- Bad ways to run a user group
- Implementing Apriori Algorithm in R

Other sites

- SAS blogs
- Jobs for R-users

Testing recommender systems in R

June 10, 2012
By saurabhat

(This article was first published on **sane.a.lytics » R**, and kindly contributed to R-bloggers)

Recommender systems are pervasive. You have encountered them while buying a book on barnesandnoble, renting a movie on Netflix, listening to music on Pandora, to finding the bar visit (FourSquare). Saar for Revolution Analytics, had demonstrated how to get started with some techniques for R here.

We will build some using Michael Hahsler's excellent package – recommenderlab. But to build something we have to learn to recognize when it is good. For this reason we will talk about some metrics quickly –

– RMSE (Root Mean Squared Error) : Here we measure far were real ratings from the ones we predicted. Mathematically, we can write it out as

$$RMSE = \sqrt{\frac{\sum_{(i,j)\in\kappa}(r_{(i,j)}-\hat{r}_{(i,j)})^2}{|\kappa|}}$$

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

Here at sane.a.lytics, I will talk about when an analysis makes sense and when it doesn't. RMSE is a great metric if you are measuring how good your predicted ratings are. But if you want to know how many people clicked on your recommendation, I have a different metric for you.

– Precision/Recall/f-value/AUC: Precision tells us how good the predictions are. In other words, how many were a hit.

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

Recall tells us how many of the hits were accounted for, or the coverage of the desirable outcome.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{total relevant documents}\}|}$$

Precision and recall usually have an inverse relationship. This becomes an even bigger issue for rare issue phenomenon like recommendations. To tackle this problem, we will use f-value. This is nothing but the harmonic mean of precision and recall.

Another popular measure is AUC. This is roughly analogous. We will go ahead and use this for now for our comparisons of recommendation effectiveness.

– ARHR (Hit Rate): Karypis likes this metric.

$$ARHR = \frac{1}{\#users}\sum_{i=1}^{\#hits}\frac{1}{p_i}$$

where p is the position of the item in a ranked list.

OK, on to the fun stuff.

They are a few different ways to build a recommender system

Collaborative Filtering : If my friend Jimmy tells me that he liked the movie "Drive", I might like it too since we have similar tastes. However if Paula tells me she liked "The Notebook", I might avoid it.

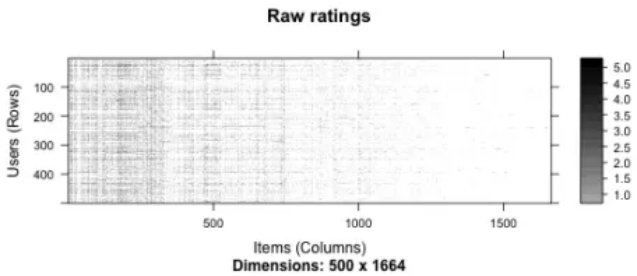
This is called UBCF (User-based collaborative filtering). Another way to think about it is that this is soft-clustering. We find Users with similar tastes (neighbourhood) and use their preferences to build yours.

Another flavour of this is IBCF (Item Based Collaborative Filtering). If I watched "Darjeeling Limited", I might be inclined to watch "The Royal Tannenbaums" but not necessarily "Die Hard". This is because the first two are more similar in the users who have watched/rated them. This is a rather simple to compute as all we need is the covariance between products to find out what this might be.

Let's compare both approaches on some real data (thanks R)

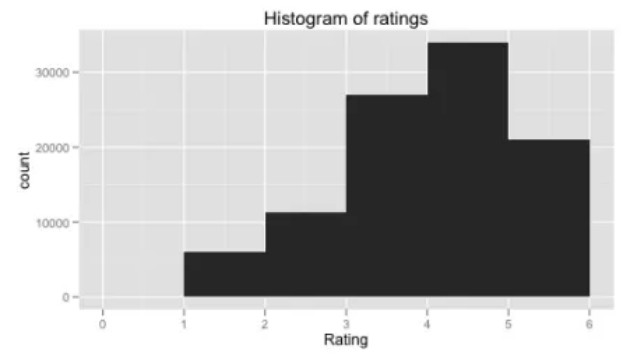
```
# Load required library
library(recommenderlab) # package being evaluated
library(ggplot2) # For plots

# Load the data we are going to work with
data(MovieLense)
MovieLense
# 943 x 1664 rating matrix of class 'realRatingMatrix' with 99

# Visualizing a sample of this
image(sample(MovieLense, 500), main = "Raw ratings")


```

recommenderlab-1-1.R hosted with by GitHub view raw



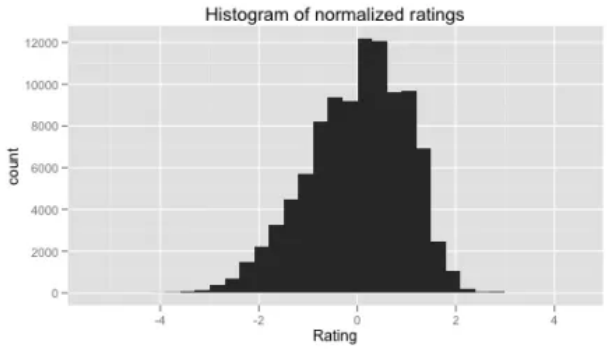

```
# Visualizing ratings
qplot(getRatings(MovieLense), binwidth = 1,
      main = "Histogram of ratings", xlab = "Rating")
summary(getRatings(MovieLense)) # Skewed to the right
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# 1.00    3.00    4.00    3.53    4.00    5.00
```

