实验5-2：SVD降维

1. 1 SVD 概述

奇异值分解（SVD, Singular Value Decomposition）：

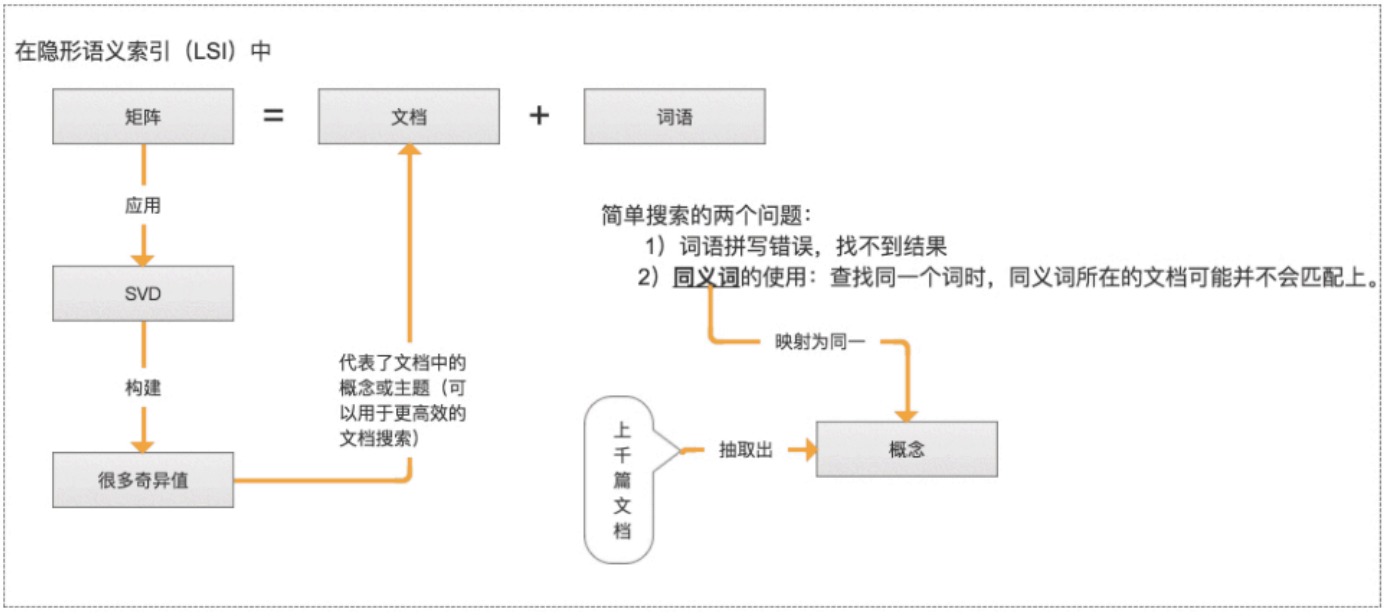
提取信息的一种方法，可以把 SVD 看成是从噪声数据中抽取相关特征。

1.2 SVD 场景

> 信息检索-隐性语义检索（Latent Semantic Indexing, LSI）或 隐形语义分析（Latent Semantic Analysis, LSA）

