Joke Recommender System

by Igor Baranov, Michael Parravani, Ariana Biagi, Hui Fang Cai

Abstract The goal of this project is to build a joke recommending application. The success of this project will be determined by the accuracy of which the system reccomends "good" jokes (RMSE and TPR). This model could be used by TV/Movie writers to tune the humor of a movie to their target demographic.

Introduction and Discussion

Since funniness is a very subjective matter, it will be very interesting to see if data science can bring out the details on what makes something funny. Having more than 50,000 users rate jokes, can an algorithm be written to identify the universal funny jokes? The purpose of the study of the cool Jester Online Joke Recommender dataset is to provide people information on recommendations of funny jokes.

In this joke recommendation model, the dataset includes over 7.6 million ratings recorded chronologically from the period between from November 2006 to May 2009. The first 90% are provided here as the training dataset and the remaining 10% as the test dataset.We use the collabrative filtering methods (used-based or item-based CF) on the rating matrix to study the algorithms between the user and the rating, and between the joke and the rating, thus is used to predict what jokes are funniest for recommendation for new users. We use RMSE/MSE/MAE to measure the accuracy of our recommendation model, and thus the most reasonable model for this study will be determined. In addition, we perform the sentiment analysis to verify if there are impacts on the rating by the sentiment score.This model could be applied to Media Industry such as TV or Moive to tune the humor to target audiences or used for the reference of humor research.

Dataset

The recommender will be trained useing a dataset of joke ratingsfrom ~73k users from http://eigentaste.berkeley.edu/dataset/. This dataset was developed to help study social information filtering. This dataset does not require much feature engineering beyond filtering out as much sparse data as is reasonable. The final dataset will be a normalized real rating matrix with a selection of jokes and users who have a sufficient number of ratings. One dataset is for the list jokes that the users rate. The another dataset is a collection of user's ratings.

- Dataset 1: jester_items.tsv contains Joke id and Joke text, including 150 jokes.
- Dataset 2: jesterfinal151cols.csv: Each row is a user, and each column is a joke. Rating are given as real vales from Ratings are given as real values from -10.00 to +10.00. The value "99"" corresponds to a null rating. The size of this dataset is 50,691 rows x 151 columns (i.e. 50,691 users).

There are many empty values and lots of null rating values presented as "99" in the jesterfinal151cols.csv rating dataset. In this study, we assume all empty values are the null rating values (NA) that are same as the null rating values presented in the dataset. There are no outliers, all the rating are ranged between -10.0 to +10.0 as expected. This dataset does not require much feature engineering beyond filtering out as much sparse data as is reasonable. The final dataset can be a normalized real rating matrix with a selection of jokes and users who have a sufficient number of ratings.

Ethical ML Framework

As the goal of this application is only to reccomend jokes, many aspects of the ethical ML framework do not directly apply. The data is open source and we can assume was collected in transparent ways. That being said, there is likely a large segment of the populace that is under represented in this ratings dataset - we assume a low income population (limited access to internet, limited time to be spent rating jokes, etc). This will potentially reduce the reccomender's accuracy for that group of the population. If the outcome of this system were to be of more social impact, this would need to be corrected with appropriate data collection methods.

Assumptions

We assume that the users are a wide distribution of individuals from various backgrounds and geographic locations across the United States.

Data preparation

Set directory to current script (works in R Studio only)

```
#this.dir <- dirname(rstudioapi::getActiveDocumentContext()$path)
#setwd(this.dir)
```

Load the jokes rating data and do the basic checks

```
df <- read_csv(file="../../../data/jesterfinal151cols.csv.zip",col_names = FALSE)
dim(df)
```

```
#> [1] 50692   151
```

Rename columns as per their "Joke ID" and columns to the userid

```
names(df)     <- c("RATINGS",1:(ncol(df)-1))
row.names(df) <- c(1:nrow(df))
```

Replace all 99 ratings with NA so we can filter them out

```
df[,2:ncol(df)][df[,2:ncol(df)]==99]<-NA
df
```

```
#> # A tibble: 50,692 x 151
#>    RATINGS   `1`   `2`   `3`   `4`     `5`   `6`    `7`     `8`   `9`
#>  *   <int> <int> <int> <int> <int>   <dbl> <int>  <dbl>   <dbl> <int>
#>  1      62    NA    NA    NA    NA  0.219     NA -9.28  -9.28      NA
#>  2      34    NA    NA    NA    NA -9.69      NA  9.94   9.53      NA
#>  3      18    NA    NA    NA    NA -9.84      NA -9.84  -7.22      NA
#>  4      82    NA    NA    NA    NA  6.91      NA  4.75  -5.91      NA
#>  5      27    NA    NA    NA    NA -0.0312    NA -9.09  -0.406     NA
#>  6      46    NA    NA    NA    NA -2.91      NA -2.34  -0.5       NA
#>  7      99    NA    NA    NA    NA  6.22      NA -7.44  -0.812     NA
#>  8      15    NA    NA    NA    NA  8.25      NA  9      8.88      NA
#>  9     104    NA    NA    NA    NA -5.75      NA  0.281  0.781     NA
#> 10      24    NA    NA    NA    NA -7.16      NA -5.91  -0.0938    NA
#> # ... with 50,682 more rows, and 141 more variables: `10` <int>,
#> #   `11` <int>, `12` <int>, `13` <dbl>, `14` <int>, `15` <dbl>,
#> #   `16` <dbl>, `17` <dbl>, `18` <dbl>, `19` <dbl>, `20` <dbl>,
#> #   `21` <dbl>, `22` <dbl>, `23` <dbl>, `24` <dbl>, `25` <dbl>,
#> #   `26` <dbl>, `27` <dbl>, `28` <dbl>, `29` <dbl>, `30` <dbl>,
#> #   `31` <dbl>, `32` <dbl>, `33` <dbl>, `34` <dbl>, `35` <dbl>,
#> #   `36` <dbl>, `37` <dbl>, `38` <dbl>, `39` <dbl>, `40` <dbl>,
#> #   `41` <dbl>, `42` <dbl>, `43` <dbl>, `44` <dbl>, `45` <dbl>,
#> #   `46` <dbl>, `47` <dbl>, `48` <dbl>, `49` <dbl>, `50` <dbl>,
#> #   `51` <dbl>, `52` <dbl>, `53` <dbl>, `54` <dbl>, `55` <dbl>,
#> #   `56` <dbl>, `57` <dbl>, `58` <dbl>, `59` <dbl>, `60` <dbl>,
#> #   `61` <dbl>, `62` <dbl>, `63` <dbl>, `64` <dbl>, `65` <dbl>,
#> #   `66` <dbl>, `67` <dbl>, `68` <dbl>, `69` <dbl>, `70` <dbl>,
#> #   `71` <dbl>, `72` <dbl>, `73` <dbl>, `74` <dbl>, `75` <dbl>,
#> #   `76` <dbl>, `77` <dbl>, `78` <dbl>, `79` <dbl>, `80` <dbl>,
#> #   `81` <dbl>, `82` <dbl>, `83` <dbl>, `84` <dbl>, `85` <dbl>,
#> #   `86` <dbl>, `87` <dbl>, `88` <dbl>, `89` <dbl>, `90` <dbl>,
#> #   `91` <dbl>, `92` <dbl>, `93` <dbl>, `94` <dbl>, `95` <dbl>,
#> #   `96` <dbl>, `97` <dbl>, `98` <dbl>, `99` <dbl>, `100` <dbl>,
#> #   `101` <dbl>, `102` <dbl>, `103` <dbl>, `104` <dbl>, `105` <dbl>,
#> #   `106` <dbl>, `107` <dbl>, `108` <dbl>, `109` <dbl>, ...
```

