DATA 612 Project 4 - Accuracy and Beyond

By: Michael D'Acampora

The goal of this assignment is give you practice working with accuracy and other recommender system metrics. In this assignment you're asked to do at least one or (if you like) both of the following: • Work in a small group, and/or • Choose a different dataset to work with from your previous projects

1. As in your previous assignments, compare the accuracy of at least two recommender system algorithms against your offline data.
2. Implement support for at least one business or user experience goal such as increased serendipity, novelty, or diversity.
3. Compare and report on any change in accuracy before and after you've made the change in #2.
4. As part of your textual conclusion, discuss one or more additional experiments that could be performed and/or metrics that could be evaluated only if online evaluation was possible. Also, briefly propose how you would design a reasonable online evaluation environment.

```r
library(recommenderlab)
library(tidyverse)
```

Data import and analysis

```r
data("Jester5k")
```

```r
dim(Jester5k)
```

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

```r
typeof(Jester5k)
```

```
## [1] "S4"
```

All of the objects in `recommenderLab` are created under the S4 Object Oriented system, which presents a different approach to thinking about how the dataset is manipulated and how models are used.

```r
# Summary of ratings data
jester_df <- as(Jester5k, 'data.frame')
summary(jester_df$rating)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -9.95   -3.06    1.46    0.85    5.10    9.90
```

The ratings are on a scale of -10 to +10. Since we have a mean rating of 0.85 and a median of 1.46 we can consider that range the average, and certainly not good by joke standards.

```r
# Total number of ratings
nratings(Jester5k)
```

```
## [1] 362106
```

```r
# Number of ratings per user
summary(rowCounts(Jester5k))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   36.00   53.00   72.00   72.42  100.00  100.00
```

```r
# Sparsity percentage
nratings(Jester5k) / (dim(Jester5k)[1] * dim(Jester5k[2]))
```

```
## [1] 72.421200  0.724212
```

```r
jester_matrix <- spread(jester_df, key = "item", value = "rating")
head(jester_matrix, 10)
```

```
##      user    j1   j10  j100   j11   j12   j13   j14   j15   j16   j17
## 1  u10005    NA    NA    NA    NA    NA -8.54    NA -4.95 -9.76 -6.60
## 2  u10014    NA    NA    NA    NA    NA -8.16 -6.17 -0.34 -9.47 -4.42
## 3   u1002  3.20 -5.44 -4.13 -8.35 -1.36  4.08 -8.69 -3.25 -8.16  2.86
## 4  u10020  7.23  9.13 -9.13  7.23  4.22 -2.91  8.11 -1.75  0.24  2.57
## 5  u10023  3.98  5.92  4.76  6.55  7.33  5.44  4.90 -6.94  5.53  2.52
## 6  u10029  1.41  1.12 -2.86  4.17  3.01  3.06  3.16  6.41 -7.62  2.04
## 7  u10037  3.20  6.36  7.67  8.11  0.97  0.34  5.34 -3.69 -4.47  1.50
## 8  u10040    NA    NA    NA    NA  5.97 -9.61  7.09 -8.01 -9.85 -1.36
## 9  u10042 -8.20  1.99  6.36  5.10  7.18 -0.63 -3.11  1.02 -7.18  3.79
## 10 u10043  5.63  5.24  3.93  3.20  6.21  4.66  6.89  4.90 -9.47  5.63
##      j18   j19    j2   j20   j21   j22   j23   j24   j25   j26   j27   j28
## 1  -9.03 -6.89    NA -9.76 -8.16    NA    NA    NA    NA -8.69 -9.81    NA
## 2  -0.44 -5.10    NA -7.48 -2.77    NA    NA    NA -6.41 -0.39  0.78 -6.60
## 3  -3.40 -7.67 -0.34  0.97 -3.93 -5.97 -0.73 -8.01  5.05 -4.27  0.68 -4.81
## 4   2.23  7.38  1.70  6.65  8.30  7.14  8.59  8.30  6.55  5.44  8.11  9.17
## 5   1.89  7.09  0.63 -2.86  7.62  1.26  5.63  4.85  3.06  0.05  7.57  0.10
## 6  -1.99 -3.59  0.68 -2.14  2.38  3.50  2.23 -7.23  4.22  3.11 -3.59  0.87
## 7   4.32  1.60 -0.58 -2.72  4.66  8.16  5.83 -2.28  3.59  2.23  8.69  8.11
## 8   5.10  3.83    NA -7.28  7.09  7.38  3.69    NA  7.62  7.82  4.08  7.48
## 9  -1.12  0.53  5.63 -6.36  8.79  5.00  0.92 -3.45 -3.69  1.84  2.96  8.98
## 10  6.02  2.14  5.24 -8.88  5.19  4.13  1.21 -2.43  6.07  7.67  4.47  3.64
##      j29    j3   j30   j31   j32   j33   j34   j35   j36   j37   j38   j39
## 1  -8.25    NA    NA -3.98 -6.80    NA -8.01 -6.46 -9.56    NA    NA    NA
## 2   0.63    NA    NA  3.88  1.65    NA -2.23  0.29  1.31    NA -0.29 -4.51
## 3  -0.39  4.22 -4.42 -6.80 -6.99 -0.53 -8.69  5.10 -3.93  2.67  5.05 -0.19
## 4   3.98  9.17  7.72  2.28  8.30  3.35  7.38  7.67  4.08 -0.15  9.22  9.22
## 5   7.04  2.23  6.99  1.46 -0.24  5.63  6.99  7.86 -0.05 -0.29  1.12  3.45
## 6   5.63  5.29 -4.95  2.23  1.36  0.05  7.43 -2.09  1.75 -4.08  3.20  3.20
## 7   6.36  7.57 -1.99  3.59  7.72 -0.34  3.11  8.16  7.72  1.75 -0.73  1.99
## 8   3.98    NA    NA  5.92  5.39    NA  7.57  6.46 -2.96    NA  3.25  6.65
## 9   4.95 -5.58 -2.38  6.80  8.25 -3.20  2.18  6.46  2.82 -4.71  5.00  5.00
```

```
## 10  8.93  1.99 -0.78  8.40  7.67 -3.20  7.43  8.83  9.03  3.30 -1.84  5.78
##        j4   j40  j41   j42   j43   j44   j45   j46   j47   j48   j49    j5
## 1      NA    NA   NA -7.14 -9.51    NA    NA -9.51    NA -6.75 -3.88 -8.45
## 2      NA    NA   NA -0.63    NA    NA    NA  4.81    NA  1.89 -0.63 -0.10
## 3   -2.23 -5.44 0.15 -6.99 -5.63 -5.44 -5.78 -0.87 -3.40 -6.99 -3.45 -0.34
## 4    8.54  9.08 9.13  9.17  8.20  1.60  8.11  5.53 -0.97  8.79  7.57  1.36
## 5    5.29  9.03 4.90 -0.83  3.30  0.44  3.74  3.30  4.13  1.21  1.41  6.21
## 6    3.40 -0.44 3.40  4.51  3.83 -8.30  1.75  3.79 -9.47 -1.84  3.88 -8.25
## 7    5.15  5.05 7.18  4.61  2.57  0.53  7.43  7.38  4.71  5.87  8.01 -5.53
## 8      NA  6.65   NA  3.45    NA    NA  3.93  6.07  4.90  7.77  5.10  1.75
## 9   -3.59  3.45 1.70  3.30 -8.54 -2.28  0.92  8.59  6.12  6.07  8.64 -3.11
## 10   2.62  4.17 5.63  4.13  5.73 -4.95  5.53  8.45  5.24  2.96  3.98  6.41
##       j50   j51   j52   j53   j54   j55   j56   j57   j58   j59    j6   j60
## 1   -8.16    NA -9.81 -5.15 -8.16    NA -9.81    NA    NA    NA    NA    NA
## 2    1.84    NA    NA  1.07 -0.44    NA -1.41    NA    NA    NA    NA    NA
## 3    3.40 -8.01 -5.97 -7.48 -0.39 -2.38  3.88 -0.39 -0.87  3.40  1.84 -2.91
## 4    7.09  4.51  3.40  7.52  2.23  4.90  6.80 -3.83  6.84  8.35 -1.70  4.08
## 5    6.46  5.63  7.04  6.07 -0.87  7.04  6.94  4.56 -9.81  5.78  6.12  6.94
## 6    7.14 -0.34  1.70 -7.14  2.33 -3.30  3.64 -7.23 -7.52 -3.64 -0.92  0.34
## 7    4.61  2.04  1.89  7.62  8.11  6.17  8.20  7.57 -6.70  1.94  5.29  6.55
## 8    3.59    NA  6.60  7.86  6.17  7.52  6.50    NA    NA    NA    NA -8.74
## 9    1.36 -6.07  1.60  7.38  5.19  3.11  1.21  0.24 -9.66 -3.25  5.39 -7.28
## 10   8.64  2.23 -0.19  8.45  8.79  5.49  9.17 -1.89 -5.58 -1.12  5.49  4.51
##       j61   j62   j63   j64   j65   j66   j67   j68   j69    j7   j70   j71
## 1   -6.99 -7.14    NA    NA -9.13 -0.24    NA -6.46 -9.81 -9.51    NA    NA
## 2    4.32  0.97    NA    NA -2.33  3.98 -6.50  2.77  1.65 -0.44    NA -8.40
## 3    0.49 -0.39  2.72 -0.34 -2.91 -5.78 -4.81  1.17  0.49  5.24  0.34 -0.19
## 4    2.82  2.52  8.74 -0.68  4.08  8.06  8.25  7.14  2.96  5.87  7.91 -2.33
## 5    6.94  6.65  0.39  5.92  5.00  7.33  5.92  3.45  8.69  7.48  5.63  6.12
## 6    2.57  4.42 -3.30 -3.69  4.03  2.23 -5.63 -0.49 -0.05 -0.53 -0.53  6.46
## 7    8.11  7.33  3.45 -0.63  7.43  6.02 -1.99  4.90  7.14  5.15 -5.24  4.03
## 8    4.17  5.78    NA    NA  6.60  8.01    NA  5.53  6.94 -9.56  8.01    NA
## 9    8.45  6.26 -0.92 -3.11  8.98  9.22 -0.63  2.77  8.45  1.70 -9.61 -1.75
## 10   8.20  5.39  5.00  2.67  8.88  8.50 -9.47  8.01  6.50 -3.25  0.34    NA
##       j72   j73   j74   j75   j76   j77   j78   j79    j8   j80   j81   j82
## 1      NA    NA    NA    NA    NA    NA    NA    NA -5.05    NA    NA    NA
## 2      NA    NA    NA    NA    NA -0.58    NA    NA  0.58    NA    NA    NA
## 3   -6.99 -0.19 -6.65  1.02 -6.31  1.17  4.42 -8.88  4.37 -1.70  2.18 -3.59
## 4   -2.86  8.64 -9.32 -9.32 -9.32 -9.32  8.69 -9.76 -3.74 -9.76 -9.76  8.74
## 5    7.91  5.63  6.80  5.44  1.02  2.38  2.38  7.77 -0.68  8.45  3.54  1.80
## 6    6.12 -7.14 -5.15  5.44  8.50 -0.92  5.29 -5.39  3.50 -3.69 -2.86 -2.77
## 7    4.03  4.03  4.03  4.03  4.03 -1.80 -2.82 -3.06  0.39  6.21 -2.28  1.46
## 8      NA    NA    NA    NA    NA    NA    NA    NA -6.55    NA  7.72    NA
## 9    7.67  1.31 -9.08 -4.90  2.18 -1.55  4.37  5.24 -3.83  0.49  4.08  2.28
## 10     NA    NA    NA    NA    NA    NA    NA    NA  2.82  6.84  9.08    NA
##      j83   j84   j85   j86   j87   j88   j89    j9   j90   j91   j92   j93
## 1     NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 2     NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## 3   1.02 -6.12 -0.68 -5.97  3.69  2.52  6.75 -0.05 -5.63  2.52 -0.34 -2.77
## 4   8.74  8.74  3.83 -9.51  8.88  9.13 -9.37  1.80  9.13 -9.51  9.03 -9.61
## 5   5.00  2.28  5.78  5.92 -0.34  6.50  6.12  4.66  3.45  1.80 -0.68  5.15
## 6   5.92 -2.52 -4.56 -3.83 -3.69  6.55  8.50  0.10  5.15 -3.30 -2.91  4.56
## 7   3.20  5.63  6.17  1.46  7.91 -0.29  2.38  4.08 -1.65  3.01 -0.53 -0.49
## 8   9.22    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

