Local Item-Item Models for Top-N Recommendation

Evangelia Christakopoulou and George Karypis

Computer Science & Engineering
University of Minnesota, Twin Cities

目录

GLSLIM背景

Top-N 推荐系统算法一般分为两大类：neighborhood-based 和 model-based。

neighborhood-based：用各种距离度量方式计算**出用户之间（user-based**）或者**物品之间（item-based）**的近似度，然后类似于KNN算法，根据该用户（物品）的最相似的 k 个用户（物品）来进行推荐，这种算法只用了用户行为数据，得益于用户行为矩阵的稀疏性，**运算非常快**。Youtube之前的视频推荐算法，是content-based和item-based结合，根据用户行为，**只计算同一topic下的视频的相似度**，这样避免了用户行为数据里的噪音，并且相对普通的item-based算法，更好地利用了长尾数据，增强了推荐系统的覆盖率，并且在改进算法中结合model-based，效果进一步提升。速度快

Model-based：一般是指根据用户行为数据矩阵进行矩阵分解或者用模型来学习用户、物品隐变量，用学习到的低rank的用户矩阵、物品矩阵相乘来预测结果，典型算法有SVD, SVD++, ALS算法等等。**推荐效果好**

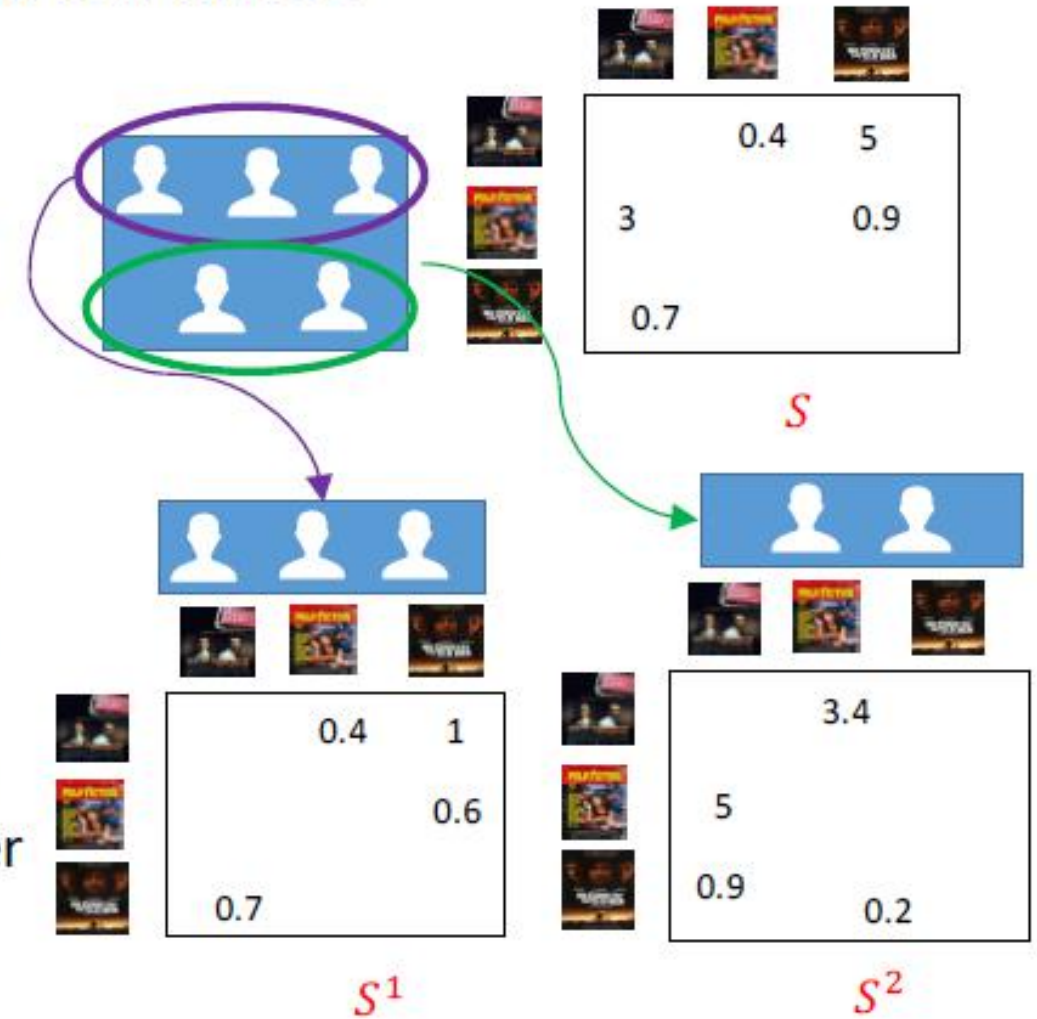
提升neighborhood-based算法的效果，又提升model-based算法的运行时间呢？答案就是**SLIM算法**。