Load the jokes text

```
jokes <- read_tsv(file="../../../data/jester_items.tsv",col_names = FALSE)
jokes <- jokes['X2']
head(jokes)
```

```
#> # A tibble: 6 x 1
#>   X2
#>   <chr>
#> 1 "A man visits the doctor. The doctor says, \"I have bad news for you. Y~
#> 2 "This couple had an excellent relationship going until one day he came ~
#> 3 Q. What's 200 feet long and has 4 teeth? A. The front row at a Willie N~
#> 4 Q. What's the difference between a man and a toilet? A. A toilet doesn'~
#> 5 Q. What's O. J. Simpson's web address? A. Slash, slash, backslash, slas~
#> 6 "Bill and Hillary Clinton are on a trip back to Arkansas. They're almos~
```

What's the distribution of rating count?

Anticipate that many jokes probably have low number of ratings. The fact that the median joke rating quantity is ~18% of users having rated it is quite low. The jokes with extremely low numbers of ratings are likely new to the system and have not been rated much (Fig. 1). Following that, there is a exponential decrease in rating count to jokes where about half of the users have rated them. The jokes where all the users rated them are "control" jokes, but it is interesting that there are no jokes between ~25k and 50k ratings.

```
ratingcount<-nrow(df)-colSums(is.na(df[,2:ncol(df)]))
summary(ratingcount)
```

```
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>       0    5340    9225   11526   14564   50692
```

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

How many users have given a reasonable amount of ratings?

There is an exponential drop in the number of ratings given by users (Fig. 2). The group with ~140 ratings is potentially a control group. We will have to establish a cut off for the data that excludes users with a low number of ratings.

```
ratingcount_u<-as.vector(df[[1]])
summary(ratingcount_u)
```

```
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>     8.0    11.0    20.0    34.1    42.0   140.0
```

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

Does sentiment score have an impact on the rating of a joke?

Initialize the Jokes dataframe to incorporate sentiment score and avg rating:

```
jokes[,2]<-1
jokes[,3]<-1
colnames(jokes)<-c("Joke","Sentiment","AvgRating")
head(jokes)
```

```
#> # A tibble: 6 x 3
#>   Joke                                                Sentiment AvgRating
#>   <chr>                                                   <dbl>     <dbl>
#> 1 "A man visits the doctor. The doctor says, \"I have~         1         1
#> 2 "This couple had an excellent relationship going un~         1         1
#> 3 Q. What's 200 feet long and has 4 teeth? A. The fro~         1         1
#> 4 Q. What's the difference between a man and a toilet~         1         1
#> 5 Q. What's O. J. Simpson's web address? A. Slash, sl~         1         1
#> 6 "Bill and Hillary Clinton are on a trip back to Ark~         1         1
```

