第七章

SVD算法及其演进

教学目标

- 了解为什么引入MF(矩阵分解)技术
- 理解SVD算法原理
- 理解LFM/Funk-SVD算法原理
- 理解加入偏置项后的LFM
- 理解SVD++算法原理

7 SVD算法及其演进

- 协同过滤的局限性
  - 协同过滤非常直观，而且可解释性强，但是它也具有很多的局限性，比如它的泛化能力比较差，协同过滤无法将两个物品相似这一信息推广到其他物品的相似度计算上。这就遇到了一个严重的问题，热门物品具有很强的头部效应，容易跟大量的商品产生相似度，而长尾商品，由于其特征向量稀疏，很少与其他物品产生相似度，导致被推荐的可能性很小。但是去发掘长尾商品，增加长尾被推荐的可能性却是推荐系统最重要的目标之一。这可以被视为是协同过滤的天然缺陷，不论是基于用户还是基于商品，协同过滤对稀疏向量的处理能力很弱，所以导致推荐结果的头部效应很明显。
  - 也正是因此，才有矩阵分解技术MF被提了出来，它可以解决上述问题，并且增加泛化能力。

目录

7.1.1 线性代数意义上的SVD

- **1. 回顾特征值和特征向量**

特征值和特征向量的定义如下：

$$Ax = \lambda x$$

其中A**是一个**$n \times n$**矩阵，**x**是一个**n**维向量，则**λ**是矩阵**A**的一个特征值，而**x**是矩阵**A**的特征值**λ**所对应的特征向量。**

7.1.1 线性代数意义上的SVD

- 为什么要求特征值和特征向量呢?

求出特征值和特征向量我们就可以将矩阵A特征分解。如果我们求出了矩阵A的n个特征值 $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ ， 以及这 n 个特征值所对应的特征向量 $w_1, w_2, \ldots, w_n$ ，

那么矩阵A就可以用下式的特征分解表示:

$$A = W\Sigma W^{-1}$$

其中W是这n个特征向量所张成的n×n维矩阵，而Σ为这n个特征值为主对角线的n×n维矩阵。

7.1.1 线性代数意义上的SVD

一般我们会把W的这n个特征向量标准化，即满足 $||w_i||_2 = 1$. 或者 $w_i^T w_i = 1$ ，此时W的n个特征向量为标准正交基，满足 $W^T W = I$ ，即 $W^T = W^{-1}$ ，也就是说W为酉矩阵。

这样我们的特征分解表达式可以写成

$$A = W\Sigma W^T$$

注意要进行特征分解，矩阵A必须为方阵。

那么如果A**不是方阵**，即行和列不相同时，我们可以对矩阵进行分解吗？答案是可以，此时我们需要使用**SVD方式进行分解**。

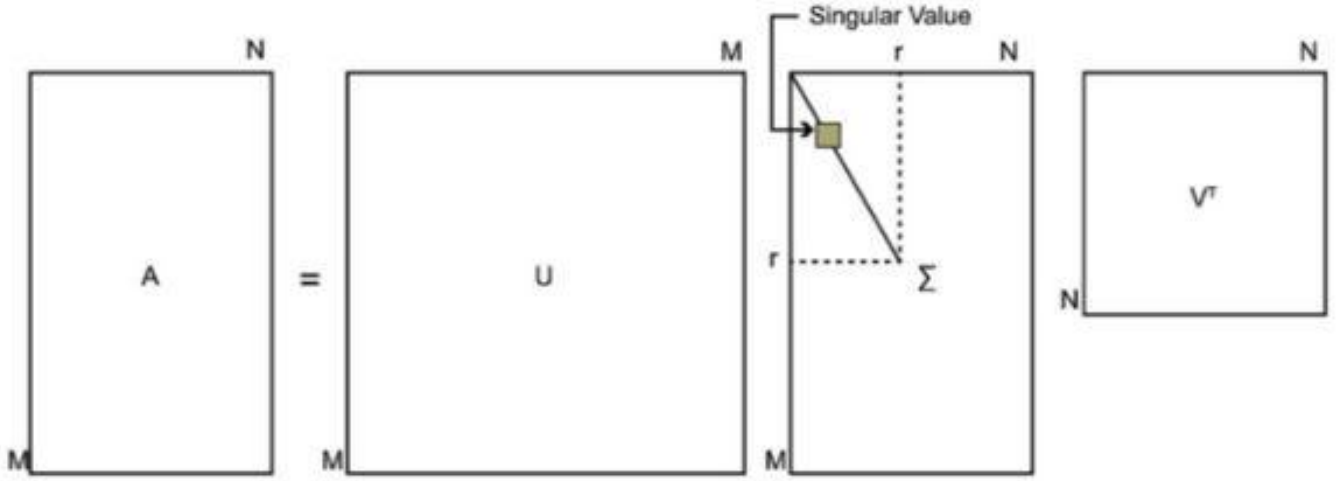
7.1.1 线性代数意义上的SVD

2. SVD的定义

SVD也是对矩阵进行分解，但是和特征分解不同，SVD并不要求要分解的矩阵为方阵。假设我们的矩阵A是一个m×n的矩阵，那么我们定义矩阵A的SVD为：

$$A = U\Sigma V^T$$

其中 U 是一个 m x m 的矩阵， Σ 是一个 m x n 的矩阵，除了主对角线上的元素以外全为0，主对角线上的每个元素都称为奇异值， V 是一个 n x n 的矩阵。 U 和 V 都是酉矩阵，即满足 $U^TU = I$ ， $V^TV = I$ 。