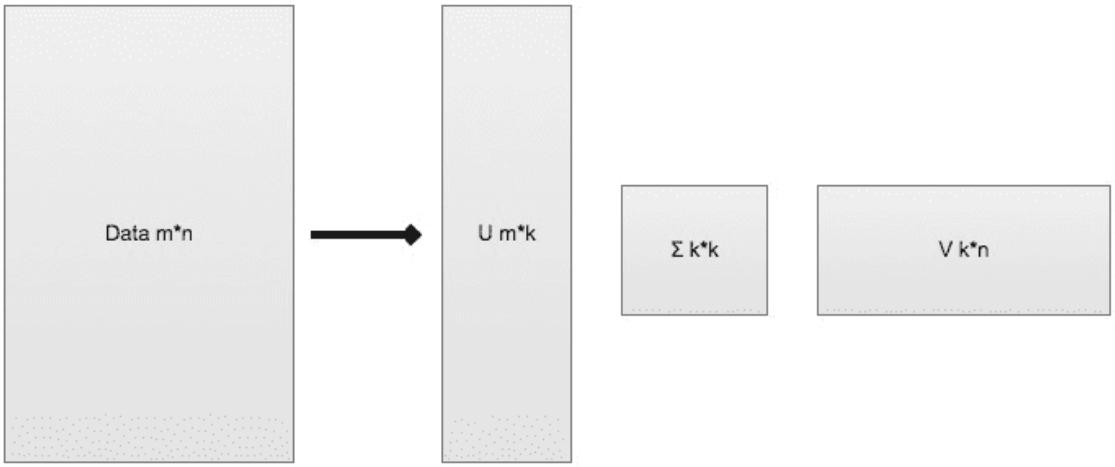
隐性语义索引: 矩阵 = 文档 + 词语

- 是最早的 SVD 应用之一，我们称利用 SVD 的方法为隐性语义索引（LSI）或隐性语义分析（LSA）。



> 图像压缩

例如: `32*32=1024 => 32*2+2*1+32*2=130`(2*1表示去掉了除对角线的0), 几乎获得了10倍的压缩比。



1.3 SVD 原理

1.3.1 SVD 工作原理

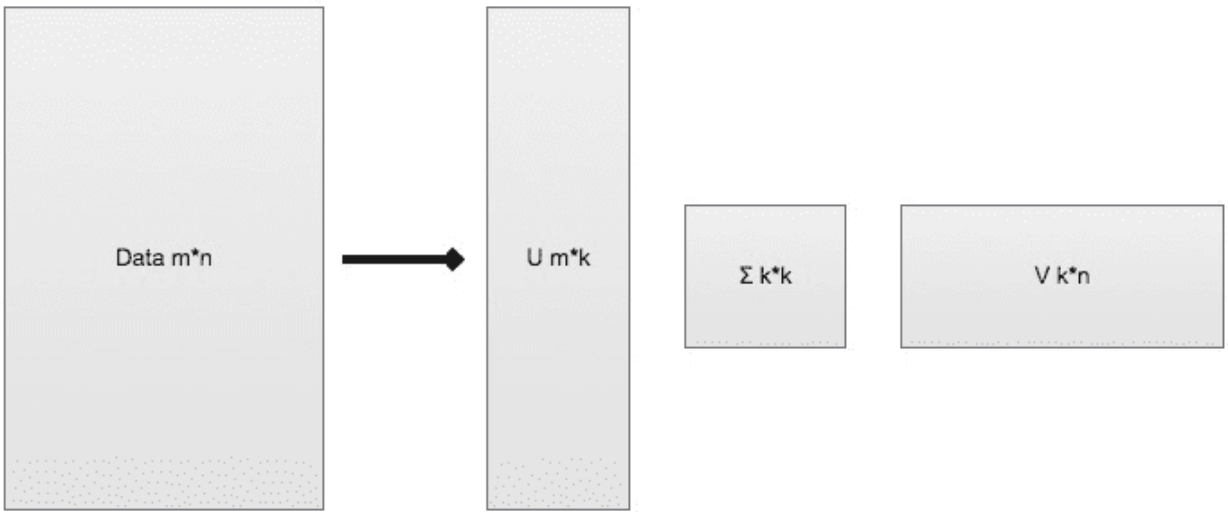
> 矩阵分解

- 矩阵分解是将数据矩阵分解为多个独立部分的过程。
- 矩阵分解可以将原始矩阵表示成新的易于处理的形式，这种新形式是两个或多个矩阵的乘积。（类似代数中的因数分解）

