《机器学习实战》源码解析

机器学习源码解析 XII

SVD 推荐算法

jackycaf

2014/5/3 Saturday

目录

第一章 SVD 与 LSI 理论

矩阵的奇异值分解

奇异值分解（Singular Value Decomposition）是线性代数中一种重要的矩阵分解，是矩阵分析中正规矩阵酉对角化的推广。在信号处理、统计学等领域有重要应用。

奇异值分解在某些方面与对称矩阵或 Hermite 矩阵基于特征向量的对角化类似。然而这两种矩阵分解尽管有其相关性，但还是有明显的不同。对称阵特征向量分解的基础是谱分析，而奇异值分解则是谱分析理论在任意矩阵上的推广。

理论描述

假设 M 是一个 m×n 阶矩阵，其中的元素全部属于域 K，也就是实数域或复数域。如此则存在一个分解使得

$$M=U\Sigma V^*$$

其中 U 是 m×m 阶酉矩阵；Σ 是半正定 m×n 阶对角矩阵；而 V*，即 V 的共轭转置，是 n×n 阶酉矩阵。这样的分解就称作 M 的奇异值分解。Σ 对角线上的元素$\Sigma_{i,i}$即为 M 的奇异值。

常见的做法是将奇异值由大而小排列。如此Σ 便能由 M 唯一确定了。(虽然 U 和 V 仍然不能确定。)

直观的解释

在矩阵 M 的奇异值分解中

$$M=U\Sigma V^*$$

V 的列(columns)组成一套对 M 的正交"输入"或"分析"的基向量。这些向量是 M*M 的特征向量。

U 的列(columns)组成一套对 M 的正交"输出"的基向量。这些向量是 MM*的特征向量。

Σ 对角线上的元素是奇异值，可视为是在输入与输出间进行的标量的"膨胀控制"。这些是 MM* 及 M*M 的特征值的非零平方根，并与 U 和 V 的行向量相对应。

奇异值和奇异向量，以及他们与奇异值分解的关系

一个非负实数σ 是 M 的一个奇异值仅当存在 Km 的单位向量 u 和 Kn 的单位向量 v 如下 ：

$$Mv= \sigma u \text{ and } M^*u = \sigma v$$

其中向量 u 和 v 分别为σ 的左奇异向量和右奇异向量。

对于任意的奇异值分解

$M = U\Sigma V^*$

矩阵Σ 的对角线上的元素等于 M 的奇异值. U 和 V 的列分别是奇异值中的左、右奇异向量。因此,上述定理表明:

一个 m × n 的矩阵至少有一个最多有 p = min(m,n)个不同的奇异值。

总是可以找到在 Km 的一个正交基 U，组成 M 的左奇异向量。

总是可以找到和 Kn 的一个正交基 V，组成 M 的右奇异向量。

如果一个奇异值中可以找到两个左（或右）奇异向量是线性相关的，则称为退化。

非退化的奇异值具有唯一的左、右奇异向量，取决于所乘的单位相位因子 eiφ （根据实际信号）。因此，如果 M 的所有奇异值都是非退化且非零，则它的奇异值分解是唯一的，因为 U 中的一列要乘以一个单位相位因子且同时 V 中相应的列也要乘以同一个相位因子。

根据定义，退化的奇异值具有不唯一的奇异向量。因为，如果 u1 和 u2 为奇异值σ 的两个左奇异向量，则两个向量的任意规范线性组合也是奇异值σ 一个左奇异向量，类似的，右奇异向量也具有相同的性质。因此，如果 M 具有退化的奇异值，则它的奇异值分解是不唯一的。

例子

观察一个 4x5 的矩阵：

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \end{bmatrix}.$$

M 矩阵的奇异值分解如下　UΣ V*

$$U = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}, \Sigma = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & \sqrt{5} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, V^* = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ \sqrt{0.8} & 0 & 0 & 0 & -\sqrt{0.2} \end{bmatrix}.$$