SVD → SLIM → GLSLIM

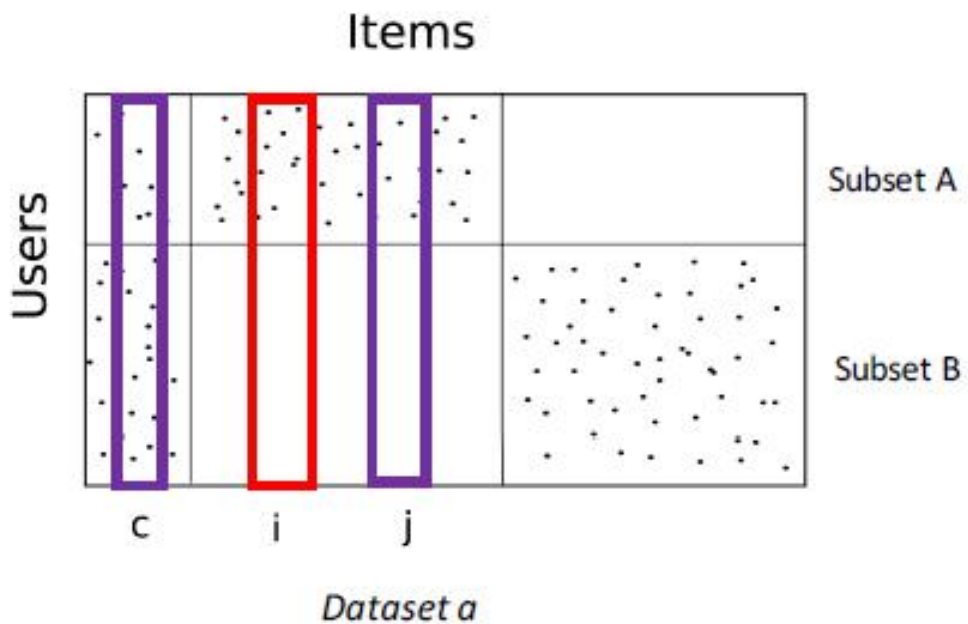
Limitation of the existing item-based approaches

Item-based methods have the drawback of estimating only a *single* model for all users.

However, there could be differences in users' behaviors, which cannot be captured by a single model.
Instead, we need *multiple* item-item models, each for every user subset!

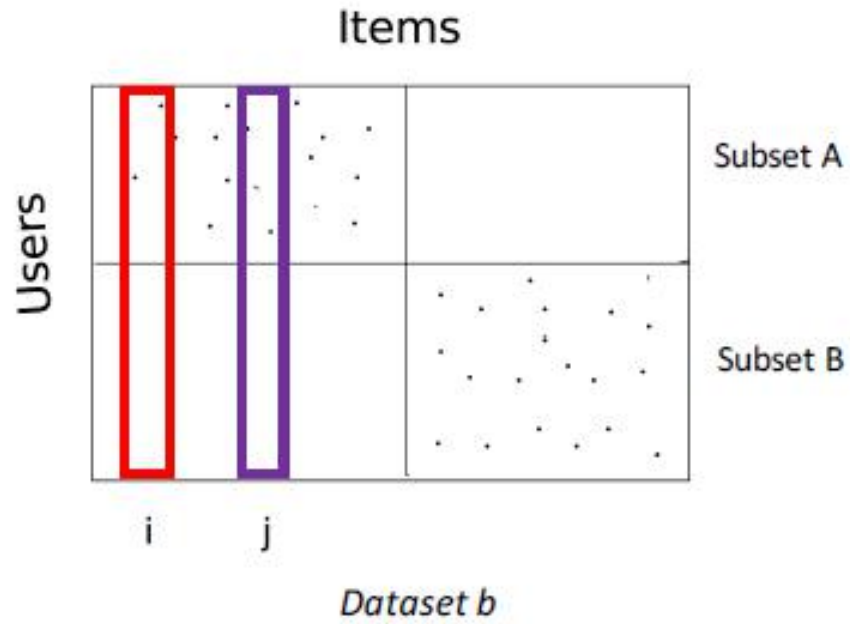


Example of when local item-item models are beneficial



Dataset a

Local item-item models improve upon global item-item model.

