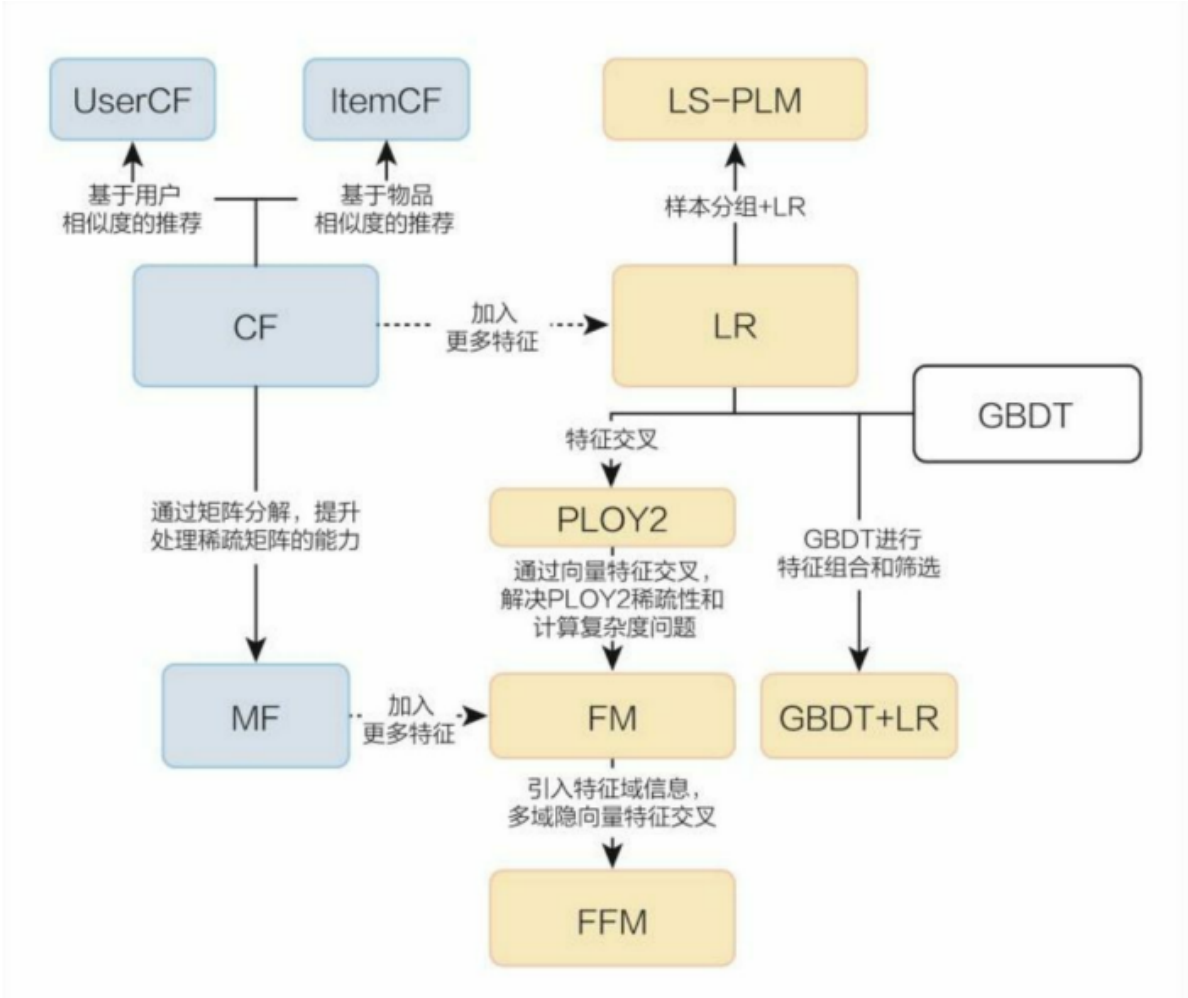
传统推荐模型

1.1 关系图



1.2 协同过滤（CF）

- 基本思想：人以类聚，物以群分
- 目标场景：
- 相似度计算方法：
  - 杰拉德（Jaccard）相似度：

$$J(A,B)=\frac{|A\bigcap B|}{|A\bigcup B|}$$

  - 余弦相似度：计算用户向量i和j之间的夹角大小，夹角越小，相似度越大。

$$sim(i,j)=cos(i,j)=\frac{i\cdot j}{||i||\cdot||j||}$$

    - 较为常用，但存在局限性：
    - 评分不规范，如有的用户天生喜欢打高分/低分

```
from sklearn.metrics.pairwise import cosine_simlarity
i = [1, 0, 0, 0]
j = [1, 0.5, 0.5, 0]
cosine_similarity([i,j])
```

- 皮尔逊相关系数：对于用户评分偏置的情况，可以考虑使用Pearson相关系数

$$\mathrm{sim}(i,j)=\frac{\sum_{p\epsilon P}(R_{i,p}-\bar{R}_i)(R_{j,p}-\bar{R}_j)}{\sqrt{\sum_{p\epsilon P}(R_{i,p}-\bar{R}_i)^2}\sqrt{\sum_{p\epsilon P}(R_{j,p}-\bar{R}_j)^2}}$$

$R_i=[R_{i1}\ R_{i2}\dots\ R_{iN}],\ R_j=[R_{j1}\ R_{j2}\dots\ R_{jN}]$

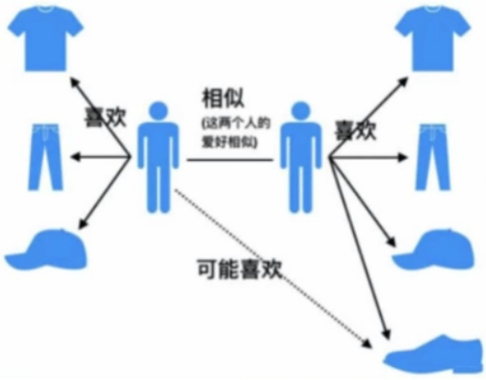
```
from scipy.stats import pearsonr

i = [1, 0, 0, 0]
j = [1, 0.5, 0.5, 0]