7.1.1 线性代数意义上的SVD

- **如何求SVD分解后的U,Σ,V这三个矩阵？**

我们将A的转置和A做矩阵乘法，那么会得到n×n的一个方阵 A^TA 。既然 A^TA 是方阵，那么我们就可以进行特征分解，得到的特征值和特征向量满足下式：

$$(A^TA)v_i = \lambda_i v_i$$

这样我们就可以得到矩阵 A^TA 的n个特征值和对应的n个特征向量v了。将 A^TA 的所有特征向量张成一个n×n的矩阵V，就是我们SVD公式里面的V矩阵了。一般我们将V中的每个特征向量叫做A的右奇异向量。

7.1.1 线性代数意义上的SVD

- **如何求SVD分解后的U,Σ,V这三个矩阵?**

如果我们将A和A的转置做矩阵乘法，那么会得到m×m的一个方阵 AA^T 。 既然 AA^T 是方阵，那么我们就可以进行特征分解，得到的特征值和特征向量满足下式:

$$(AA^T)u_i = \lambda_i u_i$$

这样我们就可以得到矩阵 AA^T 的m个特征值和对应的m个特征向量u了。将 AA^T 的所有特征向量张成一个m×m的矩阵U，就是我们SVD公式里面的U矩阵了。一般我们将U中的每个特征向量叫做A的左奇异向量。

7.1.1 线性代数意义上的SVD

- **如何求SVD分解后的U,Σ,V这三个矩阵?**

  U和V我们都求出来了，现在就剩下奇异值矩阵Σ没有求出了.

  由于Σ除了对角线上是奇异值其他位置都是0，那我们只需要求出每个奇异值σ就可以了。

  我们注意到:

$$A = U\Sigma V^T \Rightarrow AV = U\Sigma V^T V \Rightarrow AV = U\Sigma \Rightarrow Av_i = \sigma_i u_i \Rightarrow \sigma_i = Av_i / u_i$$

这样我们可以求出我们的每个奇异值，进而求出奇异值矩阵Σ。

7.1.1 线性代数意义上的SVD

- **为什么 A^TA 的特征向量组成的就是我们SVD中的V矩阵，而 AA^T 的特征向量组成的就是我们SVD中的U矩阵呢？**

  以V矩阵为例，证明过程如下：

$$A = U\Sigma V^T \Rightarrow A^T = V\Sigma U^T \Rightarrow A^TA = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T$$

上式证明使用了 $W^TW = I \quad \Sigma^T = \Sigma$ 。可以看出 A^TA 的特征向量组成的的确就是我们SVD中的V矩阵。类似的方法可以得到 AA^T 的特征向量组成的就是我们SVD中的U矩阵。

我们还可以看出我们的特征值矩阵等于奇异值矩阵的平方，也就是说特征值和奇异值满足如下关系：

$$\sigma_i = \sqrt{\lambda_i}$$

这样也就是说，我们可以不用 $\sigma_i = \frac{Av_i}{u_i}$ 来计算奇异值，也可以通过求出 A^TA 的特征值取平方根来求奇异值。

7.1.1 线性代数意义上的SVD

● SVD的一些性质

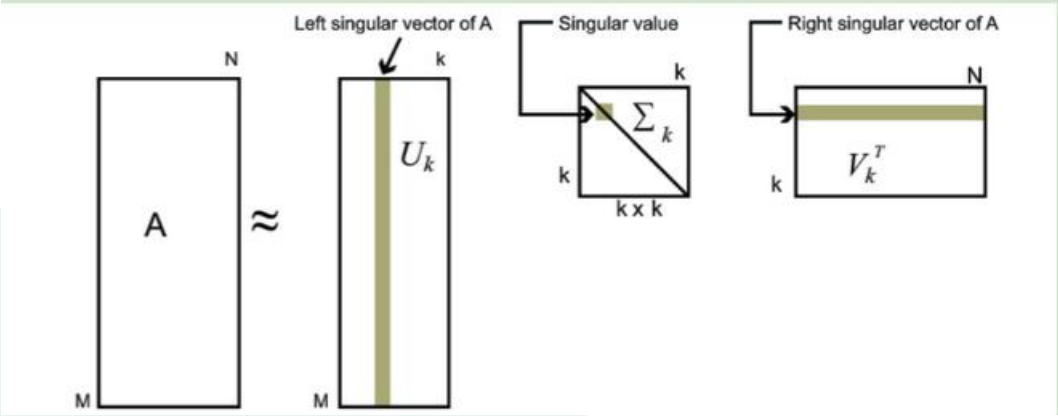
对于奇异值,它跟我们特征分解中的特征值类似，在奇异值矩阵中也是按照从大到小排列，而且奇异值的减少特别的快，在很多情况下，前10%甚至1%的奇异值的和就占了全部的奇异值之和的99%以上的比例。