```
## 9  1.89  1.94  6.50  3.11 -1.17  0.05  0.19 -8.69  4.37  1.65  2.86 -1.50
## 10   NA    NA    NA    NA    NA    NA    NA  3.06    NA    NA    NA    NA
##      j94   j95   j96   j97   j98   j99
## 1     NA    NA -5.83    NA    NA    NA
## 2     NA    NA    NA    NA    NA    NA
## 3  -7.14 -5.97  5.05  0.15 -5.78 -8.35
## 4   8.93  8.83  8.88 -9.61  7.04  9.13
## 5   8.83  2.28  5.58  5.68  2.67  1.02
## 6  -2.18  1.36  0.87  1.99  3.59 -0.68
## 7   7.96 -5.10 -3.59 -3.59 -4.81 -9.37
## 8     NA    NA    NA    NA  8.35    NA
## 9  -0.44  4.37 -1.70  1.50  3.79  7.57
## 10    NA    NA    NA    NA    NA -1.26
```

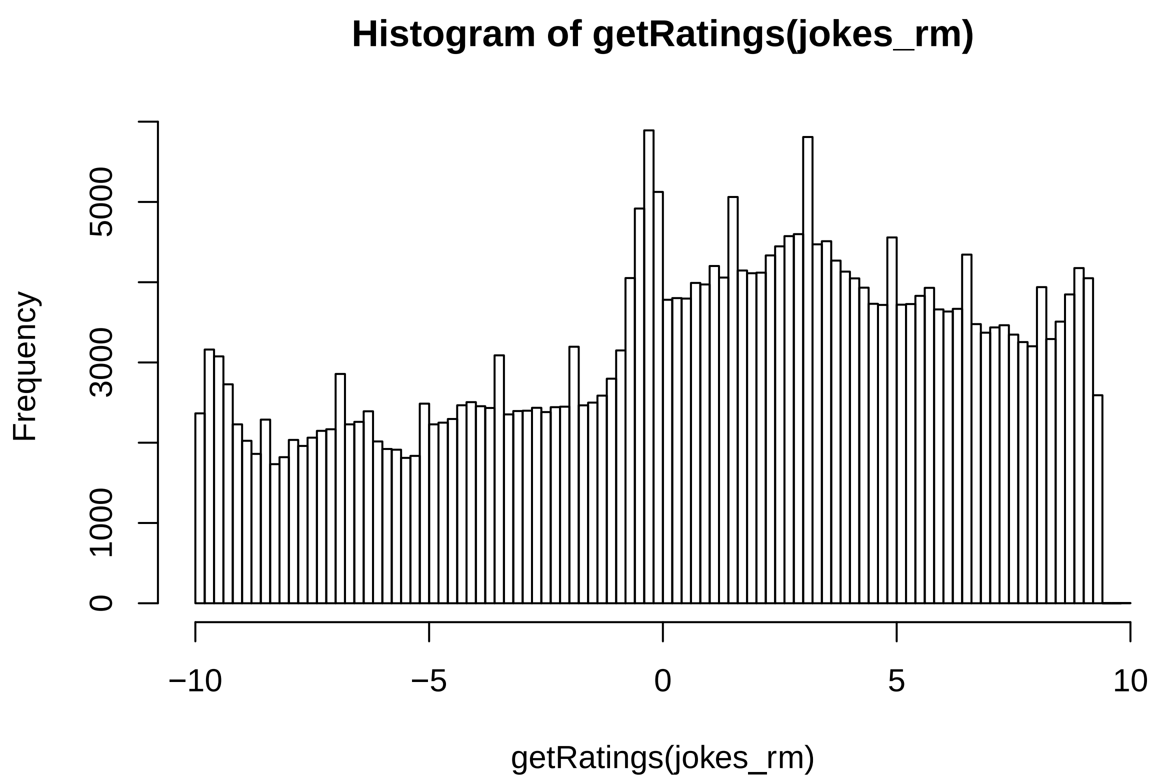
Using `tidyr` and the `gather` method we can take a quick snapshot at what the matrix looks like. However, since this was a `realratingsMatrix` S4 object, it takes more tinkering to get it to do what you're normally used to doing.

Let's reduce the dataset to include only where a user has more than 50 ratings.

