   22   23   46   15   12   23   33   20
45   72
##  j17  j18  j19  j20  j21  j22  j23  j24  j25  j26  j27  j28  j29  j30
j31  j32
##   25   31   11   22   48   16   12   60    9   28   54   13   49   25
42   59
```

```
##   j33  j34  j35  j36  j37  j38  j39  j40  j41  j42  j43  j44  j45  j46 
j47  j48 
##    47   22   52   59   57   13   19    5   12   26   43   71    3   11 
18    8 
##   j49  j50  j51  j52  j53  j54  j55  j56  j57  j58  j59  j60  j61  j62 
j63  j64 
##    54   56   34   23   61   55   11   17   73   68   23   37   54   50 
13   38 
##   j65  j66  j67  j68  j69  j70  j71  j72  j73  j74  j75  j76  j77  j78 
j79  j80 
##    37   49   39   47   60    3   20   49   27   50   21   29   23   28 
22   22 
##   j81  j82  j83  j84  j85  j86  j87  j88  j89  j90  j91  j92  j93  j94 
j95  j96 
##     8   17   19   20   14   31   12   24   54   25   17   15   26   24 
10   18 
##   j97  j98  j99 j100 
##    19   12   24   21
```

Now , let´s write the code to build the distribution chart:

```
qplot(col_sums) + stat_bin(binwidth = 1) + ggtitle("Distribution of 
the column count")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Watching the chart there are a few jokes that are similar to many others. Let’s see which are the jokes with the most elements:

```
which_max <- order(col_sums, decreasing = TRUE)[1:6]
rownames(model_details$sim)[which_max]

## [1] "j57" "j16" "j44" "j58" "j53" "j24"
```

Now, we apply the recommending model on the test set.We are on the capacity to recommend jokes to the users in the test set. So define n_recommended to specify the number of items to recommend to each user.

```
n_recommended <- 10
```

The above algorithm identifies the top n recommendations

```
r_predicted <- predict(object = r_model, newdata = r_data_test, n =
n_recommended)
r_predicted

## Recommendations as 'topNList' with n = 10 for 805 users.
```

To ilustrate I say that, the r_predicted object contains the recommendations, you can check it with this piece ofcode.

```
class(r_predicted)

## [1] "topNList"
## attr(,"package")
## [1] "recommenderlab"
```

Model II—User-based collaborative filtering

In this model we will use a new user to identify its similar users. Then, we will recommend the top-rated items. Things to have into account in this module: –Measure similarities of each user are to the new one. They are correlation and cosine.To identify the most similar we use, -(k-nearest_neighbors)and the similarity that is above a defined threshold –Apply average and Weighted average rating, using the similarities as weights. –In this model we will build a training and a test set, too.

Building "UBCF" recommendation model

To start, we present this piece of code.

```
r_models <- recommenderRegistry$get_entries(dataType =

"realRatingMatrix")

r_models$UBCF_realRatingMatrix$parameters

## $method
## [1] "cosine"
##
```

```
## $nn
## [1] 25
##
## $sample
## [1] FALSE
##
## $weighted
## [1] TRUE
##
## $normalize
## [1] "center"
##
## $min_matching_items
## [1] 0
##
## $min_predictive_items
## [1] 0
```

Parameters to have into account— method: It computes the similarity between users— nn: It´s the number of similar users

Now Let's build the recommending model.

```
r_model <- Recommender(data = r_data_train, method = "UBCF")

r_model

## Recommender of type 'UBCF' for 'realRatingMatrix'
## learned using 3070 users.
```

Now, check details of the model using getModel:

```
model_details <- getModel(r_model)
```

Now, let´s see the module components.

```
names(model_details)

##  [1] "description"          "data"                 "method"
##  [4] "nn"                   "sample"               "weighted"
##  [7] "normalize"            "min_matching_items"
"min_predictive_items"
## [10] "verbose"
```

model_details contains a dataslot too.

```
model_details$data

## 3070 x 100 rating matrix of class 'realRatingMatrix' with 249033
ratings.
## Normalized using center on rows.
```

Results

Evaluating the models

To recommend items to new users, collaborative systems estimates the ratings that are not yet seen, then, it recommends the top-rated. Now, let´s evaluate the model by comparing the estimated ratings with real users.