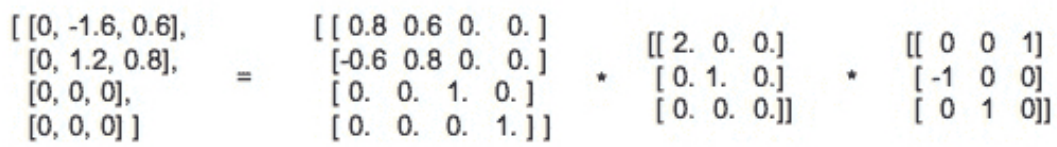
> SVD 是矩阵分解的一种类型，也是矩阵分解最常见的技术

- SVD 将原始的数据集矩阵 Data 分解成三个矩阵 U、$\sum$、V
- 举例: 如果原始矩阵 $Data_{m*n}$ 是m行n列，
  - U_{m*k} 表示m行k列
  - $\sum_{k*k}$ 表示k行k列
  - V_{k*n} 表示k行n列。

$$Data_{m*n} = U_{m*k} \sum_{k*k} V_{k*n}$$



具体的案例: （大家可以试着推导一下 ）



- 上述分解中会构建出一个矩阵 $\sum$ ，该矩阵只有对角元素，其他元素均为0(近似于0)。另一个惯例就是，$\sum$ 的对角元素是从大到小排列的。这些对角元素称为奇异值。
- 奇异值与特征值(PCA 数据中重要特征)是有关系的。这里的奇异值就是矩阵 $Data * Data^T$ 特征值的平方根。

1.3.2 SVD 算法特点

优点: 简化数据，去除噪声，优化算法的结果
缺点: 数据的转换可能难以理解
使用的数据类型: 数值型数据

2. 动手实践——推荐案例

2.1 概述

推荐系统是利用电子商务网站向客户提供商品信息和建议，帮助用户决定应该购买什么产品，模拟销售人员帮助客户完成购买过程。

2.2 推荐系统的场景

1. Amazon 会根据顾客的购买历史向他们推荐物品
2. Netflix 会向其用户推荐电影
3. 新闻网站会对用户推荐新闻频道

2.3 要点

> 基于协同过滤(collaborative filtering) 的推荐引擎

- 利用Python 实现 SVD(Numpy 有一个称为 linalg 的线性代数工具箱)
- 协同过滤: 协同过滤是推荐系统中的一种常用技术，其核心思想是通过分析用户之间的行为和偏好相似性来进行推荐。这种方法假设那些在过去对某些项目有相似喜好的用户，在未来对其他项目的喜好也可能相似。
- 当知道了两个用户或两个物品之间的相似度，我们就可以利用已有的数据来预测未知用户的喜好。

> 基于物品的相似度和基于用户的相似度: 物品比较少则选择物品相似度，用户比较少则选择用户相似度。【矩阵还是小一点好计算】

- 基于物品的相似度: 计算物品之间的距离。基于物品的协同过滤关注于物品之间的相似性。这种方法会评估项目之间的相似度，通常基于所有用户对它们的评分。如果两个项目经常被相似的用户群体同时评分，则这两个项目被视为相似。如果一个用户喜欢某个项目，系统会推荐与该项目相似的其他项目。
- 由于物品A和物品C 相似度(相关度)很高，所以给买A的人推荐C。

用户/物品	物品A	物品B	物品C
用户A	√		√
用户B	√	√	√
用户C	√		推荐

- 基于用户的相似度: 计算用户之间的距离。这种方法首先计算用户之间的相似性，通常是通过比较他们对共同项目的评分。一旦计算出用户间的相似性，就可以为一个用户推荐那些与他/她有相似喜好的其他用户喜欢的项目。例如，如果用户A与用户C有相似的电影口味，那么用户A可能会喜欢用户C评分高的一些电影，尽管A之前没有看过这些电影。
- 由于用户A和用户C 相似度(相关度)很高，所以A和C是兴趣相投的人，对于C买的物品就会推荐给A。

用户/物品	物品A	物品B	物品C	物品D
用户A	√		√	推荐
用户B		√		
用户C	√		√	√

> 相似度计算

1. 欧氏距离 (Euclidean Distance)：

- 欧氏距离是最直观的距离度量方法，它计算两个点在多维空间中的直线距离。
- 通过对欧氏距离取倒数并进行平移，将距离转换为相似度，使其范围在0到1之间。距离越近，相似度越高。

