ALS 算法学习报告(理论分析)

一、算法名称：

协同过滤算法：Alternating least squares(ALS):ALS 是 alternating least squares 的缩写 ，意为交替最小二乘法；而 ALS-WR 是 alternating-least-squares with weighted-λ -regularization 的缩写，意为加权正则化交替最小二乘法。

二、算法用途：

该方法常用于基于矩阵分解的推荐系统中。例如：将用户(user)对商品(item)的评分矩阵分解为两个矩阵：一个是用户对商品隐含特征的偏好矩阵，另一个是商品所包含的隐含特征的矩阵。在这个矩阵分解的过程中，评分缺失项得到了填充，也就是说我们可以基于这个填充的评分来给用户最商品推荐了。

三、算法输入：用户对某事物的评价矩阵

四、算法输出：对缺少值的填充和推荐结果

五、算法思路：

由于评分数据中有大量的缺失项，传统的矩阵分解 SVD（奇异值分解）不方便处理这个问题，而 ALS 能够很好的解决这个问题。对于 R(m×n)的矩阵，ALS 旨在找到两个低维矩阵 X(m×k)和矩阵 Y(n×k)，来近似逼近 R(m×n)，即

$$R_{m\times n} \approx X_{m\times k} Y_{n\times k}^T$$

其中 R(m×n)代表用户对商品的评分矩阵，X(m×k)代表用户对隐含特征的偏好矩阵，Y(n×k)表示商品所包含隐含特征的矩阵，T 表示矩阵 Y 的转置。实

际中，一般取 k<<min(m, n)，也就是相当于降维了。这里的低维矩阵，有的地方也叫低秩矩阵。

	item 1	item 2	item 3	item 4
user 1	R11	R12	R13	R14
user 2	R21	R22	R23	R24
user 3	R31	R32	R33	R34

=

	class 1	class 2	class 3
user 1	P11	P12	P13
user 2	P21	P22	P23
user 3	P31	P32	P33

×

	item 1	item 2	item 3	item 4
class 1	Q11	Q12	Q13	Q14
class 2	Q21	Q22	Q23	Q24
class 3	Q31	Q32	Q33	Q34

为了找到使低秩矩阵 X 和 Y 尽可能地逼近 R，需要最小化下面的平方误差损失函数：

$$L(X,Y)=\sum_{u,i}(r_{ui}-x_u^T y_i)^2......(1)$$

其中 xu(1×k)表示示用户 u 的偏好的隐含特征向量，yi(1×k)表示商品 i 包含的隐含特征向量，rui 表示用户 u 对商品 i 的评分，向量 xu 和 yi 的内积 xuTyi 是用户 u 对商品 i 评分的近似。

损失函数一般需要加入正则化项来避免过拟合等问题，我们使用 L2 正则化，所以上面的公式改造为：

$$L(X,Y)=\sum_{u,i}(r_{ui}-x_u^T y_i)^2+\lambda(|x_u|^2+|y_i|^2)......(2)$$

其中λ是正则化项的系数。

弗罗贝尼乌斯范数

对 p = 2，这称为弗罗贝尼乌斯范数（Frobenius norm）或希尔伯特-施密特范数（ Hilbert–Schmidt norm），不过后面这个术语通常只用于希尔伯特空

间。这个范数可用不同的方式定义：

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} = \sqrt{\operatorname{trace}(A^*A)} = \sqrt{\sum_{i=1}^{\min\{m,n\}} \sigma_i^2}$$

这里 A^* 表示 A 的共轭转置，σ_i 是 A 的奇异值，并使用了迹函数。弗罗贝尼乌斯范数与 K^n 上欧几里得范数非常类似，来自所有矩阵的空间上一个内积。

到这里，协同过滤就成功转化成了一个优化问题。由于变量 xu 和 yi 耦合到一起，这个问题并不好求解，所以我们引入了 ALS，也就是说我们可以先固定 Y（例如随机初始化 X），然后利用公式（2）先求解 X，然后固定 X，再求解 Y，如此交替往复直至收敛，即所谓的交替最小二乘法求解法。

六、算法逻辑：

使用 ALS 来解决低秩近似矩阵分解问题的步骤如下：

1. 使用指定电影的平均得分作为矩阵 M 的第一行，余下的行值使用小的随机值来填充。
2. 使用 squared errors 和的最小值来填充 U 矩阵。
3. 相似的使用 squared errors 和的最小值来填充 M 矩阵。
4. 重复 2,3 步直到满足了停止标准（stopping criterion（））。

这里使用的停止标准是基于在探测数据集上观测的 RMSE，修改 U 和 M 的第一次循环之后，如果探测数据集上的观测的 RMSE 之间的差异小于 1 个基点，那么该循环就会停止并且会使用获得的 U,M 来作为测试数据集上的最后预测根

