法律声明

- 课程详情请咨询
  - 微信公众号：北风教育
  - 官方网址：http://www.ibeifeng.com/

人工智能之推荐系统

推荐系统：推荐算法模型之隐因子模型

主讲人：Gerry

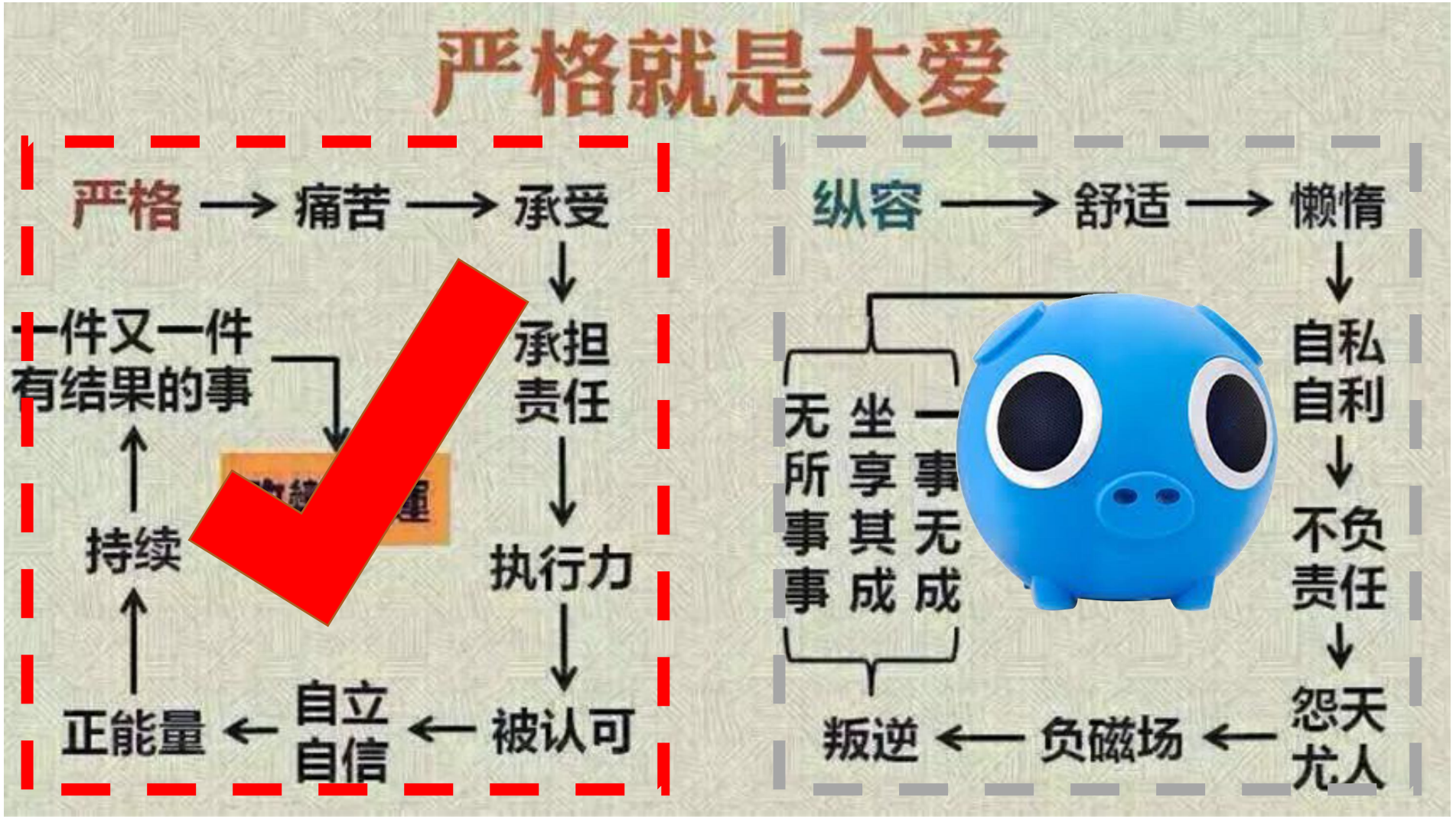
上海育创网络科技有限公司

北风网 IBEIFENG.COM

课程要求

- 课上课下“九字”真言
  - 认真听，善摘录，勤思考
  - **多温故，乐实践**，再发散
- 四不原则
  - 不懒散惰性，不迟到早退
  - 不请假旷课，不拖延作业
- 一点注意事项
  - 违反“四不原则”，不包就业和推荐就业

