对 recommenderlab 包的改进

龚识瑾 岑敏 郑宇立

2019 年 12 月 23 日

摘要

该项目旨在实现一个比 Recommenderlab 更高效，对内存更友好的的协同过滤推荐算法，并且以此重新构建推荐系统。我们将循环运算改为向量化运算，从而更快地计算相似度，同时使用矩阵的分块运算来节省内存使用。最终通过测试现实生活中的数据证明了新框架的建立具有显著效果。

目录

1 问题描述

1.1 背景介绍

1.1.1 协同过滤算法

协同过滤算法是一种较为著名和常用的推荐算法，它基于对用户历史行为数据的挖掘发现用户的喜好偏向，并预测用户可能喜好的产品进行推荐。它的主要实现由：

- 根据和你有共同喜好的人给你推荐
- 根据你喜欢的物品给你推荐相似物品
- 根据你喜欢的物品给你推荐相似物品

因此可以得出常用的协同过滤算法分为两种，基于用户的协同过滤算法 (user-based collaboratIve filtering)，以及基于物品的协同过滤算法 (item-based collaborative filtering)。

以用户的协同过滤算法为例，其具体步骤为：

1. 计算其他用户的相似度
2. 根据相似度找到与你最相似的 K 个用户
3. 在这些邻居喜欢的物品中，根据与你的相似度算出每一件物品的推荐度
4. 根据相似度推荐物品

1.1.2 Recommenderlab 包

recommenderlab 是一个用于做推荐的 R 包，提供了多种推荐算法，包括 UBCF(User Based Collaborative Filtering)，IBCF(Item Based Collaborative Filtering), Popular, Random，SVD，PCA，AR 等。recommenderlab 的核心数据结构是 RatingMatrix（评分矩阵），该矩阵记录了 user 对 item 的评分。以 RatingMatrix 为输入，利用上述算法估计用户对尚未评价 item 的评分，并依此为依据进行推荐。

1.2 问题提出

虽然 recommenderlab 包是一个广为流传的包，但在用 recommenderlab 包尝试写推荐算法的时候，存在一些问题。

1. 用 recommenderlab 包的时候运行速度很慢
2. R 语言是利用内存来跑代码的，recommenderlab 包使用时内存使用大，不能处理庞大的数据集，当数据的数量级过大时，会造成内存溢出而报错。

我们目标即是在解决以上问题的基础上重新搭建推荐系统。

2 问题解决方案

2.1 运行时间

recommenderlab 包中有很多循环语句，大大地拖慢了运行的速度。由于在 R 中利用循环做的运行的时间会比直接利用向量计算增加很多个数量级，所以我们希望将循环语句改为矩阵的乘法来减少过多的循环。在我们的改进中，最主要是优化了相似度的计算:（我们实现算法的过程中统一采用的是 Pearson 相似度来进行计算）

对于两个向量 $X=(x_1,x_2,...,x_n), Y=(y_1,y_2,...,y_n)$ 的 Pearson 相似度的公式为:

$$Similarity(X,Y)=\frac{n\sum_{i=1}^{n}x_iy_i-\sum_{i=1}^{n}x_i\sum_{i=1}^{n}y_i}{\sqrt{n\sum_{i=1}^{n}x_i^2-(\sum_{i=1}^{n}x_i)^2}\sqrt{n\sum_{i=1}^{n}y_i^2-(\sum_{i=1}^{n}y_i)^2}}$$

用户评分矩阵中，每个用户对各个物品的评分作为向量存储，我们要计算的相似度矩阵的第 ij 个元素就是第 i 个用户和第 j 个用户的 Pearson 相似度，记矩阵 $A=(X_1,X_2,...,X_k), A\in M^{n*k}$, 其中 X_i 为 n 维列向量，$i=1,2,...,l$，同理记矩阵 $B=(Y_1,Y_2,...,Y_l), B\in M^{n*l}$, 其中 Y_j 为 n 维列向量，$j=1,2,...,l$。

现在用矩阵 $Cov(A,B)$ 表示矩阵 A 的第 i 列和矩阵 B 的第 j 列的 Pearson 相似度, 则所求结果 $Cov(A,B)\in M^{k*l}$ 满足

$$\big(Cov(A,B)\big)_{ij}=Similarity(X_i,Y_j)$$

recommender 包采用循环语句对 $Cov(A,B)$ 的每个元素单独计算，所以速度缓慢，而我们将使用以下矩阵乘法公式来直接一次性求出 $Cov(A,B)$:

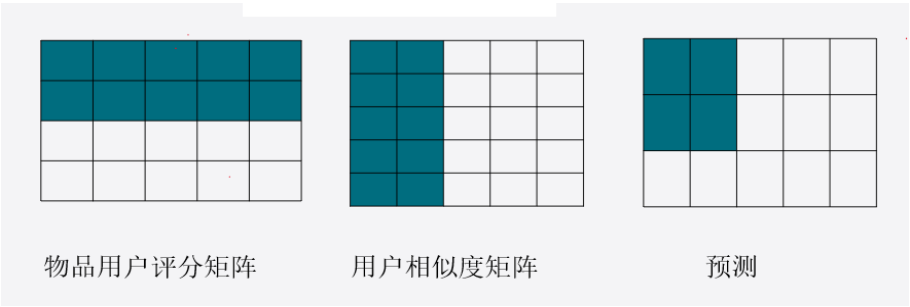
$$Cov(A,B)=\frac{A^TB}{\sqrt{[(A_2)^T(B\&G)]}*\sqrt{[(A\&G)^TB_2]}}$$

上式中 A_2 是将矩阵 A 的每个元素平方得到的矩阵，B_2 是将矩阵 B 的每个元素平方得到的矩阵,$G=(1,1,...,1)^T(1,1,...,1)$ 为全 1 矩阵，& 是逻辑算符，A&G 将 A 非零元置 1，将零元置 0，* 是将两个矩阵对应元相乘（注意不是矩阵乘法）, 利用本公式就实现了相似度矩阵的矩阵运算方法。

2.2 内存占用

以基于物品的协同过滤算法为例，若物品的数量为 n，则物品的相似度矩阵存储的数据则是 n^2 个。所以当数据集数量级很大时，建立相似度矩阵及预测矩阵时内存容易溢出。

我们提出的解决方案基于上一步的改进以及矩阵的分块运算，在我们建立了 $Cov(A,B)$ 的矩阵运算方法之后，我们就可以分块计算相似度矩阵和最终的预测矩阵。例如，取 A 为全体用户的评分矩阵，但取 B 为 A 的前两列，那我们就可以由公式得到一个 $n*2$ 的矩阵 $C=Cov(A,B)$，它存放了第 1 和第 2 位用户和其他所有用户之间的相似度，我们再取评分矩阵的前两行记为矩阵 D，那么 D 是一个 $2*n$ 的矩阵，那么 DC 就得到了预测矩阵的左上角的四个元素，图例如下：



基于这种方法，在建立相似度矩阵时就可以一次只读入相似度矩阵的一部分，从而不会出现采用 recommenderlab 包的内存溢出的情况。

3 代码实现

3.1 改进函数

在上面介绍的改进之外，我们的代码完整的重构了一个推荐系统，在实现的过程中创建了下面的函数，这一节即介绍我们包里面的各种函数。

3.1.1 cal_cor

```
# 计算皮尔森相似度
#' Calculates pearson correlations between columns of two sparse matrixs.
#'
#' @param X sparse matrix
#' @param Y sparse matrix
#' @returns correlation
#' @note Requeres {recommenderlab} package for normalization.

cal_cor <- function(X, Y){
X_not_zero <- X != 0
Y_not_zero <- Y != 0
# 正则
X <- as(normalize(as(X, "realRatingMatrix"),
  method = "center", row = FALSE), "dgCMatrix")
Y <- as(normalize(as(Y, "realRatingMatrix"),
  method = "center", row = FALSE), "dgCMatrix")
R <- Martix::crossprod(X,Y)
N <- Martix::crossprod(X^2, Y_not_zero)
M <- Martix::crossprod(X_not_zero, Y^2)
correlation <- R
correlation@x <- correlation@x/((N@x^0.5) * (M@x^0.5))
correlation
}



# 计算余弦相似度
#' Calculates cosine correlations between columns of two sparse matrixs.
#'
#' @param X sparse matrix
```

```
#' @param Y sparse matrix
#' @returns cosine correlations

cal_cos <- function(X, Y){
ones <- rep(1,nrow(X))
means <- drop(Martix::crossprod(X^2, ones)) ^ 0.5
diagonal <- Diagonal( x = means^-1 )
X <- X %*% diagonal
ones <- rep(1,nrow(Y))
means <- drop(Martix::crossprod(Y^2, ones)) ^ 0.5
diagonal <- Diagonal( x = means^-1 )
Y <- Y %*% diagonal
crossprod(X, Y)
}
```