recommenderlab-1-2.R hosted with by GitHub view raw



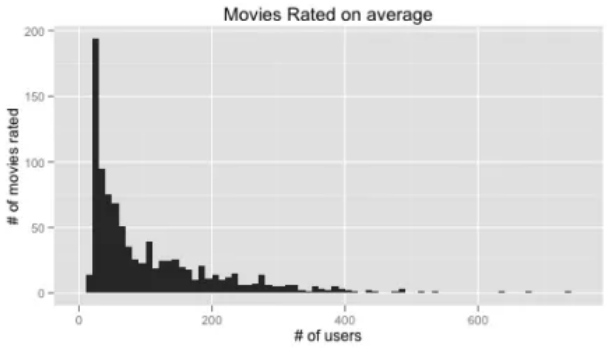

```
# How about after normalization?
qplot(getRatings(normalize(MovieLense, method = "Z-score")),
      main = "Histogram of normalized ratings", xlab = "Rating"
summary(getRatings(normalize(MovieLense, method = "Z-score")))
# Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
# -4.8520 -0.6466  0.1084  0.0000  0.7506  4.1280
```

recommenderlab-1-3.R hosted with by GitHub view raw



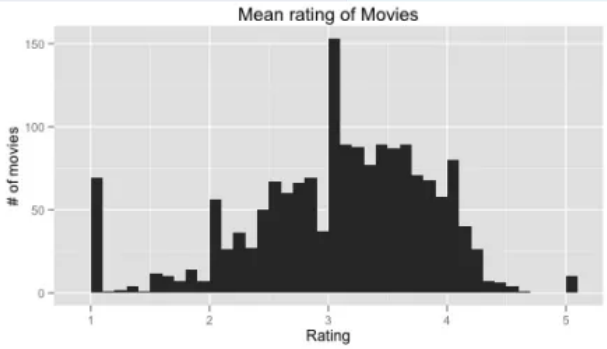

```
# How many movies did people rate on average
qplot(rowCounts(MovieLense), binwidth = 10,
      main = "Movies Rated on average",
      xlab = "# of users",
      ylab = "# of movies rated")
# Seems people get tired of rating movies at a logarithmic pace
```

recommenderlab-1-4.R hosted with by GitHub view raw




```
# What is the mean rating of each movie
qplot(colMeans(MovieLense), binwidth = .1,
      main = "Mean rating of Movies",
      xlab = "Rating",
      ylab = "# of movies")

# The big spike on 1 suggests that this could also be intepret
# In other words, some people don't want to see certain movies
# Same on 5 and on 3.
# We will give it the binary treatment later
```

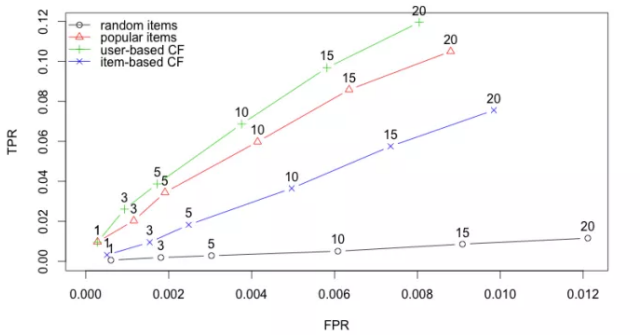
recommenderlab-1-5.R hosted with by GitHub view raw

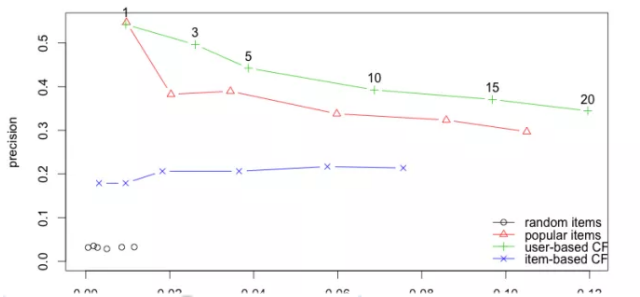



```
recommenderRegistry$get_entries(dataType = "realRatingMatrix")
# We have a few options

# Let's check some algorithms against each other
scheme <- evaluationScheme(MovieLense, method = "split", train
                           k = 1, given = 10, goodRating = 4)

scheme

```

```
algorithms <- list(
  "random items" = list(name="RANDOM", param=list(normalize =
  "popular items" = list(name="POPULAR", param=list(normalize
  "user-based CF" = list(name="UBCF", param=list(normalize = "
                                                 method="Cosin
                                                 nn=50, minRat
  "item-based CF" = list(name="IBCF2", param=list(normalize =
                                                 ))


  )

# run algorithms, predict next n movies
results <- evaluate(scheme, algorithms, n=c(1, 3, 5, 10, 15, 2

# Draw ROC curve
plot(results, annotate = 1:4, legend="topleft")

# See precision / recall
plot(results, "prec/rec", annotate=3)
```

recommenderlab-1-6.R hosted with by GitHub view raw





It seems like UBCF did better than IBCF. Then why would you use IBCF? The answer lies is when and how are you generating recommendations. UBCF saves the whole matrix and then generates the recommendation at predict by finding the closest user. IBCF saves only k closest items in the matrix and doesn't have to save everything. It is pre-calculated and predict simply reads off the closest items.

Predictably, RANDOM is the worst but perhaps surprisingly it seems, its hard to beat POPULAR. I guess we are not so different, you and I.

In the next post I will go over some other algorithms that are out there and how to use them in R. I would also recommend reading Michael's documentation on recommenderlab for more details.

Also added this to r-bloggers. Please check it out for more R goodies.

Related

Comments: 8
Partools, Recommender