

## 推荐系统中SVD算法详解

转载

2017年03月17日 20:53:12

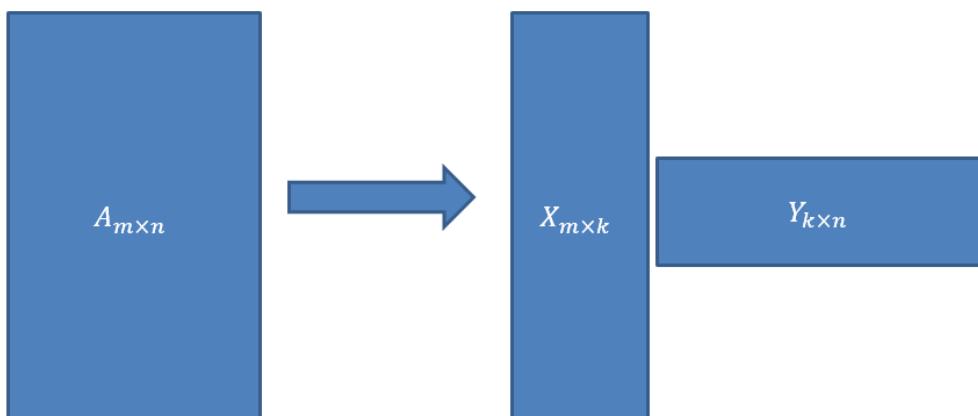
1484

### SVD算法详解

下面开始介绍SVD算法，假设存在以下user和item的数据矩阵：

User / item	1	2	3	4	5	.....
1	5	4	4.5	?	3.9	.....
2	?	4.5	?	4.5	?	.....
3	4.5	?	4.4	4	4	.....
4	?	4.8	?	?	4.5	.....
5	4	?	4.5	5	?	.....
.....	.....	.....	.....	.....	.....	.....

这是一个极其稀疏的矩阵，这里把这个评分矩阵记为R，其中的元素表示user对item的打分，“?”表示未知的，也就是要你去预测的，现在问题来了：如何去预测未知的评分值呢？上一篇文章用SVD证明了对任意一个矩阵A，都有它的满秩分解：



其中 $k = \text{Rank}(A)$

那么刚才的评分矩阵R也存在这样一个分解，所以可以用两个矩阵P和Q的乘积来表示评分矩阵R：

$$R_{U \times I} = P_{U \times K} Q_{K \times I}$$

上图中的U表示用户数，I表示商品数。然后就是利用R中的已知评分训练P和Q使得P和Q相乘的结果最好地拟合已知的评分，那么未知的评分也就可以用P的某一行乘上Q的某一列得到了：

$$\hat{r}_{ui} = p_u^T q_i$$

这是预测用户u对商品i的评分，它等于P矩阵的第u行乘上Q矩阵的第i列。这个是最基本的SVD算法，那么如何通过已知评分训练得到P和Q的具体数值呢？

假设已知的评分为：

$$r_{ui}$$

则真实值与预测值的误差为：

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

继而可以计算出总的误差平方和：

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

只要通过训练把SSE降到最小那么P、Q就能最好地拟合R了。那又如何使SSE降到最小呢？下面介绍一个常用的局部优化算法。

## 梯度下降法

为了说明梯度下降法，我找了一张PPT：

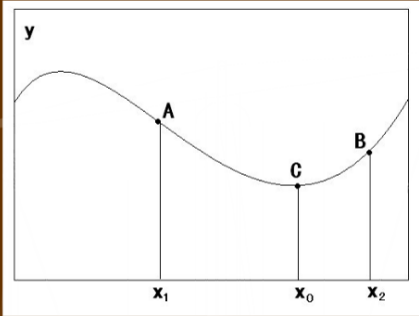
# 局部优化算法之一：梯度下降法

- 见右图。局部极小值是C点 ( $x_0$ )。
- 梯度，即导数，但是有方向，是一个矢量。曲线情况下，表达式为

$$f'(x) = \frac{dy}{dx}$$

如果， $f'(x) > 0$ ，则x增加，y也增加，相当于B点；如果 $f'(x) < 0$ ，则x增加，y减小，相当于A点。

要搜索极小值C点，在A点必须向x增加方向搜索，此时与A点梯度方向相反；在B点必须向x减小方向搜索，此时与B点梯度方向相反。总之，搜索极小值，必须向负梯度方向搜索。