3.1.2 find_similarities

```
# 计算相似度矩阵
#' calculate the similarity between some columns and the whole matrix
#' @param matrix
#' @param columns_to_consider
#' @param similarity_metric which kind of similarity to use
#' @param make_positive_similarities whether only accept positive similarities (logical)
#' @param k obtain 1~k largest similarities
#' @returns similarity matrix (dgCMatrix)

find_similarities <- function(matrix, columns_to_consider, similarity_metric,
make_positive_similarities, k){

selected_columns <- matrix[, columns_to_consider, drop = FALSE]
# 相似度矩阵应该计算
similarities <- similarity_metric(matrix, selected_columns)

similarities@x[similarities@x == 0] <- 0.000001
ind <- cbind(columns_to_consider, 1:length(columns_to_consider))
similarities[ind] <- 0
similarities <- drop0(similarities)

# 为真都变成正
if (make_positive_similarities) {
if (min(similarities@x) < 0) similarities@x
```

```
<- similarities@x + abs(min(similarities@x))
}

if (!is.null(k) && k < nrow(similarities) - 1) {

# 通过slam::simple_triplet_matrix, data.table找前推荐的
# 存下维度, 维度名字
dims_old <- similarities@Dim
dimnames_old <- similarities@Dimnames
similarities <- as.simple_triplet_matrix(similarities)
datatable <- data.table(similarities$i, similarities$j, similarities$v)
names(datatable) <- c("row", "column", "rating")

# 向量里面找k大的
kthMax <- function(vector, k){
if (length(vector) <= k) min(vector)
else{
sort(vector, partial = length(vector) - (k - 1))[length(vector) - (k - 1)]
}
}

kthMaxes <- datatable[, kthMax(rating, k), by = column]
names(kthMaxes) <- c("column", "kthMax")
datatable <- merge(datatable, kthMaxes, by = "column")
datatable <- datatable[datatable$rating >= datatable$kthMax, ]

similarities <- as(sparseMatrix(i = datatable$row, j = datatable$column,
x = datatable$rating, dims = dims_old, dimnames = dimnames_old), "dgCMatrix")
}

similarities
}
```

3.2 构建推荐算法所需其他函数

3.2.1 add_predictions_to_prediction_matrix

```
#' add part of the predictions to the prediction matrix
#'
#' @param predictions_matrix
#' @param part_predictions part of the predictions
#' @param predictions_matrix_indices indices to place part of the predictions
#' @returns prediction matrix with new part of the predictions

add_predictions_to_prediction_matrix <-
function(predictions_matrix, part_predictions, predictions_matrix_indices){

row_names <- as.integer(unlist(part_predictions@Dimnames[1]))
columns_names <- as.integer(unlist(part_predictions@Dimnames[2]))
row_info <- cbind(row_name = row_names, row_index = 1:length(row_names))
column_info <- cbind(column_name = columns_names,
 column_index = 1:length(columns_names))

all_indices <- predictions_matrix_indices
colnames(all_indices) <- c("row_name", "column_name")
all_indices <- merge(all_indices, row_info)
all_indices <- merge(all_indices, column_info)

predictions_matrix_indices <- all_indices[, c("row_name", "column_name")]
part_matrix_indices <- all_indices[, c("row_index", "column_index")]

if (nrow(predictions_matrix_indices) > 0) {
predictions_matrix[as.matrix(predictions_matrix_indices)] <-
part_predictions[as.matrix(part_matrix_indices)]
}

predictions_matrix
}
```