严格是大爱


严格就是大爱
严格
痛苦
承受
承担责任
执行力
被认可
自立自信
正能量
持续
一件又一件有结果的事
纵容
舒适
懒惰
自私自利
不负责任
怨天尤人
负磁场
叛逆
一事无成
坐享其成
无所事事

寄语

做别人不愿做的事，

做别人不敢做的事，

做别人做不到的事。

课程内容

- Slope One推荐算法模型
- 隐因子推荐模型算法

Slope One

- Slope One是一种简单、高效的协同过滤推荐算法。其核心思想是：利用均值化的思想来掩盖个体打分的差异性，算法认为所有人会采用同一个标准对待不同的物品评分。
- Slope One具有以下几个特点：
  - 算法比较容易实现和维护；
  - 对新的评分应该立即给予响应；
  - 对新用户也需要能够给出比较有效的推荐；
  - 在精度上要有比较高的竞争力。

Slope One

- 求Lucy对Item1的评分?

Sample rating database

Customer	Item 1	Item 2	Item 3
John	5	3	2
Mark	3	4	Didn't rate it
Lucy	Didn't rate it	2	5

Slope One

Customer	Item 1	Item 2	Item 3
John	5	3	2
Mark	3	4	Didn't rate it
Lucy	Didn't rate it	2	5

- 计算Item1和Item2的平均差值: ((5-3)+(3-4))/2=0.5，则Lucy对item1的打分应该：0.5+2=2.5;
- 同理计算Item1和Item3的平均差值：(5-2)/1=3，则Lucy对Item1的打分应该为：3+5=8;
- 最后最所有评分进行加权算法(Weighted Slope One)来计算出最终评分，则最终评分为：(2.5*2+8*1)/(2+1)=13/3=4.33
- 即，Lucy对于Item1的评分最终约为4.33分。

Slope One

Customer	Item 1	Item 2	Item 3
John	5	3	2
Mark	3	4	Didn't rate it
Lucy	Didn't rate it	2	5

- Slope One的最终评分计算公式为：

$$\hat{r}_{ui} = \frac{\sum_j \left(|U_{ij}| \left(\frac{1}{|U_{ij}|} \sum_{v \in U_{ij}} \left(r_{vi} - r_{vj} \right) + r_{uj} \right) \right)}{\sum_j |U_{ij}|}$$

$$= \frac{\sum_j \left(\sum_{v \in U_{ij}} \left(r_{vi} - r_{vj} \right) + |U_{ij}| * r_{uj} \right)}{\sum_j |U_{ij}|}$$

Slope One案例

- 使用Surprise中的Slope One的API对电影数据进行预测，并比较RMSE、MAE和FCP的评估指标。

隐语义模型

- 在协同过滤中，我们要求只有当两个用户对于同一个物品进行评分后我们才可以计算用户之间的相似度，或者说只有当两个物品被同一个用户评分后我们才可以计算物品之间的相似度，那么这样的计算相似度的方式其实是没有考量物品/用户背后的关联性，比如：
  - 用户1对Iphone6评价为4.2分
  - 用户2对Iphone6 plus评价为4.8分
- 问：Iphone6和Iphone6 plus之间的相似度为多少呢？用户1对于Iphone6 plus的评分是多少呢？

隐语义模型

	Iphone6	Iphone6 plus
User-1	4.2	?
User-2	?	4.8

	高端手机	大屏幕	性能	续航
Iphone6	2	1.4	1.6	1.4
Iphone6 plus	2	1.8	1.8	1.2

	高端手机	大屏幕	性能	续航
User-1	0.8	0.5	0.8	0.3
User-2	0.9	0.8	0.6	0.4

隐语义模型

- **隐语义模型**又叫做**潜在因子算法**(Latent Factor)；其算法思想是：认为每个用户都有自己的偏好，同时每个物品也包含所有用户的偏好信息；那么可以认为用户对于物品的高评分体现的是物品中所包含的偏好信息恰好就是用户喜好的信息。而这个偏好信息我们是无法明显的找出的，所以我们可以认为这个就是潜在影响用户对物品评分的因子，即潜在因子。所以只要我们可以得到**用户-潜在因子矩阵Q**和**物品-潜在因子矩阵P**就可以计算出用户对于物品的评分信息。

$$\hat{R} = QP^T$$

隐语义模型

- **用户-潜在因子矩阵Q**：1表示特别喜欢，0表示不喜欢。

	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

隐语义模型

- **物品-潜在因子矩阵P**：表示各个物品包含各个元素的成分，比如音乐A是一个偏小清新的音乐，包含小清新这个Latent Factor的成分是0.9，重口味的成分是0.1，优雅的成分是0.2....