也就是说如果要最小化目标函数，必须往其负梯度方向搜索。这就是梯度下降法，注意它是一个局部优化算法，也就是说有可能落到局部最优解而不是全局最优解。

## Basic SVD

利用梯度下降法可以求得SSE在Puk变量（也就是P矩阵的第u行第k列的值）处的梯度：

$$\frac{\partial}{\partial p_{uk}} SSE = \frac{\partial}{\partial p_{uk}} (e_{ui}^2)$$

利用求导链式法则， $e^2$ 先对e求导再乘以e对Puk的求导：

$$\frac{\partial}{\partial p_{uk}} (e_{ui}^2) = 2e_{ui} \frac{\partial}{\partial p_{uk}} e_{ui}$$

由于

$$e_{ui} = r_{ui} - \hat{r}_{ui} = r_{ui} - p_u^T q_i = r_{ui} - \sum_{k=1}^K p_{uk} q_{ki}$$

所以

$$\frac{\partial}{\partial p_{uk}} e_{ui} = \frac{\partial}{\partial p_{uk}} \left( r_{ui} - \sum_{k=1}^K p_{uk} q_{ki} \right)$$

上式中括号里的那一坨式子如果展开来看的话，其与Puk有关的项只有PukQki，其他的无关项对Puk的求导均等于0

所以求导结果为：

$$\frac{\partial}{\partial p_{uk}} e_{ui} = \frac{\partial}{\partial p_{uk}} \left( r_{ui} - \sum_{k=1}^K p_{uk} q_{ki} \right) = -q_{ki}$$

所以

$$\frac{\partial}{\partial p_{uk}} SSE = \frac{\partial}{\partial p_{uk}} (e_{ui}^2) = 2e_{ui} \frac{\partial}{\partial p_{uk}} e_{ui} = -2e_{ui} q_{ki}$$

为了让式子更简洁，令

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

这样做对结果没有影响，只是为了把求导结果前的2去掉，更好看点。得到

$$\frac{\partial}{\partial p_{uk}} SSE = -e_{ui} q_{ki}$$

现在得到了目标函数在Puk处的梯度了，那么按照梯度下降法，将Puk往负梯度方向变化：

令更新的步长（也就是学习速率）为

$$\eta$$

则Puk的更新式为

$$p_{uk} := p_{uk} - \eta(-e_{ui} q_{ki}) := p_{uk} + \eta e_{ui} q_{ki}$$

同样的方式可得到Qik的更新式为

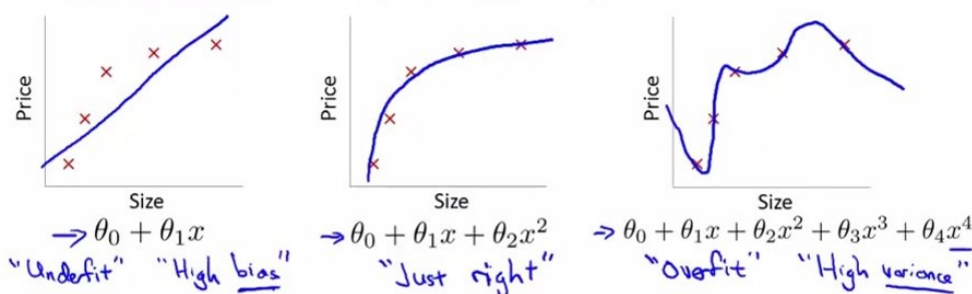
$$q_{ki} := q_{ki} - \eta(-e_{ui} p_{uk}) := q_{ki} + \eta e_{ui} p_{uk}$$

