上海对外经贸大学

2019 -2020 第 2 学期

《统计计算》课 程 论 文

论文名称：	基于迭代 SVD 的矩阵填充算法研究
学　　院：	统计与信息学院
专　　业：	统计学（经济）
学　　号：	19331008
学生姓名：	潘科良
课程教师：	郝程程
课程编号：	3.407.021.2-7636

2020　　年　6　月

目 录

摘　要

本文将使用基于迭代随机 SVD 算法的矩阵填充算法对 MovieLens 数据集构建电影推荐系统，MovieLens 数据集是一组由 MovieLens 用户提供的电影评分数据. 数据包括电影评分、电影元数据（风格类型和年代）以及关于用户的人口统计学数据（年龄、邮编、性别和职业等）。同时，本文还会介绍随机 SVD 的实现步骤，矩阵填充算法的实现以及对程序运行如何优化。在数据集巨大的稀疏矩阵下，我们还会根据不同的秩 r 来探索训练集和测试集上的 RMSE 的变化情况。另外，我们还会对数据集进行中心化或正则化，以及用其他的矩阵填充方法，以此来 RMSE 是否会因此而优化，在本文的最后，写好的程序将会被用到更大的数据集上，以此来实验算法表现是否一致。

关键词：随机 SVD，矩阵填充，稀疏矩阵，MovieLens

ABSTRACT

This paper will use matrix completion algorithm based on iterative randomized SVD algorithm to build movie recommendation system for MovieLens data set, which is a set of movie scoring data provided by MovieLens users. The data includes movie scoring, movie metadata (genres and age) and demographic data (age, postcode, gender and occupation) about users. At the same time, this paper also introduces the implementation steps of random SVD, the implementation of matrix completion algorithm and how to optimize the coding. Under the large sparse matrix of data set, we will also explore the change of RMSE on training set and test set according to different rank r. In addition, we will also centralize or regularize the dataset, and use other matrix filling methods to determine whether RMSE will be optimized. At the end of this paper, the written program will be used on a larger dataset to test whether the algorithm is uniform.

Key Words: randomized SVD, matrix completion, MovieLens, sparse matrix

基于迭代 SVD 的矩阵填充算法研究

1．引言

互联网的产生为人们获取知识提供了极大的便利，然而在这个信息爆炸的时代，大量的数据和无数的垃圾信息充斥在人们周围，用户想要的是最适合他们的信息，而不是所有信息，这个时候，个性化推荐就应运而生了。协同过滤是个性化推荐的一个核心技术，它被广泛应用于新闻，音乐，电影，书籍等推荐系统中如亚马逊，网易云音乐，MovieLens 等。当你在亚马逊购物时，亚马逊会根据你的购买历史，搜索历史，为你推荐你可能喜欢的商品；当你使用网易云音乐时，它会根据你以前点赞过的音乐，为你推荐你可能喜欢的音乐；当你不知道该看什么电影时，MovieLens 会根据你对看过的电影的打分，为你推荐你可能感兴趣的电影。

而在协同过滤，推荐系统这些机器学习领域中，这些数据通常以低秩矩阵的形式进行存储。而且这些数据矩阵中通常是不完整的，因为用户们只可能对某些商品作出评价，而不可能对所有商品给出凭借，从而造成这些数据矩阵中会存在一些缺失元素，从而引出了矩阵填充的问题[1]。

矩阵填充就是根据矩阵中观测到的部分的元素去估计缺失元素的值，从而恢复整个矩阵。越来越多的人把关注的目光投向如何从非常有限的信息中恢复出一个未知的低秩或者近似低秩的矩阵。Laurent 在 Matrix completion problems[2]中详细大量的接受了各种矩阵的基础知识以及矩阵填充方法。

一个比较典型的例子就是根据一个数据矩阵的采样来恢复整个数据矩阵。例如在推荐系统中，MovieLens 就是一个非常典型的数据集。MovieLens 是一个电影推荐系统，在 MovieLens 中用户会对数据库中他们观看过的那些电影进行打分，因为用户只对很少的电影进行了打分，所以我们会想要知道他们对为打分的电影的偏好。这个时候，我们可以把每一行用来存储一个用户对他看过的电影的打分，每一列作为一部电影的不同用户对其给出的打分，构建矩阵，应用矩阵填充技术，根据已有的打分，对用户对没有看过的电影的打分进行预测，预测他们对这些未打分电影的偏好，从而推荐那些可以他们可能感兴趣的