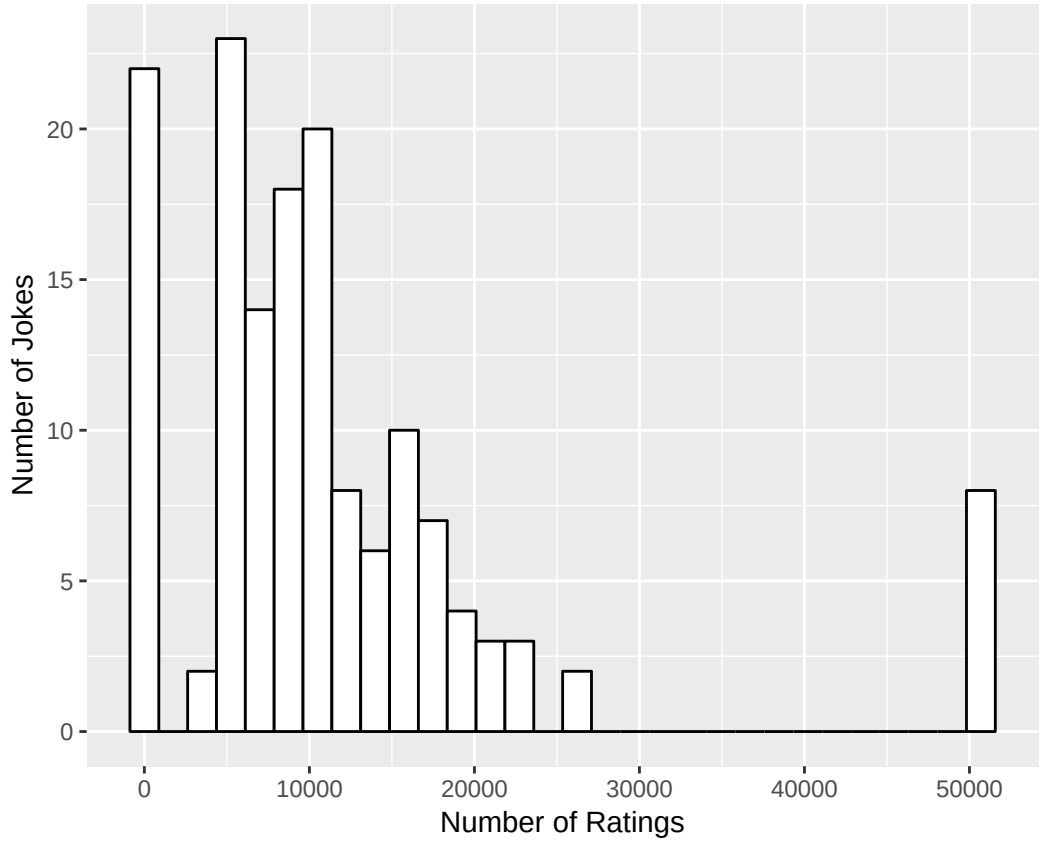

Figure 1: Joke rating distribution

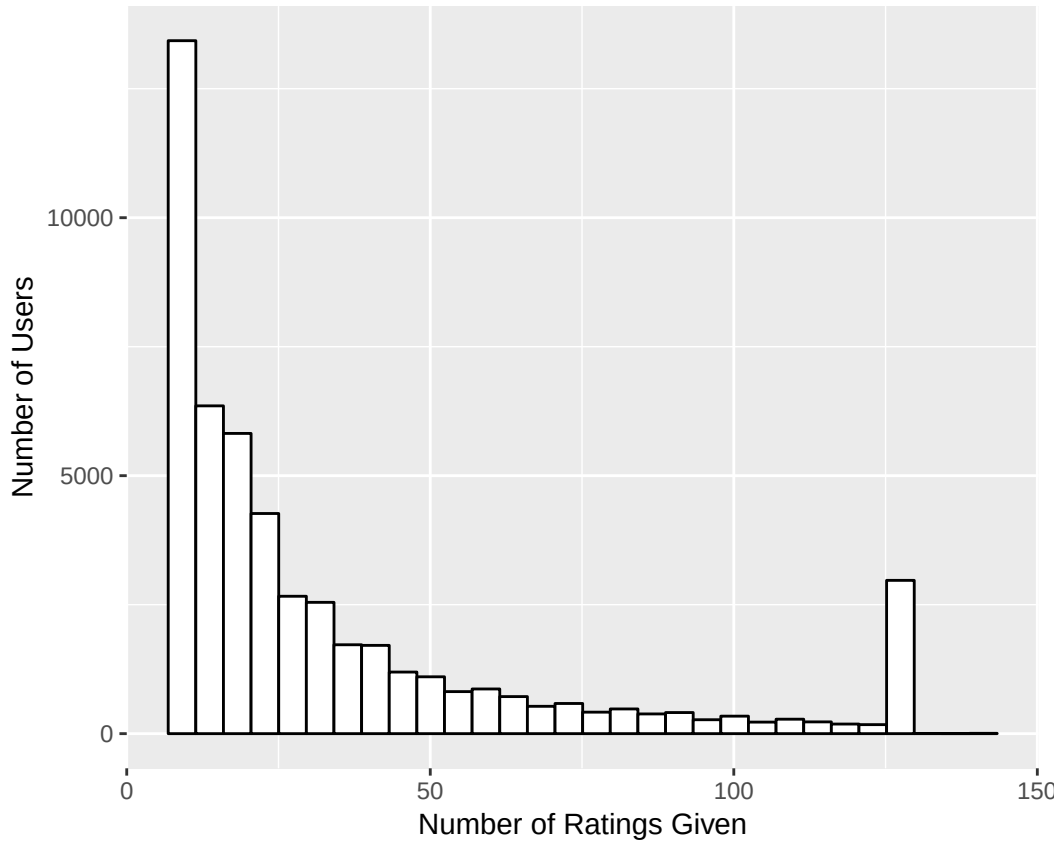

Figure 2: User Ratings Distribution

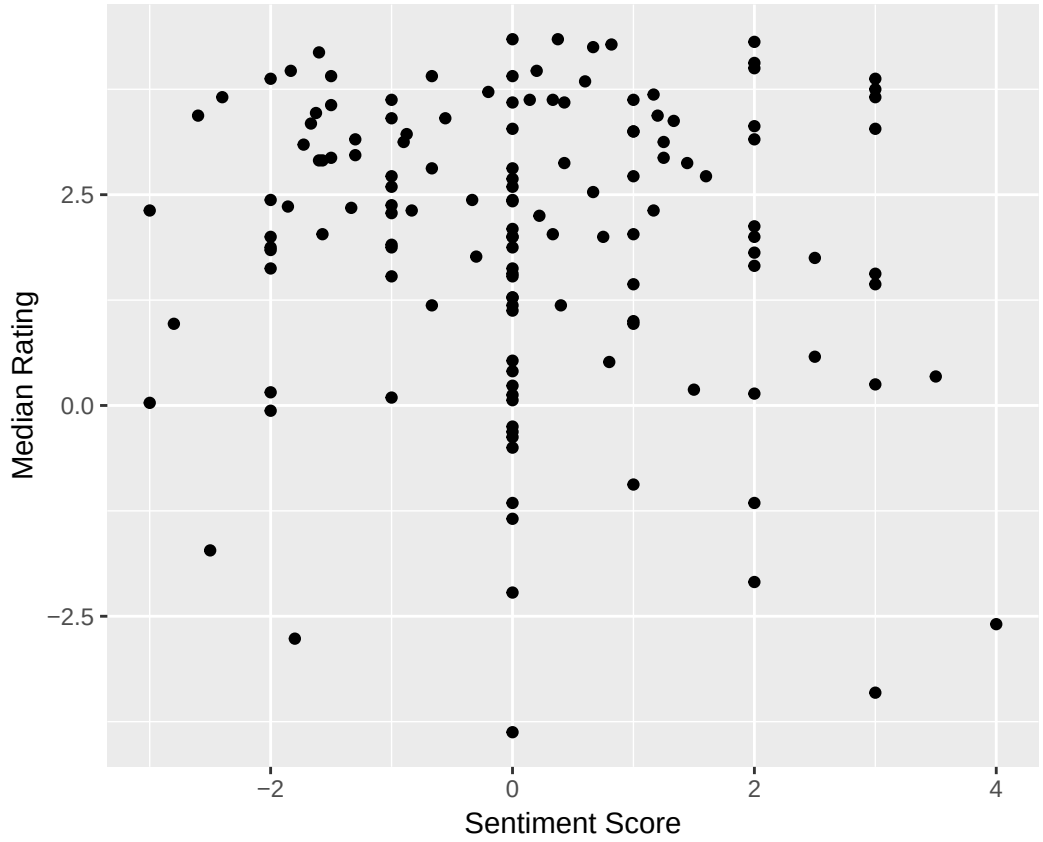

Figure 3: Sentiment vs Median Rating

Attain a sentiment score for each joke from DataScienceToolkit.com

Getting sentiment score and median for jokes to see if there's any relation between joke rating and sentiment:

```
for (i in 1:nrow(jokes)) {
  jokes[i,2]<- text2sentiment(jokes[[i,1]])
}

for (i in 1:nrow(jokes)) {
  jokes[i,3]<- median(df[[i+1]],na.rm=TRUE)
}
head(jokes)
```

```
#> # A tibble: 6 x 3
#>   Joke                                                Sentiment AvgRating
#>   <chr>                                                   <dbl>     <dbl>
#> 1 "A man visits the doctor. The doctor says, \"I have~    -0.25      NA
#> 2 "This couple had an excellent relationship going un~     0.333     NA
#> 3 Q. What's 200 feet long and has 4 teeth? A. The fro~     0         NA
#> 4 Q. What's the difference between a man and a toilet~     0         NA
#> 5 Q. What's O. J. Simpson's web address? A. Slash, sl~    -1.8       -2.77
#> 6 "Bill and Hillary Clinton are on a trip back to Ark~     3         NA

#> Warning: Removed 10 rows containing missing values (geom_point).
```

From the plot (Fig. 3), it seems as there is no correlation between sentiment score and joke rating. Further analysis into this would be to see if there are users who like distinctly high/low sentiment score jokes.

