矩阵分解（matrix_factorization）

传统的协同过滤算法（CF）Collaborative Filter 是基于启发式计算用户的相似度，例如余玄相似度，推荐是根据合计改查询用户的k个相似的用户的打分。

但是近些年来，矩阵分解（MF）成了CF的主要方法，原始的MF模型设计通过去映射user和item到一个隐因子空间而建模用户的显示反馈，这样user-item的关系就可以通过它们之间的点积所捕获。

rij表示useri对itemj的打分，我们学习到了用户向量ui∈Rr item向量vj∈R,然后用他们的点积去近似rij。然后我们只需要在已知的数据集上去优化均方误差 [

$$\min_{\mathbf{u}_*,\mathbf{v}_*} \sum_{r_{ij}\in\mathcal{K}} (r_{ij} - \mathbf{u}_i^T\mathbf{v}_j)^2 + \lambda_u\|\mathbf{u}_i\|^2 + \lambda_v\|\mathbf{v}_i\|^2,$$

where $\mathcal{K}$ is the set of known ratings; λ_u and λ_v are hyper-parameters that regularize the L^2-norm of $\mathbf{u}_*$ and $\mathbf{v}_*$.

隐反馈：直接应用到传统的MF的隐式反馈式存在问题的，通过隐式反馈我们只观察到正反馈。我们不能忽视哪些未观察的到的user-item交互，否则会导致不准确的问题。同时我们也不能假定这些未观察到的交互都是负反馈，因为我们不知道这些交互没有发生是因为用户不喜欢他们还是暂时没有什么想法（可能以后买）。

Weighted regularized matrix factorization(WRMF)将所有的未观察到的user-item交互看成是负样例,然后使用权重cijcij去减少这些不确定样本的影响。 [

$$\min_{\mathbf{u}_*,\mathbf{v}_*} \sum_{r_{ij}\in\mathcal{K}} c_{ij}(r_{ij} - \mathbf{u}_i^T\mathbf{v}_j)^2 + \lambda_u\|\mathbf{u}_i\|^2 + \lambda_v\|\mathbf{v}_i\|^2,$$

where case weight c_{ij} is larger for observed positive feedback and smaller for unobserved interactions.

上面的讨论表明了,通过隐式反馈,rating开始变得不准确了。最近的一些MF模型开始从估计一组明确的评级转为建模item之间的相对偏好。 Bayesian Personalized Ranking 就是这种方法。 [

Let $\mathcal{D}_i$ be a set of item pairs (j, k) where user i has interacted with item j but not item k, assuming user i might be more interested in item j than item k, BPR minimizes the pairwise ranking loss:

$$\min_{\mathbf{u}_*,\mathbf{v}_*} \sum_{i\in\mathcal{I}} \sum_{(j,k)\in\mathcal{D}_i} -log\ \sigma(\mathbf{u}_i^T\mathbf{v}_j - \mathbf{u}_i^T\mathbf{v}_k) + \lambda_u\|\mathbf{u}_i\|^2 + \lambda_v\|\mathbf{v}_j\|^2,$$

where σ is the sigmoid function.