电影。

要想做到预测偏好，从数学的角度来说，这就意味着这些数据矩阵满足低秩的结构，也就是说矩阵中的数据分布在一个低维度的线性子空间中，基于这个假设，矩阵填充可以以低秩矩阵进行建模，这就是我们这篇文章中所关注的问题----低秩矩阵填充。

低秩矩阵填充建模的一个直接的方法就是将其转化为以秩的最小化作为一个正则项的优化问题。不幸的是这个优化问题是一个 NP Hard 的问题。Candes 等人[3]证明了核范数（奇异值的和）是秩的最小化的一个有效的凸松弛，所以这个优化问题可以松弛成一个核范数的最小化优化问题。

同时，基于将核范数作为正规项的假设，很多学者成功地提出了很多有效的方法来解决矩阵填充的最优化问题。Mazumder 用迭代算法来取得所得到的矩阵与观测值之间有一定误差的局部最优解[4]，但是该算法需要通过迭代求解，在迭代过程中需要对数据矩阵做奇异值分解，由于奇异值分解的计算复杂度和矩阵规模呈线性关系，所以当数据矩阵规模较大时，这些算法的速度都极大的受限于奇异值分解。Nathan Halko 等人[5]讨论了如何用随机算法做主成分分析，其中用到了随机 SVD，在失去些许精度的情况，有效的提高了程序运行的效率。

本文也基于随机 SVD 来对矩阵填充问题进行探讨，并用实验分析得出一些结论。

2．本文算法

2.1 迭代 SVD

具体来说，对于$Z: m \times n$，令$\Omega \subset \{1, \ldots, m\} \times \{1, \ldots, n\}$表示$Z$中观测到的元素的索引的集合，给定这些观测值，一个自然的思路是可对应寻找Z的最低秩矩阵$\hat{Z}$，即

$$\hat{Z} = \arg\min_{M \in R^{m \times n}} rank(M), \text{其约束为 } m_{ij} = z_{ij}, (i,j) \in \Omega$$

但含缺失数据的最低秩问题计算非常困难，一般无法求解.

因此，更常见的求解方法是允许所得到的M矩阵与观测值之间有一定误差，即

$$\hat{Z}_r = \arg\min_{rank(M)\le r} \sum^{(i,j)\in\Omega} (z_{ij} - m_{ij})^2.$$

该问题是非凸优化，通常得不到最优解，但可以采用迭代算法来得到局部最优解[2]. 例如

1. 通过对Z进行随机填充，初始化$\hat{Z}$;

2. 通过计算$\hat{Z}$的 r 秩 SVD 求解M:

$$\hat{Z} = UDV^T, \quad \hat{M} \leftarrow U_r DV^T;$$

3. 基于$\hat{M}$对Z的缺失部分进行填充:

$$\hat{z}_{ij} \leftarrow \hat{m}_{ij}, (i,j) \notin \Omega;$$

4. 重复第 2-3 步，直至算法收敛.

2.2 随机 SVD

输入: m 行 n 列的初始矩阵 A，奇异值个数 k，过采样参数 p，要求满足 k+p<=min(m，n)

输出: A 的 SVD 分解结果，分别为左奇异向量 UA ，奇异值矩阵 Σ A，以及右奇异向量 VA

1.构建一个 n*(k+p)维的高斯随机矩阵 Ω

2.进行矩阵乘积运算 Y=A Ω

3.利用 QR 分解获得 Y 的正交基 Q=qr(Y)

4.构建低维矩阵 B=Q^TA

5.矩阵 B 的 SVD 分解，[UB, Σ B,VB]=SVD(B)

6.用 Q 更新左奇异向量，UB=QUB

7.得到最终结果 UA=UB(:,1:k), Σ A= Σ B(1:k,1:k),VA=VB(:,1:k)

3．实验分析

3.1 数据集

表 1　MovieLens 数据集

数据集	行数	列数	评分数
MovieLens lastest small	671	9067	10^5
Movielens1M	6040	3706	10^6

资料来源：MovieLens

MovieLens 数据集中的数据采集于 MovieLens 网站。它是最早出现的推荐系统。MovieLens 中包含了用户对已经看过的电影的评分。它有两个大小规模不同的数据集：MovieLens lastest small 和 Movielens1M，其中 MovieLens lastest small 存在的评分是 0.5，1，1.5，…，4.5，5。而 Movielens1M 的评分则为 1，2，3，4，5。两个数据集的大小详见表 1，其观测到的数据的大小约占矩阵总大小的 1.6%至 4.4%。