得到了更新的式子，现在开始来讨论这个更新要怎么进行。有两种选择：1、计算完所有已知评分的预测误差后再对P、Q进行更新。2、每计算完一个eui后立即对Pu和qi进行更新。这两种方式都有名称，分别叫：1、批梯度下降。2、随机梯度下降。两者的区别就是批梯度下降在下一轮迭代才能使用本次迭代的更新值，随机梯度下降本次迭代中当前样本使用的值可能就是上一个样本更新的值。由于随机性可以带来很多好处，比如有利于避免局部最优解，所以现在大多倾向于使用随机梯度下降进行更新。

## RSVD

上面就是基本的SVD算法，但是，问题来了，上面的训练是针对已知评分数据的，过分地拟合这部分数据有可能导致模型的测试效果很差，在测试集上面表现很糟糕。这就是过拟合问题，关于过拟合与欠拟合可以看一下这张图

### Example: Linear regression (housing prices)



**Overfitting:** If we have too many features, the learned hypothesis may fit the training set very well ( $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \approx 0$ ), but fail to generalize to new examples (predict prices on new examples).

第一个是欠拟合，第二个刚好，第三个过拟合。那么如何避免过拟合呢？那就是在目标函数中加入正则化参数（加入惩罚项），对于目标函数来说，P矩阵和Q矩阵中的所有值都是变量，这些变量在不知道哪个变量会带来过拟合的情况下，对所有变量都进行惩罚：

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

这时候目标函数对Puk的导数就发生了变化了，现在就来求加入惩罚项后的导数。

$$\begin{aligned} SSE &= \frac{1}{2} \sum_{u,i} e_{ui}^2 + \frac{1}{2} \lambda \sum_u |p_u|^2 + \frac{1}{2} \lambda \sum_i |q_i|^2 \\ &= \frac{1}{2} \sum_{u,i} e_{ui}^2 + \frac{1}{2} \lambda \sum_u \sum_{k=0}^K p_{uk}^2 + \frac{1}{2} \lambda \sum_i \sum_{k=0}^K q_{ki}^2 \end{aligned}$$

括号里第一项对Puk的求导前面已经求过了，第二项对Puk的求导很容易求得，第三项与Puk无关，导数为0，所以

$$\frac{\partial}{\partial p_{uk}} SSE = -e_{ui} q_{ki} + \lambda p_{uk}$$

同理可得SSE对qik的导数为

$$\frac{\partial}{\partial q_{ik}} SSE = -e_{ui} p_{uk} + \lambda q_{ik}$$

将这两个变量往负梯度方向变化，则更新式为

$$\begin{aligned} p_{uk} &:= p_{uk} + \eta (e_{ui} q_{ki} - \lambda p_{uk}) \\ q_{ki} &:= q_{ki} + \eta (e_{ui} p_{uk} - \lambda q_{ik}) \end{aligned}$$

这就是正则化后的SVD，也叫RSVD。

加入偏置的SVD、RSVD

关于SVD算法的变种太多了，叫法也不统一，在预测式子上加参数又会出来一个名称。由于用户对商品的打分不仅取决于用户和商品间的某种关系，还取决于用户和商品独有的性质，Koren将SVD的预测公式改成这样

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

第一项为总的平均分，bu为用户u的属性值，bi为商品i的属性值，加入的这两个变量在SSE式子中同样需要惩罚，那么SSE就变成了下面这样：

$$\begin{aligned} SSE &= \frac{1}{2} \sum_{u,i} e_{ui}^2 + \frac{1}{2} \lambda \sum_u |p_u|^2 + \frac{1}{2} \lambda \sum_i |q_i|^2 + \frac{1}{2} \lambda \sum_u b_u^2 + \frac{1}{2} \lambda \sum_i b_i^2 \\ &= \frac{1}{2} \sum_{u,i} (r_{ui} - \mu - b_u - b_i - \sum_{k=1}^K p_{uk} q_{ki})^2 + \frac{1}{2} \lambda \sum_u |p_u|^2 + \frac{1}{2} \lambda \sum_i |q_i|^2 + \frac{1}{2} \lambda \sum_u b_u^2 + \frac{1}{2} \lambda \sum_i b_i^2 \end{aligned}$$