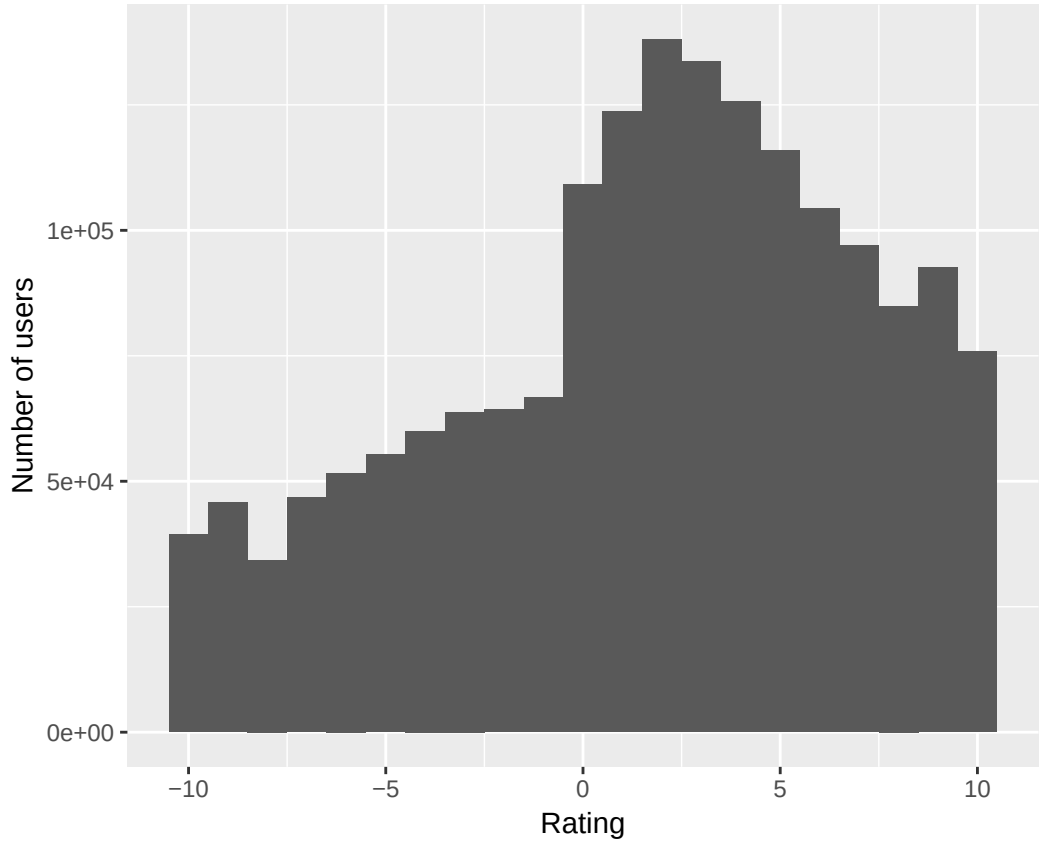

Figure 4: Histogram of rating

Build the Joke Recommender with Recommender Lab library

Convert data to realRatingMatrix object

```
s<-as(df[,2:ncol(df)], "matrix")
dimnames(s) <- list(rownames(s, do.NULL = FALSE, prefix = "u"),
                    colnames(s, do.NULL = FALSE, prefix = "j"))
s<-as(dropNA(s), "sparseMatrix")
rm <- new("realRatingMatrix",data=s)
rm
```

```
#> 50692 x 150 rating matrix of class 'realRatingMatrix' with 1728847 ratings.
```

Visualizing ratings

The ratings (Fig. 4) in the dataset are from -10 to +10. It's interesting that substantially more jokes have a positve rating than negative. With the highest density of ratings at +3. This indicates jokes are generaly enjoyed, but users don't often *love* them.

Check how many jokes have the users rated

```
summary(rowCounts(rm))
```

```
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>     8.0    11.0    20.0    34.1    42.0   140.0
```

```
nusers=dim(rm)[1]
njokes=dim(rm)[2]
```

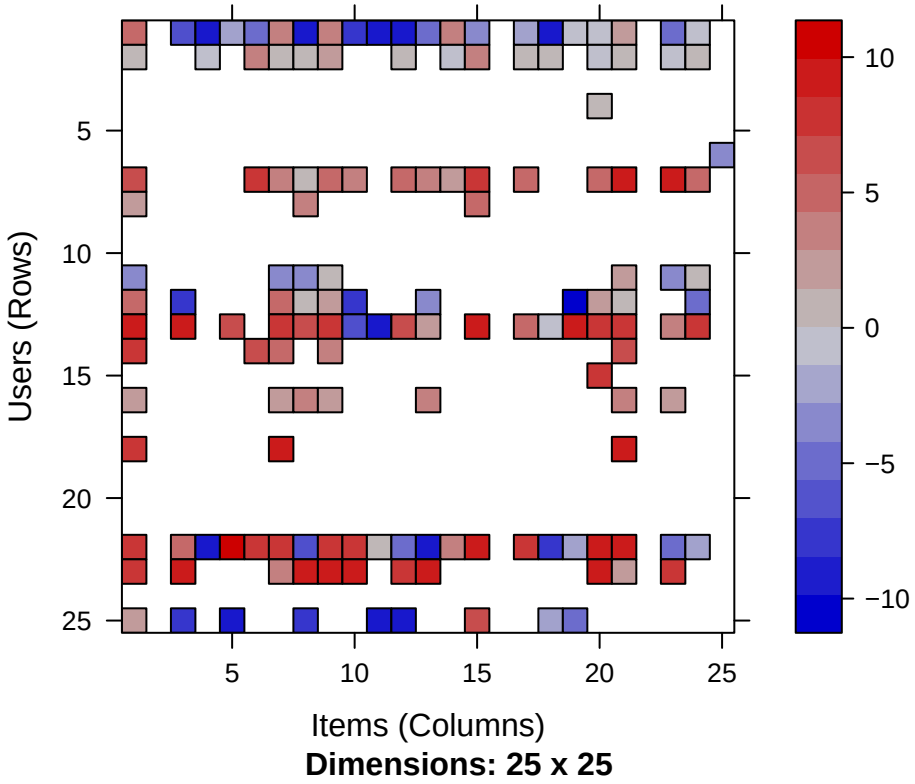

Figure 5: Ratings distribution visualization

Visualise a part of the ratings matrix.

There is lots of missing data (Fig. 5). This will potentially affect the accuracy of the reccomender algorithm as it does not have a great deal of data to train from.

Visualizing a sample of ratings

The following plot (Fig. 6) shows there is a lot of missing data. This is evident of a userbase that has not rated a substantial amount of the content.

What is the distribution of ratings

The ratings (as shown in the (Fig. 7) histogram) is right skewed. with the median at a rating of 2.25.

```
summary(getRatings(rm))
```

```
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#> -10.000  -1.938   2.250   1.657   5.719  10.000
```

With the ratings normalized (Fig. \ref{fig:fig7), the right-skewed tendency is still present. This indicates more of the jokes were "liked" by the users.