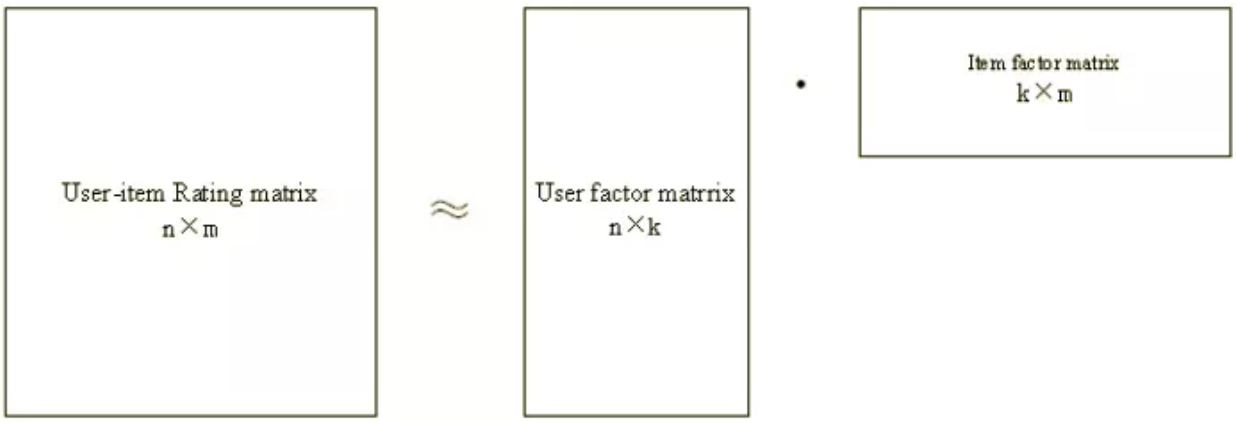
BPR损失函数存在一个问题,对low-rank item惩罚不足。

假设我先在有一个关于用户对音乐评分的矩阵如下图：

	音乐1	音乐2	音乐3	音乐4	音乐5	音乐6	音乐7	音乐8	音乐9	音乐10	音乐11	音乐12	音乐13
用户1	5					-5			5	3		1	5
用户2				3					3				4
用户3			1		2	-5	4			-2	-2		-2
用户4		4	4	3			-2		-5			3	
用户5		5	-5		-5		4	3			4		
用户6			4			3			4				
用户7		-2				5				4		4	-2
用户8		-2				5		5		4			-2

这就是上面讲到的稀疏数据

只有上述的数据是很难使用户相互推荐音乐的，因为可以看出用户本身听过的歌就不够多，那么如何使数据更加“饱满”呢？这时正是需要矩阵分解的时候，矩阵分解算法的数学理论基础是矩阵的行列变换。行列变换中又有以下规则，我们知道矩阵A进行行变换相当于A左乘一个矩阵，矩阵A进行列变换等价于矩阵A右乘一个矩阵，因此矩阵A可以表示为A=PEQ=PQ（E是标准阵）。



矩阵分解的目的就是把一个稀疏的用户评分矩阵分解成用户因子矩阵和项目因子矩阵相乘的形式R=U(转置)*I，我们的目的就是最后再让两个因子矩阵反乘回去得到饱满的用户评分矩阵。那么这个用户,项目因子是个什么东西呢？我们接着上面的音乐评分的形式说，一首歌可能包含多种音乐风格，我们可以量化风格，体现各种风格在一首歌中的比重，那么这里的“潜在因子”我们就可以当作“音乐风格”，K个因子就可以看作K种风格。譬如下图：

	小清新	重口味	优雅	伤感	五月天
张三	0.6	0.8	0.1	0.1	0.7
李四	0.1	0	0.9	0.1	0.2
王五	0.5	0.7	0.9	0.9	0

	小清新	重口味	优雅	伤感	五月天
音乐A	0.9	0.1	0.2	0.4	0
音乐B	0.5	0.6	0.1	0.9	1
音乐C	0.1	0.2	0.5	0.1	0

可以说，这些因子就是我们的模型中的重要参数，个人理解分解出来的这两个因子矩阵就可以说是基于模型的CF中的，“模型”的了，其实我觉得可以类比线性模型中的参数，我们的回归模型最终重要的不就是公式中的各项参数吗，这两个因子矩阵其实就是我们这个模型中的重要参数，参数知道了模型也就求出来了。

矩阵分解的缺陷

撇开矩阵分解对推荐任务的成功，它并非没有缺陷。 一个主要问题是点积限制了矩阵分解的表达[18]。 点产品关心两个向量的宏观和角度。 在某种程度上，它在大小和角度3方面测量了两个方面的相似性而不是距离。 这里我们从[18]中扩展一个例子。 如图1所示，我们尝试使用用户潜在因素（P1，P2，P3，P4）来模拟用户的相似性并将其与用户相似度由来自交互矩阵的Jaccard相似度计算。假设我们有三个用户: u1、U2和3，基于交互矩阵的相似性是: s23> s12> s13，那么相应的用户潜在因素sP1、P2P和3c会被定位成如图1 (中间)。假设我们有另一个用户u4，并且41> s43> s42，在这种情况下，如果我们将p41和s42做得相似，那么无论我们如何定义同一个空间中的P4，都不会满足约束s43> s42。然而，如果我们将用户视为同一空间中的点，并使相似的用户彼此更接近(即，使D23<D12<D13，和41<D43<D42，Ddenotes距离)，我们可以很容易地满足上述两个约束。