	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
音乐D	0	0.6	0.1	0.2	0

隐语义模型

- 根据Q和P这两个矩阵，就可以计算出每个人对每个商品的喜好程度，比如：张三对音乐A的喜好程度是：张三对**小清新**的偏好*音乐A含有的**小清新**的成分 + 张三对**重口味**的偏好*音乐A含有的**重口味**的成分 + 张三对**优雅**的偏好*音乐A含有的**优雅**的成分 + ...

	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

	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

- 即：0.6*0.9+0.8*0.1+0.1*0.2+0.1*0.4+0.7*0=0.68

隐语义模型

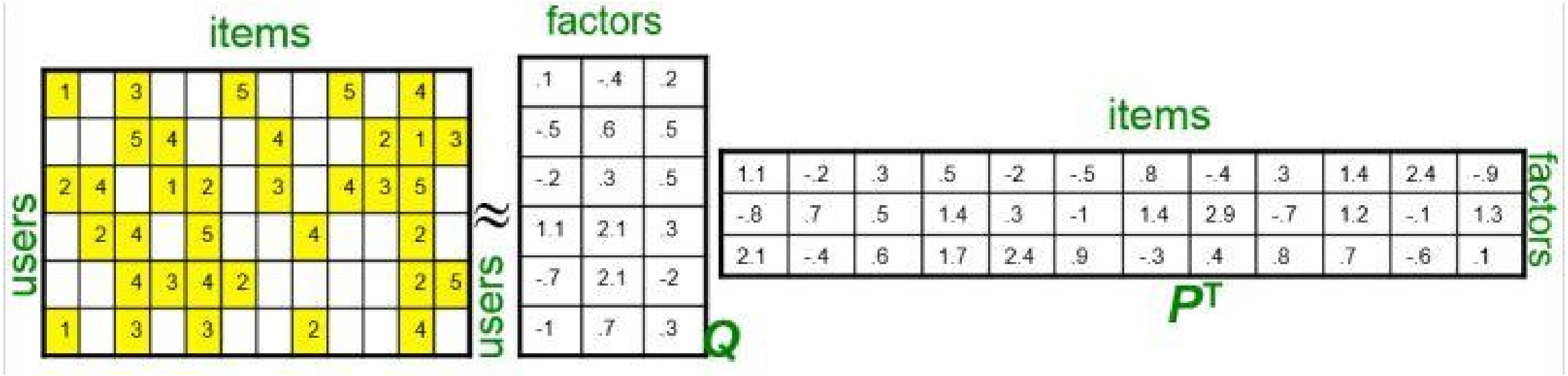
- 根据Q和P通过这种计算方式就可以得到不同用户对于不同物品的评分矩阵，最终的推荐结果就是每个人评分比较高的那些物品，比如张三就推荐音乐B，李四推荐音乐C，王五推荐音乐B。

	音乐A	音乐B	音乐C	音乐D
张三	0.68	1.58	0.28	0.51
李四	0.31	0.43	0.47	0.11
王五	1.06	1.57	0.73	0.69

隐语义模型

- 求解用户/物品-潜在因子矩阵
  - 假定有n个用户，m个物品，k/K个隐因子；R为用户-物品评分矩阵，Q为用户-潜在因子矩阵，P为物品-潜在因子矩阵。

$$R_{n*m} \approx Q_{n*k} \times P^T_{m*k} \qquad \hat{r}_{ui} = q_u p_i^T = \sum_{k=1}^{K} q_{uk} p_{ik}$$