注意到Σ 在对角线上包含 0. 一个对角线元素是 0. 并且, 因为矩阵 U 和 V*是 酉, 乘以各自个共轭系数并得出结果确定矩阵, 如下所示. 在这个例子中，由于 U 和 V*是实值，他们各自是 正交矩阵.

$$UU^* = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \equiv I_4$$

$$VV^* = \begin{bmatrix} 0 & 0 & \sqrt{0.2} & 0 & \sqrt{0.8} \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \sqrt{0.8} & 0 & -\sqrt{0.2} \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ 0 & 0 & 0 & 1 & 0 \\ \sqrt{0.8} & 0 & 0 & 0 & -\sqrt{0.2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \equiv I_5.$$

必须注意的是这个奇异值分解值不是唯一的. 选择 V 使得

$$V^* = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sqrt{0.2} & 0 & 0 & 0 & \sqrt{0.8} \\ \sqrt{0.4} & 0 & 0 & \sqrt{0.5} & -\sqrt{0.1} \\ -\sqrt{0.4} & 0 & 0 & \sqrt{0.5} & \sqrt{0.1} \end{bmatrix}$$

这也是一个合法的奇异值

与特征值分解的联系

奇异值分解在意义上看很一般，因此它可以适用于任意 $m \times n$ 矩阵，而特征分解只能适用于特定类型的方阵。不过，这两个分解之间是有关联的。 给定一个 M 的奇异值分解，根据上面的论述，两者的关系式如下：

$$M^*M = V\Sigma^*U^*U\Sigma V^* = V(\Sigma^*\Sigma)V^*$$

$$MM^* = U\Sigma V^*V\Sigma^*U^* = U(\Sigma\Sigma^*)U^*$$

关系式的右边描述了关系式左边的特征值分解。于是：

- V（右奇异向量）的列是 M^*M 的特征向量。
- U（左奇异向量）的列是 MM^* 的特征向量。
- Σ （非零奇异值）的非零元素是 M^*M 或者 MM^* 中非零特征值的平方根。

特殊情况下，当 M 是一个正规矩阵，即是一个方阵时，根据谱定理，M 可以被一组特征向量单一对角化，所以它可以被写为：

$$M=UDU^*$$

其中 U 为一个酉矩阵，D 为一个对角阵。如果 M 是半正定的， $M=UDU^*$ 的分解也是一个奇异值分解。

然而，在其他所有的 M 矩阵上，特征分解跟奇异值分解是不同。特征分解如下：

$$M=UDU^{-1}$$

其中 U 是不需要是单式的，D 也不需要是正半定的。而奇异值分解如下：

$$M=U\Sigma V^*$$

其中 Σ 是对角半正定矩阵，U 和 V 是酉矩阵，两者除了通过矩阵 M 没有必然的联系。

隐性语义索引(LSI)

latent Semantic Indexing-隐性语义索引，也可译为隐含语义索引，是近年来逐渐兴起的不同于关键词检索的搜索引擎解决方案，其检索结果的实际效果更接近于人的自然语言，在一定程度上提高检索结果的相关性，目前已被逐渐的应用到图书馆、数据库和搜索引擎的算法当中。Google 就是典型的代表。

隐性语义索引的概念

所谓隐性语义索引指的是，怎样通过海量文献找出词汇之间的关系。当两个词或一组词大量出现在同一个文档中时，这些词之间就可以被认为是语义相关。机器并不知道某个词究竟代表什么，不知道某个词是什么意思。

比如：

（1）电脑和计算机这两个词在人们写文章时经常混用，这两个词在大量的网页中同时出现，搜索引擎就会认为这两个词是极为语义相关的。

（2）SEO 和搜索引擎优化（虽然一个是英语，一个是中文）这两个词大量出现在相同的网页中，虽然搜索引擎还不能知道搜索引擎优化或 SEO 指的是什么，但是却可以从语义上把”SEO”,”搜索引擎优化”,”Search Engine optimization”,”SEM”等词紧紧的连在一起。可见潜在语义索引并不依赖于语言。