3.2.2 predict_cf

```
# 这个函数分段计算预测评分
# 数据大保护内存
#' This function implements the whole algorithm by part with protecting big memory
#'
```

```
#' @param ratings_matrix
#' @param predictions_indices the part which need predictions in rating matrix
#' @param alg_method (string) "ubcf" or "ibcf"
#' @param normalization (logical) whether normalize the rating matrix
#' @param similarity_metric which kind of similarity to use
#' @param k obtain 1~k largest similarities
#' @param make_positive_similarities (logical)
whether only accept positive similarities
#' @param rowchunk_size (integer)
the number of rows of rating_matrix considered in one iteration
#' @param columnchunk_size (integer)
the number of columns of similarity_matrix considered in one iteration
#' @returns Predictions matrix.

predict_cf <- function(ratings_matrix, predictions_indices, algorithm_method,
normalization, similarity_metric, k, make_positive_similarities,
rowchunk_size, columnchunk_size){

if (normalization) {
# 剪掉平均值来正则
if (algorithm_method == "ubcf") ratings_matrix <-
normalize(as(ratings_matrix, "realRatingMatrix"), method = "center", row = FALSE)
if (algorithm_method == "ibcf") ratings_matrix <-
normalize(as(ratings_matrix, "realRatingMatrix"), method = "center", row = TRUE)
ratings_matrix@data@x[ratings_matrix@data@x == 0] <- 0.000001
# Prevent droping zeros obtained after applying normalization.
normalization_info <- ratings_matrix@normalize
ratings_matrix <- as(ratings_matrix, "dgCMatrix")
}

predictions_matrix <- as(sparseMatrix(i = c(), j = c(), dims = ratings_matrix@Dim,
  dimnames = ratings_matrix@Dimnames), "dgCMatrix")

num_row_splits <- ceiling(nrow(ratings_matrix)/rowchunk_size)
num_column_splits <- ceiling(ncol(ratings_matrix)/columnchunk_size)

for (i in 1:num_column_splits) {
start_column <- columnchunk_size * (i - 1) + 1 # 最左的列
end_column <- columnchunk_size * i # 右列
if (ncol(ratings_matrix) < end_column) {
```

```
end_column <- ncol(ratings_matrix)
}
columns_to_consider <- intersect(start_column:end_column, predictions_indices[, 2])
if (length(columns_to_consider) == 0) next
ratings_matrix@Dimnames[[1]] <- as.character(1:nrow(ratings_matrix))
ratings_matrix@Dimnames[[2]] <- as.character(1:ncol(ratings_matrix))
similarities <- find_similarities(ratings_matrix,
  columns_to_consider, similarity_metric, make_positive_similarities, k)
for (j in 1:num_row_splits) {
start_row <- rowchunk_size * (j - 1) + 1 # 开始行
end_row <- rowchunk_size * j # 结束行
if (nrow(ratings_matrix) < end_row) {
end_row <- nrow(ratings_matrix)
}
rows_to_consider <- intersect(start_row:end_row, predictions_indices[, 1])
if (length(rows_to_consider) == 0) next
part_predictions <- calculate_predictions(ratings_matrix[rows_to_consider, ,
  drop = FALSE], similarities)
predictions_indices_to_consider <-
subset(predictions_indices,
predictions_indices[, 1] %in% rows_to_consider &
predictions_indices[, 2] %in% columns_to_consider)
predictions_matrix <- add_predictions_to_prediction_matrix(predictions_matrix,
part_predictions, predictions_indices_to_consider)
}

}

if (normalization) {
temp <- as(predictions_matrix, "realRatingMatrix")
temp@normalize <- normalization_info
predictions_matrix <- denormalize(temp)
predictions_matrix <- as(predictions_matrix, "dgCMatrix")
}

predictions_matrix
}
```

3.2.3 calculate_predictions

```
#' This function calculate the predictions
```

```
#'
#' @param rating_matrix matrix of rating (dgCMatrix)
#' @param similarity_matrix similarity matrix(dgCMatrix)
#' @returns prediction matrix

calculate_predictions <- function(ratings_matrix, similarity_matrix){

predictions <- ratings_matrix %*% similarity_matrix
# 计算归一化子
# 在有评分的地方放1
ratings_matrix@x <- rep(1, length(ratings_matrix@x))
similarity_matrix@x <- abs(similarity_matrix@x)
sum_abs_similarities <- ratings_matrix %*% similarity_matrix

predictions@x <- predictions@x / sum_abs_similarities@x
predictions
}
```

4 实例应用

4.1 数据处理

4.1.1 数据集介绍

我们采用了 342 万条中国移动用户在一段时间内阅读公众号及文章的信息。数据存放在一 txt 文件中，其每一行作为一条数据，每一条数据包含了用户 id，文章名称，文章 id 以及公众号名称。我们预期通过此数据集构建公众号的推荐系统。

4.1.2 数据预处理

首先对数据进行预处理:

1. 将 txt 文件读入到 Rstudio, 检查数据格式是否有误，剔除错误数据
2. 将数据存入数据框，使得其每列分别为用户 id，文章名称，文章 id 以及公众号名称
3. 采用以下的评分标准构建用户评分矩阵: 将用户历史所看的该公众号的文章的篇数作为用户对该公众号的评分。若用户没看过该公众号的文章，则评分为零。

code

```
lines <- readLines("test.txt", encoding = "UTF-8")
lines <- lines[-1] #去除第一行错误数据
user.ids <- c()
official.account.names <- c()
```

```
for (i in 1:length(lines)) {
words <- strsplit(lines[i], split = "\001")[[1]]
user.ids <- c(user.ids, words[1])
official.account.names <- c(official.account.names, words[5])
}

lines.df <- data.frame(cbind(user.ids, official.account.names)) #建dataframe

names(lines.df) <- c("user_id", "official_name")

user <- unique(lines.df$user_id)
user.idx <- match(lines.df$user_id, user) #为用户id建索引

official <- unique(lines.df$official_name)
official.idx <- match(lines.df$official_name, official) #为公众号索引

temp <- cbind(user.idx, official.idx)

M <- matrix(0, length(official), length(user))
for (k in 1:nrow(temp)) {
M[temp[k,2],temp[k,1]] <- M[temp[k,2],temp[k,1]] + 1
}

ratings_matrix <- as(M, "sparseMatrix") #转化为我们的包的所需的格式
```

4.2 运行程序

用我们的函数来给用户推荐公众号, 所用的电脑内存为 8g。

code

```
# ==================================  our data using "ubcf"
start <- Sys.time()
#调用evaluate_cf函数来评估,函数代码可见4.3.1
res <- evaluate_cf(ratings_matrix, number_of_folds = 10,
alg_method = "ubcf", normalization = TRUE,
similarity_metric = cal_cor, k = 300, make_positive_similarities = FALSE,
rowchunk_size = 1000, columnchunk_size = 2000)
end <- Sys.time()
print(end - start)
print(paste("RMSE: ", res[[2]]))
```

```
# ================================== our data using "ibcf"
# nearly the same with "ubcf" except for we transpose the rating_matrix
ratings_matrix <- t(as(M, "sparseMatrix"))
start <- Sys.time()
res <- evaluate_cf(ratings_matrix, number_of_folds = 10,
alg_method = "ubcf", normalization = TRUE,
similarity_metric = cal_cor, k = 300, make_positive_similarities = FALSE,
rowchunk_size = 1000, columnchunk_size = 2000)
end <- Sys.time()
print(end - start)
print(paste("RMSE: ", res[[2]]))
```

4.3 评估

4.3.1 评估方法

我们使用 10 折的 cross-validation 来评估结果。具体为分别计算 recommenderlab 和我们的改进方法的运行时间及 RMSE（标准误差）。部分代码已列在上一个部分，用到了 evaluate_cf 函数的代码如下：

```
evaluate_cf <- function(ratings_matrix, number_of_folds, ...) {
  #函数内部创建rmse计算函数
  rmse_function <- function(predicted_values, actual_values){
    differences <- predicted_values - actual_values
    sqrt ( (1/length(differences)) * sum(differences^2) )
  }
  positions_non_zero_values <- as.data.frame(which(ratings_matrix != 0, arr.ind = T))
  names(positions_non_zero_values) <- c("row", "column")
  size <- nrow(positions_non_zero_values)
  fold_train_indices <- createFolds(y = 1:size, k = number_of_folds, list = TRUE, returnTrain =
TRUE)

  # 建立空矩阵存放预测结果
  all_predictions <- as(sparseMatrix(i = c(), j = c(), dims = ratings_matrix@Dim, dimnames =
ratings_matrix@Dimnames), "dgCMatrix")

  for(i in 1:number_of_folds){
    print(paste("iteration", i))
    # 创建test和train set
    train_index <- fold_train_indices[[i]]
    test_index <- setdiff(1:size, train_index)
    train_subset <- as.matrix(positions_non_zero_values[train_index, ])
    test_subset <- as.matrix(positions_non_zero_values[test_index, ])

    train_matrix <- ratings_matrix
    train_matrix[test_subset] <- 0
    train_matrix <- drop0(train_matrix)

    predictions_matrix <- predict_cf(train_matrix, test_subset, ...)
    all_predictions[as.matrix(test_subset)] <- predictions_matrix[as.matrix(test_subset)]
  }
  # 处理超过范围的值
  all_predictions@x[all_predictions@x > 5] <- 5
  all_predictions@x[all_predictions@x < 1] <- 1

  prediction_indices <- as.matrix(which(all_predictions != 0, arr.ind = T))
  actual_values <- ratings_matrix[prediction_indices]
  predicted_values <- all_predictions[prediction_indices]
  rmse <- rmse_function(predicted_values, actual_values)

  list(all_predictions, rmse)
}
```