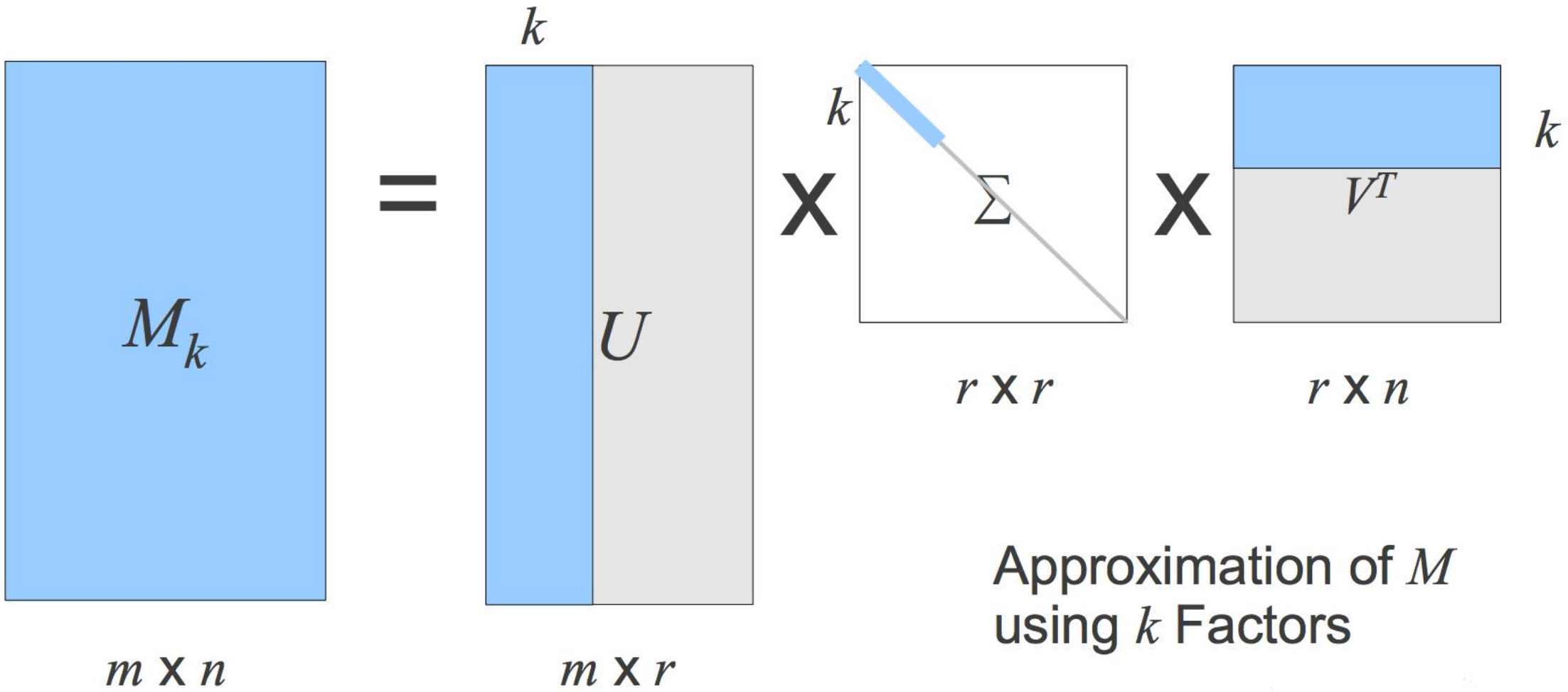


矩阵因子分解(SVD)推荐算法

- SVD(Singular Value Decomposition)的想法是根据已有的评分情况，分析出评分者对各个物品因子的喜好程度，以及各个物品对于这些因子的包含程度，最后再反过来根据分析结果预测评分。通过SVD的方式可以找出影响评分的显示因子和隐藏因子，这样更多有意义的关联关系就会被发现出来。
- SVD的数学定义：将给定评分矩阵R分解成为三个矩阵的乘积，其中U、V称为**左、右奇异向量**，Σ对角线上的值称为**奇异值**；其中R为n*m的矩阵，U为n*n的矩阵，Σ为n*m的矩阵，V为m*m的矩阵；可以使用前k个奇异值来近似的替代R矩阵，因为前1%的奇异值的和就占了全部奇异值和的99%以上。

$$R_{n*m} = U_{n*n} * \Sigma_{n*m} * V_{m*m} \qquad R_{n*m} \approx U_{n*k} * \Sigma_{k*k} * V^{T}_{k*m}$$

矩阵因子分解(SVD)推荐算法



Approximation of M
using k Factors

矩阵因子分解(SVD)推荐算法

- SVD分解要求矩阵是稠密的，也就是说矩阵的所有位置不能有空白。有空白时我们的是没法直接去SVD分解的。而且如果这个矩阵是稠密的，那不就是说我们都已经找到所有用户物品的评分了嘛，那还要SVD干嘛呢？ 的确，这是一个问题，传统SVD采用的方法是对评分矩阵中的缺失值进行简单的补全，比如用全局平均值或者用用户物品平均值补全，得到补全后的矩阵。接着可以用SVD分解。
- 使用上述简单的矩阵分解方式虽然可以解决缺省值的情况，但是由于推荐系统中的用户量和物品量是特别大的，所以直接使用原始SVD的矩阵分解是比较困难的。