pearsonr(i,j)
```

- 其他
  - 欧式距离
  - 曼哈顿距离
  - 马氏距离
- 欧式距离 vs 余弦相似度
  - 欧式距离体现数值上的绝对差异，余弦距离体现方向上的相对差异
  - 欧式距离强调绝对数值，余弦相似强调夹角
  - 例子：

场景	选择
统计两部剧用户观看行为，用户A观看向量（0,1）用户B观看向量（1,0）。分析两用户对不同视频的喜好。（此时余弦距离很大，而欧式距离很小）	选择余弦距离
分析用户活跃度。以登录次数和平均观看时长为特征。用户A（1,10）用户B（1,100）。（此时余弦距离会很近，但是欧式距离很远）	选择欧氏距离

- **基于User的协同过滤（UserCF）：计算用户之间的相似度**
  - **步骤一：计算Alice和其他用户相似度**



	物品1	物品2	物品3	物品4	物品5
Alice	5	3	4	4	?
用户1	3	1	2	3	3
用户2	4	3	4	3	5
用户3	3	3	1	5	4
用户4	1	5	5	2	1

用户向量Alice, user1, user2, user3, user4

(1) 余弦相似性：sim(Alice, user1) = cos(Alice, user1) = $\frac{15+3+8+12}{\sqrt{25+9+16+16}*\sqrt{9+1+4+9}} = 0.975$

(2) Pearson相关系数 Alice_ave = 4 user1_ace =2.25

sim(Alice, user1) = 0.852

```
from sklearn.metrics.pairwise import cosine_similarity
users = np.array([[5, 3, 4, 4],[3, 1, 2, 3],[4, 3, 4, 3],[3, 3, 1, 5],[1, 5, 5, 2]])
cosine_similarity(users)
np.corrcoef(users)
```