由上式可以看出SSE对Puk和qik的导数都没有变化，但此时多了bu和bi变量，同样要求出其更新式。首先求SSE对bu的导数，只有第一项和第四项和bu有关，第一项对bu的求导和之前的求导类似，用链式法则即可求得，第四项直接求导即可，最后可得偏导数为

$$\frac{\partial}{\partial b_u} SSE = -e_{ui} + \lambda b_u$$

同理可得对bi的导数为

$$\frac{\partial}{\partial b_i} SSE = -e_{ui} + \lambda b_i$$

所以往其负梯度方向变化得到其更新式为

$$\begin{aligned} b_u &:= b_u + \eta(e_{ui} - \lambda b_u) \\ b_i &:= b_i + \eta(e_{ui} - \lambda b_i) \end{aligned}$$

这就是修改后的SVD（RSVD）。

## ASVD

全称叫Asymmetric-SVD，即非对称SVD，其预测式子为

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T (|R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} (r_{uj} - \mu - b_u - b_j) x_j + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j)$$

R(u)表示用户u评过分的商品集合，N(u)表示用户u浏览过但没有评过分的商品集合，Xj和Yj是商品的属性。这个模型很有意思，看预测式子，用户矩阵P已经被去掉了，取而代之的是利用用户评过分的商品和用户浏览过尚未评分的商品属性来表示用户属性，这有一定的合理性，因为用户的行为记录本身就能反应用户的喜好。而且，这个模型可以带来一个很大的好处，一个商场或者网站的用户数成千上万甚至过亿，存储用户属性的二维矩阵会占用巨大的存储空间，而商品数却没有那么多，所以这个模型的好处显而易见。但是它有个缺点，就是迭代时间太长了，这是可以预见的，以时间换空间嘛。

同样的，需要计算其各个参数的偏导数，求出更新式，显然，bu和bi的更新式和刚才求出的一样

$$\begin{aligned} SSE &= \frac{1}{2} \sum_{u,i} e_{ui}^2 + \frac{1}{2} \lambda \sum_{j \in R(u)} |x_j|^2 + \frac{1}{2} \lambda \sum_{j \in N(u)} |y_j|^2 + \frac{1}{2} \lambda \sum_i |q_i|^2 + \frac{1}{2} \lambda \sum_u b_u^2 + \frac{1}{2} \lambda \sum_i b_i^2 \\ &= \frac{1}{2} \sum_{u,i} (r_{ui} - \mu - b_u - b_i - \sum_{m=1}^F z_m q_{mi})^2 + \frac{1}{2} \lambda \sum_{j \in R(u)} |x_j|^2 + \frac{1}{2} \lambda \sum_{j \in N(u)} |y_j|^2 + \frac{1}{2} \lambda \sum_i |q_i|^2 \\ &\quad + \frac{1}{2} \lambda \sum_u b_u^2 + \frac{1}{2} \lambda \sum_i b_i^2 \end{aligned}$$

其中的向量z等于下面这坨

$$z = |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} (r_{uj} - \mu - b_u - b_j) x_j + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$$

现在要求qik、x和y的更新式，在这里如果把z看成Pu则根据前面求得的更新式可得qik的更新式：

$$q_{ki} = q_{ki} + \eta (e_{ui} z_k - \lambda q_{ik})$$

求Xj的导数需要有点耐心，SSE的第二项对Xj的导数很容易得到，现在来求第一项对Xj的导数：

$$\frac{\partial}{\partial x_{jk}} \left( \frac{1}{2} \sum_{u,i} e_{ui}^2 \right) = \sum_{i,j \in R(u)} (e_{ui} \frac{\partial}{\partial x_{jk}} e_{ui}) = \sum_{i,j \in R(u)} (e_{ui} \frac{\partial}{\partial x_{jk}} (r_{ui} - \mu - b_u - b_i - \sum_{m=1}^F z_m q_{mi}))$$