矩阵因子分解(FunkSVD)推荐算法

- 由于现在在SVD矩阵分解中，将矩阵分解为三个矩阵效率比较低，那么我们可以通过矩阵分解为两个矩阵来降低执行的消耗； 即将U、V矩阵进行转换得到用户因子矩阵Q和物品因子矩阵P

$$Q_{n*k} = U_{n*k} * \left(\Sigma_{k*k}\right)^{1/2} \qquad P_{m*k} = \left(\left(\Sigma_{k*k}\right)^{1/2} \times V^{T}_{m*k}\right)^{T}$$

$$R_{\mathrm{n}*m} \approx Q_{n*k} * P^{T}_{m*k}$$

- 评分预测为r_{ui}为：

$$\hat{r}_{ui} = q_u p_i^T = \sum_{k=1}^{K} q_{uk} p_{ik}$$

矩阵因子分解(FunkSVD)推荐算法

- 用户因子矩阵Q和物品因子矩阵P的计算可以利用线性回归的思想，通过随机梯度下降的方式进行学习，迭代式的更新相关参数即可，使用SVD矩阵因子分解推荐算法对于评分稀疏矩阵也可以进行正常处理，对于没有评分的不用计算误差值，直接令误差值为0.

$$\hat{r}_{ui} = q_u p_i^T \qquad e_{ui} = r_{ui} - \hat{r}_{ui}$$

$$\min_{p_*, q_*} \frac{1}{2} \sum_{u,i} \left(r_{u,i} - q_u p_i^T \right)^2$$

$$p_i^{k+1} = p_i^k + \alpha \cdot e_{ui} \cdot q_u^k$$

$$q_u^{k+1} = q_u^k + \alpha \cdot e_{ui} \cdot p_i^k$$

矩阵因子分解(FunkSVD)推荐算法

- 在普通的SVD求解过程中，和机器学习类似，我们也需要防止Q和P的隐因子值不能过大，所以我们加入正则化项；从而可以得到下列目标函数：

$$\hat{r}_{ui} = q_u p_i^T \qquad e_{ui} = r_{ui} - \hat{r}_{ui}$$

$$\min_{q_*, p_*} \frac{1}{2}\left(\sum_{u,i} \left(r_{u,i} - q_u p_i^T \right)^2 + \lambda \left(\|q_u\|^2 + \|p_i\|^2 \right) \right)$$

$$p_i^{k+1} = p_i^k + \alpha \cdot \left(e_{ui} \cdot q_u^k - \lambda \cdot p_i^k \right)$$

$$q_u^{k+1} = q_u^k + \alpha \cdot \left(e_{ui} \cdot p_i^k - \lambda \cdot q_u^k \right)$$

矩阵因子分解(BiasSVD)推荐算法

- 同普通的协同过滤算法一样，我们可以更改一下FunkSVD预测值公式，可以认为最终的预测值是在基准评分/偏置项基础上的一个变化，从而我们可以得到下列预测公式：

$$\hat{r}_{ui} = \mu + b_u + b_i + q_u p_i^T$$

- 同样的我们可以得到一个加入正则化项后的目标函数：

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

$$\min_{q_u, p_i, b_u, b_i} \frac{1}{2}\left(\sum_{u,i}\left(r_{u,i} - \hat{r}_{u,i}\right)^2 + \lambda\left(b_u^2 + b_i^2 + \|q_u\|^2 + \|p_i\|^2\right)\right)$$

矩阵因子分解(BiasSVD)推荐算法

- 通过梯度下降法，我们最终可以得到b、q、p的迭代计算公式：

$$e_{ui} = r_{ui} - \hat{r}_{ui} \qquad \min_{q_u, p_i, b_u, b_i} \frac{1}{2}\left(\sum_{u,i}\left(r_{u,i} - \hat{r}_{u,i}\right)^2 + \lambda\left(b_u^2 + b_i^2 + \left\|q_u\right\|^2 + \left\|p_i\right\|^2\right)\right)$$

$$b_u^{k+1} = b_u^k + \alpha \cdot \left(e_{ui} - \lambda \cdot b_u^k\right)$$

$$b_i^{k+1} = b_i^k + \alpha \cdot \left(e_{ui} - \lambda \cdot b_i^k\right)$$

$$p_i^{k+1} = p_i^k + \alpha \cdot \left(e_{ui} \cdot q_u^k - \lambda \cdot p_i^k\right)$$

$$q_u^{k+1} = q_u^k + \alpha \cdot \left(e_{ui} \cdot p_i^k - \lambda \cdot q_u^k\right)$$