数学公式:

$$\text{Similarity} = \frac{1}{1+\text{Euclidean Distance}(inA,inB)}$$

其中，欧氏距离定义为：

$$\text{Euclidean Distance}(inA, inB) = \sqrt{\sum_{i=1}^{n}(inA_i - inB_i)^2}$$

2. 皮尔逊相关系数 (Pearson Correlation Coefficient)：

- 皮尔逊相关系数度量两个向量之间的线性相关程度。系数值范围从-1到+1，其中+1表示完全正相关，-1表示完全负相关，0表示无相关。
- 通过将原始的相关系数进行线性变换，可以将其调整到0到1的范围，以适应相似度度量。

数学公式:

$$\text{Similarity} = 0.5 + 0.5 \times \text{corrcoef}(inA, inB)$$

其中，皮尔逊相关系数定义为：

$$\text{corrcoef}(inA, inB) = \frac{\sum(inA_i - \bar{inA})(inB_i - \bar{inB})}{\sqrt{\sum(inA_i - \bar{inA})^2 \sum(inB_i - \bar{inB})^2}}$$

3. 余弦相似度 (Cosine Similarity)：

- 余弦相似度是通过计算两个向量的夹角余弦来度量它们的方向相似度。夹角越小，余弦值越接近1，表示方向越相似。
- 此方法特别适用于文本数据的相似度计算，因为它只关注向量的方向而不是向量的大小。

数学公式:

$$\text{Similarity} = 0.5 + 0.5 \times \cos(\theta)$$

其中，余弦值定义为：

$$\cos(\theta) = \frac{inA \cdot inB}{\|inA\| \cdot \|inB\|} = \frac{\sum_{i=1}^{n} inA_i \times inB_i}{\sqrt{\sum_{i=1}^{n} inA_i^2} \times \sqrt{\sum_{i=1}^{n} inB_i^2}}$$

2.4 原理

- 推荐系统的工作过程: 给定一个用户，系统会为此用户返回N个最好的推荐菜。
- 实现流程大致如下:
    1. 寻找用户没有评级的菜肴，即在用户-物品矩阵中的0值。
    2. 在用户没有评级的所有物品中，对每个物品预计一个可能的评级分数。这就是说: 我们认为用户可能会对物品的打分（这就是相似度计算的初衷）。
    3. 对这些物品的评分从高到低进行排序，返回前N个物品。

2.5 项目案例一: 餐馆菜肴推荐系统

项目概述

假如一个人在家决定外出吃饭，但是他并不知道该该点什么菜。推荐系统可以帮他做到这两点。

> 准备数据

	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	三文鱼汉堡	鲁宾三明治	印度烤鸡	麻婆豆腐	宫保鸡丁	印度奶酪咖喱	俄式汉堡
Bratt	2	0	0	4	4	0	0	0	0	0	0
Rob	0	0	0	0	0	0	0	0	0	0	5
Drew	0	0	0	0	0	0	0	1	0	4	0
Scott	3	3	4	0	3	0	0	2	2	0	0
Mary	5	5	5	0	0	0	0	0	0	0	0
Brent	0	0	0	0	0	0	5	0	0	5	0
Kyle	4	0	4	0	0	0	0	0	0	0	5
Sara	0	0	0	0	0	4	0	0	0	0	4
Shaney	0	0	0	0	0	0	5	0	0	5	0
Brendan	0	0	0	3	0	0	0	0	4	5	0
Leanna	1	1	2	1	1	2	1	0	4	5	0