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

Take subset of data users with more than 50 ratings

This subset of users who have rated more than 1/3 of the jokes will allow for our training set to be more dense, and subsequently improve the accuracy of the reccomender. This distribution seems better than the one done for all the users.

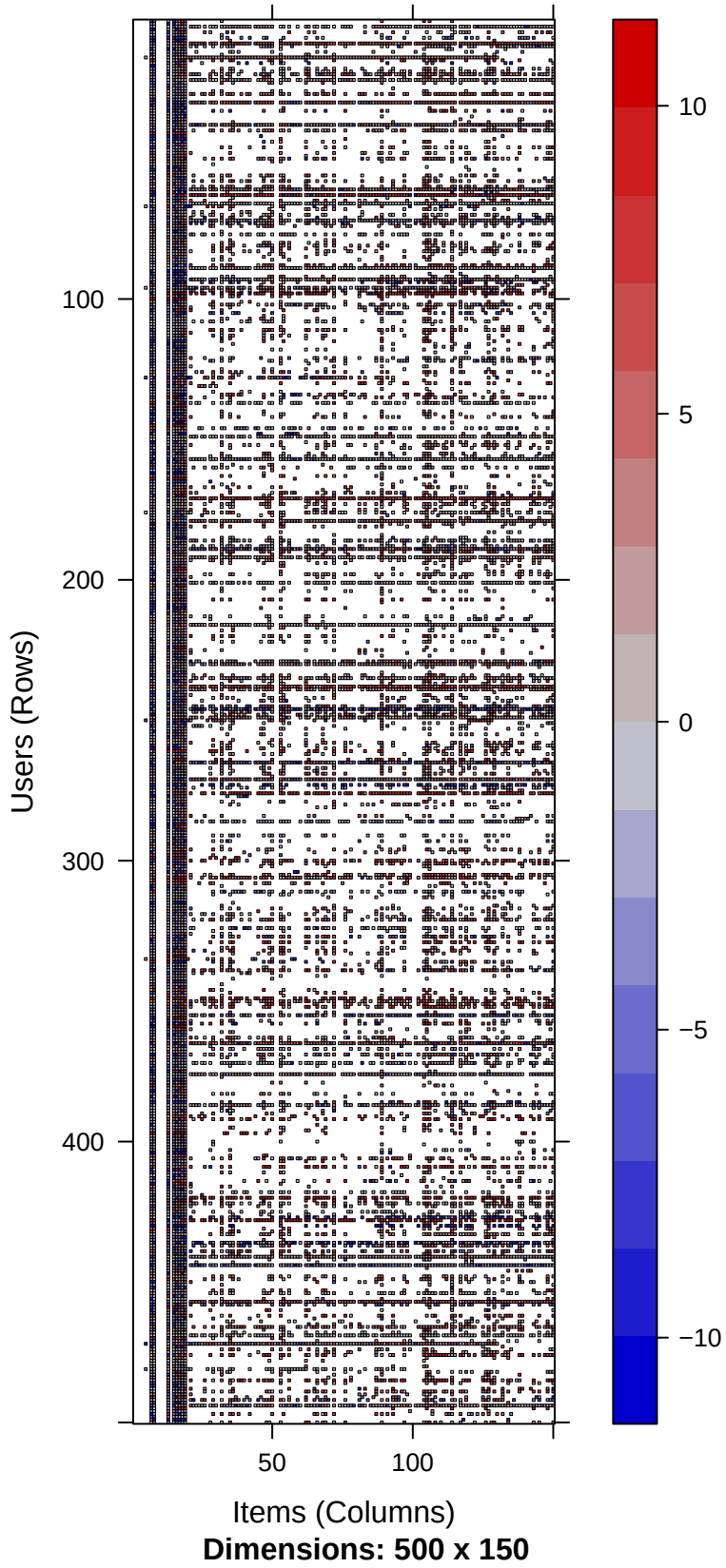

Figure 6: Raw ratings distribution visualization

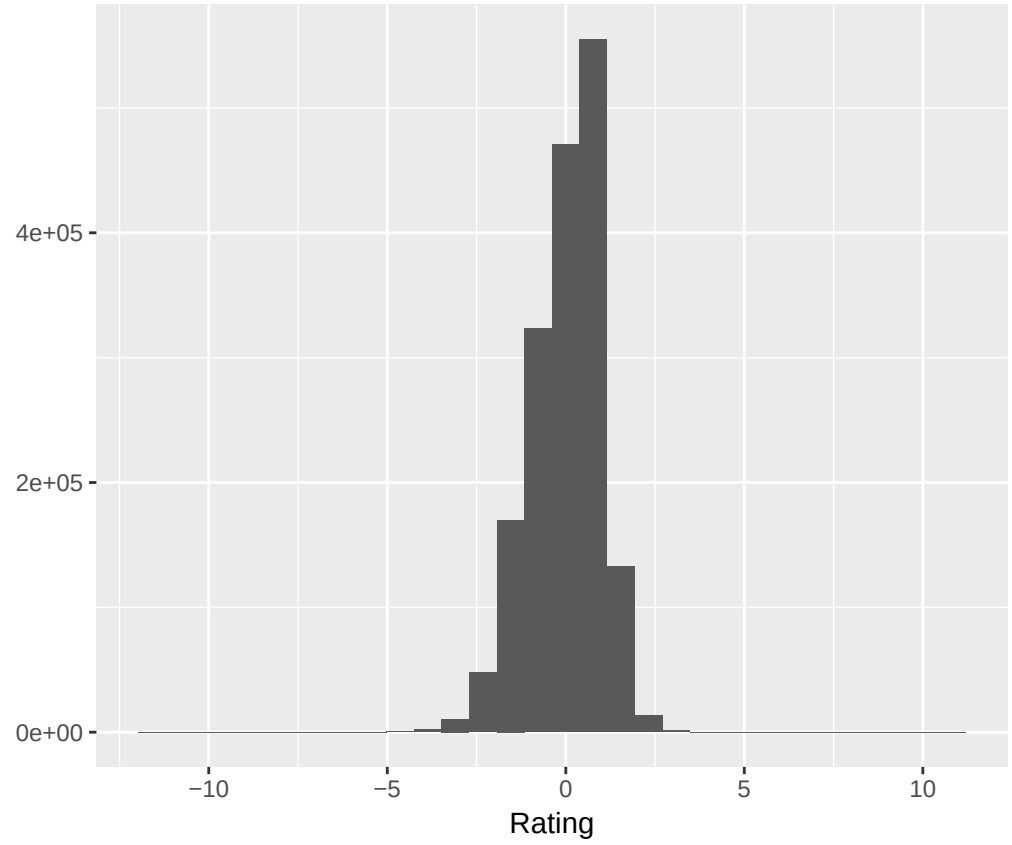

Figure 7: Histogram of normalized ratings

```
Jokes50 <- rm[rowCounts(rm) >50,]
Jokes50
```

```
#> 10290 x 150 rating matrix of class 'realRatingMatrix' with 947608 ratings.
```

```
summary(getRatings(normalize(Jokes50, method = "Z-score")))
```

```
#>     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
#> -11.2139  -0.6239   0.1741   0.0000   0.6879  11.2135
```

Similar logarithmic curve of ratings counts decreasing quickly (Fig. 8). And what looks to be a control group of users with ~140 ratings.