3.2 实验结果

我们将 MovieLens lastest small 数据集随机抽取 75%左右的评分为训练集, 25%为测试集。在实验中，我们用均方根误差 RMSE 来评判结果的好坏，均方根误差的定义如下：

$$RMSE = \sqrt{\sum_{(i,j)\in\Omega} \left(Z_{ij} - M_{ij}\right)^2 / |\Omega|}$$

其中，Ω表示所用数据集，而|Ω|表示所用数据集所观测到的元素的个数。M_{ij}是通过低秩矩阵填充算法所填充后的矩阵的元素，而Z_{ij}是原矩阵所观测到的真实的值。RMSE 可以用来评判算法填充之后的矩阵和真实矩阵之间的差异，RMSE 越小，算法的效果越好，准确率越高。

本次实验的算法的迭代终止条件是

$$\sum_{(i,j)\notin\Omega} (M_{ij}^{k+1} - M_{ij}^{k})^2 < \varepsilon$$

其中M_{ij}^{k}表示第 k 次迭代之后填充得到的矩阵。

我们将实验结果以两种形式呈现出来：图片和表格。图片的横轴代表着秩 r，纵轴代表着 RMSE，我们可以通过图片来查看当秩不同时 RMSE 如何变化，同时也能看到随着代码的优化，运行速度的大幅提升。

表 2　代码不同时的运行速度

不同类型的代码	单次迭代的运行时间
1.用双层循环遍历矩阵	60 秒
2.用矩阵运算来获得结果	6 秒
3.矩阵运算+随机 SVD	1 秒

资料来源：实验所得

其中第一代是通过双层循环遍历矩阵求的 RMSE，速度可谓是极慢，根本无法满足快速任务的需求，在通过我们用矩阵运算来求 RMSE 之后，除了代码速度大幅变快外，代码简洁性也大幅提高。在加上随机 SVD 的运算后，单次迭代时间已经缩短为 1 秒，同时，对其中的参数进行微调，可以在获得极快运行速度的同时还能使得 RMSE 几乎不变。

实验中填充的方法有多种，通过层层筛选，最终发现，填充值为矩阵的观察值的均值 RMSE 最小，因此我们选择矩阵观察值的均值作为初始的填充值。

图 1，图 2，图 3，图 4 分别给出了初始值为矩阵均值、初始值选好后对行或列中心化、用 softimpute[6]进行初始值填充以及用现有的算法对 MovieLens1M 进行分析的 RMSE 图。

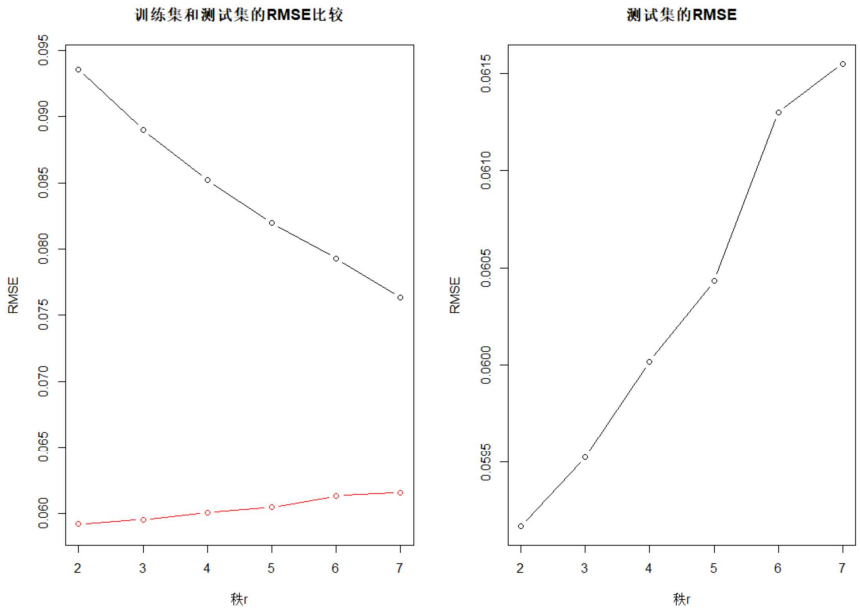

图 1　使用随机 SVD 之后，用平均值填充矩阵填充结果

算法在使用随机 SVD 并迭代 30 次之后达到稳定，同时由图一可知，秩在 2 的时候，测试集的 RMSE 是最小的，随后 RMSE 则不断增大，而训练集的 RMSE 是不断减小的，说明超过 2 之后算法容易出现过拟合。