4.3.2 评估结果

时间比较 在不同的数据集规模条件 IBCF 和 UBCF 运行时间如表 1 和表 2 所示。

可以看到我们的方法在时间上比用 recommenderlab 的方法要快了两到三个数量级。

另一方面，当我们使用的数据集数据量在 25 万条左右时，recommenderlab 的运行速度已经非常缓慢了，在处理 100 万或者更多数据的情况下，recommenderlab 会报错内存溢出，而我们的改进方法仍然能在几十分钟之内完成整个算法，这说明我们显著的提升了算法的效率并且防止了处理大规模数据时的内存溢出。

方法 / 数量	recommenderlab	改进之后的方法
5 万条	15.0595	0.0523
10 万条	45.6724	0.0691
25 万条	>180	1.2981
100 万条	内存溢出	14.9879
342 万条	内存溢出	33.7796

表 1: IBCF 下两种方法的时间/mins

方法 / 数量	recommenderlab	改进之后的方法
5 万条	17.9386	0.0689
10 万条	47.3827	0.0444
25 万条	>180	1.5625
100 万条	内存溢出	11.2056
342 万条	内存溢出	19.0865

表 2: UBCF 下两种方法的时间/mins

标准误差比较 同样的，标准误差如表 3 和表 4 所示。

方法 / 数量	recommenderlab	改进之后的方法
5 万条	14.3716	12.6765
10 万条	11.2491	12.2847
25 万条	暂无	2.3618
100 万条	内存溢出	5.8931
342 万条	内存溢出	13.4584

表 3: IBCF 下两种方法的 RMSE

方法 / 数量	recommenderlab	改进之后的方法
5 万条	15.6765	13.0231
10 万条	13.9157	12.6478
25 万条	暂无	2.5275
100 万条	内存溢出	6.3875
342 万条	内存溢出	15.0356

表 4: UBCF 下两种方法的 RMSE

我们的标准误差也在正常范围之内，说明推荐的效果依然能够得到保证。

5 打包

我们为函数添加了注释以及帮助文档，并将整个项目打包成 Rpackage，名为 rCF。

5.1 Description

```
Package: racf
Type: Package
Version: 0.1.0
Date: 2019-11-30
Title: An improved method for Developing and Testing CF Recommender Algorithms
Author: YL Zheng, SJ Gong, M Cen.
Maintainer: YL Zheng <1194789973@qq.com>
Description: Improves the UBCF, IBCF algorithm for developing recommender system provided by packa
License: GPL-2
Depends: R (>= 3.2.0), Matrix, slam, data.table, recommenderlab, lattice, arules, caret
Imports: ggplot2
BugReports: 1194789973@qq.com
Encoding: UTF-8
RoxygenNote: 7.0.2
```