也就是说，我们也可以用最大的k个的奇异值和对应的左右奇异向量来近似描述矩阵。

也就是说:

$$A_{m\times n} = U_{m\times m}\,\Sigma_{m\times n}\,V^T_{n\times n} \approx U_{m\times k}\,\Sigma_{k\times k}\,V^T_{k\times n}$$

其中k要比n小很多，也就是一个大的矩阵A可以用三个小的矩阵 $U_{m\times k}, \sum_{k\times k}, V^T_{k\times n}$ 来表示。



由于这个重要的性质，SVD可以用于PCA降维，来做数据压缩和去噪。也可以用于推荐算法，将用户和喜好对应的矩阵做特征分解，进而得到隐含的用户需求来做推荐。同时也可以用于NLP中的算法，比如潜在语义索引（LSI）。

7.1.2 SVD在推荐系统中的应用

- 假设有评分矩阵如下：

User\Item	item1	item2	item3	item4
user1	5	5	?	5
user2	5	?	3	4
user3	3	4	?	3
user4	?	?	5	3
user5	5	4	4	5
user6	5	4	5	5

“?”代表用户对该Item没有评分。实际上在真实的推荐系统中，我们会得到一个很大的稀疏矩阵。

7.1.2 SVD在推荐系统中的应用

- SVD分解需要矩阵是稠密矩阵，所以我们需要对上面的矩阵做填充

User\Item	item1	item2	item3	item4	item5
user1	1	5	0	5	4
user2	5	4	4	3	2
user3	0	4	0	0	5
user4	4	4	1	4	0
user5	0	4	3	5	0
user6	2	4	3	5	3

我们这里使用0来填充空白项，也有用平均值来填充的。

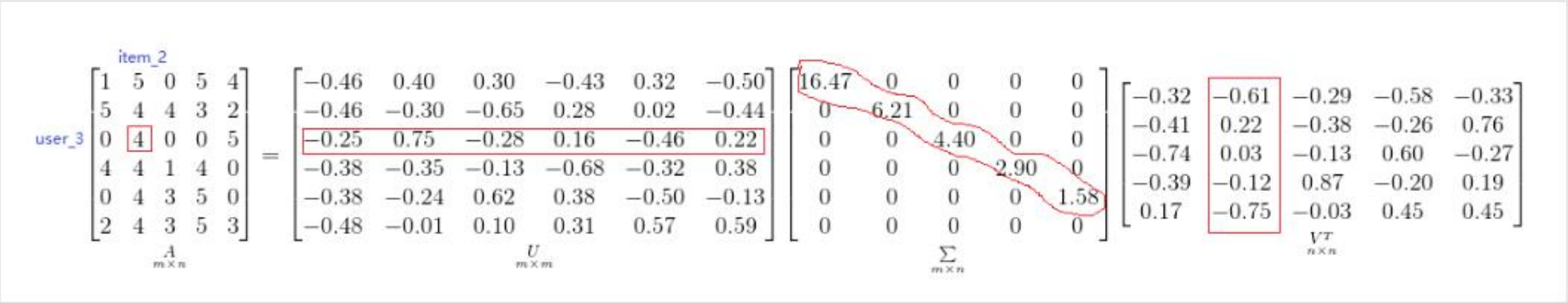
7.1.2 SVD在推荐系统中的应用

- 使用svd分解得:

$$
\underset{m\times n}{A}=\begin{bmatrix}1&5&0&5&4\\5&4&4&3&2\\0&4&0&0&5\\4&4&1&4&0\\0&4&3&5&0\\2&4&3&5&3\end{bmatrix}=\underset{m\times m}{U}\begin{bmatrix}-0.46&0.40&0.30&-0.43&0.32&-0.50\\-0.46&-0.30&-0.65&0.28&0.02&-0.44\\-0.25&0.75&-0.28&0.16&-0.46&0.22\\-0.38&-0.35&-0.13&-0.68&-0.32&0.38\\-0.38&-0.24&0.62&0.38&-0.50&-0.13\\-0.48&-0.01&0.10&0.31&0.57&0.59\end{bmatrix}\underset{m\times n}{\sum}\begin{bmatrix}16.47&0&0&0&0\\0&6.21&0&0&0\\0&0&4.40&0&0\\0&0&0&2.90&0\\0&0&0&0&1.58\\0&0&0&0&0\end{bmatrix}\underset{n\times n}{V^T}\begin{bmatrix}-0.32&-0.61&-0.29&-0.58&-0.33\\-0.41&0.22&-0.38&-0.26&0.76\\-0.74&0.03&-0.13&0.60&-0.27\\-0.39&-0.12&0.87&-0.20&0.19\\0.17&-0.75&-0.03&0.45&0.45\end{bmatrix}
$$