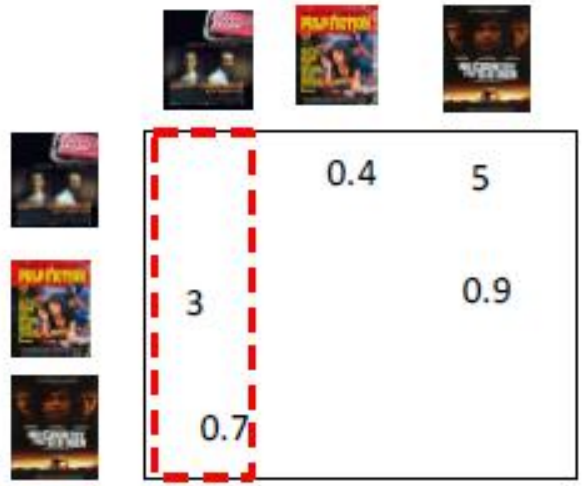


Dataset b

Global item-item model and local item-item models yield the same results.

i: item for which we will compute predictions

A few words on SLIM (Sparse LInear Method)



- Computes the item-item relations, by estimating an items × items sparse aggregation coefficient matrix S.
- The recommendation score of an unrated item i for user u is:

$$\hat{r}_{ui} = \mathbf{r}_u^T \mathbf{s}_i.$$

$$\begin{aligned} \underset{S}{\text{minimize}} \quad & \tfrac{1}{2}\textstyle\sum_{u,i}(r_{ui} - \hat{r}_{ui})^2 + \tfrac{\beta}{2}||S||_F^2 + \lambda||S||_1, \\ \text{subject to} \quad & S \geq 0, \text{and} \\ & \text{diag}(S) = 0. \end{aligned}$$

GLSLIM model

If user u belongs to user subset p_u, then the predicted rating is:

$$\hat{r}_{ui} = \mathbf{r}_u^T (g_u \mathbf{s}_i + (1 - g_u)\mathbf{s}_i^{p_u}).$$

global local

$$\underset{S,\{S^1,\ldots,S^k\},\mathbf{p},\mathbf{g}}{\text{minimize}} \quad \frac{1}{2}\sum_{u,i}(r_{u,i} - \hat{r}_{u,i})^2 +$$

$$\frac{1}{2}\beta_g ||S||_F^2 + \lambda_g ||S||_1 + \sum_{p_u=1}^{k} \left[\frac{1}{2}\beta_l ||S^{p_u}||_F^2 + \lambda_l ||S^{p_u}||_1\right],$$

global local

$$\text{subject to} \quad 0 \le g_u \le 1, \ \forall u$$

$$p_u \in \{1, \ldots, k\}, \ \forall u$$

$$S \ge 0, \ S^1 \ge 0, \ldots, \ S^k \ge 0$$

$$\text{diag}(S) = 0, \ \text{diag}(S^1) = 0, \ldots, \ \text{diag}(S^k) = 0.$$

了解SVD系列推荐系统的都知道，SVD系列算法的精髓在于找出两个rank远低于原矩阵的小矩阵，矩阵B代表用户的特征，矩阵C代表物品的特征，B的第 i 行即为用户 i 的特征，C 的第 j 列即为物品 j 的特征，两个向量相乘，得到的即为用户 i 对于 物品 j 的得分预测。由于预测过程只需要矩阵相乘，训练出两个 low rank 矩阵后，得到速度非常快。SLIM和SVD系列相似，区别在于 SVD是把矩阵 A 分解为了两个 low rank 的小矩阵，而SLIM是直接用矩阵 A 当做要学习得到的用来相乘的两个矩阵中的一个。