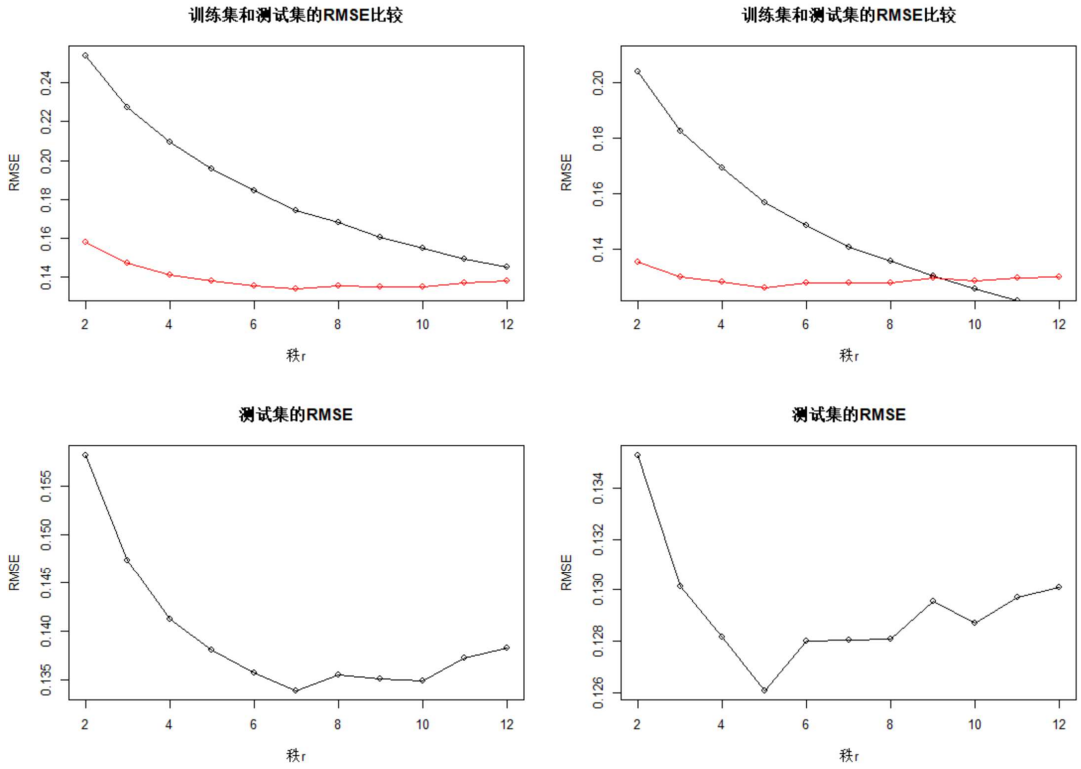

图 2　左列为使用列中心化的 RMSE，右列为使用行中心化的 RMSE

左列为列中心化后的 RMSE 比较，我们可以看到训练集的 RMSE 一如既往

的不停下降，而测试集的 RMSE 在秩为 7 的时候达到最低点，然后震荡上升。说明秩为 7 时算法达到最佳。

右列为行中心化后的 RMSE 比较，我们可以看到在秩为 5 的时候测试集的 RMSE 达到了最小。

同时我们也发现了在中心化之后，RMSE 明显比没有中心化的 RMSE 提升很多，因此我们可以认为，以 RMSE 为参照的话，中心化是没有必要的。

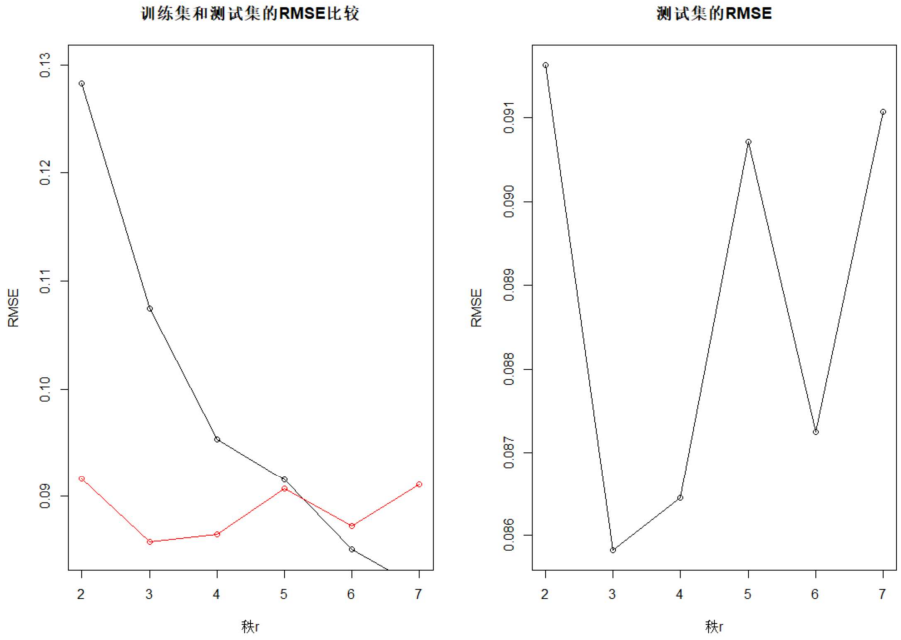

图 3　使用 softimpute 进行填充的 RMSE

使用 softimpute 之后，测试集的 RMSE 变得有些震荡，但是在秩为 3 的时候取得最小值，之后便是震荡上升，同时我们也看到，使用 softimpute 是没有单纯的使用均值填充的效果要好的。

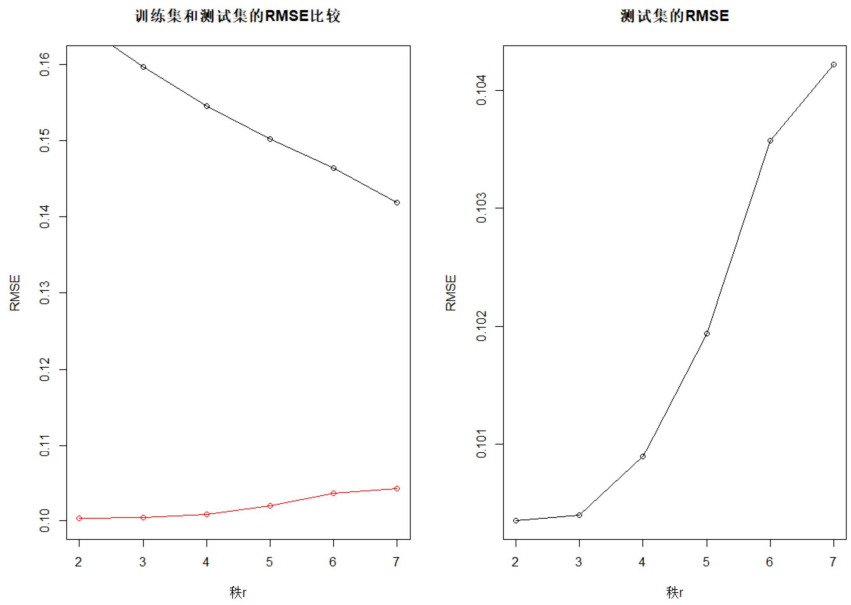

图 4　用随机 SVD，均值填充的方式来运行 1M 的数据

从图 4 我们可以看出，测试集的 RMSE 的秩在 2 的时候是最小的，和之前的小数据集类似，因此我们可以判定该算法在这两个数据集上的效果是稳定的。

4．总结与展望

我们可以看到，随着参数的改变以及填充方式的改变，使 RMSE 最小的秩在 2-7 的范围内变化，我们也发现了单纯的使用矩阵的均值对缺失值进行填充，RMSE 的效果比其他的填充方式都要好，同时，算法在不同大小的数据集上 RMSE 最小的秩都为 2，因此我们认为该算法在这两个数据集上来说效果是稳定的。

但是该算法还有许多需要提升的地方，一是填充方式的不合理，简单的使用矩阵均值进行填充，这是毫无逻辑的。这种方式容易将烂片或者是用户喜欢类型截然相反的电影推荐给用户，在此情况下的 RMSE 是没有意义的。

二是 RMSE 评判的不够完好，没有理由说明 RMSE 最小的情况下，能给用户推荐到好的电影，就如同平时刷抖音，刷到的视频都不是用户喜欢的，有些视频甚至是用户反感的。

基于这些情况，想要弄出一个优秀的推荐系统，现有的算法很多方面都需要完善，现在的推荐系统只能说是勉勉强强合格。

参考文献

[1]曹翔. 基于随机投影的大规模矩阵补全方法[D].上海交通大学,2016.

[2] Laurent, M. (2001), Matrix completion problems, in The Encyclopedia of Optimization, Kluwer Academic, pp. 221–229.

[3]Candès, Emmanuel J,Recht, Benjamin. Exact Matrix Completion via Convex Optimization[J]. Foundations of Computational Mathematics,2009,9(6).

[4] Mazumder, R., Hastie, T. and Tibshirani, R. (2010), Spectral regularization algorithms for learning large incomplete matrices, Journal of Machine Learning Research 11, 2287–2322.

[5] Nathan Halko, Per-Gunnar Martinsson, Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev., Survey and Review section, Vol. 53, num. 2, pp. 217-288, June 2011.