（3）如苹果和橘子这两个词，也是大量出现在相同文档中，不过紧密度低于同义词。所以搜索引擎不会认为它们是语义相关的。

为什么需要隐性语义索引

搜索引擎是使用机器算法来替代过去人工搜索的工作。但机器算法和人的工作有一个很一样的地方就是人可以直接理解词的意思，文章的意思，机器和算法却无法理解——人看到苹果这两个字就知道指的是那个圆圆的，有水的挺好吃的东西，搜索引擎却不能从感性上理解。

其原因和自然语言的特点有关。从自然语言的角度，大部分词具有一词多义的特点，机器算法无法确定在何环境下使用何种词义，这就导致了搜索结果与用户的理想值便存在很大的距离：

（1）一词多义将导致基于精确匹配的搜索算法在结出的结果中包含很多并非用户真正要查找的内容；

（2）一义多词则使用得基于精确匹配的搜索算法在给出的结果中遗漏很多用户真正要查找的内容。

Latent Semantic Indexing (LSI：隐性语义索引)便是搜索引擎试图尽可能弱化这一弊端的可行解决方案之一。Latent Semantic Indexing 通过绕开自然语言理解，以大样本数量的统计分析找出不同的词(词组、短语)间的相关性，以使搜索结果进一步接近于用户真正要查找的内容，同时，也能够保证搜索的效率。

隐性语义索引的机制

单纯从理论上看，Latent semantic indexing (隐性语义索引)的实现机制并不复杂，它只不过是在正常的网页收录与索引过程中增添了一个步骤：

（1）先统计、分析网页及链接中的关键词；

（2）将该网页与索引数据库中其他包含相同关键词或部分相同关键词的网页进行比对，以确定不同网页间的语义相关性以及网页与特定关键词间的相关性

（3）同时将该网页与具有高语义相关性的网页进行比对分析，从中找出特定网页中存在关键词的相关项，即找出特定网页中虽然并不存在但与其内容相关的关键词。

可以看出，虽然搜索引擎本身并不知道某个词究竟代表什么，不知道某个词是什么意思，但通过 Latent Semantic Indexing 算法，与单纯的关键词匹配相比，搜索引擎能够以一种更准确的方式判断特定网

页中内容与搜索项间的相关性，从而给出用户要寻找的内容，甚至从某种角度上看，更接近于“人”分析、查找内容时的判断方式。

隐性语义索引的其他关键点

对 Latent Semantic Indexing，可能大多数人注意的是其中的 Semantic（语义有关的），但 Latent Semantic Indexing 方面的技术文档则往往更强调 Latent（潜在的、隐含的），而非简单意义的语义相关。比如说对“水”一词而言，与其语义相关的可能是“热水”、“凉水”之类，但潜在相关的则可以是“蒸汽”、“冰”等，这里有很大区别。

需要强调的是，Latent Semantic Indexing(隐性语义索引)只是目前搜索引擎排名算法中关键词匹配技术的补充（排名算法会为不同的影响因素赋予不同的权重，Latent Semantic Indexing 的权重值已经慢慢加大），但绝不是取代现有的关键词匹配算法。

第二章 SVD 与相似性验证

实现奇异值分解

代码：testSvd01.py

```
# -*- coding: utf-8 -*-
# Filename : testSvd01.py

from numpy import *
import numpy as np
import operator
import svdRec2
import matplotlib.pyplot as plt

eps = 1.0e-6
# numpy 求矩阵的奇异分解
# U:酉矩阵
# Sigma:半正定 m×n 阶对角矩阵
# VT:V 的共轭转置
U,Sigma,VT = linalg.svd([[1,1],[7,7]])
print "U:",U
print "Sigma:",Sigma
print "VT:",VT
```

输出：

```
U: [[-0.14142136 -0.98994949]
 [-0.98994949  0.14142136]]
Sigma: [ 10.   0.]
VT: [[-0.70710678 -0.70710678]
 [-0.70710678  0.70710678]]
```