图14-4 一个更大的用户－菜肴矩阵，其中有很多物品都没有评分，这比一个全填充的矩阵更接近真实情况

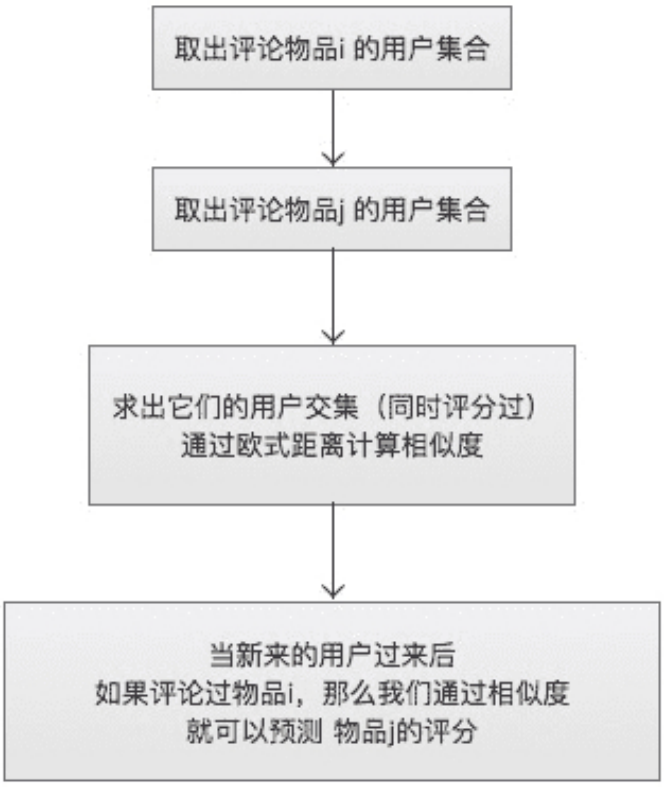
```
def loadExData3():
    # 利用SVD提高推荐效果，菜肴矩阵
    """
    行：代表人
    列：代表菜肴名词
    值：代表人对菜肴的评分，0表示未评分
    """
    return[[2, 0, 0, 4, 4, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5],
           [0, 0, 0, 0, 0, 0, 0, 1, 0, 4, 0],
           [3, 3, 4, 0, 3, 0, 0, 2, 2, 0, 0],
           [5, 5, 5, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0, 0, 5, 0, 0, 5, 0],
           [4, 0, 4, 0, 0, 0, 0, 0, 0, 0, 5],
           [0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 4],
           [0, 0, 0, 0, 0, 0, 5, 0, 0, 5, 0],
           [0, 0, 0, 3, 0, 0, 0, 0, 4, 5, 0],
           [1, 1, 2, 1, 1, 2, 1, 0, 4, 5, 0]]
```

> 分析数据: 这里不做过多的讨论(当然此处可以对比不同距离之间的差别)