```
jokes_rm <- Jester5k[rowCounts(Jester5k) > 50]

min(rowCounts(jokes_rm))
```

```
## [1] 51
```

```
hist(getRatings(jokes_rm), breaks=100)
```

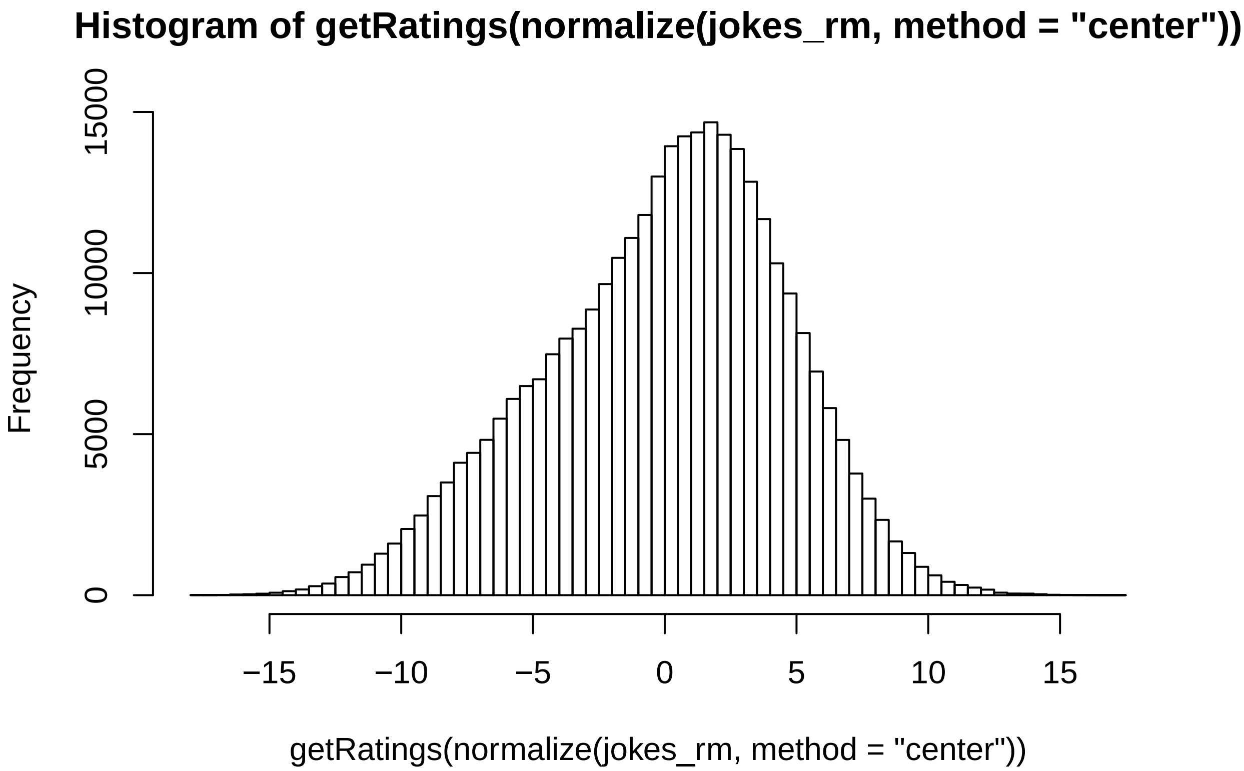


We can see when we plot a histogram that shows the negative vales occur with similar frequencies and the positive ratgins are more frequent but slope off as you get towards the max rating of 10.

Let's take a look at the same distribution after normalization.

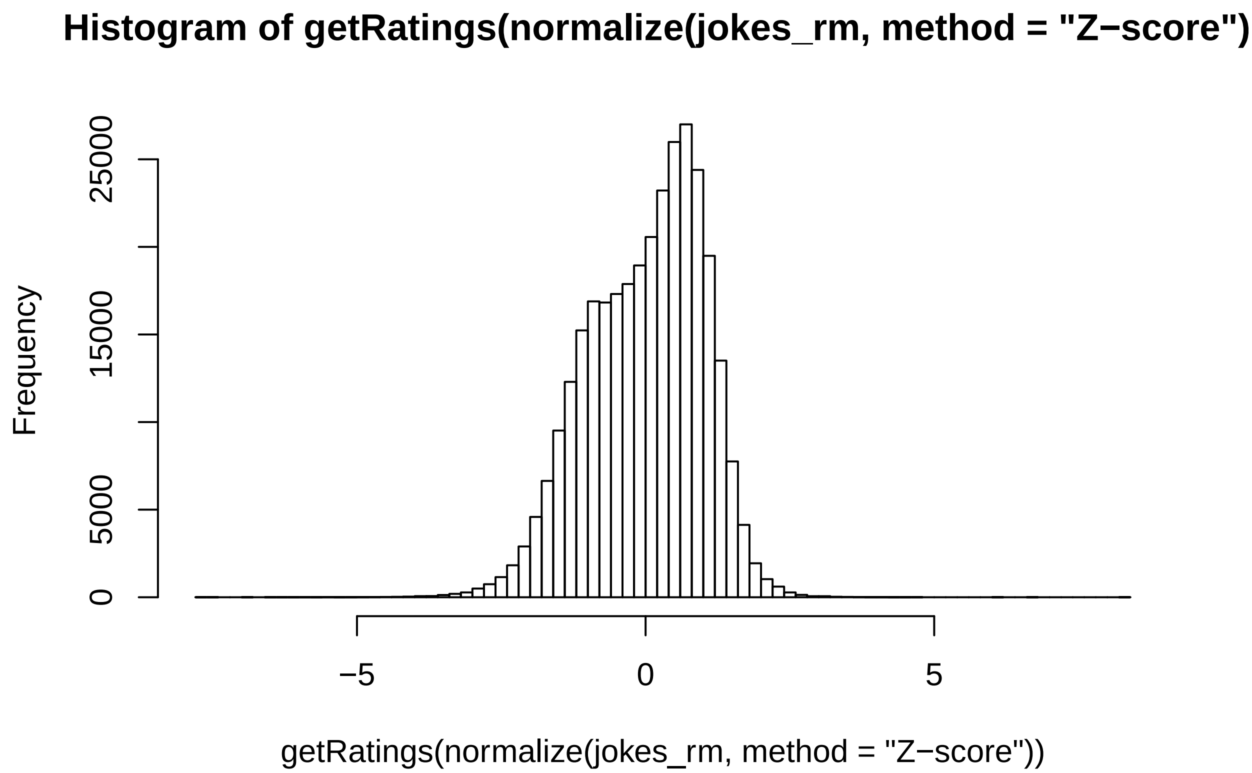
One by row centering...