7.1.2 SVD在推荐系统中的应用

- 对于评分矩阵，该分解意味着什么呢？我们来计算一下user_3对item_2的评分



$$
\begin{bmatrix} 1 & 5 & 0 & 5 & 4 \\ 5 & 4 & 4 & 3 & 2 \\ 0 & \boxed{4} & 0 & 0 & 5 \\ 4 & 4 & 1 & 4 & 0 \\ 0 & 4 & 3 & 5 & 0 \\ 2 & 4 & 3 & 5 & 3 \end{bmatrix}_{A_{m\times n}} = \begin{bmatrix} -0.46 & 0.40 & 0.30 & -0.43 & 0.32 & -0.50 \\ -0.46 & -0.30 & -0.65 & 0.28 & 0.02 & -0.44 \\ \boxed{-0.25} & \boxed{0.75} & \boxed{-0.28} & \boxed{0.16} & \boxed{-0.46} & \boxed{0.22} \\ -0.38 & -0.35 & -0.13 & -0.68 & -0.32 & 0.38 \\ -0.38 & -0.24 & 0.62 & 0.38 & -0.50 & -0.13 \\ -0.48 & -0.01 & 0.10 & 0.31 & 0.57 & 0.59 \end{bmatrix}_{U_{m\times m}} \begin{bmatrix} 16.47 & 0 & 0 & 0 & 0 \\ 0 & 6.21 & 0 & 0 & 0 \\ 0 & 0 & 4.40 & 0 & 0 \\ 0 & 0 & 0 & 2.90 & 0 \\ 0 & 0 & 0 & 0 & 1.58 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{\Sigma_{m\times n}} \begin{bmatrix} -0.32 & \boxed{-0.61} & -0.29 & -0.58 & -0.33 \\ -0.41 & \boxed{0.22} & -0.38 & -0.26 & 0.76 \\ -0.74 & \boxed{0.03} & -0.13 & 0.60 & -0.27 \\ -0.39 & \boxed{-0.12} & 0.87 & -0.20 & 0.19 \\ 0.17 & \boxed{-0.75} & -0.03 & 0.45 & 0.45 \end{bmatrix}_{V^T_{n\times n}}
$$

=>

$$
4 = (-0.25)\times 16.47\times(-0.61)+0.75\times 6.21\times 0.22+(-0.28)\times 4.4\times 0.03+ \\ 0.16\times 2.9\times(-0.12)+(-0.46)\times 1.58\times(-0.75)
$$

也就是说A中各个位置的元素来自于U的对应位置行向量和V^T的对应位置列向量按照奇异值进行加权点积。

7.1.2 SVD在推荐系统中的应用

将U的行向量看作user向量在新特征下的表示，V^T的列向量看作item向量在新特征下的表示，同时只截取权重最大的前两个特征，对数据进行降维：

$$
\underset{m\times n}{A}=\begin{bmatrix}1&5&0&5&4\\5&4&4&3&2\\0&4&0&0&5\\4&4&1&4&0\\0&4&3&5&0\\2&4&3&5&3\end{bmatrix}=
\begin{array}{c}\\ \text{user\_1}\\ \text{user\_2}\\ \text{user\_3}\\ \text{user\_4}\\ \text{user\_5}\\ \text{user\_6}\end{array}
\begin{array}{c}\begin{matrix}\text{特征1}&\text{特征2}&\text{特征3}&\text{特征4}&\text{特征5}&\text{特征6}\end{matrix}\\ \begin{bmatrix}-0.46&0.40&0.30&-0.43&0.32&-0.50\\-0.46&-0.30&-0.65&0.28&0.02&-0.44\\-0.25&0.75&-0.28&0.16&-0.46&0.22\\-0.38&-0.35&-0.13&-0.68&-0.32&0.38\\-0.38&-0.24&0.62&0.38&-0.50&-0.13\\-0.48&-0.01&0.10&0.31&0.57&0.59\end{bmatrix}\end{array}
\underset{m\times m}{U}
\begin{array}{c}\begin{matrix}\text{权重1}&\text{权重2}&\text{权重3}&\text{权重4}&\text{权重5}\end{matrix}\\ \begin{bmatrix}16.47&0&0&0&0\\0&6.21&0&0&0\\0&0&4.40&0&0\\0&0&0&2.90&0\\0&0&0&0&1.58\\0&0&0&0&0\end{bmatrix}\end{array}
\underset{m\times n}{\Sigma}
\begin{array}{c}\begin{matrix}\text{item\_1}&\text{item\_2}&\text{item\_3}&\text{item\_4}&\text{item\_5}\end{matrix}\\ \begin{bmatrix}-0.32&-0.61&-0.29&-0.58&-0.33\\-0.41&0.22&-0.38&-0.26&0.76\\-0.74&0.03&-0.13&0.60&-0.27\\-0.39&-0.12&0.87&-0.20&0.19\\0.17&-0.75&-0.03&0.45&0.45\end{bmatrix}\end{array}
\begin{array}{l}\\ \text{特征1}\\ \text{特征2}\\ \text{特征3}\\ \text{特征4}\\ \text{特征5}\end{array}
\underset{n\times n}{V^T}
$$