矩阵因子分解(SVD++)推荐算法

- SVD++算法是在BiasSVD算法的基础上加入了用户的隐式反馈；在BiasSVD矩阵分解中其实是没有考虑过用户隐式的反馈信息，比如浏览行为、点击行为等；所以可以在目标函数中加入这些隐式数据作为新的参数，从而得到一个新的预测公式为：

$$\hat{r}_{ui} = \mu + b_u + b_i + \left(q_u + |I_u|^{-\frac{1}{2}} \sum_{j \in I_u} y_j \right) p_i^T$$

- 对于该公式同样可以使用梯度下降法分别求解出b_u、b_i、q_u、y_j、p_i的值

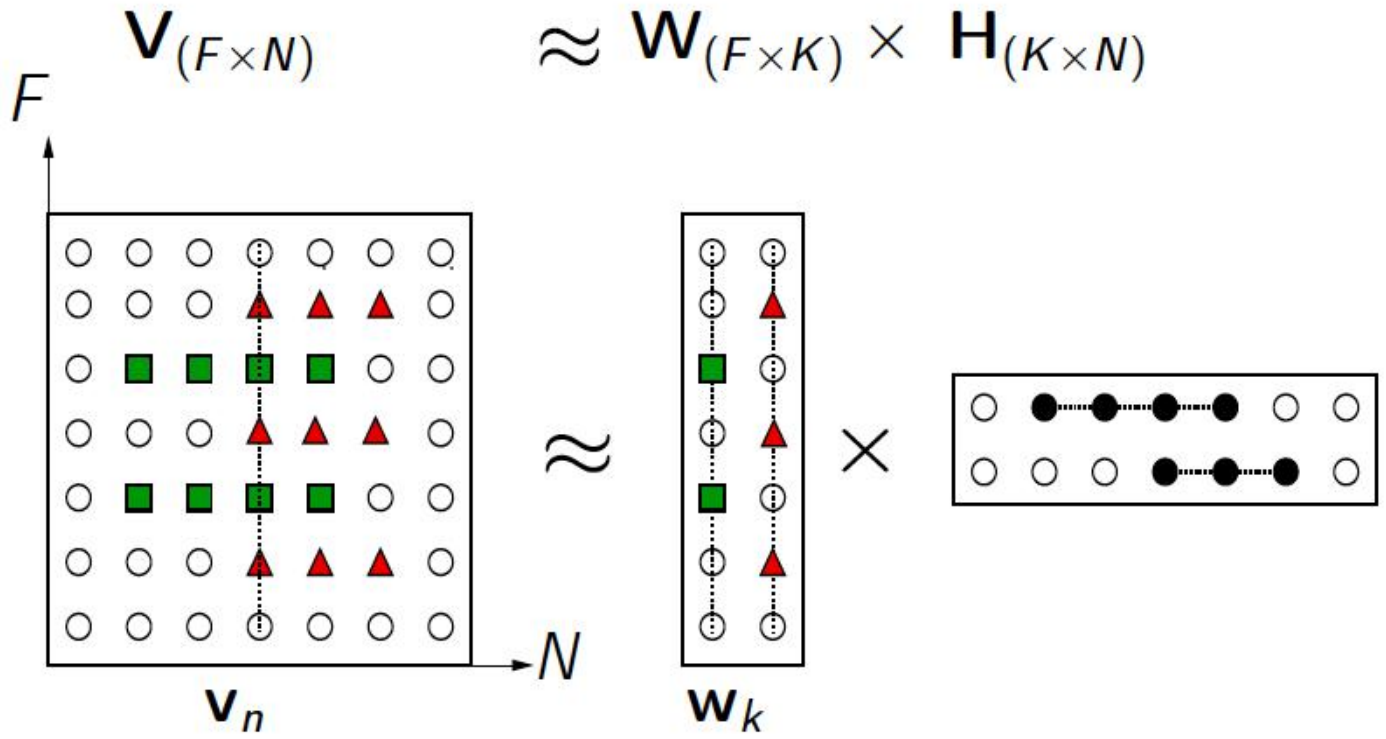
矩阵因子分解(NMF)推荐算法

- **非负矩阵分解**(Non-negative Matrix Factorization, NMF)是一种常用的矩阵分解方式，常用于矩阵分解、降维、主题模型、推荐系统等应用场景。
- NMF虽然和SVD一样都是矩阵分解，但是NMF不同的是：它的目标希望是将矩阵分解成为两个子矩阵， 并且这两个子矩阵中的元素都是非负数。
- 参考：https://www.csie.ntu.edu.tw/~cjlin/papers/pgradnmf.pdf