```
hist(getRatings(normalize(jokes_rm, method="center")), breaks = 100)
```



And the other by Z-score.

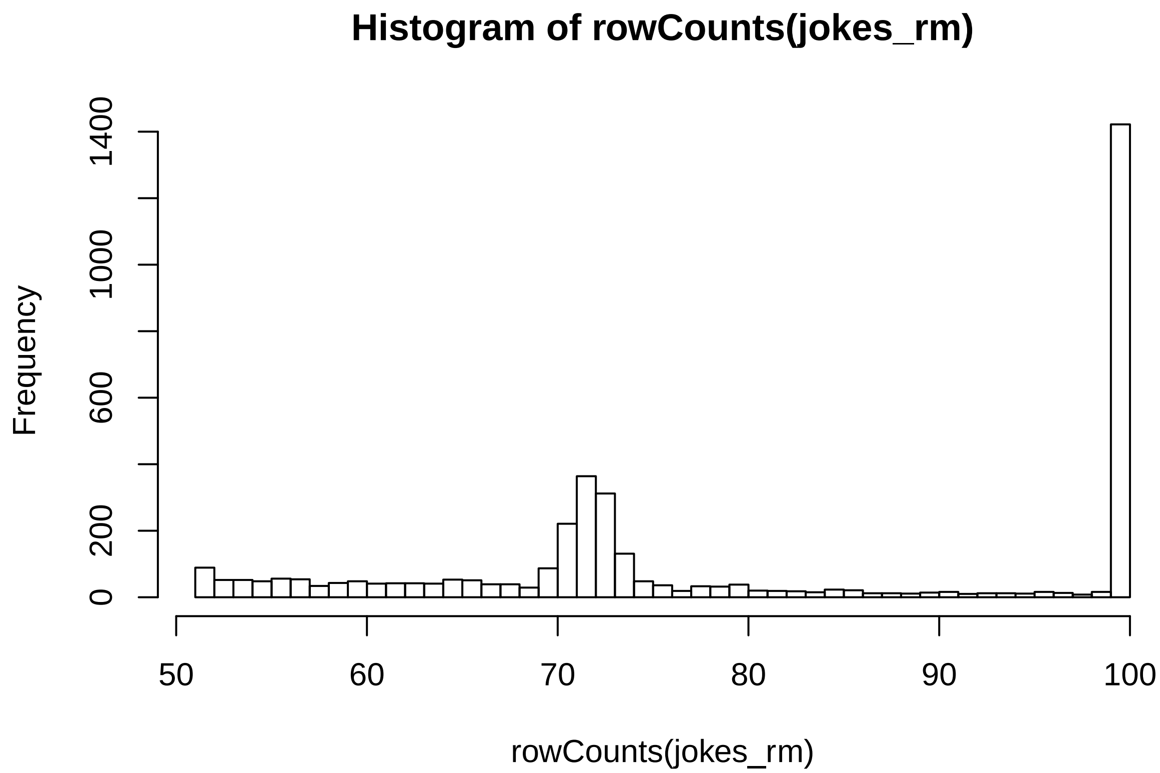
```
hist(getRatings(normalize(jokes_rm, method="Z-score")), breaks = 100)
```



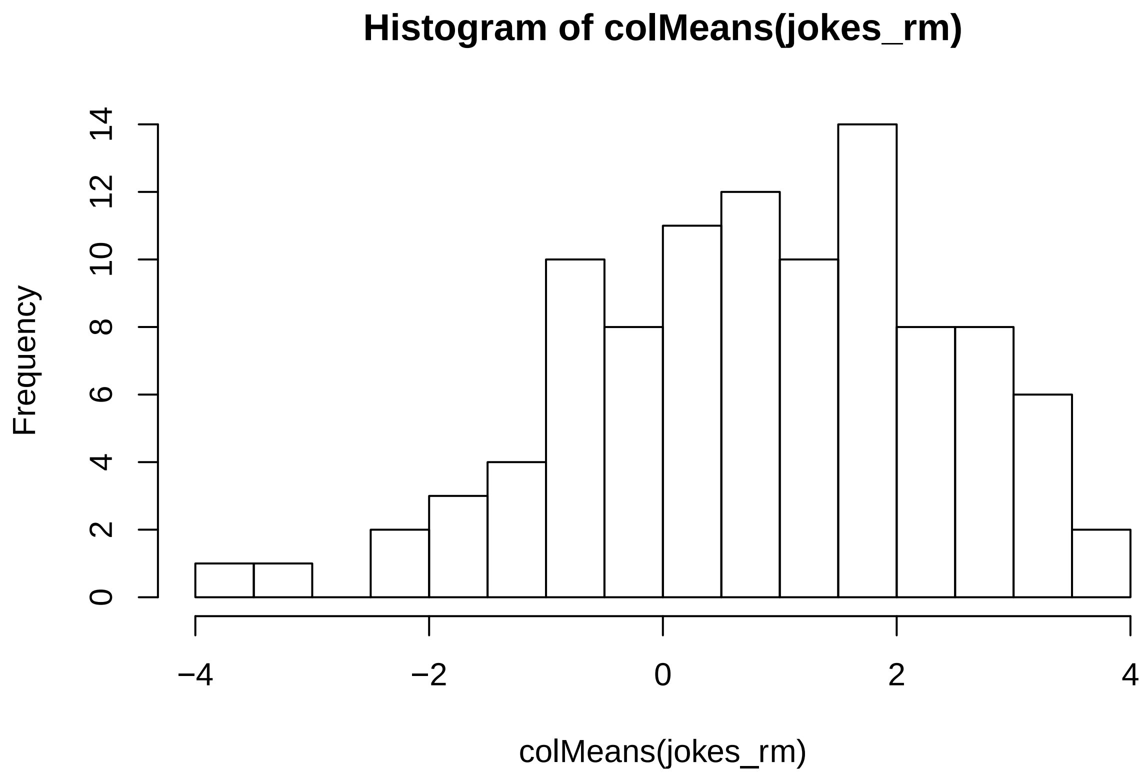
We can see the peak ratings in this reduced set range from about 0-1.

Lastly we can take a quick look at how many jokes each user rated and the average rating per joke, by taking the row count and column mean, respectively.

```
hist(rowCounts(jokes_rm), breaks = 50)
```




```
hist(colMeans(jokes_rm), breaks = 20)
```



Teehee

As a quick aside, we can find the max value, or "funniest" joke. Here it is...

```
funniest <- which.max(colMeans(Jester5k))
cat(JesterJokes[funniest])
```

```
## A guy goes into confession and says to the priest, "Father, I'm 80 years old, widower, with 11 grand
```

Evaluation

We will take four different recommender models. User-based Collaborative Filtering (UBCF), Item-based Collaborative Filtering (IBCF), Random recommendations (RANDOM), and selections based on Popularity (POPULAR). For each of the four models we will apply six combinations of similarity (cosine and pearson) and normalization (row center, Z-score, none) to them, for a total of 24 models.

First we build an evaluation set, which we use on the `jokes_rm` dataset. The data set is split at 80% training, 20% test. Jokes often have tough critics so we will consider a rating of 5 a "good" rating, which is at the edge of the 3rd interquartile range.

```
eval_set <- evaluationScheme(data = jokes_rm,
                             method = "split",
                             train = 0.8,
                             given = 30,
```