=>

$$
\underset{m\times n}{\tilde{A}}=
\begin{array}{c}\\ \text{user\_1}\\ \text{user\_2}\\ \text{user\_3}\\ \text{user\_4}\\ \text{user\_5}\\ \text{user\_6}\end{array}
\begin{array}{c}\begin{matrix}\text{特征1}&\text{特征2}\end{matrix}\\ \begin{bmatrix}-0.46&0.40\\-0.46&-0.30\\-0.25&0.75\\-0.38&-0.35\\-0.38&-0.24\\-0.48&-0.01\end{bmatrix}\end{array}
U
\begin{array}{c}\begin{matrix}\text{权重1}&\text{权重2}\end{matrix}\\ \begin{bmatrix}16.47&0\\0&6.21\end{bmatrix}\end{array}
\Sigma
\begin{array}{c}\begin{matrix}\text{item\_1}&\text{item\_2}&\text{item\_3}&\text{item\_4}&\text{item\_5}\end{matrix}\\ \begin{bmatrix}-0.32&-0.61&-0.29&-0.58&-0.33\\-0.41&0.22&-0.38&-0.26&0.76\end{bmatrix}\end{array}
\begin{array}{l}\text{特征1}\\ \text{特征2}\end{array}
V^T
$$

7.1.2 SVD在推荐系统中的应用

- 在此维度下，寻找与user5最接近的user。每个用户向量抽离出来是2维向量：

  user1 = (-0.46, 0.40)
  user2 = (-0.46, -0.30)
  . .
  . .
  . .
  user6 = (-0.48, -0.01)

- 计算user5与其他用户的相似度。计算相似度的方法有很多种，比如：余弦相似度、欧氏距离、皮尔逊相关系数等。我们这里使用欧氏距离，即：$d = \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2}$ 欧式距离越小，向量相似度越高。
- 计算得知user2与user5相似度最高，所以回到原矩阵查看user2对item1和item5的评分：item1评分更高，所以将item1推荐给user5。

7.1.2 SVD在推荐系统中的应用

- 向新用户推荐item

如果现在有一个新的用户X=(2,0,5,2,0)，它的数据并不在上面的样本矩阵里，怎么办？

直接加入原样本数据进行扩展再求解？可以，但是这样做在大数据情况下时间消耗太大。我们需要基于已经有的模型对新加入的用户进行item推荐。

首先，将新用户投影到降维后的user向量空间中去，用如下公式：

$$X_k = X \underset{n\times k}{V} \sum_{k\times k}^{-1}$$

计算过程如下：

$$X_2 = (2,0,5,2,0)\begin{bmatrix} -0.32 & -0.41 \\ -0.61 & 0.22 \\ -0.29 & -0.38 \\ -0.58 & -0.26 \\ -0.33 & 0.76 \end{bmatrix}\begin{bmatrix} 16.47 & 0 \\ 0 & 6.21 \end{bmatrix}^{-1} = (-0.20, -0.52)$$

接下来通过欧式距离或者其它方式,找出user1~user6中与$\mathbf{X}_k$相似度最高且对item2和item5评过分的user，然后把该user对item2和item5中评分更高的item推荐给X用户。

7.1.2 SVD在推荐系统中的应用

- 以上对新用户做推荐的公式是怎么来的?

  将SVD分解的原始矩阵A和分解矩阵U都写成行向量形式:

$$A = U\sum V^T$$

=>

$$\begin{pmatrix} A_1 \\ A_2 \\ \dots \\ A_m \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ \dots \\ u_m \end{pmatrix} \sum V^T = \begin{pmatrix} u_1 \sum V^T \\ u_2 \sum V^T \\ \dots \\ u_m \sum V^T \end{pmatrix}$$