Data preparation for validation using k-fold

```
n_fold <- 6

items_to_keep <- 12

rating_threshold <- 3

eval_sets <- evaluationScheme(data = ratings_jokes, method = "cross-
validation",
                              k = n_fold, given = items_to_keep,
goodRating = rating_threshold)
```

Now, let´s define the model to evaluate and list parameters.

```
model_to_evaluate <- "IBCF"

model_parameters <- NULL
```

Now, let´s construct the model using the next chunk of code.

```
eval_recommender <- Recommender(data = getData(eval_sets, "train"),
                                method = model_to_evaluate, parameter =
model_parameters)
```

Now, we specify the number of items to recommend.

```
items_to_recommend <- 10
```

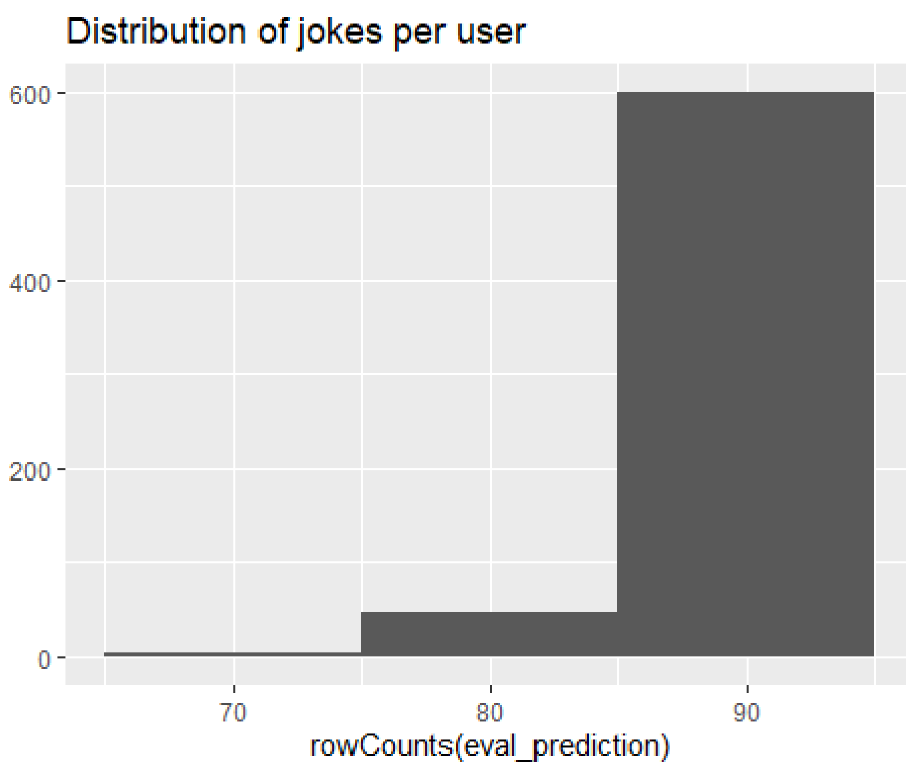
Now, let´s make the matrix using the predict function.

```
eval_prediction <- predict(object = eval_recommender, newdata =
                           getData(eval_sets, "known"), n =
items_to_recommend, type = "ratings")

class(eval_prediction)
```

```
## [1] "realRatingMatrix"
## attr(,"package")
## [1] "recommenderlab"
```