- **步骤二：预测得分**
  - 方式一：加权平均

    用户u对于商品p的评分：

    $$R_{u,p} = \frac{\sum_{s \in S} w_{u,s} R_{s,p}}{\sum_{s \in S} w_{u,s}}$$

    $w_{u,s}$表示用户u和s之间的相似度

    （部分用户喜欢打高分，有的喜欢打低分，容易不客观）
  - 方式二：

    $$P_{i,j} = \overline{R_i} + \frac{\sum_{k=1}^{n}(S_{i,k}(R_{k,j} - \overline{R_k}))}{\sum_{k=1}^{n} S_{i,k}}$$

    这里$P_{i,j}$表示用户i对商品j的评分，S表示相似度，R同样表示评分

    $$P_{Alice,物品5} = \overline{R_{Alice}} + \frac{\sum_{k=1}^{2}(S_{Alice,userk}(R_{userk,物品5} - \overline{R_{userk}}))}{\sum_{k=1}^{2} S_{Alice,userk}}$$

=4.87

- **步骤三：基于用户评分进行推荐**

  设定阈值，超过阈值可以推荐给用户

UserCF的缺点：

- 数据稀疏性：商品多，用户之间买的重叠性比较低，导致难以找到一个用户的邻居（偏好相似用户）。即使找到了也准确性不高，所以UserCF不适用于正反馈获取困难的应用场景

  （如酒店预订，大件商品购买的低频应用）
- 用户相似度矩阵维护难度大：
  - 互联网场景中用户数一般远大于无评书，维护用户相似度矩阵难度大
  - 基于用户协同过滤需要维护用户相似度矩阵以便快速找出Top n的相似用户，该矩阵的存储开销巨大，不适用于用户数据量大的情况使用

使用场景：

- 适用于用户少，物品多，时效性强的场合（如新闻推荐场景）

- **基于Item的协同过滤（ItemCF）：计算物品之间的相似度（电商早期使用）**

	物品1	物品2	物品3	物品4	物品5
Alice	5	3	4	4	?
用户1	3	1	2	3	3
用户2	4	3	4	3	5
用户3	3	3	1	5	4
用户4	1	5	5	2	1

计算过程类似，但是是计算列向量之间的相似度，即商品向量之间的相似度

$$P_{Alice,物品5} = \overline{R_{物品5}} + \frac{\sum_{k=1}^{2}(S_{物品5,物品k}(R_{Alice,物品k} - \overline{R_{物品k}}))}{\sum_{k=1}^{2} S_{物品k,物品5}}$$
$$= \frac{13}{4} + \frac{0.97*(5-3.2)+0.58*(4-3.4)}{0.97+0.58} = 4.6$$

ItemCF优点：

- Item-based的预测效果更好，余弦计算好物品相似度，在线预测性能更好
  （物品增长速度较慢，较稳定，能在较长时间内维护较稳定的相似度）

ItemCF缺点：

- 稀疏性
- 相似度维护难度大（但相对UserCF可能较小）

适用场景：

- 兴趣变化较为稳定的应用。更接近个性化的推荐
- 用户数量远大于商品数目，用户兴趣固定持久，商品更新速度不是太快
  （推荐艺术品、音乐、电影等）

- **UserCF和ItemCF的优缺点对比**

	UserCF	ItemCF
性能	适用于用户较少场合，如果用户很多，计算用户相似矩阵代价大	适用于物品数明显校友用户数的场合，如果物品很多，计算武平相似度矩阵代价也很大
领域	时效性强，用户个性化兴趣不太明显的领域（强调人与人之间的共性，周围人都在看）	长尾物品丰富，用户个性化需求强烈的领域（强调个性）
实时性	用户有新行为，不一定造成推荐结果的立即变换	用户有新行为，一定会导致推荐结果实时变化
冷启动	在新用户对很少的物品产生行为以后，不能立即对他进行个性化推荐，因为用户相似度表每隔一段时间离线计算（需要更新相似用户，才能做出准确推荐）	新用户只要对一个物品产生行为，就可以给他推荐和该物品相关的其他物品
新物品	新物品上线一段时间，一旦有用户对物品产生行为，就可以将新物品推荐给对它产生行为的用户兴趣相似的其他用户	没有办法在不离线更新物品相似度表的情况下，将新物品推荐给新用户
推荐理由	难提供令用户信服的推荐解释	利用用户的历史行为给用户做推荐解释，可以令用户比较信服