$$V_{m*n} \approx W_{m*k} H_{k*n}$$

矩阵因子分解(NMF)推荐算法

- 在NMF中求解出来的W和H，分别体现的是用户和隐因子之间的概率相关度，以及物品和隐因子之间的概率相关度；



矩阵因子分解(NMF)推荐算法

- NMF的期望是找到两个W、H矩阵，使得WH的矩阵乘积结果和对应的原矩阵V对应位置的值相比误差尽可能的小。

$$\min_{W,H} \quad f(W,H) \equiv \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(V_{ij}-(WH)_{ij}\right)^2$$

$$\text{subject to} \quad W_{ia} \geq 0, H_{bj} \geq 0, \quad \forall i,a,b,j.$$

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\left(V_{ij}-(WH)_{ij}\right)^2 = \|V-WH\|_F^2$$

矩阵因子分解(NMF)推荐算法

- NMF的目标函数中总共包含了m*k+k*n个参数，可以直接使用梯度下降法或者拟牛顿法来进行求解。

$$W^{k+1} = \max(0, W^k - \alpha_k \nabla_W f(W^k, H^k))$$

$$H^{k+1} = \max(0, H^k - \alpha_k \nabla_H f(W^k, H^k))$$

矩阵因子分解(NMF)推荐算法

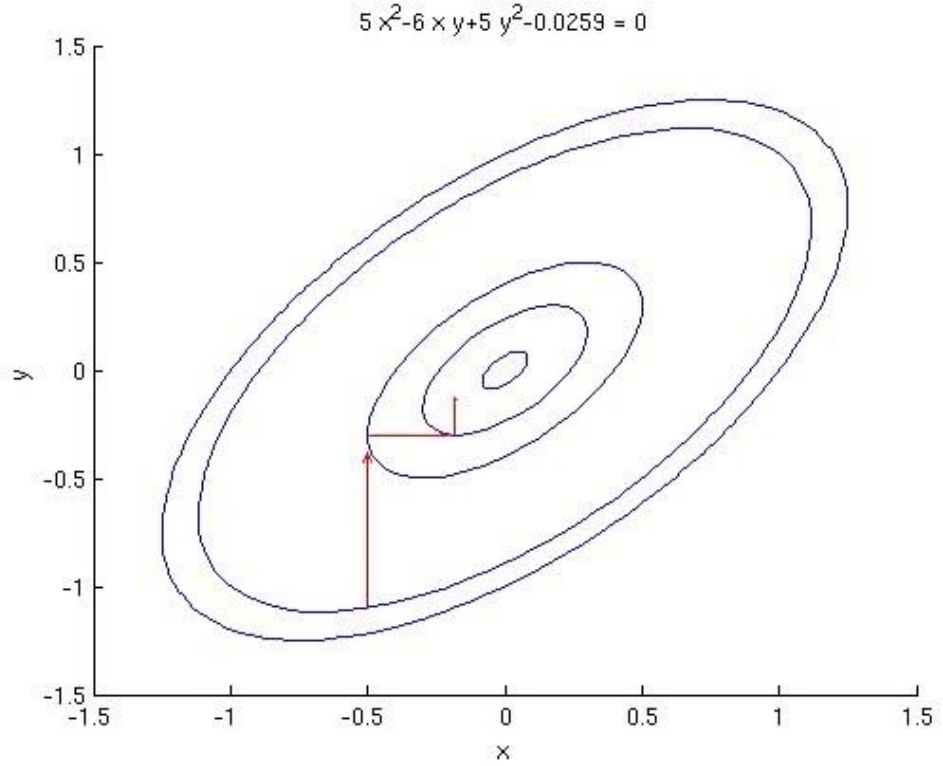
- 为了防止过拟合，也可以在NMF的目标函数的基础上添加一个正则化项

$$\frac{1}{2}||X-WH||_{Fro}^2+\alpha\rho||W||_1+\alpha\rho||H||_1+\frac{\alpha(1-\rho)}{2}||W||_{Fro}^2+\frac{\alpha(1-\rho)}{2}||H||_{Fro}^2$$