=>

$$A_i = u_i \sum V^T , \quad i = 1, 2, \dots, m$$

=>

$$u_i = A_i (V^T)^{-1} (\sum)^{-1}$$

=>

$$u_i = A_i V (\sum)^{-1}$$

上述即原始user向量向新特征下的user向量的转换公式。

7.1.2 SVD在推荐系统中的应用

将SVD分解的原始矩阵A和分解矩阵都写成列向量形式：

$$A = U \sum V^T$$

=>

$$(A_1, A_2, ..., A_n) = U \sum (v_1, v_2, ..., v_n)$$

=>

$$A_i = U \sum v_i \ , \quad i = 1, 2, ..., n$$

=>

$$v_i = (\sum)^{-1} U^{-1} A_i$$

=>

$$v_i^T = A_i^T (U^{-1})^T ((\sum)^{-1})^T$$

=>

$$v_i^T = A_i^T U (\sum)^{-1}$$

上述即原始item向量向新特征下的item向量的转换公式。

目录

7.2 推荐系统对SVD的改进-LFM算法

- 传统SVD算法有什么缺点：

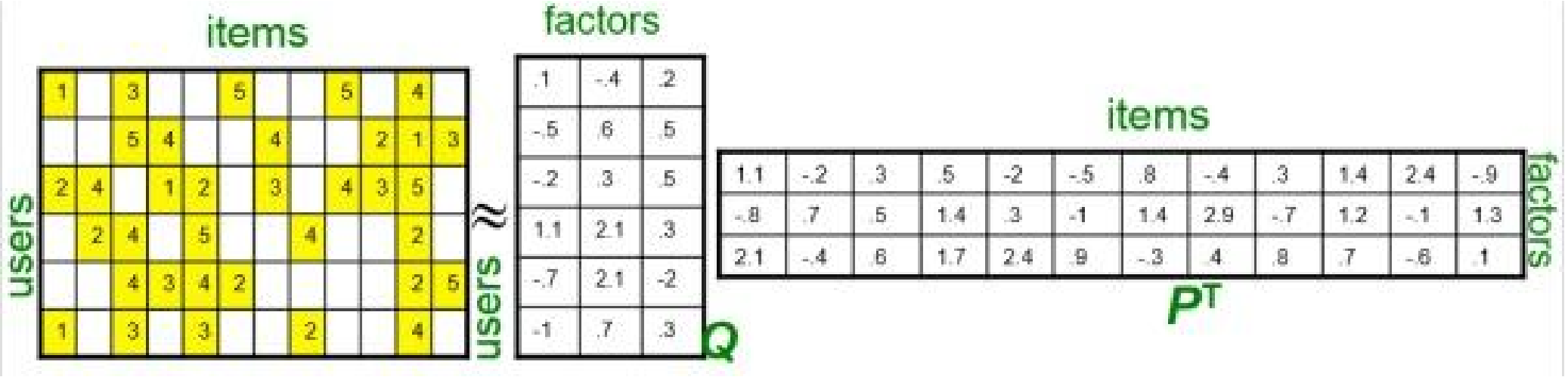
  1、时间复杂度比较高(O(n^3))

  假设矩阵A为m*n，则矩阵A的转置B为n*m，则AxB，如下计算过程：
  矩阵A中第一行的元素与矩阵B的第一列元素对应相乘，得到
  结果中的第一行的第一个元素，所以要得到一个元素需要进行n次乘法运算，所以要得到第一行m个元素需要进行n*m次运算，故总的需要n*m*m次乘法运算。

  2、要求原始矩阵必须是稠密矩阵

7.2 推荐系统对SVD的改进-LFM算法

- LFM算法，又叫Funk-SVD，其矩阵分解方式如下图所示：



  - 上图的大矩阵可以就是user-item的评分矩阵，而后面的就是分解出来的两个小矩阵分别为Q矩阵和P矩阵。
  - 其中 Q矩阵是用户的因子矩阵；而P矩阵就是物品的因子矩阵。
  - 两个小矩阵通过K个隐因子相连，最后通过最优化的方式计算出小矩阵的元素。
  - 从物理意义上讲，可以理解为这是将稀疏的大矩阵中的信息，浓缩到了两个小矩阵中。

7.2 推荐系统对SVD的改进-LFM算法

- 假设我们已有有了一个评分矩阵 $\mathrm{R}_{m,n}$，m个用户对n个物品的评分全在这个矩阵里，这是一个高度稀疏的矩阵，我们用 $\mathrm{r}_{u,i}$ 表示用户u对物品i的评分。
- LFM认为 $\mathrm{R}_{m,n} = P_{m,F} \cdot Q_{F,n}$ 即R是两个矩阵的乘积；F是隐因子的个数；P的每一行代表一个用户对各个隐因子的喜好程度；Q的每一列代表一个物品在各个隐因子上的概率分布。
- 下面的公式即为LFM的模型：$\hat{r}$ 表示的是预测评分

$$\hat{r}_{ui} = \sum_{f=1}^{F} P_{uf} Q_{fi} \tag{1}$$

7.2 推荐系统对SVD的改进-LFM算法

- $r_{u,i}$ 表示用户u对物品i的真实评分，$\hat{r}_{u,i}$ 表示用户u对物品i的预测评分，则LFM的目标函数如下：

$$min: \quad Loss = \sum_{r_{ui} \neq 0} (r_{u,i} - \hat{r}_{ui})^2 \tag{2}$$

- 为了防止overfitting，添加正则项控制过拟合

$$min: \quad Loss = \sum_{r_{ui} \neq 0} (r_{u,i} - \hat{r}_{ui})^2 + \lambda(\sum P_{uf}^2 + \sum Q_{fi}^2) = f(P, Q) \tag{3}$$