```r
                              goodRating = 5)
eval_set
```

```
## Evaluation scheme with 30 items given
## Method: 'split' with 1 run(s).
## Training set proportion: 0.800
## Good ratings: >=5.000000
## Data set: 3875 x 100 rating matrix of class 'realRatingMatrix' with 314302 ratings.
```

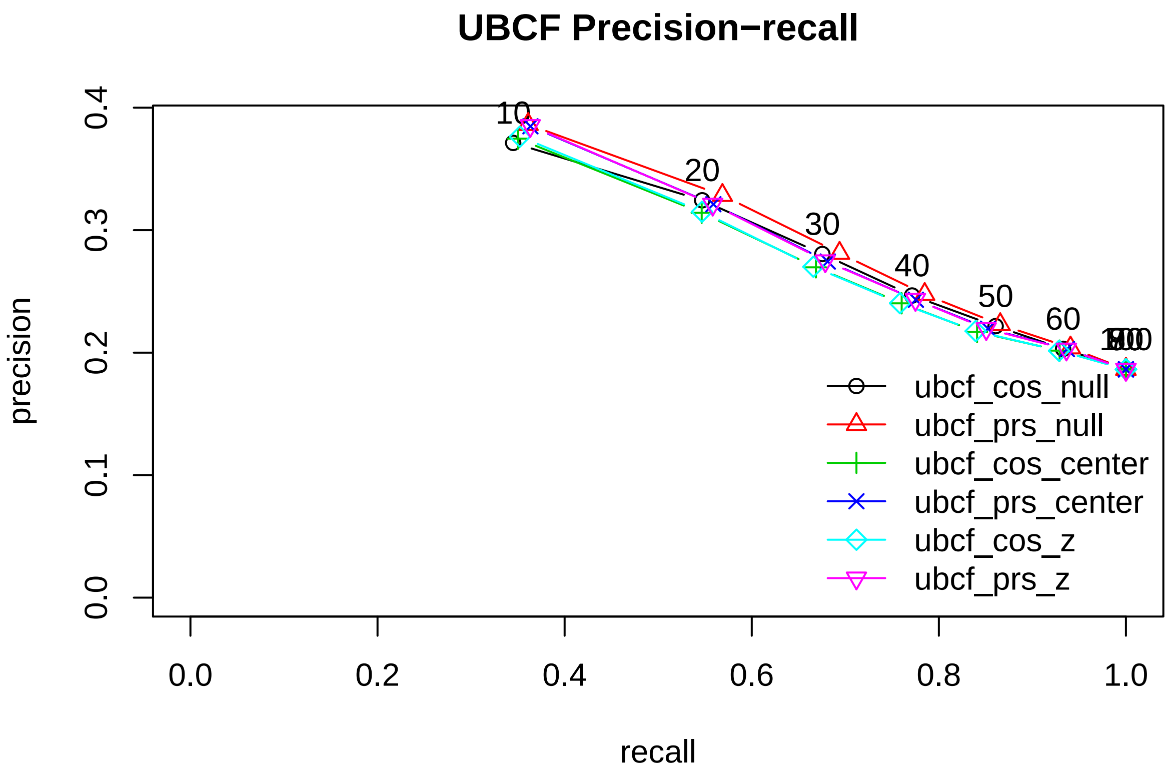
Now that the evaluation set is created, we will create and evaluate each of the four subject models, with their six subcomponents, and plot Precision-recall and ROC curves to visually evaluate model performance.

UBCF models

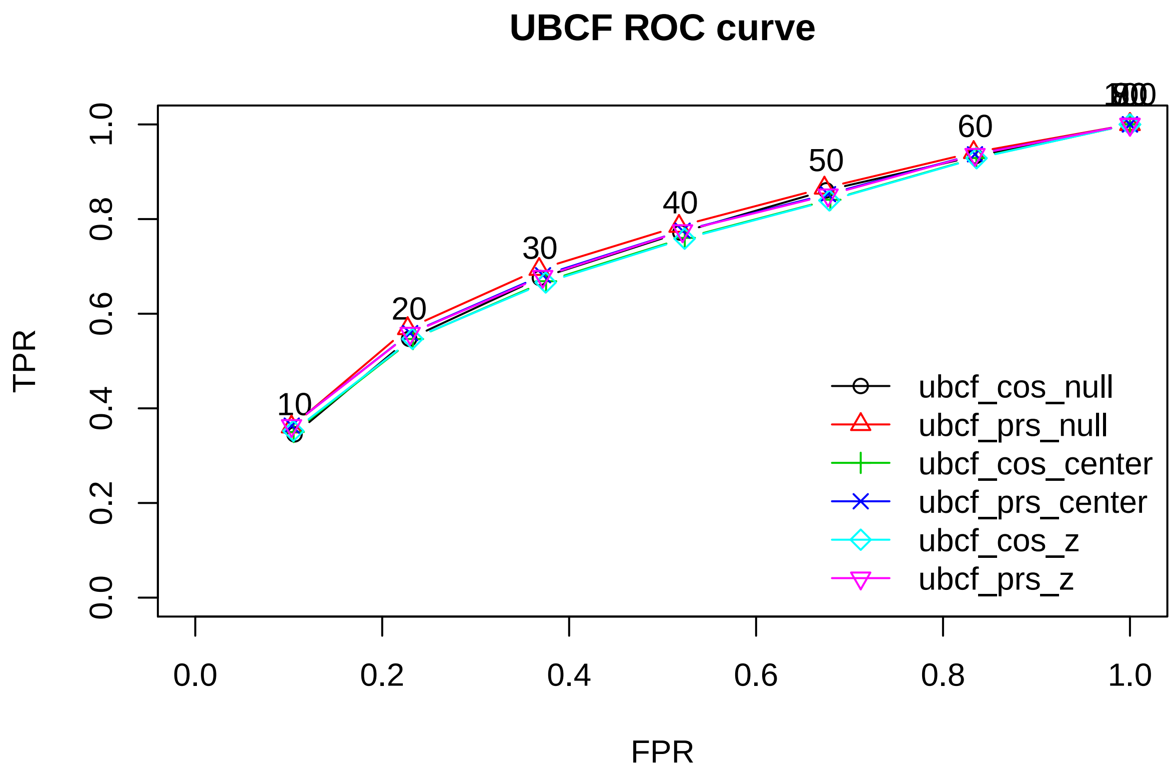
```r
ubcf_models <- list(
  ubcf_cos_null = list(name = "UBCF", param = list(method = "cosine", normalize = NULL)),
  ubcf_prs_null = list(name = "UBCF", param = list(method = "pearson", normalize = NULL)),

  ubcf_cos_center = list(name = "UBCF", param = list(method = "cosine", normalize = "center")),
  ubcf_prs_center = list(name = "UBCF", param = list(method = "pearson", normalize = "center")),

  ubcf_cos_z = list(name = "UBCF", param = list(method = "cosine", normalize = "Z-score")),
  ubcf_prs_z = list(name = "UBCF", param = list(method = "pearson", normalize = "Z-score"))
)

ubcf_eval_results <- evaluate(x = eval_set,
                              method = ubcf_models,
                              n = seq(10, 100, 10)
                              )
```

```
## UBCF run fold/sample [model time/prediction time]
##   1  [0.011sec/3.878sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.001sec/4.094sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.04sec/3.696sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.06sec/3.681sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.113sec/3.768sec]
## UBCF run fold/sample [model time/prediction time]
##   1  [0.155sec/3.921sec]
```

```r
plot(ubcf_eval_results, "prec/rec", annotate = T, main = "Precision Recall")
title("UBCF Precision-recall")
```




```
plot(ubcf_eval_results, annotate = T)
title("UBCF ROC curve")
```



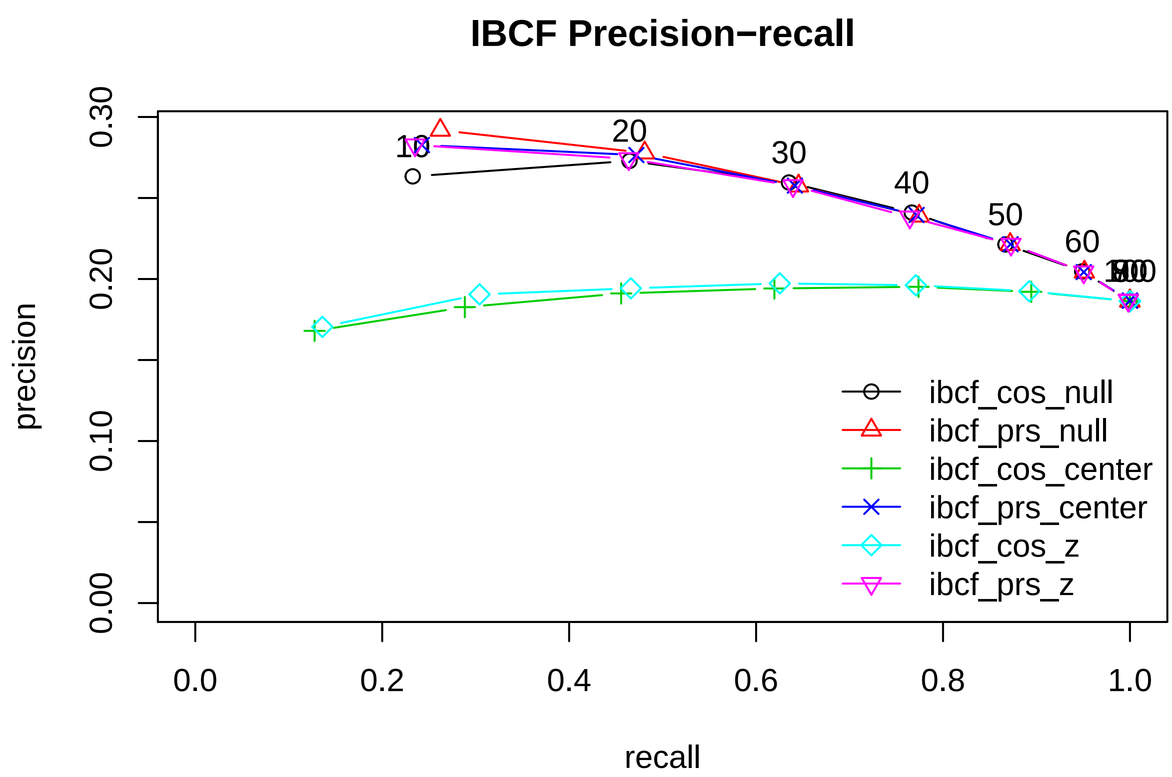
IBCF Models