矩阵相似形验证

我们是如何知道仅需保留前3个奇异值的呢？确定要保留的奇异值的数目有很多启发式的策略，其中一个典型的做法就是保留矩阵中90%的能量信息。为了计算总能量信息，我们将所有的奇异值求其平方和。于是可以将奇异值的平方和累加到总值的90%为止。另一个启发式策略就是，当矩阵上有上万的奇异值时，那么就保留前面的2000或3000个。尽管后一种方法不太优雅，但是在实际中更容易实施。之所以说它不够优雅，就是因为在任何数据集上都不能保证前3000个奇异值就能够包含90%的能量信息。但在通常情况下，使用者往往都对数据有足够的了解，从而就能够做出类似的假设了。

现在我们已经通过三个矩阵对原始矩阵进行了近似。我们可以用一个小很多的矩阵来表示一个大矩阵。有很多应用可以通过SVD来提升性能。下面我们将讨论一个比较流行的SVD应用的例子——推荐引擎。

代码：testSvd01.py

```
# 矩阵相似形验证
Data = svdRec2.loadExData()
# 数据矩阵化
Data = mat(Data)
print "Data:\n",Data
# 进行奇异值分解
U,Sigma,VT = linalg.svd(Data)
# 只取前 3 项重构相似阵
Sig3 = mat(Sigma[:3]*eye(3))
U3 = U[:,:3]
VT3 = VT[:3,:]
# D3 == Data 矩阵是相似的
D3 = U3*Sig3*VT3
# 返回 D3 中大于 0 的所有元素的下标
indxs = nonzero(abs(D3.A)<eps)
D3[indxs]=0
# 输出 D3
print "D3:\n", D3
D_D3 = Data-D3
# 返回 Data-D3 中大于 0 的所有元素的下标
idxs = nonzero(abs(D_D3.A)<eps)
D_D3[idxs]=0
print "如果输出为全零矩阵，Data 与 D3 两矩阵相似",D_D3
```

输出：

```
Data:
[[0 0 0 2 2]
 [0 0 0 3 3]
 [0 0 0 1 1]
 [1 1 1 0 0]
```

```
 [2 2 2 0 0]
 [5 5 5 0 0]
 [1 1 1 0 0]]
D3:
[[ 0.  0.  0.  2.  2.]
 [ 0.  0.  0.  3.  3.]
 [ 0.  0.  0.  1.  1.]
 [ 1.  1.  1.  0.  0.]
 [ 2.  2.  2.  0.  0.]
 [ 5.  5.  5.  0.  0.]
 [ 1.  1.  1.  0.  0.]]
如果输出为全零矩阵，Data 与 D3 两矩阵相似 [[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
```

代码：testSvd01.py

```
# 奇异值分解
U,Sigma,VT = linalg.svd(Data)
SigmaE = mat(Sigma[:4]*eye(4))
# VT 的转置:V1 是 dataMat 的相似矩阵
V1 = VT.T[:,:4]
print "V1:\n",V1
# 验证相似性
V2 = DataMat.T*U[:,:4]*SigmaE.I
print "V2:\n",V2
V1_V2 = V1-V2
idxs = nonzero(abs(V1_V2.A)<eps)
V1_V2[idxs] =0
print "V1-V2:\n", V1_V2[idxs]
```

输出：

```
Data:
[[0 0 0 2 2]
 [0 0 0 3 3]
 [0 0 0 1 1]
 [1 1 1 0 0]
 [2 2 2 0 0]
```

```
  [5 5 5 0 0]
  [1 1 1 0 0]]
V1:
[[-0.72891572 -0.29288354  0.37735484 -0.49041252]
  [-0.51851171  0.38695254 -0.76111202 -0.04606326]
  [-0.35413431  0.55605121  0.48940585  0.57085713]
  [-0.19289424 -0.47712368 -0.13927263  0.46448675]
  [-0.19289424 -0.47712368 -0.13927263  0.46448675]]
V2:
[[-0.72891572 -0.29288354  0.37735484 -0.49041252]
  [-0.51851171  0.38695254 -0.76111202 -0.04606326]
  [-0.35413431  0.55605121  0.48940585  0.57085713]
  [-0.19289424 -0.47712368 -0.13927263  0.46448675]
  [-0.19289424 -0.47712368 -0.13927263  0.46448675]]
V1-V2:
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
   0.  0.]]
```