共同缺点
不能彻底解决数据稀疏性的问题
泛化能力弱：热门商品具有很强的头部效应，容易和大量物品产生相似，而尾部物品由于特征向量系数，很少被推荐（为有效解决头部效应，矩阵分解技术被提出）
无法利用更多信息，如用户和物品本身的特征

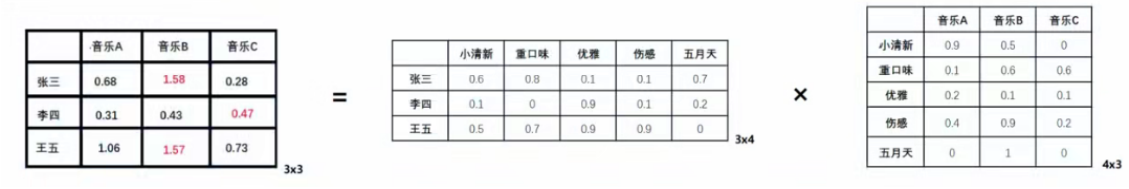
1.3 MF矩阵分解 -- SVD、LFM、RSVD、SVD++（Matrix Factorization）

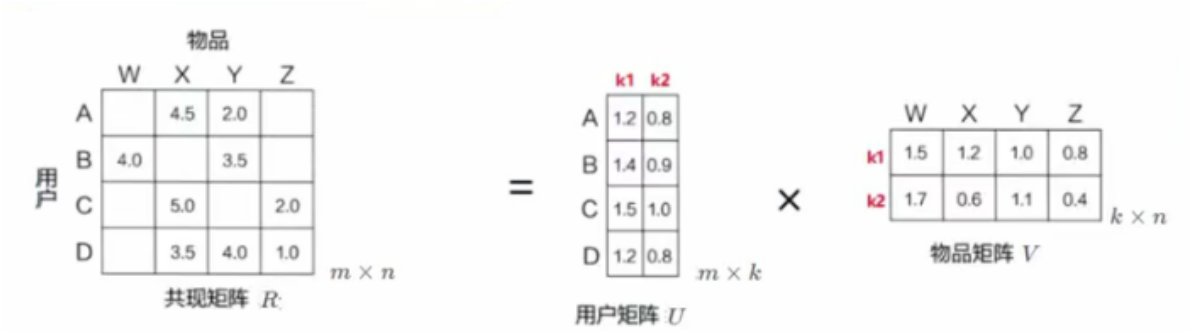
- **针对问题：**

  协同过滤处理稀疏矩阵的能力较弱

  协同过滤中，相似度矩阵维护难度大

- **解决思路：**





-
  - 隐含特征是不可解释的，需要模型自己学习
  - k的大小决定隐向量表达能力强弱，k越大表达能力越强，用户兴趣和物品分类具体
  - 通过用户矩阵和物品矩阵预测评分计算公式：

$$Preference(n,i) = r_{ui} = \sum_{f=1}^{F} p_{u,k} q_{k,i} \quad \text{(对应向量内积)}$$

- **MF方式**
  - **特征值分解**
    - 特征值，特征向量： $Av = \lambda v$

      v是特征向量，λ是特征向量
    - 特征值分解： $A = Q \sum Q^{-1}$

      Q代表矩阵A的特征向量构成的矩阵

      $\sum$是对角阵，对角线的元素是特征值

- **奇异值分解（SVD）：**
  - 定义：$A = U\sum V^T$

    其中　A是实矩阵，$UU^T = I, VV^T = I$

    $\sum$是对角矩阵，对角线元素非负且降序排列

  - 计算步骤

    A是一个m*n的实矩阵

    1. 构造n阶实对称矩阵 $W = A^T A$
    2. 计算W的特征值与特征向量

求解特征方程

$$(W - \lambda I)x = 0$$

得到特征值 λ_i，并将特征值由大到小排列

$$\lambda_1 \geqslant \lambda_2 \geqslant \cdots \geqslant \lambda_n \geqslant 0$$