据。该探测数据集是由 Netflix 提供的，它与那些隐藏的测试数据集有相同的结构。

具体求解方法说明如下：

先固定 Y，将损失函数 L(X,Y)对 xu 求偏导，并令导数=0，得到：

$$
\begin{aligned}
& \frac{1}{2}\frac{\partial f}{\partial u_{ki}} = 0, \quad \forall i, k \\
\Rightarrow & \sum_{j \in I_i^U} (\mathbf{u}_i^T \mathbf{m}_j - r_{ij}) m_{kj} + \lambda n_{u_i} u_{ki} = 0, \quad \forall i, k \\
\Rightarrow & \sum_{j \in I_i^U} m_{kj} \mathbf{m}_j^T \mathbf{u}_i + \lambda n_{u_i} u_{ki} = \sum_{j \in I_i^U} m_{kj} r_{ij}, \quad \forall i, k \\
\Rightarrow & \left(M_{I_i^U} M_{I_i^U}^T + \lambda n_{u_i} E \right) \mathbf{u}_i = M_{I_i^U} \ R^T(i, I_i^U), \quad \forall i \\
\Rightarrow & \mathbf{u}_i = A_i^{-1} V_i, \quad \forall i
\end{aligned}
$$

这里，

$$A_i = M_{I_i^U} M_{I_i^U}^T + \lambda n_{u_i} E, \ V_i = M_{I_i^U} \ R^T(i, I_i^U),$$

E 是 $N_f \times N_f$ 的单位矩阵同理固定 X，

同理可得：

$$\mathbf{m}_j = A_j^{-1} V_j, \quad \forall j,$$

其中，

$$A_j = U_{I_i^M} U_{I_i^M}^T + \lambda n_{m_j} E \text{ and } V_j = U_{I_i^M} R(I_i^M, j).$$

迭代步骤：首先随机初始化 Y，利用公式(3)更新得到 X， 然后利用公式(4)更新 Y， 直到均方根误差变 RMSE 化很小或者到达最大迭代次数。

$$\tilde{R} = XY$$

$$RMSE = \sqrt{\frac{\sum (R - \tilde{R})^2}{N}}$$

上文提到的模型适用于解决有明确评分矩阵的应用场景，然而很多情况下，用户没有明确反馈对商品的偏好，也就是没有直接打分，我们只能通过用户的某些行为来推断他对商品的偏好。比如，在电视节目推荐的问题中，对电视节目收看的次数或者时长，这时我们可以推测次数越多，看得时间越长，用户的偏好程度越高，但是对于没有收看的节目，可能是由于用户不知道有该节目，或者没有途径获取该节目，我们不能确定的推测用户不喜欢该节目。

七、算法扩展：

ALS-WR 通过置信度权重来解决这些问题：对于更确信用户偏好的项赋以较大的权重，对于没有反馈的项，赋以较小的权重。ALS-WR 模型的形式化说明如下：

ALS-WR 的目标函数：

$$\min_{x_u,y_i} L(X,Y) = \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(|x_u|^2 + |y_i|^2) \; ......(5)$$

$$p_{ui} = \begin{cases} 1 & \text{if } r_{ui} > 0 \\ 0 & \text{if } r_{ui} = 0 \end{cases}$$

$$c_{ui} = 1 + \alpha r_{ui}$$

其中α是置信度系数。

求解方式还是最小二乘法：

$$x_u = (Y^T C^u Y + \lambda I)^{-1} Y^T C^u r_u \; ......(6)$$

$$y_i = (X^T C^i X + \lambda I)^{-1} X^T C^i r_i ......(7)$$

其中 C^u 是 n×n 的对角矩阵，Ci 是 m×m 的对角矩阵；$C^u_{ii} = c_{ui}$， $C^i_{ii} = c_{ii}$。

八、算法效率分析：

算法的时间复杂度是：运算的每一步都需要更新 U 和 M，更新 U 的时间复杂度是：更新 U 的时间复杂度是、$O\left(n_f^2(n_r + n_f n_u)\right)$，更新 M 的时间复杂度是：$O\left(n_f^2(n_r + n_f n_m)\right)$，程序运行了 Nt 步迭代完成执行，总的时间复杂度是：$O\left(n_f^2(n_r + n_f n_u + n_f n_m) n_t\right)$.

参考文献：

[1] http://blog.csdn.net/oucpowerman/article/details/49847979
[2] http://www.educity.cn/wenda/387428.html

两篇论文：

[1] Large-Scale Parallel Collaborative Filtering for the Netflix Prize
[2] Matrix factorization techniques for recommender systems