5.2 载入与使用

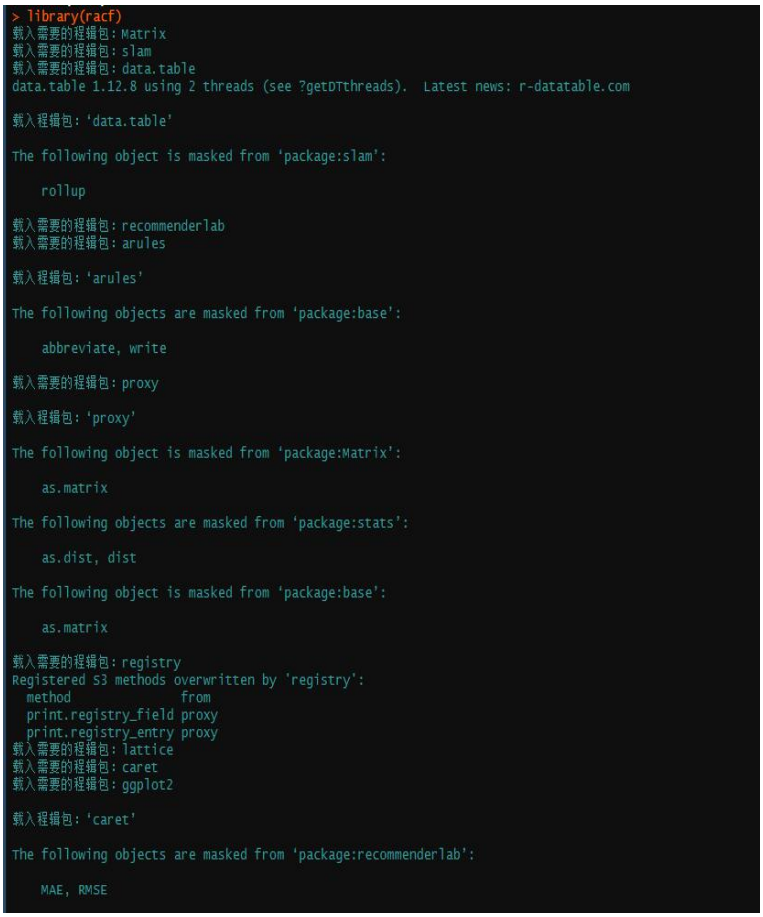
5.2.1 install

从磁盘 install 打包好的 racf 包

```
> install.packages("C:/Users/hp-pc/Desktop/R/code/racf_0.1.0.tar.gz", repos = NULL, type
Installing package into 'C:/Users/hp-pc/Documents/R/win-library/3.6'
(as 'lib' is unspecified)
* installing *source* package 'racf' ...
** using staged installation
** R
** byte-compile and prepare package for lazy loading
** help
*** installing help indices
  converting help for package 'racf'
    finding HTML links ... 好了
    add_predictions_to_prediction_matrix    html
    cal_cor                                 html
    cal_cos                                 html
    calculate_predictions                   html
    evaluate_cf                             html
    find_similarities                       html
    predict_cf                              html
** building package indices
** testing if installed package can be loaded from temporary location
** testing if installed package can be loaded from final location
** testing if installed package keeps a record of temporary installation path
* DONE (racf)
>
```