- 但是当加入L1正则项后，由于没法求解出正常的导函数出来(导函数不是连续的)，也就没法使用梯度下降法和拟牛顿法求解参数，此时一般采用坐标轴下降法来进行参数的求解。

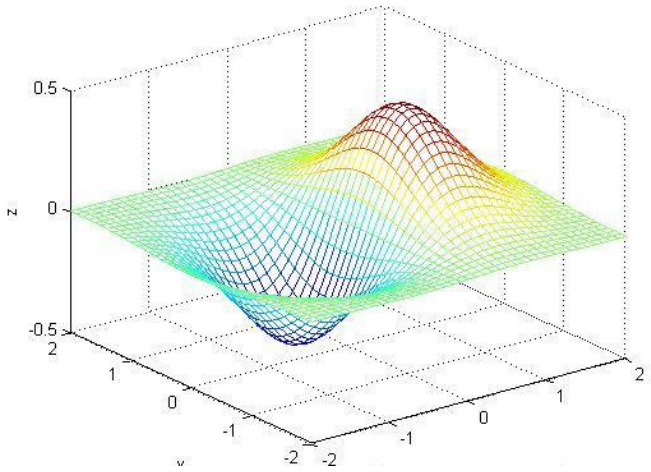
坐标轴下降法

- 坐标轴下降法(Coordinate Descent， CD)是一种迭代法，通过启发式的方法一步步的迭代求解函数的最小值，和梯度下降法(GD)不同的时候，坐标轴下降法是沿着坐标轴的方向去下降，而不是采用梯度的负方向下降。



坐标轴下降法

- 坐标轴下降法利用EM算法的思想，在参数更新过程中，每次均先固定m-1个参数值，求解剩下的一个参数的局部最优解；然后进行迭代式的更新操作。
- 坐标轴下降法的核心思想是多变量函数F(X)可以通过每次沿着一个方向优化来获取最小值；其数学依据是：对于一个可微凸函数f(θ)，其中θ为n*1的向量，如果对于一个解$\theta=(\theta_1,\theta_2,...,\theta_n)$，使得f(θ)在某个坐标轴$\theta_i(i=1,2,..,n)$上都能达到最小值，则$\theta=(\theta_1,\theta_2,...,\theta_n)$就是的f(θ)全局的最小值点



坐标轴下降法

- 在坐标轴下降法中，优化方向从算法的一开始就固定了，即沿着坐标的方向进行变化。在算法中，循环最小化各个坐标方向的目标函数。即：如果x^k给定，那么x^{k+1}的第i维度为:

$$\mathbf{x}_i^{k+1} = \arg\min_{y \in \mathbb{R}} f(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, y, x_{i+1}^{k}, \ldots, x_n^{k});$$

- 因此，从一个初始的x0求得函数F(x)的局部最优解，可以迭代获取x0、x1、x2...的序列，从而可以得到:

$$F(\mathbf{x}_0) \geq F(\mathbf{x}_1) \geq F(\mathbf{x}_2) \geq \cdots$$

坐标轴下降法算法过程

- 1. 给θ向量随机选取一个初值，记做θ^0；
- 2. 对于第k轮的迭代，从θ_1^k开始计算，θ_n^k到为止，计算公式如下：

$$\theta_1^k = \arg\min_{\theta_1} J\left(\theta_1, \theta_2^{k-1}, \theta_3^{k-1}, ..., \theta_n^{k-1}\right)$$

$$\theta_2^k = \arg\min_{\theta_2} J\left(\theta_1^k, \theta_2, \theta_3^{k-1}, ..., \theta_n^{k-1}\right)$$

.....

$$\theta_n^k = \arg\min_{\theta_n} J\left(\theta_1^k, \theta_2^k, \theta_3^k, ..., \theta_n\right)$$

- 检查θ^k和θ^{k-1}向量在各个维度上的变化情况，如果所有维度的变化情况都比较小的话，那么认为结束迭代，否则继续k+1轮的迭代。
- 备注：在求解每个参数局部最优解的时候可以求导的方式来求解。

矩阵分解推荐案例

- 分别使用SVD、NMF、SVD++算法对Movieline数据构建推荐，并比较三种方式的性能指标。

THANK YOU

上海育创网络科技有限公司