在下 SLIM 这一节中，u代表用户，t代表物品，集合U代表所有用户，集合T代表所有物品。用户行为矩阵为A（m*n，m个用户，n个物品）如果用户 i 对 物品 j 有过点击、购买等行为，Aij 就是1（或者一个正数），否则为0 。 $\mathbf{a}_i^T$ 代表矩阵A的一行，即用户 i 对所有物品的行为记录。 $\mathbf{a}_j$ 代表矩阵A的一列，即所有用户对物品 j 的记录。字母上面带下划线的是预测的结果，例如 $\tilde{a}_{ij}$ 。

SLIM的预测公式是 $\tilde{a}_{ij} = \mathbf{a}_i^T \mathbf{w}_j$ ，写成矩阵的形式就是 $\tilde{A} = AW$ 。W 是一个n*n的矩阵，也就是我们需要训练学习得到的权重矩阵。本质上，权重矩阵 W 是物品之间的相似度矩阵。大家可能会有疑惑，SLIM 的权重矩阵比 SVD 的两个小矩阵大好多啊，看起来并没有什么训练时间上的优化。而且 A 和 W 的 rank 都高，相乘得到预测结果的时候计算量也更大。不要着急，下面会对 SLIM 算法速度提升在哪进行解释。

权重矩阵 W 的学习过程如下，损失函数是大家都很熟悉的最小二乘法 + L2正则化+L1正则化。值得注意的是那两个约束条件，一个是 W 非负，这样我们只学习 item 之间的正相关关系。那么为什么 W 的对角线都要为 0 呢？ 留给大家思考一下。

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \frac{1}{2}\|A - AW\|_F^2 + \frac{\beta}{2}\|W\|_F^2 + \lambda\|W\|_1 \\ \text{subject to} \quad & W \geq 0 \\ & \text{diag}(W) = 0, \end{aligned}$$

保证一个已知的行为得分不会用于预测他自己的计算

SLIM对于SVD的提升

1. W 的各列之间是独立的，也就是说，我们可以将 W 的学习过程分解为下图所示的一个个小过程，对 W 中的每一列 独立求解。很方便并行化。

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \frac{1}{2}\|A - AW\|_F^2 + \frac{\beta}{2}\|W\|_F^2 + \lambda\|W\|_1 \\ \text{subject to} \quad & W \geq 0 \\ & \text{diag}(W) = 0, \end{aligned}$$

$$\begin{aligned} \underset{\mathbf{w}_j}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{a}_j - A\mathbf{w}_j\|_2^2 + \frac{\beta}{2}\|\mathbf{w}_j\|_2^2 + \lambda\|\mathbf{w}_j\|_1 \\ \text{subject to} \quad & \mathbf{w}_j \geq \mathbf{0} \\ & w_{j,j} = 0, \end{aligned}$$

由上面各列的损失函数，可以发现拟合 的时候其实是用剩下的 n-1 列来拟合的，这就引来了一个优化方向，如果我们**只选择那些和相似度较高的列来拟合** ，就像特征选择一样，那么训练过程会大大缩短。思考一下矩阵 A 中每一列的意义，是所有用户对一个物品 j 的行为，取那些和 相似度高的列，也就是取那些用户行为和物品 j 相似的物品们的向量。这里其实很像 item-based。

GLSLIM item-item model(第一步)

公式（3）是 GLSLIM 的拟合公式，GLSLIM 会像 SLIM 一样拟合一个权重矩阵 S (SLIM算法中的W)，然后再训练 k 个 local 模型，学习 k 个不同的类中的信息（论文中聚类方法采用的是CLUTO）。代表第 pu 个类的权重矩阵，pu ∈{1,2 … k}。 s_{li} 代表物品 l 和 物品 i 的全局相似度， $s_{li}^{p_u}$ 代表物品 l 和物品 i 的在第pu个用户群里的局部相似度。 g_u 是每个用户独有的权重系数，代表该用户更倾向于全局模型还是局部模型，该系数位于【0,1】区间内，也是模型需要学习的参数。下图意为预测 用户 u 对 物品 i 的得分，其中用户 u 属于 pu 类。

$$\tilde{r}_{ui} = \sum_{l \in \mathscr{R}_u} g_u s_{li} + (1 - g_u) s_{li}^{p_u}. \qquad (3)$$