```
ibcf_models <- list(
  ibcf_cos_null = list(name = "IBCF", param = list(method = "cosine", normalize = NULL)),
  ibcf_prs_null = list(name = "IBCF", param = list(method = "pearson", normalize = NULL)),

  ibcf_cos_center = list(name = "IBCF", param = list(method = "cosine", normalize = "center")),
  ibcf_prs_center = list(name = "IBCF", param = list(method = "pearson", normalize = "center")),

  ibcf_cos_z = list(name = "IBCF", param = list(method = "cosine", normalize = "Z-score")),
  ibcf_prs_z = list(name = "IBCF", param = list(method = "pearson", normalize = "Z-score"))
)

ibcf_eval_results <- evaluate(x = eval_set,
                              method = ibcf_models,
                              n = seq(10, 100, 10)
                              )
```

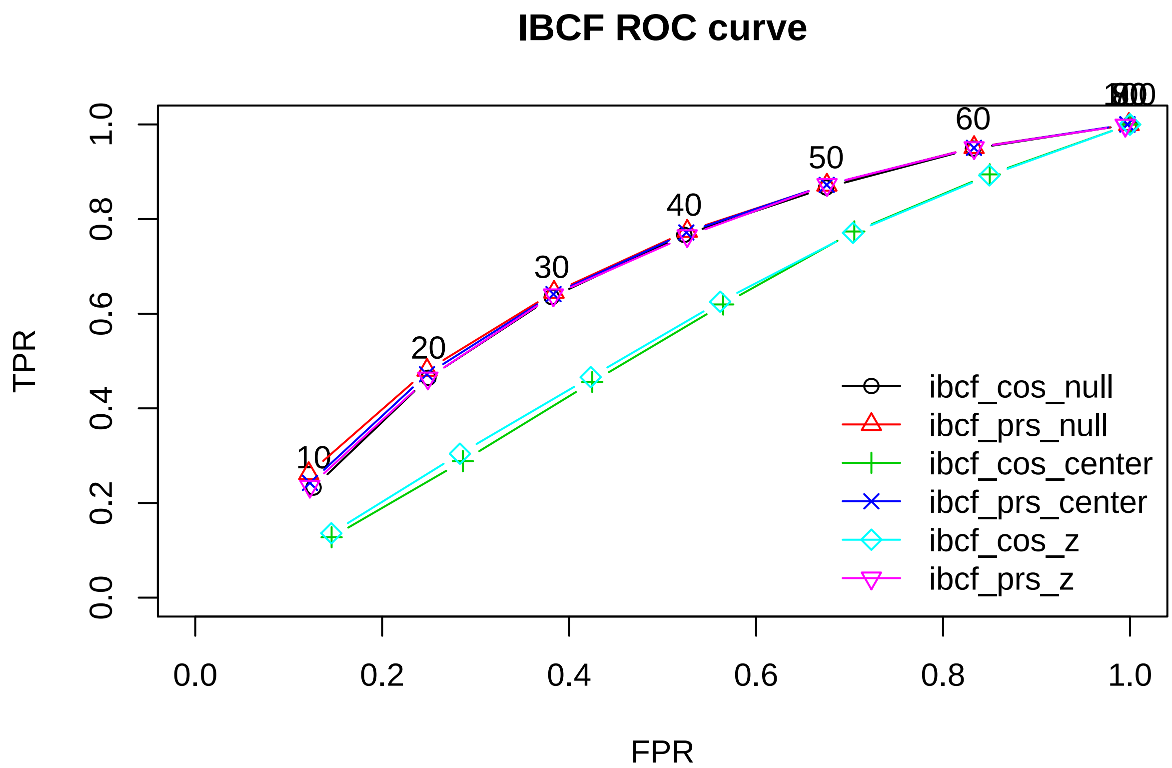
```
## IBCF run fold/sample [model time/prediction time]
##   1  [0.179sec/0.201sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.25sec/0.182sec]
## IBCF run fold/sample [model time/prediction time]
##   1  [0.191sec/0.186sec]
```

```
## IBCF run fold/sample [model time/prediction time]
##    1  [0.27sec/0.431sec]
## IBCF run fold/sample [model time/prediction time]
##    1  [0.294sec/0.195sec]
## IBCF run fold/sample [model time/prediction time]
##    1  [0.361sec/0.222sec]
```

```
plot(ibcf_eval_results, "prec/rec", annotate = T, main = "Precision Recall")
title("IBCF Precision-recall")
```




```
plot(ibcf_eval_results, annotate = T)
title("IBCF ROC curve")
```



RANDOM models

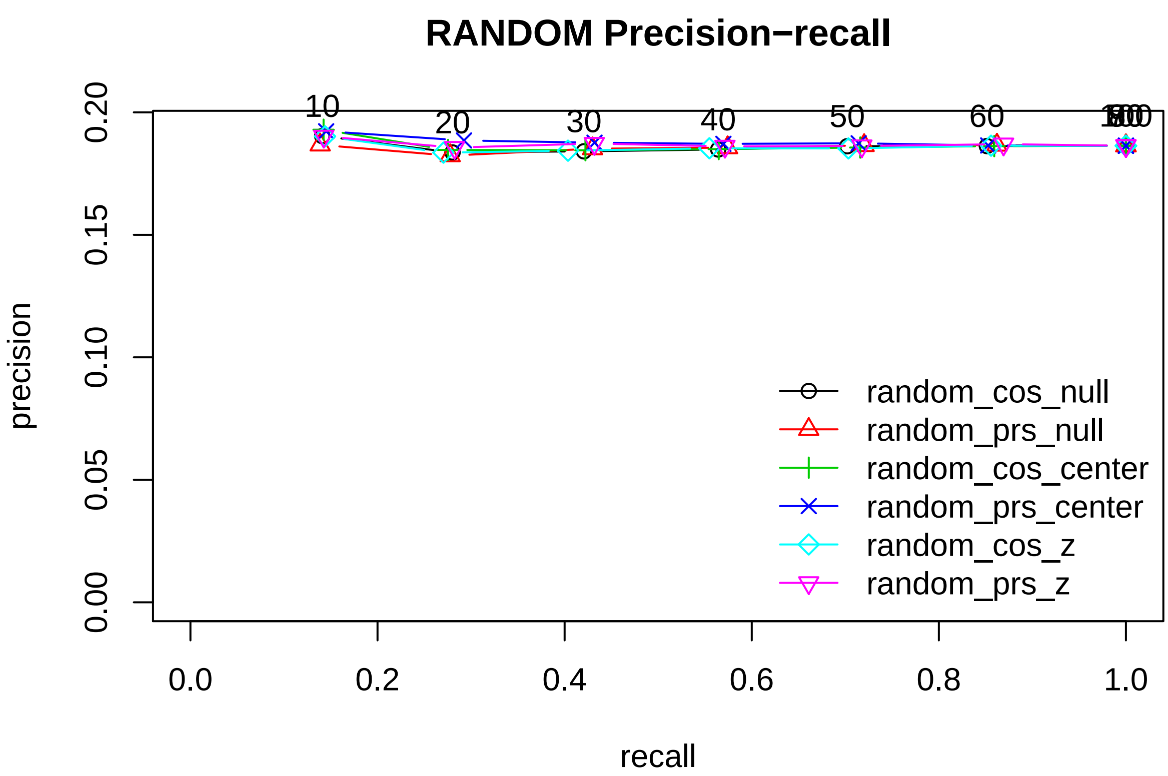
The next two models are to implement support for at least one business or user experience goal such as increased serendipity, novelty, or diversity. The hope is that the RANDOM and POPULAR models can bring serendipity and novelty to the systems.