> 训练算法: 通过调用 recommend() 函数进行推荐

recommend() 会调用 基于物品相似度或者是基于SVD，得到推荐的物品评分。

- 1.基于物品相似度



	鳗鱼饭	日式炸鸡排	寿司饭	烤牛肉	手撕猪肉
Jim	2	0	0	4	4
John	5	5	5	3	3
Sally	2	4	2	1	2

图14-3　用于展示相似度计算的简单矩阵

我们计算一下手撕猪肉和烤牛肉之间的相似度。一开始我们使用欧氏距离来计算。手撕猪肉和烤牛肉的欧氏距离为：

$$\sqrt{(4-4)^2+(3-3)^2+(2-1)^2}=1$$

而手撕猪肉和鳗鱼饭的欧氏距离为：

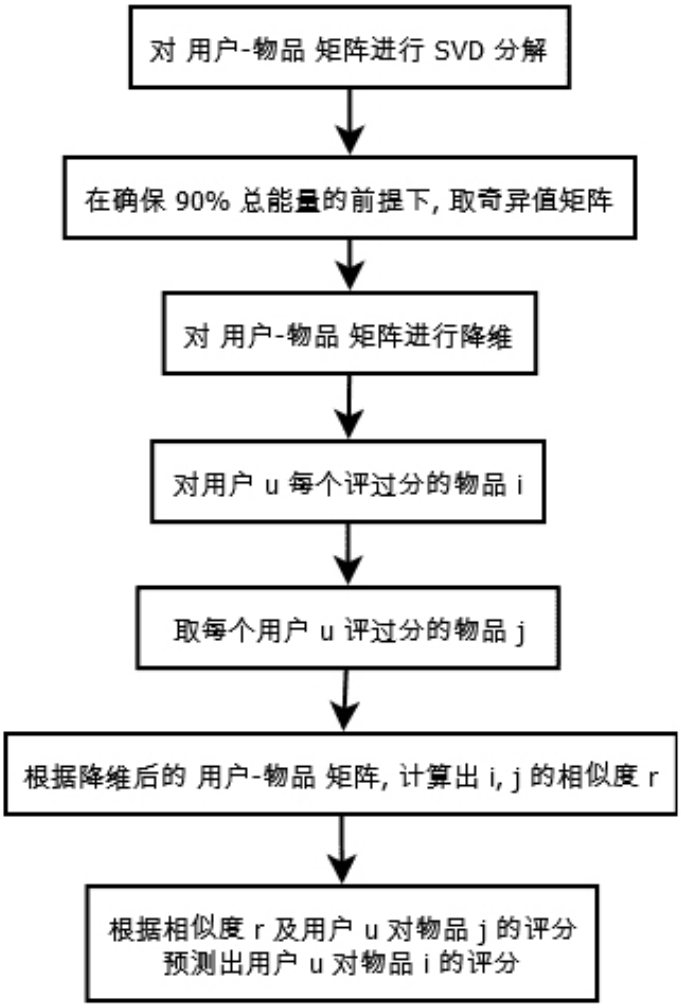
$$\sqrt{(4-2)^2+(3-5)^2+(2-2)^2}=2.83$$

```
# 基于物品相似度的推荐引擎
def standEst(dataMat, user, simMeas, item):
    """standEst(计算某用户未评分物品中，以对该物品和其他物品评分的用户的物品相似度，然后进行综合评分)

    Args:
        dataMat           训练数据集
        user              用户编号
        simMeas           相似度计算方法
        item              未评分的物品编号
    Returns:
        ratSimTotal/simTotal      评分（0～5之间的值）
```

```
    """
    # 得到数据集中的物品数目
    n = shape(dataMat)[1]
    # 初始化两个评分值
    simTotal = 0.0
    ratSimTotal = 0.0
    # 遍历行中的每个物品（对用户评过分的物品进行遍历，并将它与其他物品进行比较）
    for j in range(n):
        userRating = dataMat[user, j]
        # 如果某个物品的评分值为0，则跳过这个物品
        if userRating == 0:
            continue
        # 寻找两个用户都评级的物品
        # 变量 overLap 给出的是两个物品当中已经被评分的那个元素的索引ID
        # logical_and 计算x1和x2元素的真值。
        overLap = nonzero(logical_and(dataMat[:, item].A > 0, dataMat[:, j].A > 0))[0]
        # 如果相似度为0，则两着没有任何重合元素，终止本次循环
        if len(overLap) == 0:
            similarity = 0
        # 如果存在重合的物品，则基于这些重合物重新计算相似度。
        else:
            similarity = simMeas(dataMat[overLap, item], dataMat[overLap, j])
        # print 'the %d and %d similarity is : %f'(iten,j,similarity)
        # 相似度会不断累加，每次计算时还考虑相似度和当前用户评分的乘积
        # similarity  用户相似度,    userRating 用户评分
        simTotal += similarity
        ratSimTotal += similarity * userRating
    if simTotal == 0:
        return 0
    # 通过除以所有的评分总和，对上述相似度评分的乘积进行归一化，使得最后评分在0~5之间，这些评分用来对预测值进行排序
    else:
        return ratSimTotal/simTotal
```