GLSLIM 的损失函数如下，看起来比 SLIM 复杂很多，其实只是加入了对各局部模型的拟合、正则化，约束项中对和的约束也类似于全局模型，保证只学习物品间的正相似度，以及不用已知的行为去拟合它自己。

$$\begin{aligned}
\underset{\mathbf{s}_i, \{\mathbf{s}_i^1, \ldots, \mathbf{s}_i^k\}}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{r}_i - \mathbf{g} \odot R\mathbf{s}_i - \mathbf{g}' \odot \textstyle\sum_{p_u=1}^{k} R^{p_u}\mathbf{s}_i^{p_u}||_2^2 + \\
& \tfrac{1}{2}\beta_g||\mathbf{s}_i||_2^2 + \lambda_g||\mathbf{s}_i||_1 + \\
& \textstyle\sum_{p_u=1}^{k} \tfrac{1}{2}\beta_l||\mathbf{s}_i^{p_u}||_2^2 + \lambda_l||\mathbf{s}_i^{p_u}||_1, \\
\text{subject to} \quad & \mathbf{s}_i \geq 0, \\
& \mathbf{s}_i^{p_u} \geq 0, \forall p_u \in \{1, \ldots, k\}, \\
& s_{ii} = 0, \\
& s^{p_u}{}_{ii} = 0, \forall p_u \in \{1, \ldots, k\},
\end{aligned} \qquad (4)$$

GLSLIM user-subsets的最优划分(第二步)

第一步聚类的结果只是初始化用，GLSLIM 算法本身还会对聚类结果进行调整。
可以看到在while循环里，对每个用户，会把他分到训练误差最小的那一类去。当聚类的结果改变超过 1% 时，会一直迭代下去

Algorithm 1 GLSLIM

1: Assign $g_u = 0.5$, to every user u.
2: Compute the initial clustering of users with CLUTO [1].
3: **while** number of users who switched clusters $> 1\%$ of the total number of users **do**
4: Estimate S and S^{p_u}, $\forall p_u \in \{1, \ldots, k\}$ with Equation 4.
5: **for all** user u **do**
6: **for all** cluster p_u **do**
7: Compute g_u for cluster p_u with Equation 5.
8: Compute the training error.
9: **end for**
10: Assign user u to the cluster p_u that has the smallest training error and update g_u to the corresponding one for cluster p_u.
11: **end for**
12: **end while**

用户偏好系数由公式（3）误差计算

$$g_u = \frac{\sum_{i=1}^{m} (\sum_{l \in \mathscr{R}_u} s_{li} - \sum_{l \in \mathscr{R}_u} s_{li}^{p_u})(r_{ui} - \sum_{l \in \mathscr{R}_u} s_{li}^{p_u})}{\sum_{i=1}^{m} (\sum_{l \in \mathscr{R}_u} s_{li} - \sum_{l \in \mathscr{R}_u} s_{li}^{p_u})^2}.$$

评价指标

本文使用留一交叉验证方法来评估模型的效果，对于每个用户，随机选择一个打分物品放在测试集里，剩下的当做训练集。通过计算物品在top-N列表里的hit次数和位置来评估该模型。HR表示hit-rate，ARHR是hit rank的倒数的平均值，计算方法如下：

$$HR = \frac{\#hits}{\#users},$$

$$ARHR = \frac{1}{\#users} \sum_{i=1}^{\#hits} \frac{1}{p_i},$$

HR的分母表示用户数目，分子是hit次数，如果该用户的测试集里的打分物品正好出现在推荐列表里，那么hit加一，ARHR中的p_i表示该hit物品在list中的位置，位置越靠前，认为效果越好。

模型比较

- LSLIMr0: Local SLIM without refinement.
- GLSLIMr0: Global and Local SLIM without refinement.
- LSLIM: Local SLIM with refinement.
- GLSLIM: Global and Local SLIM with refinement.

◆ LSLIM 与 GLSLIM 相比，没有去训练全局模型，只训练了各用户群的局部模型。效果会比GLSLIM稍差一些，毕竟训练过程缩短，没有训练全局模型，捕捉不到全局信息