将特征值 λ_i（$i = 1, 2, \cdots, n$）代入特征方程求得对应的特征向量。

    3. 求得n阶正交矩阵V

将特征向量单位化，得到单位特征向量 $v_1, v_2, \cdots, v_n$，构成 n 阶正交矩阵 V：

$$V = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

    4. 求得m*n对角矩阵

计算 A 的奇异值

$$\sigma_i = \sqrt{\lambda_i}, \quad i = 1, 2, \cdots, n$$

构造 $m \times n$ 矩形对角矩阵 Σ，主对角线元素是奇异值，其余元素是零，

$$\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \cdots, \sigma_n)$$

    5. 求得m阶正交矩阵U（求得上述）
       - 求U1

对 A 的前 r 个正奇异值，令

$$u_j = \frac{1}{\sigma_j} A v_j, \quad j = 1, 2, \cdots, r$$

得到

$$U_1 = \begin{bmatrix} u_1 & u_2 & \cdots & u_r \end{bmatrix}$$

- 求U2

  求 A^{T} 的零空间的一组标准正交基 $\{u_{r+1}, u_{r+2}, \cdots, u_m\}$，令

  $$U_2 = [\ u_{r+1} \quad u_{r+2} \quad \cdots \quad u_m\ ]$$

- 得到U=[U1, U2]

- 缺点：

  传统SVD分解会要求原始矩阵是稠密的，所以我们需要对缺失进行填补，空间复杂度非常高，基本无法解决大规模稀疏矩阵的矩阵分解问题

- **Basic SVD（LFM，Funk SVD）**

  - 将矩阵分解问题转化为**最优化问题**，通过梯度下降进行优化
  - 预测函数：

  $$\text{Preference}(u,i) = r_{ui} = p_u^T q_i = \sum_{f=1}^{F} p_{u,k} q_{k,i}$$

  - 损失函数（误差平方和）：

    式子2便于优化

  $$\text{SSE} = \sum_{u,i} e_{ui}^2 = \sum_{u,i} \left(r_{ui} - \sum_{k=1}^{K} p_{u,k} q_{k,i} \right)^2$$

  $$\text{SSE} = \frac{1}{2} \sum_{u,i} e_{ui}^2 = \frac{1}{2} \sum_{u,i} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ki} \right)^2$$

  - 步骤：

    1. 首先先初始化这两个矩阵
    2. 把用户评分矩阵里面已经评过分的那些样本当做训练集的label， 把对应的用户和物品的隐向量当做features， 这样就会得到(features, label)相当于训练集
    3. 通过两个隐向量乘积得到预测值pred
    4. 根据label和pred计算损失
    5. 然后反向传播， 通过梯度下降的方式，更新两个隐向量的值
    6. 未评过分的那些样本当做测试集， 通过两个隐向量就可以得到测试集的label值
    7. 这样就填充完了矩阵， 下一步就可以进行推荐了

- **RSVD**

  在Basic SVD基础上，加入正则化参数（惩罚项）

  预测函数：

  $$\text{Preference}(u,i) = r_{ui} = p_u^T q_i = \sum_{f=1}^{F} p_{u,k} q_{k,i}$$

  目标函数：

$$SSE = \frac{1}{2}\sum_{u,i} e_{ui}^2 + \frac{1}{2}\lambda \sum_u |p_u|^2 + \frac{1}{2}\lambda \sum_i |q_i|^2$$

$$= \frac{1}{2}\sum_{u,i} e_{ui}^2 + \frac{1}{2}\lambda \sum_u \sum_{k=0}^{K} p_{u,k}^2 + \frac{1}{2}\lambda \sum_i \sum_{k=0}^{K} q_{k,i}^2$$