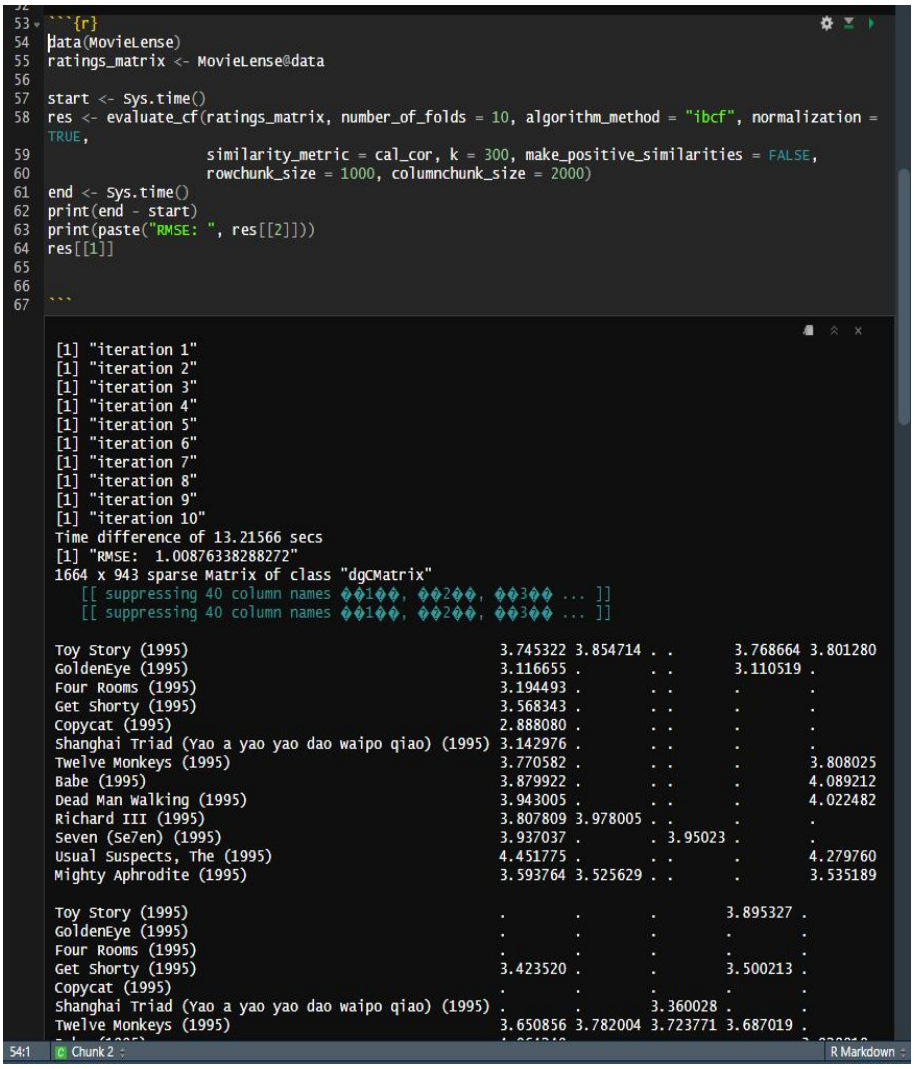
5.2.2 library

用 library 载入包，同时包含在 dependencies 里面的包都被同时载入。



5.2.3 test

使用 recommenderlab 的 data 来作为 test, 结果如下图所示



```{r}
data(MovieLense)
ratings_matrix <- MovieLense@data

start <- Sys.time()
res <- evaluate_cf(ratings_matrix, number_of_folds = 10, algorithm_method = "ibcf", normalization = TRUE,
                   similarity_metric = cal_cor, k = 300, make_positive_similarities = FALSE,
                   rowchunk_size = 1000, columnchunk_size = 2000)
end <- Sys.time()
print(end - start)
print(paste("RMSE: ", res[[2]]))
res[[1]]


```

```
[1] "iteration 1"
[1] "iteration 2"
[1] "iteration 3"
[1] "iteration 4"
[1] "iteration 5"
[1] "iteration 6"
[1] "iteration 7"
[1] "iteration 8"
[1] "iteration 9"
[1] "iteration 10"
Time difference of 13.21566 secs
[1] "RMSE:  1.00876338288272"
1664 x 943 sparse Matrix of class "dgCMatrix"
   [[ suppressing 40 column names ??1??, ??2??, ??3?? ... ]]
   [[ suppressing 40 column names ??1??, ??2??, ??3?? ... ]]

Toy Story (1995)                                    3.745322 3.854714 . .       3.768664 3.801280
GoldenEye (1995)                                    3.116655 .        . .       3.110519 .
Four Rooms (1995)                                   3.194493 .        . .       .        .
Get Shorty (1995)                                   3.568343 .        . .       .        .
Copycat (1995)                                      2.888080 .        . .       .        .
Shanghai Triad (Yao a yao yao dao waipo qiao) (1995) 3.142976 .        . .       .        .
Twelve Monkeys (1995)                               3.770582 .        . .       .        3.808025
Babe (1995)                                         3.879922 .        . .       .        4.089212
Dead Man Walking (1995)                             3.943005 .        . .       .        4.022482
Richard III (1995)                                  3.807809 3.978005 . .       .        .
Seven (Se7en) (1995)                                3.937037 .        . 3.95023 .        .
Usual Suspects, The (1995)                          4.451775 .        . .       .        4.279760
Mighty Aphrodite (1995)                             3.593764 3.525629 . .       .        3.535189

Toy Story (1995)                                    .        .        .        3.895327 .
GoldenEye (1995)                                    .        .        .        .        .
Four Rooms (1995)                                   .        .        .        .        .
Get Shorty (1995)                                   3.423520 .        .        3.500213 .
Copycat (1995)                                      .        .        .        .        .
Shanghai Triad (Yao a yao yao dao waipo qiao) (1995) .       .        3.360028 .        .
Twelve Monkeys (1995)                               3.650856 3.782004 3.723771 3.687019 .
```

5.3 函数帮助文档

利用 roxygen2 为每个函数加了帮助文档，下图是函数 predict_cf 的部分。

predict_cf {racf} R Documentation

This function implements the whole algorithm by part with protecting big memory

Description

This function implements the whole algorithm by part with protecting big memory

Usage

```
predict_cf(
  ratings_matrix,
  predictions_indices,
  algorithm_method,
  normalization,
  similarity_metric,
  k,
  make_positive_similarities,
  rowchunk_size,
  columnchunk_size
)
```

Arguments

predictions_indices	the part which need predictions in rating matrix
normalization	(logical) whether normalize the rating matrix
similarity_metric	which kind of similarity to use
k	obtain 1~k largest similarities
make_positive_similarities	(logical) whether only accept positive similarities
rowchunk_size	(integer) the number of rows considered in one iteration (rating matrix)
columnchunk_size	(integer) the number of columns considered in one iteration(similarity matrix)
algrithm_method	(string) "ubcf" or "ibcf"

Value

Predictions matrix.