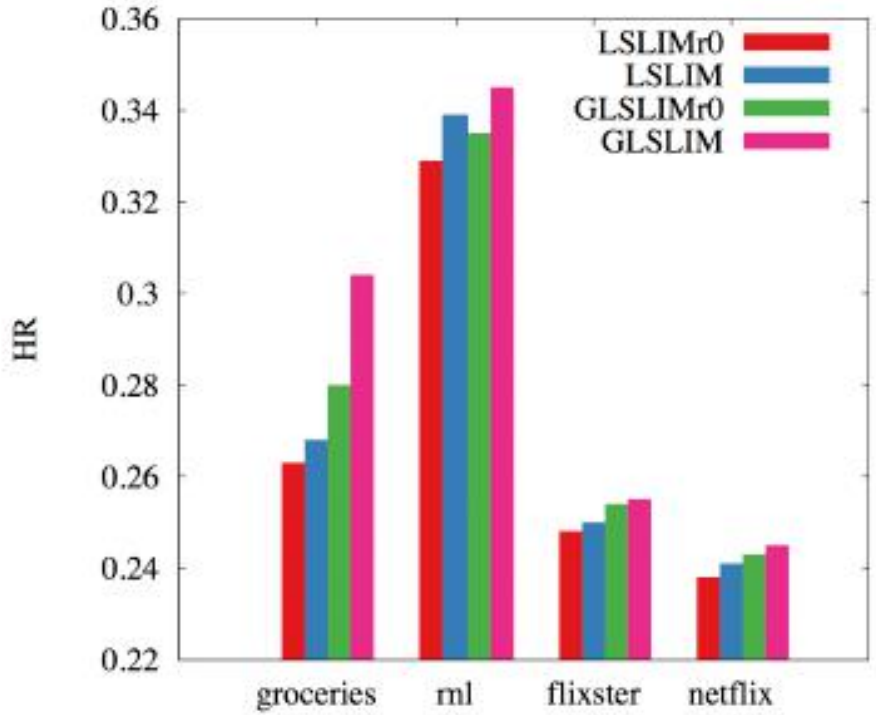
$$\begin{aligned}&\underset{\{\mathbf{s}_i^1,\ldots,\mathbf{s}_i^k\}}{\text{minimize}} \quad \tfrac{1}{2}||\mathbf{r}_i - \textstyle\sum_{p_u=1}^{k} R^{p_u}\mathbf{s}_i^{p_u}||_2^2 + \\ &\qquad\qquad \textstyle\sum_{p_u=1}^{k} \tfrac{1}{2}\beta_l||\mathbf{s}_i^{p_u}||_2^2 + \lambda_l||\mathbf{s}_i^{p_u}||_1, \qquad (8)\\ &\text{subject to} \\ &\qquad\qquad \mathbf{s}_i^{p_u} \geq 0, \forall p_u \in \{1,\ldots,k\}, \\ &\qquad\qquad {s^{p_u}}_{ii} = 0, \forall p_u \in \{1,\ldots,k\},\end{aligned}$$

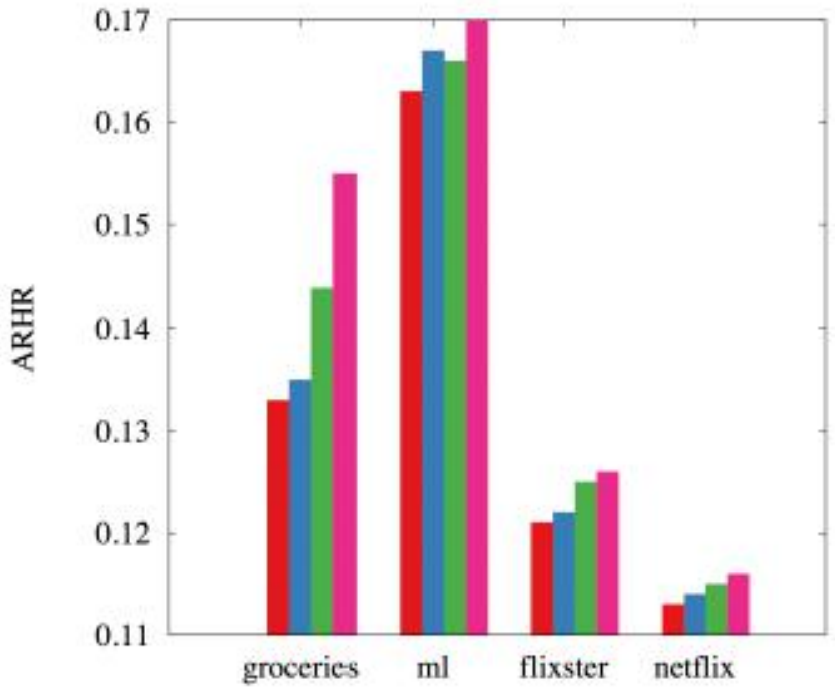
$$\tilde{r}_{ui} = \sum_{l \in \mathscr{R}_u} s_{li}^{p_u}. \qquad (9)$$