```
random_models <- list(
  random_cos_null = list(name = "RANDOM", param = list(method = "cosine", normalize = NULL)),
  random_prs_null = list(name = "RANDOM", param = list(method = "pearson", normalize = NULL)),

  random_cos_center = list(name = "RANDOM", param = list(method = "cosine", normalize = "center")),
  random_prs_center = list(name = "RANDOM", param = list(method = "pearson", normalize = "center")),

  random_cos_z = list(name = "RANDOM", param = list(method = "cosine", normalize = "Z-score")),
  random_prs_z = list(name = "RANDOM", param = list(method = "pearson", normalize = "Z-score"))
)

random_eval_results <- evaluate(x = eval_set,
                               method = random_models,
                               n = seq(10, 100, 10)
                               )
```

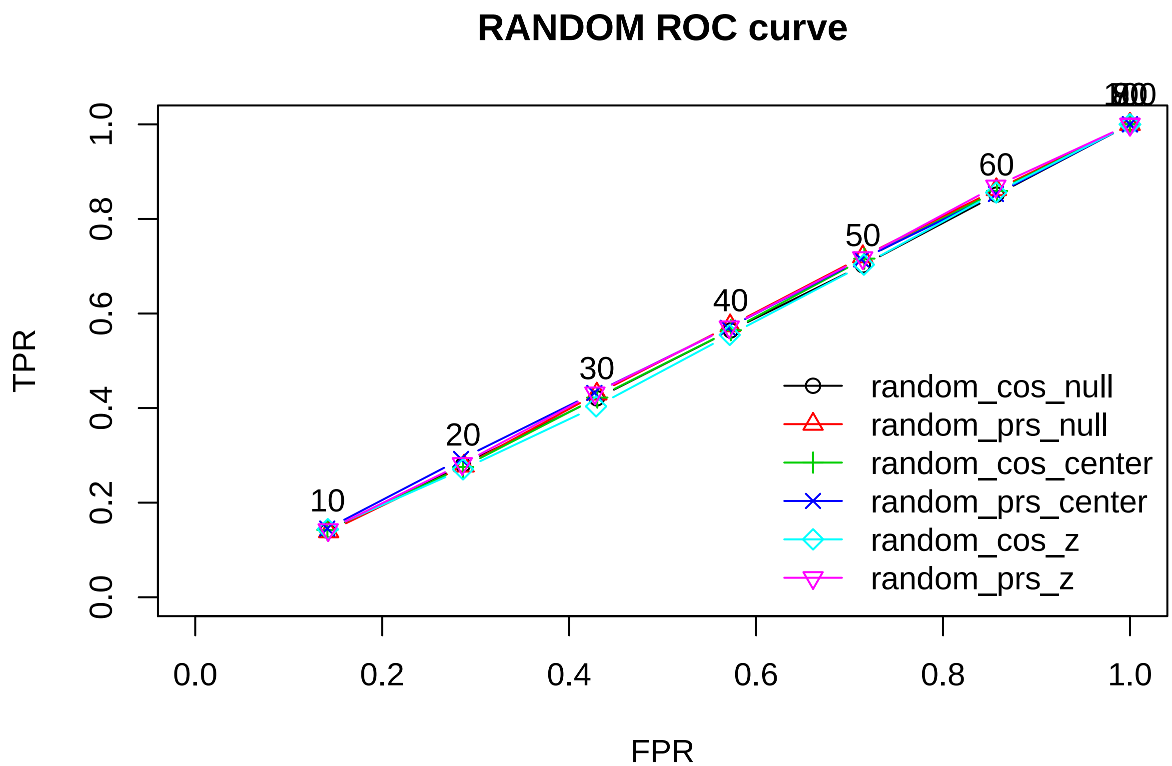
```
## RANDOM run fold/sample [model time/prediction time]
##   1  [0.004sec/0.207sec]
## RANDOM run fold/sample [model time/prediction time]
```

```
##    1  [0.058sec/0.218sec]
## RANDOM run fold/sample [model time/prediction time]
##    1  [0.003sec/0.185sec]
## RANDOM run fold/sample [model time/prediction time]
##    1  [0.003sec/0.186sec]
## RANDOM run fold/sample [model time/prediction time]
##    1  [0.003sec/0.186sec]
## RANDOM run fold/sample [model time/prediction time]
##    1  [0.003sec/0.194sec]
```

```r
plot(random_eval_results, "prec/rec", annotate = T, main = "Precision Recall")
title("RANDOM Precision-recall")
```

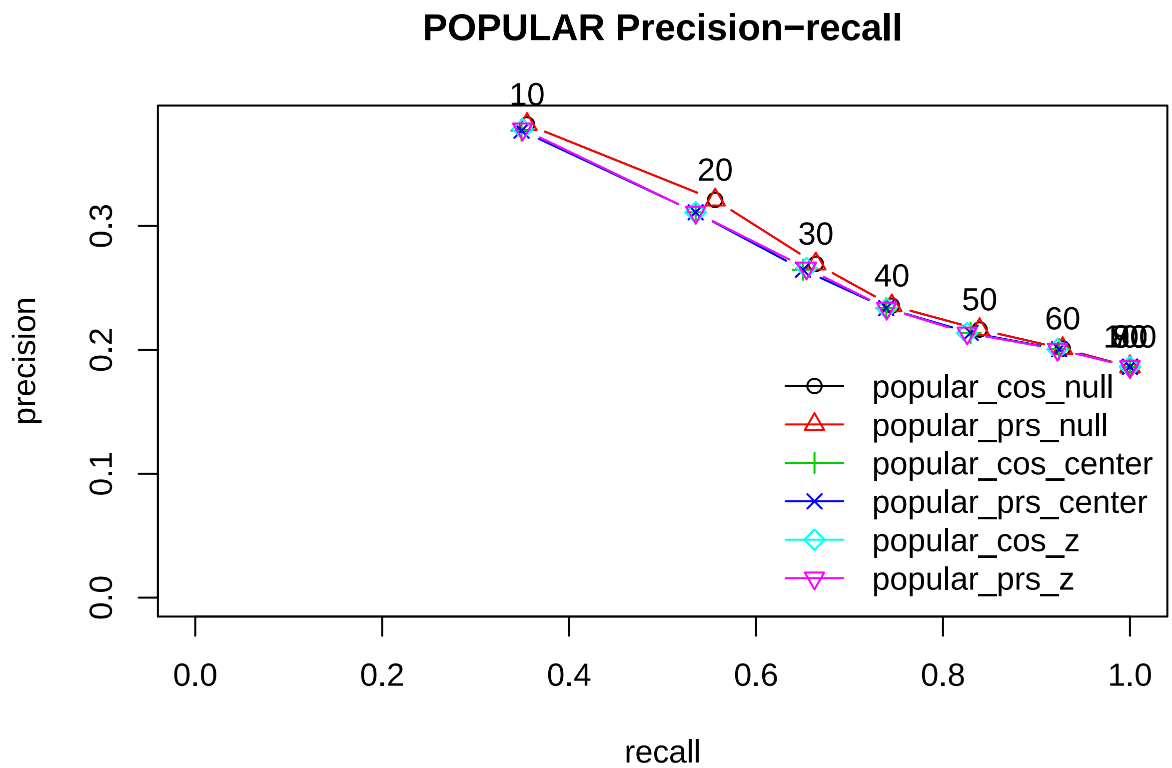