λ是一个超参数，需要根据具体应用场景反复实验得到。

7.2 推荐系统对SVD的改进-LFM算法

- 求解最优解的方式采用梯度下降法，下面采用梯度下降法求解上面的无约束最优化

$$P^{(t+1)} = P^{(t)} - \alpha \frac{\partial Loss}{\partial P^{(t)}}, Q^{(t+1)} = Q^{(t)} - \alpha \frac{\partial Loss}{\partial Q^{(t)}} \tag{4}$$

$$\frac{\partial Loss}{\partial P^{(t)}} = \begin{bmatrix} \frac{\partial Loss}{\partial P_{11}^{(t)}} & \cdots & \frac{\partial Loss}{\partial P_{1F}^{(t)}} \\ \cdots & \frac{\partial Loss}{\partial P_{uf}^{(t)}} & \cdots \\ \frac{\partial Loss}{\partial P_{m1}^{(t)}} & \cdots & \frac{\partial Loss}{\partial P_{mF}^{(t)}} \end{bmatrix} \tag{5}$$

$$\frac{\partial Loss}{\partial P_{uf}^{(t)}} = \sum_{i, r_{ui} \neq 0} -2(r_{u,i} - \hat{r}_{ui}) \frac{\partial \hat{r}_{ui}}{\partial P_{uf}^{(t)}} + 2\lambda P_{uf}^{(t)} = \sum_{i, r_{ui} \neq 0} -2(r_{u,i} - \hat{r}_{ui}) Q_{fi}^{(t)} + 2\lambda P_{uf}^{(t)} \tag{6}$$

$$\frac{\partial Loss}{\partial Q_{fi}^{(t)}} = \sum_{u, r_{ui} \neq 0} -2(r_{u,i} - \hat{r}_{ui}) \frac{\partial \hat{r}_{ui}}{\partial Q_{fi}^{(t)}} + 2\lambda Q_{fi}^{(t)} = \sum_{u, r_{ui} \neq 0} -2(r_{u,i} - \hat{r}_{ui}) P_{uf}^{(t)} + 2\lambda Q_{fi}^{(t)} \tag{7}$$

以上就是梯度下降法的所有公式，我们注意到：

1. 求$\frac{\partial Loss}{\partial P_{uf}^{(t)}}$时用到了用户$u$对物品的所有评分
2. 求$\frac{\partial Loss}{\partial P^{(t)}}$时用到了整个评分矩阵$R$，时间复杂度为$m \times F \times n'$，$n'$是平均一个用户对多少个物品有过评分

7.2 推荐系统对SVD的改进-LFM算法

- 我们采用随机梯度下降法，随机梯度下降法（SGD）没有严密的理论证明，但是在实践中他通常比传统的梯度下降法需要更少的迭代次数就可以收敛，他有两个特点：

1. 单独更新参数$P_{uf}^{(t+1)} = P_{uf}^{(t)} - \alpha \frac{\partial Loss}{\partial P_{uf}^{(t)}}$，而原始的梯度下降法要整体更新参数$P^{(t+1)} = P^{(t)} - \alpha \frac{\partial Loss}{\partial P^{(t)}}$。

   在$t+1$轮次中计算其他参数的梯度时直接使用P_{uf}的最新值$P_{uf}^{(t+1)}$

2. 计算$\frac{\partial Loss}{\partial P_{uf}^{(t)}}$时只利用用户$u$对一个物品的评分，而不是利用用户$u$的所有评分，即

$$\frac{\partial Loss}{\partial P_{uf}^{(t)}} = -2(r_{u,i} - \hat{r}_{ui})Q_{fi}^{(t)} + 2\lambda P_{uf}^{(t)} \tag{8}$$

从而

$$P_{uf}^{(t+1)} = P_{uf}^{(t)} + \alpha[(r_{u,i} - \hat{r}_{ui})Q_{fi}^{(t)} - \lambda P_{uf}^{(t)}] \tag{9}$$

同理可得

$$Q_{fi}^{(t+1)} = Q_{fi}^{(t)} + \alpha[(r_{u,i} - \hat{r}_{ui})P_{uf}^{(t+1)} - \lambda Q_{fi}^{(t)}] \tag{10}$$

7.2 推荐系统对SVD的改进-LFM算法

- SGD单轮迭代的时间复杂度也是 m * F * n'，但由于它是单个参数地更新，且更新单个参数时只利用到一个样本（一个评分），更新后的参数立即可用于更新剩下的参数，所以SGD比传统的梯度下降需要更少的迭代次数。
- 在训练模型的时候我们只要求模型尽量拟合 $r_{u,i}$ 不为0的位置即可，对于为0的情况我们也不希望 $\hat{r}_{u,i}$ 为0，因为 $\hat{r}_{u,i}$ 为0只表示用户没有评价过，并不代表用户对物品的喜好为0。而恰恰 $\hat{r}_{u,i}$ 能反映用户u对物品i的喜好程度，然后根据喜好程度排序，就可以完成推荐。