上式求和符中的i,j都是属于R(u)的，可以看到Zm只有当m=k时才与Xjk有关，所以

$$\frac{\partial}{\partial x_{jk}} (r_{ui} - \mu - b_u - b_i - \sum_{m=1}^F z_m q_{mi}) = - \frac{\partial}{\partial x_{jk}} \sum_{m=1}^F z_m q_{mi} = -q_{ki} \frac{\partial z_k}{\partial x_{jk}}$$

因为

$$z_k = |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} (r_{uj} - \mu - b_u - b_j) x_{jk} + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_{jk}$$

所以

$$\frac{\partial z_k}{\partial x_{jk}} = |R(u)|^{-\frac{1}{2}} (r_{uj} - \mu - b_u - b_j)$$

所以得到SSE对Xjk的导数为

$$\frac{\partial}{\partial x_{jk}} (SSE) = - \left[ \sum_{j \in R(u), i \in R(u)} (e_{ui} q_{ki} |R(u)|^{-\frac{1}{2}} (r_{uj} - \mu - b_u - b_j)) \right] + \lambda x_{jk}$$

同理可得SSE对Yjk的导数为

$$\frac{\partial}{\partial y_{jk}} (SSE) = - \left[ \sum_{j \in N(u), i \in R(u)} (e_{ui} q_{ki} |N(u)|^{-\frac{1}{2}}) \right] + \lambda y_{jk}$$

所以得到Xjk和Yjk的更新方程

$$x_{jk} = x_{jk} + \eta \left\{ \left[ \sum_{j \in R(u), i \in R(u)} (e_{ui} q_{ki} |R(u)|^{-\frac{1}{2}} (r_{uj} - \mu - b_u - b_j)) \right] - \lambda x_{jk} \right\}$$

$$y_{jk} = y_{jk} + \eta \left\{ \left[ \sum_{j \in N(u), i \in R(u)} (e_{ui} q_{ki} |N(u)|^{-\frac{1}{2}}) \right] - \lambda y_{jk} \right\}$$

这就叫ASVD。。。。。

**SVDPP**

最后这个模型也是Koren文章中提到的，SVDPlusPlus (SVD++)，它的预测式子为

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

这里的N(u)表示用户u行为记录（包括浏览的和评过分的商品集合）。看了ASVD的更新式推导过程再来看这个应该很简单，Puk和qik的更新式子不变，Yjk的更新式子和ASVD的更新式一样：

$$y_{jk} := y_{jk} + \eta \left\{ \left[ \sum_{j \in N(u), i \in R(u)} (e_{ui} q_{ki} |N(u)|^{-\frac{1}{2}}) \right] - \lambda y_{jk} \right\}$$

这些就是Koren在NetFlix大赛中用到的SVD算法，最后，还有一个需要提的：

## 对偶算法

将前面预测公式中的u和i调换位置得到其对偶算法，对于RSVD而言，u和i的位置是等价的、对称的，所以其对偶算法和其本身没有区别，但对于ASVD和SVD++则不同，有时候对偶算法得到的结果更加精确，并且，如果将对偶算法和原始算法的预测结果融合在一起的话，效果的提升会让你吃惊！

对偶的ASVD预测公式：

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T(|R(i)|^{-\frac{1}{2}} \sum_{v \in R(i)} (r_{vi} - \mu - b_i - b_v)x_v + |N(i)|^{-\frac{1}{2}} \sum_{v \in N(i)} y_v)$$

这里R(i)表示评论过商品i的用户集合，N(i)表示浏览过商品i但没有评论的用户集合。由于用户数量庞大，所以对偶的ASVD会占用很大空间，这里需要做取舍了。

对偶的SVD++预测公式：

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T(q_i + \frac{1}{\sqrt{|N(i)|}} \sum_{v \in N(i)} y_v)$$

这里N(i)表示对商品i有过行为（浏览或评分）的用户集合。

实现这一对偶的操作其实很简单，只要读取数据的时候把用户id和商品id对调位置即可，也就是将R矩阵转置后再训练。

暂时实现的算法就这些，代码都在github上了，有兴趣的可以看看，地址：<https://github.com/jingchenUSTC/SVDRecommenderSystem>