Now, let´s see the number of jokes to recommend to each user, visualizing them.

```
qplot(rowCounts(eval_prediction)) + geom_histogram(binwidth = 10) +

  ggtitle("Distribution of jokes per user")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

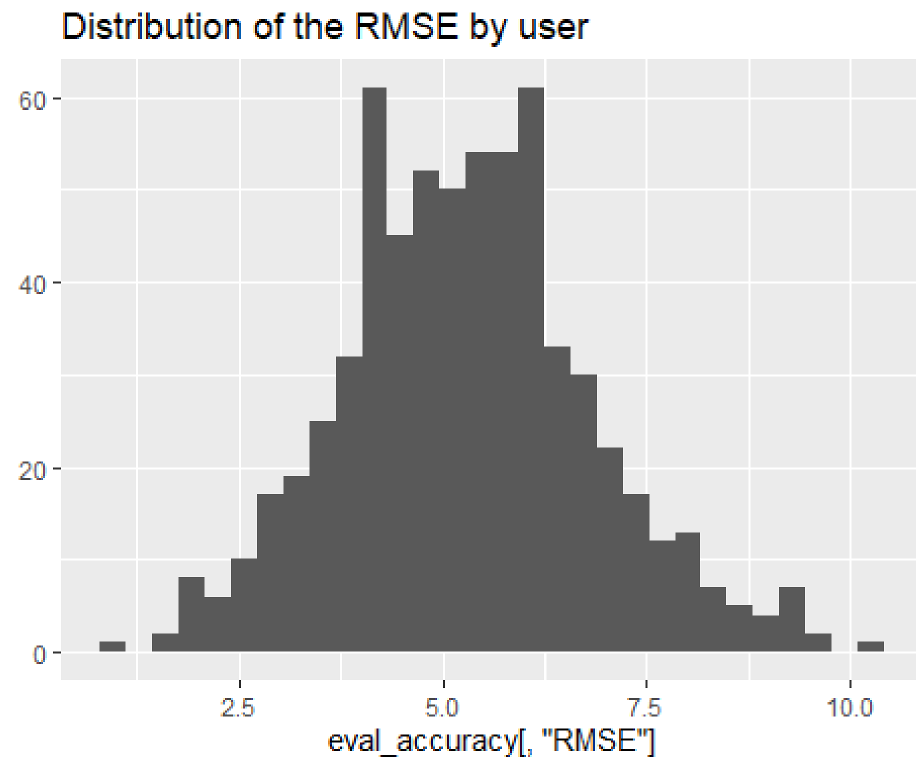


Now, let´s measure the accuracy and compute(RMSE,MSE and MAE)

```
eval_accuracy <- calcPredictionAccuracy(
  x = eval_prediction, data = getData(eval_sets, "unknown"), byUser =
    TRUE)

head(eval_accuracy)

##             RMSE       MSE      MAE
## u15547 6.330312 40.072847 5.166835
## u999   3.161013  9.992004 2.686781
## u4519  5.083688 25.843885 4.358402
## u18953 6.177501 38.161524 5.137254
## u16123 4.233197 17.919958 3.659190
## u17883 7.516146 56.492451 6.913729
```

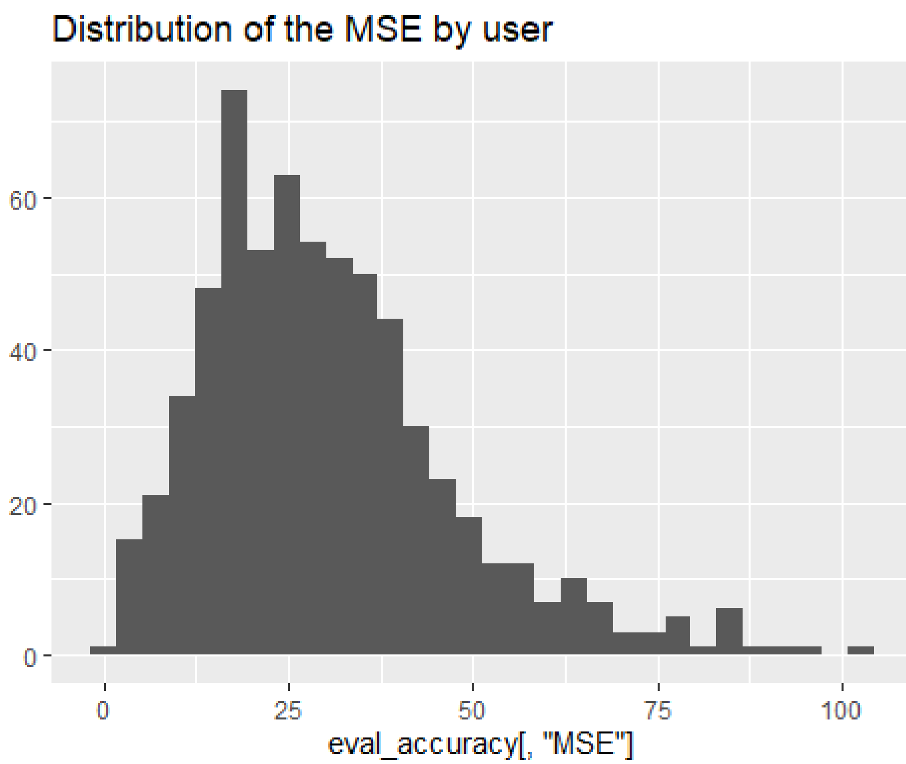
Now, let's check the RMSE by a user.