Present mean rating of jokes

The ratings of this small group are still right skewed (Fig. 9).

```
#> Warning: Removed 10 rows containing non-finite values (stat_bin).
```

Train a User-Based Collaborative Filtering Recommender using a small training set.

We have a few options.

```
scheme <- evaluationScheme(Jokes50, method = "split", train = .9,
                           k = 1, given = 10, goodRating = 4)
scheme
```

```
#> Evaluation scheme with 10 items given
#> Method: 'split' with 1 run(s).
```

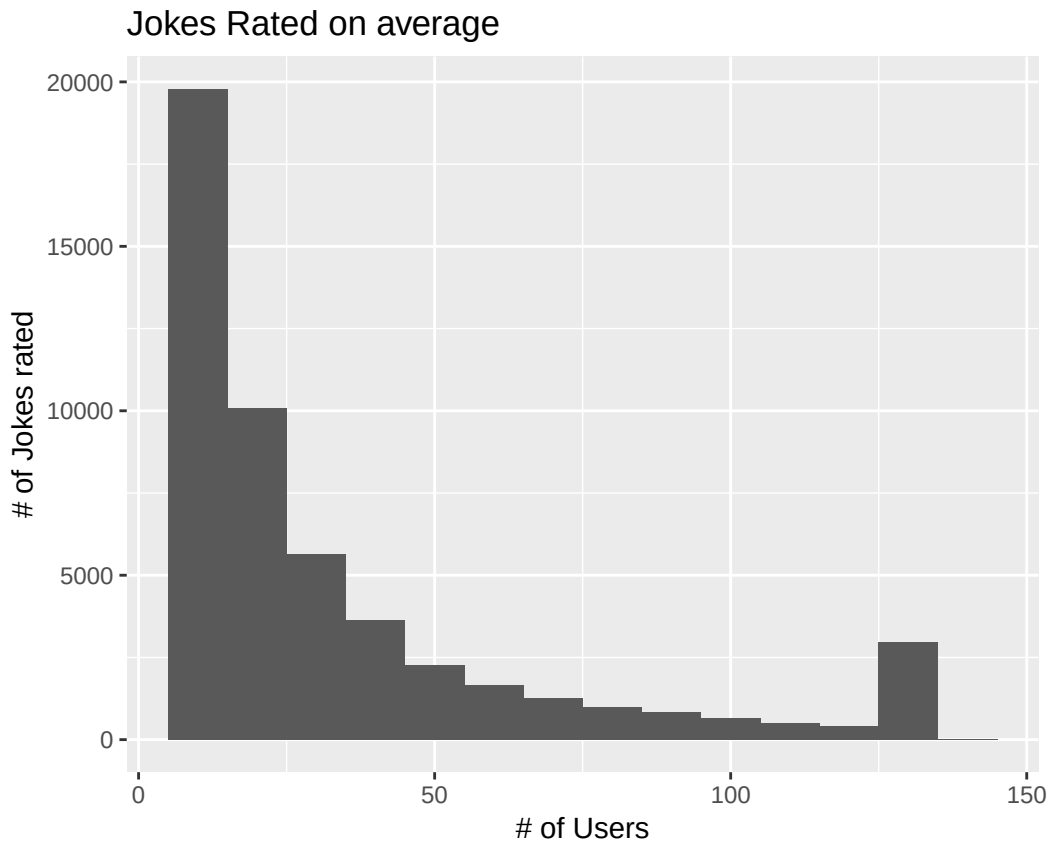

Figure 8: Rating count distribution for this subset

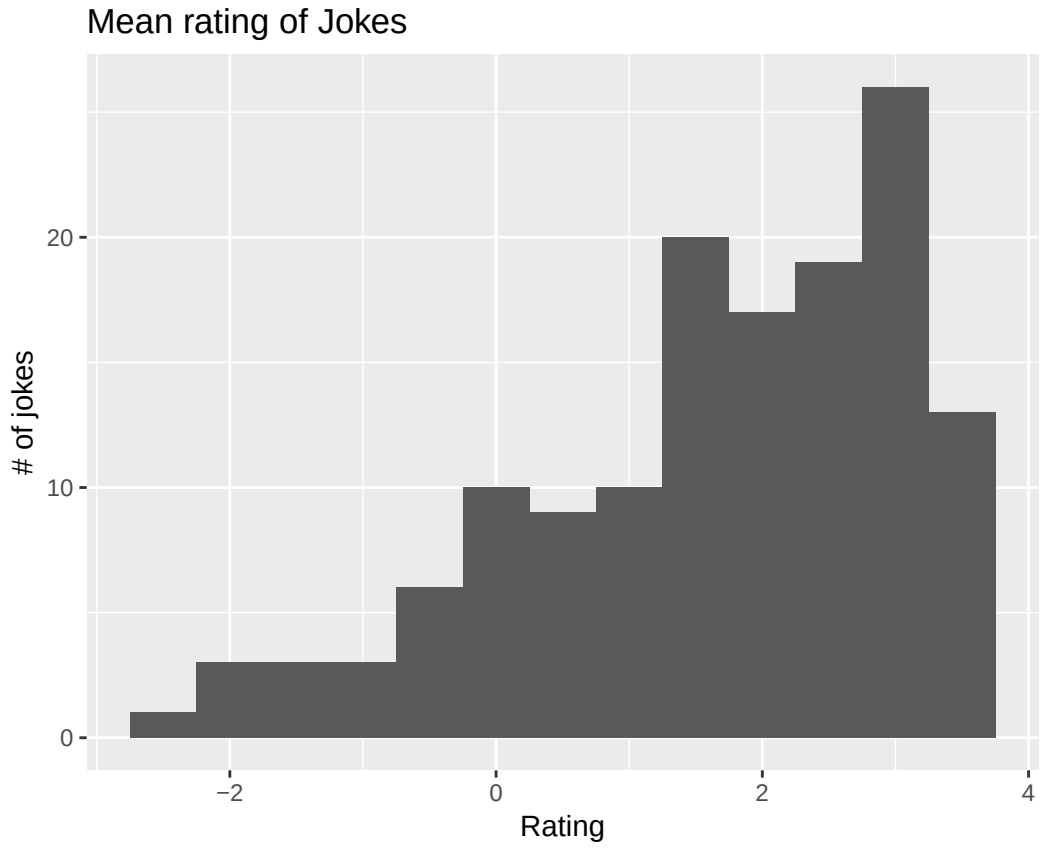

Figure 9: Rating count distribution for this subset

```
#> Training set proportion: 0.900
#> Good ratings: >=4.000000
#> Data set: 10290 x 150 rating matrix of class 'realRatingMatrix' with 947608 ratings.
```