-
  - **改进（LFM）：**

    Netflix提出另外一种LFM，在原有基础上加偏置项，消除用户和物品打分的偏差

    原因：不同用户打分体系不同，不同物品衡量标准有区别，导致评分偏差

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T \cdot q_i$$

用户评分偏差

评分的全局平均数

物品得分偏差

目标函数：

$$\text{SSE} = \frac{1}{2}\sum_{u,i} e_{ui}^2 + \frac{1}{2}\lambda \sum_u |\boldsymbol{p}_u|^2 + \frac{1}{2}\lambda \sum_i |\boldsymbol{q}_i|^2 + \frac{1}{2}\lambda \sum_u \boldsymbol{b}_u^2 + \frac{1}{2}\lambda \sum_u \boldsymbol{b}_i^2$$
$$= \frac{1}{2}\sum_{u,i} \left(\boldsymbol{r}_{ui} - \boldsymbol{\mu} - \boldsymbol{b}_u - \boldsymbol{b}_i - \sum_{k=1}^{K} \boldsymbol{p}_{uk}\boldsymbol{q}_{ki}\right)^2 + \frac{1}{2}\lambda \sum_u |\boldsymbol{p}_u|^2 + \frac{1}{2}\lambda \sum_i |\boldsymbol{q}_i|^2 + \frac{1}{2}\lambda \sum_u \boldsymbol{b}_u^2 + \frac{1}{2}\lambda \sum_u \boldsymbol{b}_i^2$$

  - **SVD++**

    改进方向：用户历史记录会对新评分产生影响（即物品间存在某些联系），交给模型学习

    首先先把ItemCF的预测算法改成一个可以学习的模型， 就行LFM那样， 怎么改？ ItemCF的预测算法公式如下：

$$\hat{r}_{ui} = \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} w_{ij}$$

    还记得ItemCF吗？ 这个式子是预测用户u对于物品i的打分， $N(u)$表示用户u打过分的历史物品， w_{ij}表示物品ij的相似度， 当然这里的这个相似度不再是ItemCF那样， 通过向量计算的， 而是想向LFM那样， 让模型自己学出这个参数来， 那么相应的就可以通过优化的思想嘛：

$$SSE = \sum_{(u,i) \in \text{Train}} \left(r_{ui} - \sum_{j \in N(u)} w_{ij} r_{uj}\right)^2 + \lambda w_{ij}^2$$

    但是呢， 这么模型有个问题， 就是w比较稠密， 存储需要很大的空间， 因为如果有n个物品， 那么模型的参数就是n^2， 参数一多， 就容易造成过拟合。 所以Koren提出应该对w矩阵进行分解， 将参数降到了$2 * n * F$:

$$\hat{r}_{ui} = \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} x_i^T y_j = \frac{1}{\sqrt{|N(u)|}} x_i^T \sum_{j \in N(u)} y_j$$

    相当于用$x_i^T y_j$代替了w_{ij}， 这里的x_i, y_j是两个F维的向量。 **有没有发现在这里，就出现了点FM的改进身影了。**这里其实就是又对物品i和某个用户u买过的历史物品又学习一波隐向量， 这次是F维， 为了衡量出物品i和历史物品j之间的相似性来。 这时候， 参数的数量降了下来， 并同时也考虑进来了用户的历史物品记录。 所以这个和之前的LFM相加就得到了：

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T \cdot q_i + \frac{1}{\sqrt{|N(u)|}} x_i^T \sum_{j \in N(u)} y_j$$

前面的是我们之前分析的LFM模型， 而后面的这个是考虑进了用户购买的历史物品。 但是这样感觉参数太多了， 所以Koren提出令$x = q$, 因为既然同是商品i， 就没有必要学习两个隐向量了嘛， 所以得到了该模型的最终预测方式：

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_j \right)$$

这一个就是SVD++模型了。 有了预测函数， 然后也知道真实值， 就可以由损失函数对各个参数求偏导， 和上面的一样了， 这里直接给出导数了， 不推了：

$$e_{ui} = r_{ui} - \hat{r}_{ui}$$

$$b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u)$$

$$b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i)$$

$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot (p_u + \frac{1}{\sqrt{\|R_u\|}} \sum_{j \in R_u} y_j) - \lambda \cdot q_i)$$

$$y_j \leftarrow y_j + \gamma (e_{ui} \cdot \frac{1}{\sqrt{\|R_u\|}} \cdot q_i - \lambda \cdot q_i)$$