```
qplot(eval_accuracy[, "RMSE"]) + geom_histogram(binwidth = 0.1) +
  ggtitle("Distribution of the RMSE by user")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



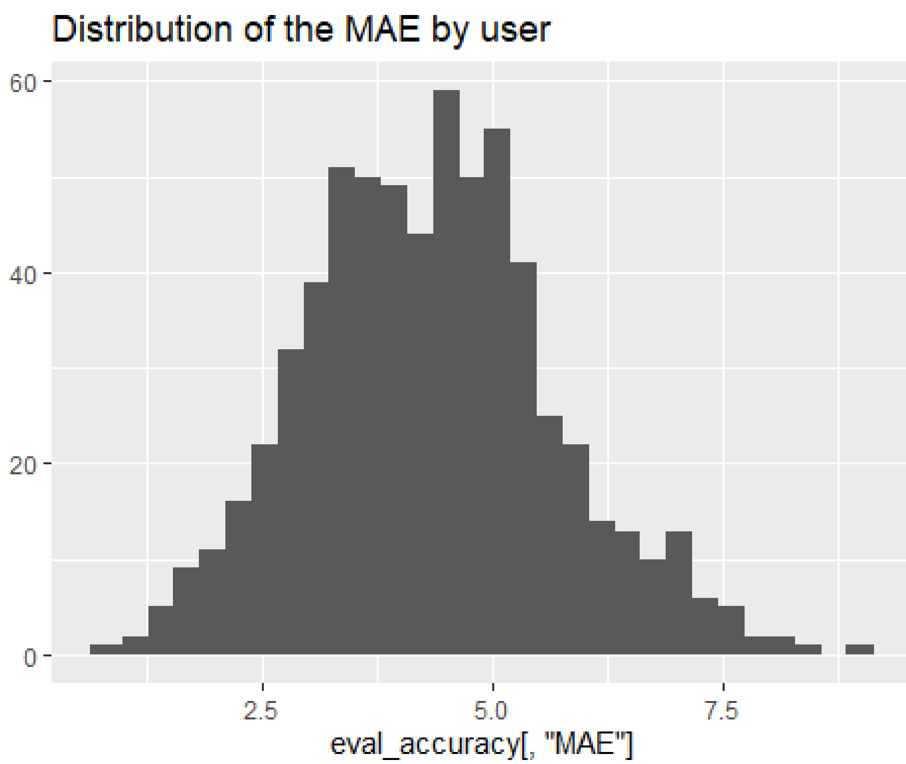
Now, let's check the MSE by a user.

```
qplot(eval_accuracy[, "MSE"]) + geom_histogram(binwidth = 0.1) +
  ggtitle("Distribution of the MSE by user")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Now, let's check the MAE by a user.

```
qplot(eval_accuracy[, "MAE"]) + geom_histogram(binwidth = 0.1) +
  ggtitle("Distribution of the MAE by user")

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Having a performance index in the whole model.

Most of the RMSEs are in the range of 0.8 to 1.4. The model was evaluated in each user. We use this code.

```
eval_accuracy <- calcPredictionAccuracy(x = eval_prediction, data =
getData(eval_sets, "unknown"), byUser =
                                                      FALSE)
eval_accuracy

##       RMSE       MSE       MAE
##   5.530651 30.588097  4.356794
```

Discussing the models performance.

With these measures we can compare the performance of different models with the same data, as shown using qplot, geom_histogram and ggtitle.

Conclusion

This project dealing to Item and users-based collaborative filtering, would help to know the importance of recommendation systems in our everyday life and business. It leave us as a big content of learning that I am sure will impact when applying to other kind of data becoming in a powerful tool to recommend items.

The IBCF and the UBCF Collaborative filtering have some limitations When dealing with new users and/or new items.

Taking into account the limitations that have the IBCF and the UBCF Collaborative filtering models leave us the necessity to explore new models to apply in a future work and get the 100% of effectiveness to achieve better results.