Let's check some algorithms against each other

```
tr <- getData(scheme, "train"); tr
```

```
#> 9261 x 150 rating matrix of class 'realRatingMatrix' with 851317 ratings.
```

```
tst_known <- getData(scheme, "known"); tst_known
```

```
#> 1029 x 150 rating matrix of class 'realRatingMatrix' with 10290 ratings.
```

```
tst_unknown <- getData(scheme, "unknown"); tst_unknown
```

```
#> 1029 x 150 rating matrix of class 'realRatingMatrix' with 86001 ratings.
```

```
algorithms <- list(
  "random items" = list(name="RANDOM", param=list(normalize = "Z-score")),
  "popular items" = list(name="POPULAR", param=list(normalize = "Z-score")),
  "user-based CF" = list(name="UBCF", param=list(normalize = "Z-score", method="Cosine", nn=50)),
  "item-based CF" = list(name="IBCF", param=list(normalize = "Z-score", method="Cosine"))
)
```

run algorithms, predict next n jokes (list)

```
results <- evaluate(scheme, algorithms, n=c(1, 3, 5, 10, 15, 20))
```

```
#> RANDOM run fold/sample [model time/prediction time]
#>   1  [0.02sec/0.61sec]
#> POPULAR run fold/sample [model time/prediction time]
#>   1  [1.3sec/3.1sec]
#> UBCF run fold/sample [model time/prediction time]
#>   1  [1.27sec/15.54sec]
#> IBCF run fold/sample [model time/prediction time]
#>   1  [3.17sec/0.63sec]
```

Visualize ROC curves

One can see (Fig. 10) that the Popular and User-based algorithms have better true positive rate performance than random item selection. We will use the UserBased model as it is more relevant to the assignment.

Find RMSE for User-Based CF Recommender

```
rcmnd_ub <- Recommender(tr, "UBCF", param=list(method="pearson",nn=50))
rcmnd_ub_c <- Recommender(tr, "UBCF", param=list(method="Cosine",nn=50))
rcmnd_Ib <- Recommender(tr, "IBCF", param=list(method="Cosine"))
```

Create predictions for the test users using known ratings

```
pred_ub <- predict(rcmnd_ub, tst_known, type="ratings"); pred_ub
```

```
#> 1029 x 150 rating matrix of class 'realRatingMatrix' with 144060 ratings.
```

```
pred_ub_c<- predict(rcmnd_ub_c, tst_known, type="ratings"); pred_ub_c
```

```
#> 1029 x 150 rating matrix of class 'realRatingMatrix' with 144060 ratings.
```

```
pred_ib <- predict(rcmnd_Ib, tst_known, type="ratings"); pred_ib
```

```
#> 1029 x 150 rating matrix of class 'realRatingMatrix' with 117800 ratings.
```

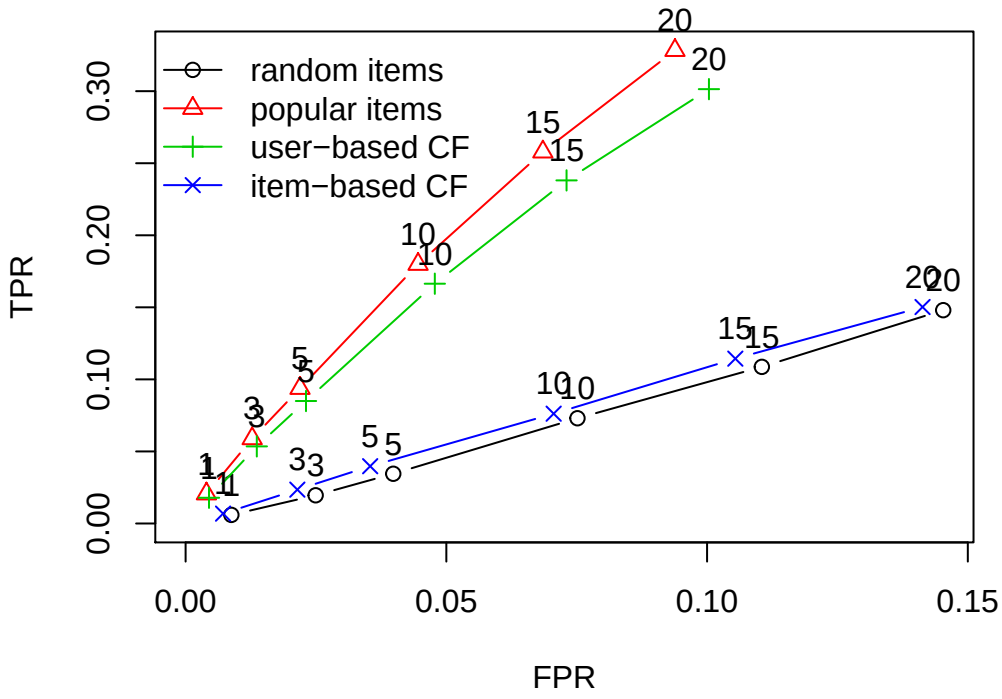

Figure 10: Performance of Popular and User-based algorithms

Evaluate recommendations on "unknown" ratings

```
#UB Pearson
acc_ub <- calcPredictionAccuracy(pred_ub, tst_unknown)
as(acc_ub,"matrix")

#>           [,1]
#> RMSE  4.363748
#> MSE  19.042298
#> MAE   3.277631

#UB Cosine
acc_ub_c <- calcPredictionAccuracy(pred_ub_c, tst_unknown)
as(acc_ub_c,"matrix")

#>           [,1]
#> RMSE  4.369451
#> MSE  19.092098
#> MAE   3.280443

#IB Cosine
acc_ib <- calcPredictionAccuracy(pred_ib, tst_unknown)
as(acc_ib,"matrix")

#>           [,1]
#> RMSE  5.636133
#> MSE  31.765996
#> MAE   4.080273
```

The RMSE is higher than we'd like for all algorithms, and is likely due to the fact that there is so much missing data. Unfortunately, we will have to continue with this model. We will continue with the UB Pearson as it has the lowest RMSE value.

compare predictions with true "unknown" ratings

```
as(tst_unknown, "matrix")[1:8,1:5]
```

```
#>       1  2  3  4        5
#> u4   NA NA NA NA  6.90625
#> u14  NA NA NA NA  0.46875
#> u18  NA NA NA NA  6.31250
#> u38  NA NA NA NA -0.18750
#> u50  NA NA NA NA  9.87500
#> u68  NA NA NA NA  6.28125
#> u109 NA NA NA NA -3.56250
#> u210 NA NA NA NA  1.43750
```

```
as(pred_ub, "matrix")[1:8,1:5]
```

```
#>              1         2         3         4         5
#> u4    2.243750  2.243750  2.243750  2.243750  2.243750
#> u14  -3.028125 -3.028125 -3.028125 -3.028125 -3.028125
#> u18  -0.793750 -0.793750 -0.793750 -0.793750 -0.793750
#> u38   4.165625  4.165625  4.165625  4.165625  4.165625
#> u50   6.321875  6.321875  6.321875  6.321875  6.321875
#> u68  -1.109375 -1.109375 -1.109375 -1.109375 -1.109375
#> u109  2.990625  2.990625  2.990625  2.990625  2.990625
#> u210  0.784375  0.784375  0.784375  0.784375  0.784375
```

From a cursery visual check, the model accuracy is unfortunately quite low. This again is due to the sparse dataset.