```r
plot(random_eval_results, annotate = T)
title("RANDOM ROC curve")
```

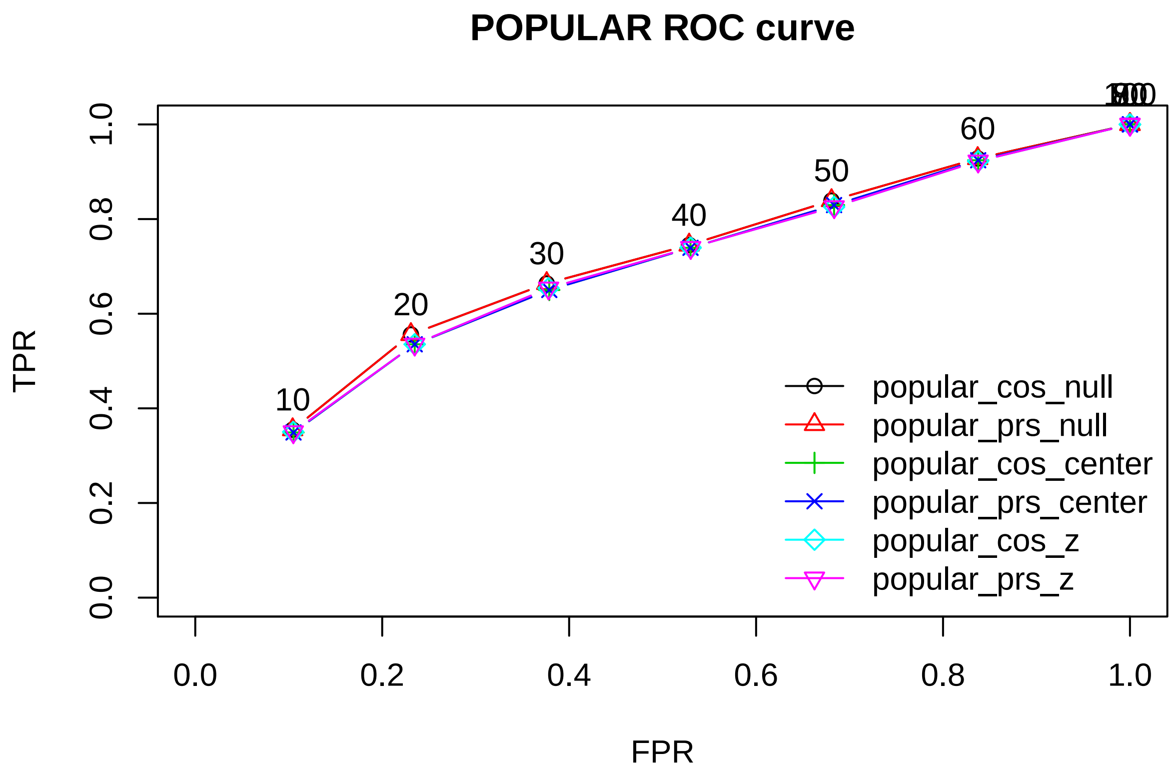



```r
popular_models <- list(
  popular_cos_null = list(name = "POPULAR", param = list(method = "cosine", normalize = NULL)),
  popular_prs_null = list(name = "POPULAR", param = list(method = "pearson", normalize = NULL)),

  popular_cos_center = list(name = "POPULAR", param = list(method = "cosine", normalize = "center")),
  popular_prs_center = list(name = "POPULAR", param = list(method = "pearson", normalize = "center")),

  popular_cos_z = list(name = "POPULAR", param = list(method = "cosine", normalize = "Z-score")),
  popular_prs_z = list(name = "POPULAR", param = list(method = "pearson", normalize = "Z-score"))
)

popular_eval_results <- evaluate(x = eval_set,
                                 method = popular_models,
                                 n = seq(10, 100, 10))
```

```r
plot(popular_eval_results, "prec/rec", annotate = T, main = "Precision Recall")
title("POPULAR Precision-recall")
```




```r
plot(popular_eval_results, annotate = T)
title("POPULAR ROC curve")
```



Model Selection

After looking at the Precision and ROC curves on the four methods, it appears that the different subsets of models were more accurate than others. Three out of the four included Pearson correlation and two of the four had a Z-score normalization. The below code chunks provide a litle more information to the selected models.

```
# Set the training, known and unknown sets
training_set <- getData(eval_set, "train")

known_set <- getData(eval_set, "known")

unknown_set <- getData(eval_set, "unknown")
```

```
ubcf_rec <- Recommender(data = training_set, method = "UBCF")

ibcf_rec <- Recommender(data = training_set, method = "IBCF")

popular_rec <- Recommender(data = training_set, method = "POPULAR")

random_rec <- Recommender(data = training_set, method = "RANDOM")
```

```
ubcf_model <- predict(ubcf_rec, known_set, type = "ratingMatrix")
```

```r
ibcf_model <- predict(ibcf_rec, known_set, type = "ratingMatrix")

popular_model <- predict(popular_rec, known_set, type = "ratingMatrix")

random_model <- predict(random_rec, known_set, type = "ratingMatrix")
```

Selected models

I relied on the ROC curver over the Precision-Recall curves since it seems like we have a fairly balanced dataset.

```r
# UBCF Pearson Z-score
ubcf_prs_z_rec <- Recommender(data = training_set, method = "UBCF", parameter = list(method = "pearson"

# IBCF Pearson Row Centering
ibcf_prs_c_rec <- Recommender(data = training_set, method = "IBCF", parameter = list(method = "pearson"

# POPULAR Cosine similarity Z-score
popular_cos_z_rec <- Recommender(data = training_set, method = "POPULAR", parameter = list(method = "co
```

```
## Available parameter (with default values):
## normalize     =  center
## aggregationRatings    =  new("standardGeneric", .Data = function (x, na.rm = FALSE, dims = 1, ...)
## aggregationPopularity     =  new("standardGeneric", .Data = function (x, na.rm = FALSE, dims = 1, ..
## verbose   =  FALSE
```

```r
# RANDOM Pearson WITHOUT normalization
random_prs_n_rec <- Recommender(data = training_set, method = "RANDOM", parameter = list(method = "pear
```

Predictions

```r
ubcf_prs_z_model <- predict(ubcf_prs_z_rec, known_set, type = "ratingMatrix")

ibcf_prs_c_model <- predict(ibcf_prs_c_rec, known_set, type = "ratingMatrix")

popular_cos_z_model <- predict(popular_cos_z_rec, known_set, type = "ratingMatrix")

random_prs_n_model <- predict(random_prs_n_rec, known_set, type = "ratingMatrix")
```

##Accuracy

The results below show before and after results with different models selected based on specific similarity normalization methods. The UBCF with Pearson Similarity with Z-score normalization was the model with the lowest error rate across all three measures.

```r
error <- rbind(
  UBCF = calcPredictionAccuracy(ubcf_model, unknown_set),
  UBCF_prs_z = calcPredictionAccuracy(ubcf_prs_z_model, unknown_set),
  IBCF = calcPredictionAccuracy(ibcf_model, unknown_set),
  IBCF_prs_c = calcPredictionAccuracy(ibcf_prs_c_model, unknown_set),
  POPULAR = calcPredictionAccuracy(popular_model, unknown_set),
```

```r
  POPULAR_cos_z = calcPredictionAccuracy(popular_cos_z_model, unknown_set),
  RANDOM = calcPredictionAccuracy(random_model, unknown_set),
  RANDOM_prs_n = calcPredictionAccuracy(random_prs_n_model, unknown_set)
)
error
```

```
##                   RMSE      MSE      MAE
## UBCF          4.388843 19.26194 3.442074
## UBCF_prs_z    4.320129 18.66351 3.353867
## IBCF          4.901839 24.02802 3.901594
## IBCF_prs_c    4.396341 19.32782 3.412049
## POPULAR       4.418263 19.52105 3.491181
## POPULAR_cos_z 4.403166 19.38787 3.448060
## RANDOM        6.297851 39.66292 4.868979
## RANDOM_prs_n  6.330309 40.07281 4.906319
```

Another item that can be tested is a hybrid recommender system that can take features from one more more recommenders on a weighted basis to obtain a little bit of user/item accuracy coupled with novelty and serentipity from the popularity and random models. There were datatype inconsistencies regarding testing the hybrid system, which is a class object in `recommenderLab`. With a little more time I could have created and evaluated that as well.

The main difference between offline and online datasets is the accuracy testing. With offline, as we used, the recommendations are tested against some "unknown" portion of test set, whereas if we were online that unknown group could be a live user being given a recommendation on the spot. The system can then learn based on users' click rates which would further improve accuracy and tie together even more interesting recommendations. It also seems judging accuracy for serendipity and novelty would be easier on a live online user since these are off-hand recommendations that might be tougher to assess on a cold offline dataset.

As shown here, one could put a bunch of model in a list and run them all, evaluate and choose a model for production. This type of method will continue to get easier with more computing power, but one must slow down and really think through what the goals of the system are, and what kind of experience you want the end user to see.

References

recommenderlab: A Framework for Developing and Testing Recommendation Algorithms

Buidling a Recommendation System with R

Package 'recommenderlab'