◆ GLSLIMr0 与GLSLIM 相比，相信聚类算法的结果，没有在每次迭代中对用户所在的用户群进行更改，固定住每个类里的用户，当某次迭代和上一次的训练误差相比变化不大时，停止训练。

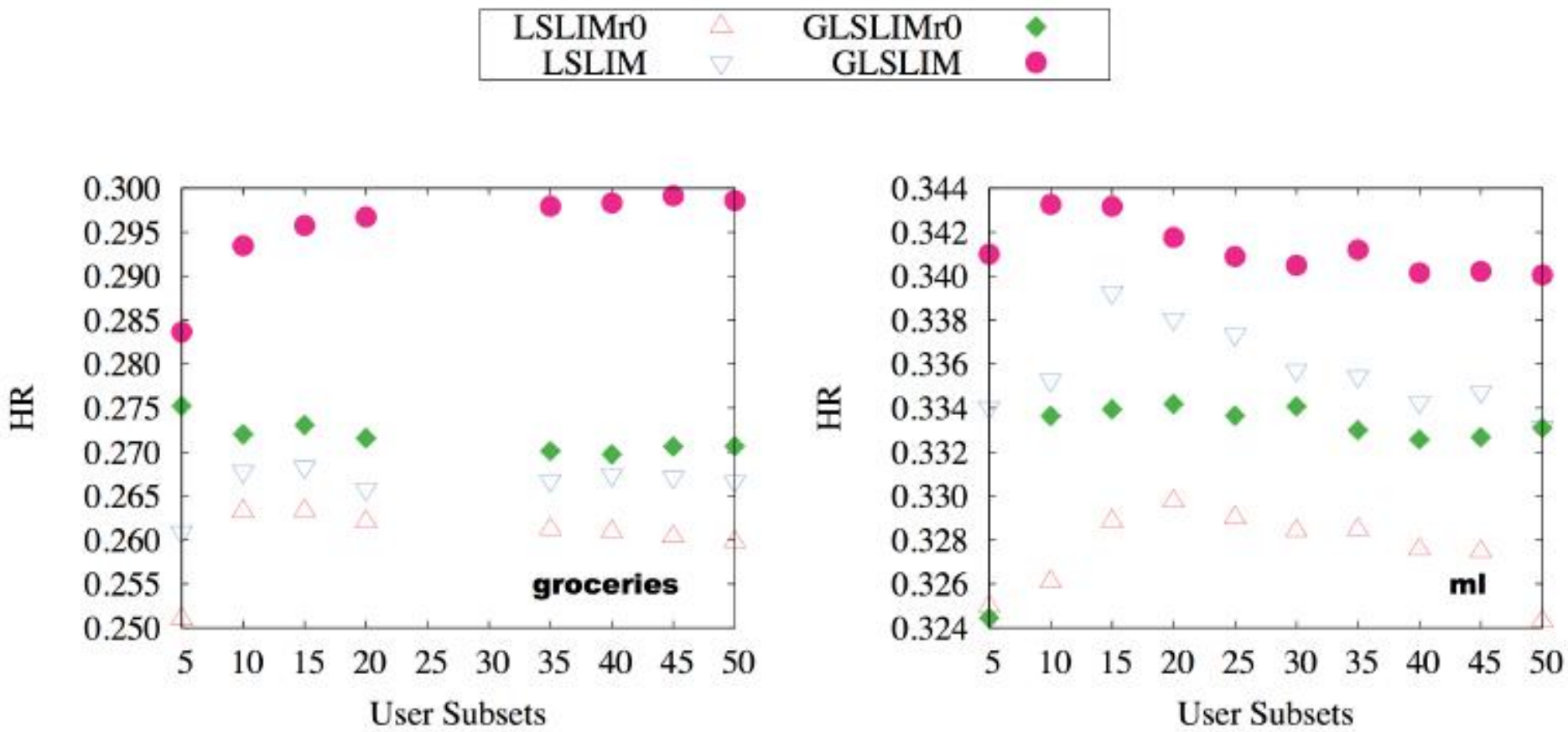
试验结果

Name	#Users	#Items	#Transactions	Density
groceries	63,034	15,846	2,060,719	0.21%
ml	69,878	10,677	10,000,054	1.34%
flixster	29,828	10,085	7,356,146	2.45%
netflix	274,036	17,770	31,756,784	0.65%