Create top-N recommendations for new users (users 1000 and 1001)

```
pre <- predict(rcmnd_ub, Jokes50[1000:1001], n = 10)
pre
```

```
#> Recommendations as 'topNList' with n = 10 for 2 users.
```

Recommendations as 'topNList' with n = 10 for 2 users.

```
as(pre, "list")
```

```
#> $u3469
#>  [1] "119" "126" "49"  "68"  "150" "104" "110" "92"  "97"  "122"
#>
#> $u3471
#>  [1] "116" "31"  "80"  "61"  "27"  "1"   "2"   "3"   "4"   "6"
```

Create a 'new' user ratings and recommend next joke

```
# create a random matrix with ratings
rr <- sample(c(NA,0:5),ncol(df)-1, replace=TRUE, prob=c(.7,rep(.3/6,6)))
rr
```

```
#>   [1] NA  2 NA NA  3  3 NA NA NA NA NA NA NA NA  5  5 NA NA NA NA NA NA NA
#>  [24] NA NA NA NA NA NA NA  3 NA NA NA NA  3  5 NA NA NA NA NA NA  3  4  4
#>  [47] NA NA  3 NA  4 NA NA NA NA NA NA NA NA NA  3 NA NA NA  5  1  0  3 NA
#>  [70]  0 NA  5 NA NA NA NA NA  3 NA  5 NA NA  3 NA  2 NA NA  2 NA  1  1 NA
#>  [93] NA  5  0  2  5 NA NA NA NA  3  1 NA NA NA  1 NA NA  1  1  0 NA  4 NA
#> [116]  3 NA NA NA NA  5  2 NA NA NA NA NA NA NA NA NA NA  5 NA NA NA NA NA
#> [139]  0  0 NA NA  0  4  0  4 NA NA NA  1
```

```
## coerce into a realRatingMAtrix
m <- matrix(rr,
    nrow=1, ncol=(ncol(df)-1), dimnames = list(
        user=paste('u', 1:1, sep=''),
        item=paste('i', 1:(ncol(df)-1), sep='')
```

```
    ))
m

#>     item
#> user i1 i2 i3 i4 i5 i6 i7 i8 i9 i10 i11 i12 i13 i14 i15 i16 i17 i18 i19
#>   u1 NA  2 NA NA  3  3 NA NA NA  NA  NA  NA  NA  NA   5   5  NA  NA  NA
#>     item
#> user i20 i21 i22 i23 i24 i25 i26 i27 i28 i29 i30 i31 i32 i33 i34 i35 i36
#>   u1  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA   3  NA  NA  NA  NA   3
#>     item
#> user i37 i38 i39 i40 i41 i42 i43 i44 i45 i46 i47 i48 i49 i50 i51 i52 i53
#>   u1   5  NA  NA  NA  NA  NA  NA   3   4   4  NA  NA   3  NA   4  NA  NA
#>     item
#> user i54 i55 i56 i57 i58 i59 i60 i61 i62 i63 i64 i65 i66 i67 i68 i69 i70
#>   u1  NA  NA  NA  NA  NA  NA  NA   3  NA  NA  NA   5   1   0   3  NA   0
#>     item
#> user i71 i72 i73 i74 i75 i76 i77 i78 i79 i80 i81 i82 i83 i84 i85 i86 i87
#>   u1  NA   5  NA  NA  NA  NA  NA   3  NA   5  NA  NA   3  NA   2  NA  NA
#>     item
#> user i88 i89 i90 i91 i92 i93 i94 i95 i96 i97 i98 i99 i100 i101 i102 i103
#>   u1   2  NA   1   1  NA  NA   5   0   2   5  NA  NA   NA   NA    3    1
#>     item
#> user i104 i105 i106 i107 i108 i109 i110 i111 i112 i113 i114 i115 i116 i117
#>   u1   NA   NA   NA    1   NA   NA    1    1    0   NA    4   NA    3   NA
#>     item
#> user i118 i119 i120 i121 i122 i123 i124 i125 i126 i127 i128 i129 i130 i131
#>   u1   NA   NA   NA    5    2   NA   NA   NA   NA   NA   NA   NA   NA   NA
#>     item
#> user i132 i133 i134 i135 i136 i137 i138 i139 i140 i141 i142 i143 i144 i145
#>   u1   NA    5   NA   NA   NA   NA   NA    0    0   NA   NA    0    4    0
#>     item
#> user i146 i147 i148 i149 i150
#>   u1    4   NA   NA   NA    1

ratings.new <- as(m, "realRatingMatrix")
print (ratings.new)

#> 1 x 150 rating matrix of class 'realRatingMatrix' with 49 ratings.

# recommend next joke
recNext <- predict(rcmnd_ub, ratings.new, n = 1)
recNum<-as(recNext, "list")
recNum

#> $u1
#> [1] "53"

sprintf("The next recommended joke is %s", recNum[[1]])

#> [1] "The next recommended joke is 53"

print (jokes[recNum[[1]],1])

#> # A tibble: 1 x 1
#>   Joke
#>   <chr>
#> 1 "One Sunday morning William burst into the living room and said, \"Dad!~
```

Prepare Shiny App for Deployment

Saving recommender data objects

```
saveRDS(rcmnd_ub, file = "jokeRecommender.Rds")
saveRDS(jokes, file = "jokes.Rds")
```

Deployment Discussion

This model is currently not of much use given its accuracy but it will serve as a proof of concept. This model could be used to help writers of movies/tv shows write jokes appropriate for a specific or large audience.

More data should be collected from this userbase to fill a training dataset. The dataset in its current state is quite sparse. The data would need to be updated every 3-5 years as people's taste changes and people within certian age groups mature. The Shiny app developed would be a deployment method to collect more data.

Further analysis could be done (with the appropriate data) to see how similar taste in humor is related to age.

The model developed in this project was used to create Shiny application currently deployed at ivbsoftware.shinyapps.io/JokeRecommender/. Code of the application could be found in Github.

Note from the Authors

This file was generated using *The R Journal* style article template, additional information on how to prepare articles for submission is here - Instructions for Authors. The article itself is an executable R Markdown file that could be downloaded from Github with all the necessary artifacts.

Igor Baranov
York University School of Continuing Studies

https://learn.continue.yorku.ca/user/profile.php?id=21219

Michael Parravani
York University School of Continuing Studies

Ariana Biagi
York University School of Continuing Studies

Hui Fang Cai
York University School of Continuing Studies