第三章 推荐系统

各类距离公式

代码：svdRec2.py

```
# 欧氏距离：
# 二维空间的欧氏距离公式：sqrt((x1-x2)^2+(y1-y2)^2)
def ecludSim(inA,inB):
    return 1.0/(1.0 + la.norm(inA - inB))

# 皮尔逊相似度：corrcoef 相关系数:衡量 X 与 Y 线性相关程度,其绝对值越大,则表明 X 与 Y 相关度越高。
# E((X-EX)(Y-EY))/sqrt(D(X)D(Y))
def pearsSim(inA,inB):
    if len(inA) < 3 : return 1.0
    return 0.5+0.5*corrcoef(inA, inB, rowvar = 0)[0][1]

# 夹角余弦：计算空间内两点之间的夹角余弦
# 两个 n 维样本点 a(x11,x12,…,x1n)和 b(x21,x22,…,x2n)的夹角余弦
# cos(theta) = a*b/(|a|*|b|)
def cosSim(inA,inB):
    num = float(inA.T*inB)
    denom = la.norm(inA)*la.norm(inB)
    return 0.5+0.5*(num/denom)
```

执行程序

代码：testRecomm02.py

```
# -*- coding: utf-8 -*-
# Filename : testRecomm02.py

from numpy import *
import numpy as np
import operator
import svdRec2
import matplotlib.pyplot as plt
```

```
eps = 1.0e-6
# 加载修正后数据
Data = svdRec2.loadReData()
dataMat = mat(Data)

# 相似公式：夹角余弦
output1 = svdRec2.recommend(dataMat,1,estMethod=svdRec2.svdEst)
print output1

# 相似公式：欧氏距离
output2 = svdRec2.recommend(dataMat,1,estMethod=svdRec2.svdEst,simMeas=svdRec2.ecludSim)
print output2

# 相似公式：相关系数
output3 = svdRec2.recommend(dataMat,1,estMethod=svdRec2.svdEst,simMeas=svdRec2.pearsSim)
print output3
```

输出：

```
第 1 列和第 0 列的相似度是: 0.500000
第 1 列和第 3 列的相似度是: 0.500000
第 1 列和第 4 列的相似度是: 0.500000
第 2 列和第 0 列的相似度是: 0.500000
第 2 列和第 3 列的相似度是: 0.500000
第 2 列和第 4 列的相似度是: 0.500000
[(1, 3.3333333333333339), (2, 3.333333333333333)]
第 1 列和第 0 列的相似度是: 0.414214
第 1 列和第 3 列的相似度是: 0.449490
第 1 列和第 4 列的相似度是: 0.449490
第 2 列和第 0 列的相似度是: 0.414214
第 2 列和第 3 列的相似度是: 0.449490
第 2 列和第 4 列的相似度是: 0.449490
[(2, 3.3154247298392598), (1, 3.315424729839259)]
第 1 列和第 0 列的相似度是: 0.316849
第 1 列和第 3 列的相似度是: 0.433183
第 1 列和第 4 列的相似度是: 0.433183
第 2 列和第 0 列的相似度是: 0.780318
第 2 列和第 3 列的相似度是: 0.602265
第 2 列和第 4 列的相似度是: 0.602265
[(2, 3.3931372876394206), (1, 3.2677865084349662)]
```