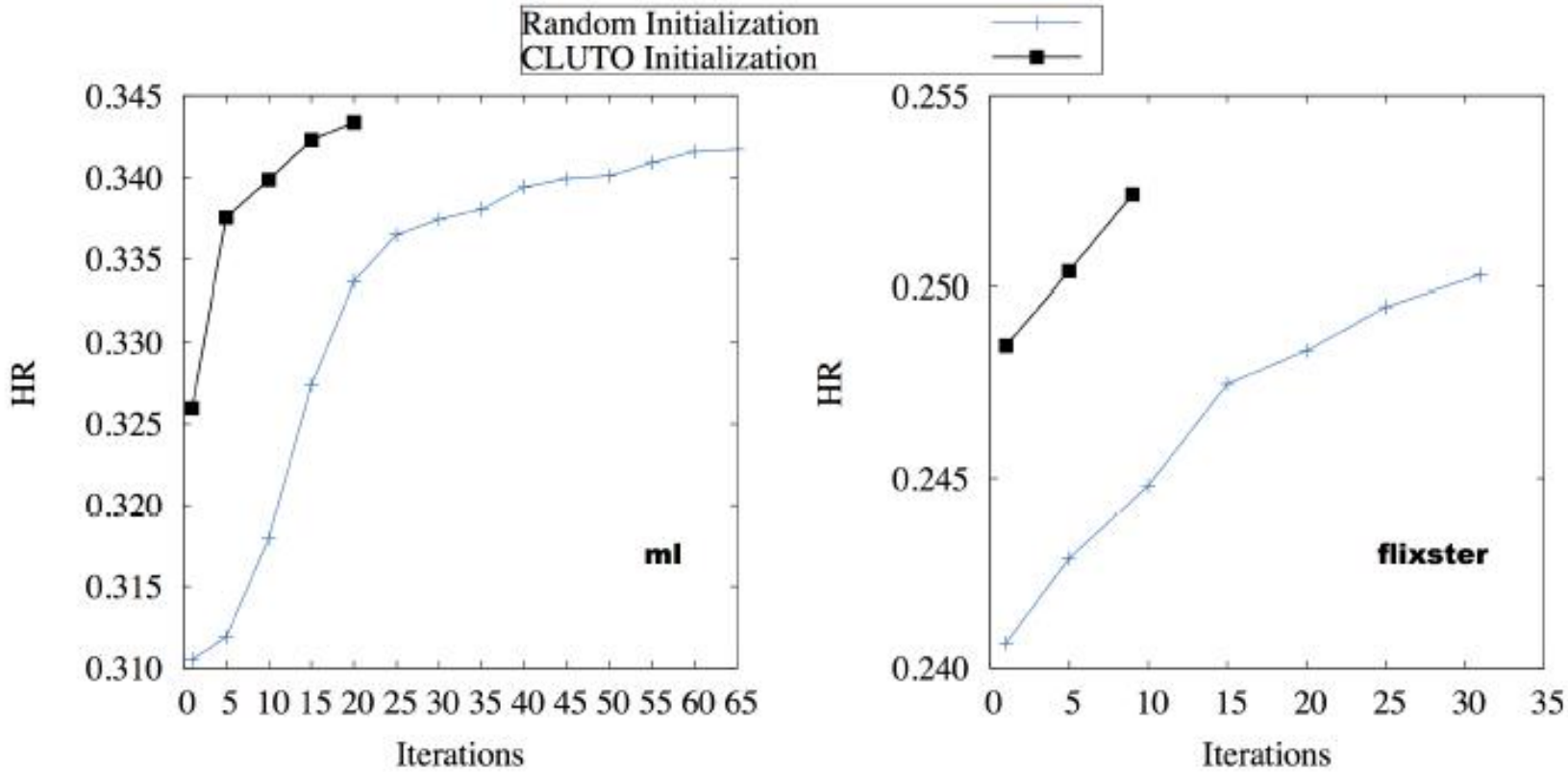




Sensitivity on the number of User Subsets



Initializing with Random User Subsets



Performance against Competing Approaches