目录

7.3 加入偏置项后的LFM

- 偏置：把独立于用户或者独立于物品的的因素称为偏置部分。
- 个性化：将用户和物品的交互，也就是表示用户对物品的喜好的部分称为个性化部分。
- 在Netflix Prize推荐比赛中，Yehuda Koren仅使用偏置部分可以将评分误差降低32%，而加入个性化部分能降低42%，也就是说只有10%是个性化部分的作用，这也说明了偏置部分的重要性，剩下的58%的误差Yehuda Koren将其称之为模型不可解释部分，包括数据的噪音等因素。
- 将上面的公式（1）中加入偏置

$$\hat{r}_{ui} = \sum_{f=1}^{F} P_{uf} Q_{fi} + \mu + b_u + b_i \tag{11}$$

7.3 加入偏置项后的LFM

- 对偏置的解释：
  - μ 代表训练集中所有评分记录的全局平均数，表示了训练数据的总体评分情况，对于固定的数据集，他是一个常数。
  - b u 代表用户偏置，独立于物品特征的因素，表示某一特定用户的打分习惯。比如，对于性格较为严苛的用户，打分会比较低；对于较为温和的用户，打分会偏高。
  - b i代表物品偏置，独立于用户兴趣的因素，表示一特定物品得到打分的情况。比如，好的影片的总体评分偏高，而烂片获得的评分普遍偏低，物品偏置捕获的就是这样的特征。
  - 其中要说明的一点，μ 是一个统计值； b u 和 b i 需要通过模型训练得到，那么如此对比上面的公式（3）目标函数变为：

$$min: \quad Loss = \sum_{r_{ui} \neq 0} (r_{u,i} - \hat{r}_{ui})^2 + \lambda(\sum P_{uf}^2 + \sum Q_{fi}^2 + \sum b_u^2 + \sum b_i^2) \qquad (12)$$

7.3 加入偏置项后的LFM

- 由梯度下降法得到 b u 和 b i 的更新方式:

$$b_u^{(t+1)} = b_u^{(t)} + \alpha * (r_{u,i} - \hat{r}_{ui} - \lambda * b_u^{(t)}) \tag{13}$$

$$b_i^{(t+1)} = b_i^{(t)} + \alpha * (r_{u,i} - \hat{r}_{ui} - \lambda * b_i^{(t)}) \tag{14}$$

- 个性化部分的更新方式不变。初始化时 b u 和 b i 全部初始化为0即可。

目录

7.4 引入隐反馈的SVD++算法

- 由带偏置的LFM继续进化，就可以得到SVD++。
- 值得注意的是，在推荐系统中使用的SVD和线性代数中的SVD并不完全相同。
- 在实际的生产中，用户的评分数据很稀少，也就是说显示反馈比隐式反馈要少很多，那么可以不可以把隐式反馈的因素加入到模型呢？答案是肯定的。
- SVD++就是在模型融入了隐式反馈的因素，索引评分数据可以理解为：评分 = 显示兴趣 + 隐式兴趣 + 偏见。
- 从另一个角度来看，任何用户只要对物品i有过评分，不管评分多少，就已经在一定程度上反映了他对各个隐因子的喜好程度 $y_i = (y_{i1}, y_{i2}, y_{i3} \cdots\cdots y_{iF})$，y 是物品所携带的属性，就如同Q一样，在公式（11）的基础上，SVD++进化为：

$$\hat{r}_{ui} = \sum_{f=1}^{F} (P_{uf} + \frac{\sum_{j \in N(u)} Y_{jf}}{\sqrt{|N(u)|}}) Q_{fi} + \mu + b_u + b_i \qquad (15)$$

7.4 引入隐反馈的SVD++算法

- N(u)是用户u评价过的物品集合。
- 和上面的方式一样，还是基于评分的误差平方和建立目标函数，正则项里加一个 $\lambda\Sigma Y_{if}^2$ ，采用随机梯度下降法优化。其他项不变，隐式兴趣部分优化。

$$\frac{\partial \hat{r_{ui}}}{\partial Q_{fi}} = P_{uf} + \frac{\sum_{j \in N(u)} Y_{jf}}{\sqrt{|N(u)|}} \tag{16}$$

- 另外引入了 Y 矩阵，所以也需要计算它的偏导：

$$\frac{\partial \hat{r_{ui}}}{\partial Y_{jf}} = \frac{Q_{fi}}{\sqrt{|N(u)|}} \tag{17}$$

内容回顾

1、线性代数意义上的SVD

2、推荐系统对SVD的改进-LFM算法

3、加入偏置项后的LFM

4、引入隐反馈的SVD++算法

Thank you!