评估主函数

代码：svdRec2.py

```
# 产生推荐结果的主方法
# simMeas 取值:cosSim, pearsSim, ecludSim
# estMethod 取值:standEst,svdEst
# user 用户项目矩阵中进行评估的行下标
# N=3 返回前 3 项
def recommend(dataMat, user, N=3, simMeas=cosSim, estMethod=standEst):
    unratedItems = nonzero(dataMat[user,:].A==0)[1] # 查找未评级的项目--即用户--项目矩阵中 user 行对应的零值
    # print "unratedItems:",unratedItems
    # unratedItems: 为用户--项目矩阵中 user 行对应零值的列下标
    if len(unratedItems) == 0: return "已经对所有项目评级"
    # 初始化项目积分数据类型,是一个二维矩阵
    # 元素 1:item;元素 2:评分值
    itemScores = []
    # 循环进行评估:将每个未评估项目于已评估比较，计算相似度
    # 本例中未评估项目取值 1,2
    for item in unratedItems:
        estimatedScore = estMethod(dataMat, user, simMeas, item) # 使用评估方法对数据评估，返回评估积分
        itemScores.append((item, estimatedScore)) # 并在项目积分内加入项目和对应的评估积分
    # 返回排好序的项目和积分， N=3 返回前 3 项
    return sorted(itemScores, key=lambda jj: jj[1], reverse=True)[:N]
```

STD 评估算法：标准评估

代码：svdRec2.py

```
# 标准相似度计算方法下的用户估计值
def standEst(dataMat, user, simMeas, item):
    n = shape(dataMat)[1]                # 列数
    simTotal = 0.0; ratSimTotal = 0.0
    for j in range(n):
        userRating = dataMat[user,j]    # 数据集第 user 行第 j 列的元素值
        if userRating == 0: continue    # 跳过未评估项目
        # logical_and:矩阵逐个元素运行逻辑与,返回值为每个元素的 True,False
        # dataMat[:,item].A>0: 第 item 列中大于 0 的元素
        # dataMat[:,j].A: 第 j 列中大于 0 的元素
        # overLap: dataMat[:,item],dataMat[:,j]中同时都大于 0 的那个元素的行下标
```

```
        overLap = nonzero(logical_and(dataMat[:,item].A>0,dataMat[:,j].A>0))[0]
        # 计算相似度
        if len(overLap) == 0: similarity = 0
        else: similarity = simMeas(dataMat[overLap,item],dataMat[overLap,j]) # 计算 overLap 矩阵的相似度
        # print "第%d 列和第%d 列的相似度是: %f" %(item, j, similarity)
        # 累计总相似度
        simTotal += similarity
        # ratSimTotal = 相似度*元素值
        ratSimTotal += similarity * userRating
    if simTotal == 0: return 0 # 如果总相似度为 0，返回 0
    # 返回相似度*元素值/总相似度
    else:
        # print "ratSimTotal:",ratSimTotal
        # print "simTotal:",simTotal
        return ratSimTotal/simTotal
```

SVD 评估算法

代码：svdRec2.py

```
#使用 svd 进行估计
def svdEst(dataMat, user, simMeas, item):
    n = shape(dataMat)[1]
    simTotal = 0.0; ratSimTotal = 0.0
    # svd 相似性计算的核心
    U,Sigma,VT = la.svd(dataMat)    # 计算矩阵的奇异值分解
    # Sig4 = mat(eye(4)*Sigma[:4]) # 取 Svd 特征值的前 4 个构成对角阵
    # xformedItems = dataMat.T * U[:,:4] * Sig4.I  # 创建变换后的项目矩阵 create transformed items
    V = VT.T        # V 是 dataMat 的相似矩阵
    xformedItems = V[:,:4]
    print "xformedItems:",xformedItems
    # 逐列遍历数据集
    for j in range(n):
        userRating = dataMat[user,j]   # 未评级用户为 0,因此不会计算。其他的均有值
        # print "userRating:",userRating
        # 跳过未评级的项目
        if userRating == 0 or j==item: continue
        # 使用指定的计算公式计算向量间的相对度
        similarity = simMeas(xformedItems[item,:].T,xformedItems[j,:].T)   # 相似度计算公式
        print "待评估%d 列和第%d 列的相似度是: %f" % (item, j, similarity)
        simTotal += similarity                  # 计算累计总相似度
        ratSimTotal += similarity * userRating  # ratSim = 相似度*项目评估值
    if simTotal == 0: return 0
```

```
    else: return ratSimTotal/simTotal
```