[6] Hastie, T., Mazumder, R.. softImpute: matrix completion via iterative soft-thresholded svd, 2013. URL http://CRAN.R-project.org/ package=softImpute. R package version 1.0.

附件

读取数据

```
library(dslabs)
data("movielens")
head(movielens, 3)

round(table(movielens$rating)/length(movielens$rating),3)

library(reshape)
Z <- cast(movielens, userId~movieId, value="rating")
dim(Z)

Z <- as.matrix(Z)
image(Z, col=grey.colors(10))
```

分割训练测试集

```
set.seed(1234)
x <- sample(1:100004)
train <- movielens
test <- movielens
train[x[75003:100004],'rating']=NA
test[x[1:75003],'rating']=NA

library(reshape)
traindata <- cast(train, userId~movieId, value="rating")
dim(traindata)

testdata <- cast(test, userId~movieId, value="rating")
dim(testdata)

traindata <- as.matrix(traindata)
testdata <- as.matrix(testdata)
```

普通 SVD 的用矩阵运算的矩阵填充，“#”是双层循环下的矩阵遍历，用平均值填充

```
completion <- function(traindata, r, n){
   h = Sys.time()
   zhat <- traindata
   zhat[is.na(zhat)] <- 3

   for (k in 1:n) {
      s = Sys.time()
      svd <- svd(zhat)
      U <- svd$u
      d <- svd$d
      V <- svd$v

      mhat <- U[,1:r]%*%diag(d[1:r])%*%t(V[,1:r])

      zhatc <- traindata

      #for (i in 1:dim(zhatc)[1]) {
      #    for (j in 1:dim(zhatc)[2]) {
      #        if(is.na(zhatc[i,j]) == TRUE){
      #            zhatc[i,j] <- mhat[i,j]
      #        }else{next}
      #    }
```

```
    #}
    zhatc[is.na(zhatc)] <- mhat[is.na(zhatc)]
    x <- zhatc - zhat
    mx <- x * x
    fn <- sqrt(sum(mx))
    cat('第', k, '次迭代结束，F 范式：', fn, '\n')
    zhat <- zhatc
    e = Sys.time()
    cat('第', k, '次迭代，共用：', e-s, '\n')
    if(fn < 10){
      print('fn 小于 10，算法收敛')
      break
    }
  }
  ha = Sys.time()
  cat('迭代达到上限,共用：', ha-h, '分钟')
  return(mhat)
}

Mat <- completion(traindata, 50, 20)
```

使用随机 SVD 的矩阵填充，用平均值填充

```
#随机 SVD
library(rsvd)
rc <- function(train, r, n){
  h = Sys.time()
  zhat <- train
  zhat[is.na(zhat)] <- mean(zhat, na.rm = TRUE)

  for (k in 1:n) {
    s = Sys.time()
    rsvd <- rsvd(zhat, k = r)
    U <- rsvd$u
    d <- rsvd$d
    V <- rsvd$v

    mhat <- U%*%diag(d)%*%t(V)
    zhatc <- train
    #for (i in 1:dim(zhatc)[1]) {
    #  for (j in 1:dim(zhatc)[2]) {
    #    if(is.na(zhatc[i,j]) == TRUE){
```

```
    #           zhatc[i,j] <- mhat[i,j]
    #       }else{next}
    #   }
    #}
    zhatc[is.na(zhatc)] <- mhat[is.na(zhatc)]
    x <- zhatc - zhat
    mx <- x * x
    fn <- sqrt(sum(mx))
    cat('第', k, '次迭代结束，F 范式：', fn, '\n')
    zhat <- zhatc
    e = Sys.time()
    cat('第', k, '次迭代，共用：', e-s, '\n')
    if(fn < 1){
      print('fn 小于 1，算法收敛')
      break
    }
  }
  ha = Sys.time()
  cat('迭代达到上限,共用：', ha-h)
  return(mhat)
}
```

RMSE 公式计算

```
RMSE <- function(train, M){
  rmse <- c()
  len <- sum(is.na(train))
  #for (i in 1:dim(train)[1]){
  #   for (j in 1:dim(train)[2]){
  #       if(is.na(train[i,j]) == FALSE){
  #           x <- (M[i,j] - train[i,j])**2
  #           rmse <- c(rmse, x)
  #       }
  #   }
  #}
  #rmse <- sqrt(sum(rmse))
  train[is.na(train)] <- M[is.na(train)]
  x <- (train - M)
  mx <- x * x
  rmse <- sqrt(sum(mx)/len)
  return(rmse)
}
```