- 2.基于SVD




```
# 基于SVD的评分估计
# 在recommend() 中，这个函数用于替换对standEst()的调用，该函数对给定用户给定物品构建了一个评分估计值
def svdEst(dataMat, user, simMeas, item):
    """svdEst(计算某用户未评分物品中，以对该物品和其他物品评分的用户的物品相似度，然后进行综合评分)

    Args:
        dataMat           训练数据集
        user              用户编号
        simMeas           相似度计算方法
        item              未评分的物品编号
    Returns:
        ratSimTotal/simTotal      评分（0～5之间的值）
    """
    # 物品数目
    n = shape(dataMat)[1]
    # 对数据集进行SVD分解
    simTotal = 0.0
    ratSimTotal = 0.0
    # 奇异值分解
    # 在SVD分解之后，我们只利用包含了90%能量值的奇异值，这些奇异值会以NumPy数组的形式得以保存
    U, Sigma, VT = la.svd(dataMat)

    # # 分析 Sigma 的长度取值
```

```
    # analyse_data(Sigma, 20)

    # 如果要进行矩阵运算，就必须要用这些奇异值构建出一个对角矩阵
    Sig4 = mat(eye(4) * Sigma[: 4])

    # 利用U矩阵将物品转换到低维空间中，构建转换后的物品(物品+4个主要的"隐形"特征)
    # 公式1(目的是：降维-改变形状，也改变大小)  xformedItems = dataMat.T * U[:, :4] * Sig4.I
    # 公式2(目的是：压缩-不改变形状，改变大小)       reconMat = U[:, :4] * Sig4.I * VT[:4, :]
        # 其中：imgCompress() 是详细的案例
    xformedItems = dataMat.T * U[:, :4] * Sig4.I
    # 对于给定的用户，for循环在用户对应行的元素上进行遍历，
    # 这和standEst()函数中的for循环的目的一样，只不过这里的相似度计算时在低维空间下进行的。
    for j in range(n):
        userRating = dataMat[user, j]
        if userRating == 0 or j == item:
            continue
        # 相似度的计算方法也会作为一个参数传递给该函数
        similarity = simMeas(xformedItems[item, :].T, xformedItems[j, :].T)
        # for 循环中加入了一条print语句，以便了解相似度计算的进展情况。如果觉得累赘，可以去掉
        print 'the %d and %d similarity is: %f' % (item, j, similarity)
        # 对相似度不断累加求和
        simTotal += similarity
        # 对相似度及对应评分值的乘积求和
        ratSimTotal += similarity * userRating
    if simTotal == 0:
        return 0
    else:
        # 计算估计评分
        return ratSimTotal/simTotal
```

上面的之所以使用4这个数字，是因为通过预先计算得到能满足**90%**的奇异值能量的前N个奇异值。判断计算如下：

```
# 选出奇异值能量大于90%的所有奇异值
myMat = mat(loadExData2())
U,sigma,VT = linalg.svd(myMat)
sigma = sigma**2    # 对奇异值求平方
cnt = sum(sigma)    # 所有奇异值的和
print(cnt)
value = cnt*0.9     # 90%奇异值能量
print(value)
cnt2 = sum(sigma[:3])   # 2小于90%，前3个则大于90%，所以这里选择前三个奇异值
print(cnt2)

# 541.9999999999995
# 487.7999999999996
# 500.5002891275793
```

排序获取最后的推荐结果

```
# recommend()函数，默认调用standEst()函数，产生了最高的N个推荐结果。
# 如果不指定N的大小，则默认值为3。该函数另外的参数还包括相似度计算方法和估计方法
def recommend(dataMat, user, N=3, simMeas=cosSim, estMethod=standEst):
    # 寻找未评级的物品
    # 对给定的用户建立一个未评分的物品列表
    unratedItems = nonzero(dataMat[user, :].A == 0)[1]
    # 如果不存在未评分物品，那么就退出函数
    if len(unratedItems) == 0:
        return 'you rated everything'
    # 物品的编号和评分值
    itemScores = []
    # 在未评分物品上进行循环
    for item in unratedItems:
        estimatedScore = estMethod(dataMat, user, simMeas, item)
        # 寻找前N个未评级物品，调用standEst()来产生该物品的预测得分，该物品的编号和估计值会放在一个元素列表itemScores中
        itemScores.append((item, estimatedScore))
        # 按照估计得分，对该列表进行排序并返回。列表逆排序，第一个值就是最大值
    return sorted(itemScores, key=lambda jj: jj[1], reverse=True)[: N]
```