生成一组当秩不同时的 RMSE 序列，其中秩的范围是 2-7

```
list1 <- seq(2,7)

Compute <- function(train, test, n, list, rc){
   train_rmse <- c()
   test_rmse <- c()
   for (r in list) {
      s = Sys.time()
      rmat <- rc(train, r, n)
      xrmse <- RMSE(train, rmat)
      crmse <- RMSE(test, rmat)
      train_rmse <- c(train_rmse, xrmse)
      test_rmse <- c(test_rmse, crmse)
      cat('秩为', r, '时，训练集的 RMSE 为', xrmse, '\n')
      cat('秩为', r, '时，测试集的 RMSE 为', crmse, '\n')
      e = Sys.time()
      cat('共用：', e-s, '\n')
   }
   return(cbind(train_rmse, test_rmse))
}
```

画出使用随机 SVD，在不同秩下的 RMSE 图，其中秩的范围是 2-7

```
rmsedata <- Compute(traindata, testdata, 30, list = list1, rc=rc)
opar <- par(no.readonly = TRUE)

par(mfcol=c(1,2))
plot(list1, rmsedata[,1], type='b', ylim = c(0.059,0.094), main = "训练集和测试集的 RMSE 比较
", xlab = "秩 r", ylab = "RMSE")
lines(list1, rmsedata[,2], type = 'b', col='red')

plot(list1, rmsedata[,2], type = 'b', main = "测试集的 RMSE", xlab = "秩 r", ylab = "RMSE")
```

随机 SVD，平均值填充后，对矩阵进行列中心化，其中秩的范围是 2-12

```
list3 <- seq(2,12)
library(rsvd)
nrc1 <- function(train, r, n){
   h = Sys.time()
   zhat <- train
   zhat[is.na(zhat)] <- mean(zhat, na.rm = TRUE)
```

```
  cmean <- colMeans(zhat)
  meanmat <- matrix(rep(cmean, dim(train)[1]),nrow = dim(train)[1], byrow = TRUE)
  zhat <- zhat - meanmat

  for (k in 1:n) {
    s = Sys.time()
    rsvd <- rsvd(zhat, r)
    U <- rsvd$u
    d <- rsvd$d
    V <- rsvd$v

    mhat <- U%*%diag(d)%*%t(V)
    zhatc <- train

    zhatc[is.na(zhatc)] <- mhat[is.na(zhatc)]

    x <- zhatc - zhat
    mx <- x * x
    fn <- sqrt(sum(mx))
    cat('第', k, '次迭代结束，F 范式：', fn,'\n')
    zhat <- zhatc

    cmean <- colMeans(zhat)
    meanmat <- matrix(rep(cmean, dim(train)[1]),nrow = dim(train)[1], byrow = TRUE)
    zhat <- zhat - meanmat

    e = Sys.time()
    cat('第', k, '次迭代，共用：', e-s,'\n')
    if(fn < 10){
      print('fn 小于 10，算法收敛')
      break
    }
  }
  ha = Sys.time()
  cat('迭代达到上限,共用：', ha-h, '分钟')
  return(mhat)
}
```

随机 SVD，平均值填充后，对矩阵用行中心化

```
nrc2 <- function(train, r, n){
  h = Sys.time()
  zhat <- train
  zhat[is.na(zhat)] <- mean(zhat, na.rm = TRUE)
```

```
  rmean <- rowMeans(zhat)
  meanmat <- matrix(rep(rmean, dim(train)[2]),ncol = dim(train)[2])
  zhat <- zhat - meanmat

  for (k in 1:n) {
    s = Sys.time()
    rsvd <- rsvd(zhat, r)
    U <- rsvd$u
    d <- rsvd$d
    V <- rsvd$v

    mhat <- U%*%diag(d)%*%t(V)
    zhatc <- train

    zhatc[is.na(zhatc)] <- mhat[is.na(zhatc)]

    x <- zhatc - zhat
    mx <- x * x
    fn <- sqrt(sum(mx))
    cat('第', k, '次迭代结束，F 范式：', fn,'\n')
    zhat <- zhatc

    rmean <- rowMeans(zhat)
    meanmat <- matrix(rep(rmean, dim(train)[2]),ncol = dim(train)[2])
    zhat <- zhat - meanmat

    e = Sys.time()
    cat('第', k, '次迭代，共用：', e-s,'\n')
    if(fn < 10){
      print('fn 小于 10，算法收敛')
      break
    }
  }
  ha = Sys.time()
  cat('迭代达到上限,共用：', ha-h, '分钟')
  return(mhat)
}
```

画出行列中心化后的 RMSE，其中秩的范围是 2-12

```
rmsedata1 <- Compute(traindata, testdata, 30, list = list3, rc=nrc1)
rmsedata1_1 <- Compute(traindata, testdata, 30, list = list3, rc=nrc2)
```

```
par(mfcol=c(2,2))
plot(list3, rmsedata1[,1], type='o', ylim = c(0.133,0.254), main = "训练集和测试集的 RMSE 比较
", xlab = "秩 r", ylab = "RMSE")
lines(list3, rmsedata1[,2], type = 'o', col='red')
plot(list3, rmsedata1[,2], type = 'o', main = "测试集的 RMSE", xlab = "秩 r", ylab = "RMSE")

plot(list3, rmsedata1_1[,1], type='o', ylim = c(0.125,0.21), main = "训练集和测试集的 RMSE 比
较", xlab = "秩 r", ylab = "RMSE")
lines(list3, rmsedata1_1[,2], type = 'o', col='red')
plot(list3, rmsedata1_1[,2], type = 'o', main = "测试集的 RMSE", xlab = "秩 r", ylab = "RMSE")
```

随机 SVD，用 softimpute 进行填充

```
#带有 softimpute 的随机 SVD
library(rsvd)
library(softImpute)
src <- function(train, r, n){
  h = Sys.time()
  zhat <- train
  fit <- softImpute(zhat,rank=5)
  zhat <- complete(zhat, fit)

  for (k in 1:n) {
    s = Sys.time()
    rsvd <- rsvd(zhat, r)
    U <- rsvd$u
    d <- rsvd$d
    V <- rsvd$v

    mhat <- U%*%diag(d)%*%t(V)
    zhatc <- train

    zhatc[is.na(zhatc)] <- mhat[is.na(zhatc)]

    x <- zhatc - zhat
    mx <- x * x
    fn <- sqrt(sum(mx))
    cat('第', k, '次迭代结束，F 范式：', fn,'\n')
    zhat <- zhatc
    e = Sys.time()
    cat('第', k, '次迭代，共用：', e-s,'\n')
```

```
      if(fn < 10){
        print('fn 小于 10，算法收敛')
        break
      }
    }
    ha = Sys.time()
    cat('迭代达到上限,共用：', ha-h, '分钟')
    return(mhat)
}

srMat <- src(traindata, 4, 30)
```

画出填充值用 softimpute 的 RMSE，其中秩的范围是 2-7

```
list4 <- seq(2,7)

rmsedata2 <- Compute(traindata, testdata, 30, list = list4, rc=src)

par(mfcol=c(1,2))
plot(list4, rmsedata2[,1], type='o', ylim = c(0.085,0.13), main = "训练集和测试集的 RMSE 比较
", xlab = "秩 r", ylab = "RMSE")
lines(list4, rmsedata2[,2], type = 'o', col='red')

plot(list4, rmsedata2[,2], type = 'o', main = "测试集的 RMSE", xlab = "秩 r", ylab = "RMSE")
```

使用 1 百万的数据集进行测试

```
bigdata <- read.table("ratings.dat",sep = ":")

bigdata <- bigdata[,c(-2,-4,-6)]

names(bigdata) <- c("userId", "movieId", "rating", "timestamp")

head(bigdata, 3)

round(table(bigdata$rating)/length(bigdata$rating),3)



bigdata <- bigdata
set.seed(666)
x <- sample(1:1000209)
trainbig <- bigdata
```

```
testbig <- bigdata
trainbig[x[750160:1000209],'rating']=NA
testbig[x[1:750159],'rating']=NA

library(reshape)
trainbigdata <- cast(trainbig, userId~movieId, value="rating")
dim(trainbigdata)

testbigdata <- cast(testbig, userId~movieId, value="rating")
dim(testbigdata)

trainbigdata <- as.matrix(trainbigdata)
testbigdata <- as.matrix(testbigdata)


library(rsvd)

#bigMat <- rc(trainbigdata, 50, 20)
```

画出 RMSE，其中秩的范围是 2-7

```
rmsebigdata <- Compute(trainbigdata, testbigdata, 30, list = list1, rc=rc)

par(mfcol=c(1,2))
plot(list1, rmsebigdata[,1], type='o', ylim = c(0.1,0.16), main = "训练集和测试集的 RMSE 比较",
xlab = "秩 r", ylab = "RMSE")
lines(list1, rmsebigdata[,2], type = 'o', col='red')

plot(list1, rmsebigdata[,2], type = 'o', main = "测试集的 RMSE", xlab = "秩 r", ylab = "RMSE")
```