

推荐系统编程作业1: PMF

作者：邓梓烽

时间：2021/04/11

一、PMF的基本原理

(一) 问题描述

(二) 基本思路

1. 任务目标

2. 朴素PMF模型

(1) 基本假设

(2) 求解方法

(3) 控制模型复杂度

二、悬而未决的问题

三、代码实现

(一) 所使用的数据集

(二) 参数设置

四、结果分析

(一) 原文的实验方法以及结果分析

1. 数据集

2. 训练方法

3. 结果分析

(二) 复现代码的结果分析

1. 数据集

2. 结果展示

(1) 运行截图

(2) 结果分析

五、总结与收获

参考资料

本次作业主要参考的文章是 [Ruslan Salakhutdinov, Andriy Mnih](#) 在2007的NIPS上发表的“[Probabilistic Matrix Factorization \(PMF\)](#)”[1](#)。

一、PMF的基本原理

（一）问题描述

假设有 N 个用户， M 部电影，则对应的评分系统就可以用一个 $N \times M$ 的矩阵 R 来表示。

推荐系统所面对的问题是： R 中只有部分entries是已知的（每个用户都只给部分电影打了分），而且 R 往往非常稀疏，我们需要求出 R 中某些缺失entries的值。

（二）基本思路

1. 任务目标

文章采用了**low-dimension factor model**，也即**low rank model**来对问题进行建模。其核心思想就是：用户和电影之间的关系（或称用户的“兴趣”）由少数几个显著因素所决定（具体运算为线性组合）。

类似于PCA的思想，只不过PCA求出来的几个主成分是原来所有成分的线性组合，是virtual的。

举个例子：假设一个用户是否喜欢一部电影取决于三个因素——电影分类（文艺片 via 娱乐片）、电影的语言（外语 via 华语）以及演员的咖位（小众 via 明星）。我们可以用一个三维向量来描述一个用户的这种偏好，例如 $\mathbf{x} = [0.6, 1.0, -0.2]^T$ 就表示该用户比较喜欢娱乐片，而且只看华语片，对演员的咖位没有很强的倾向性，但小众一点会更好。同样地，我们也可以用三维向量来表征一部电影，例如 $\mathbf{y} = [0.9, -1.0, 0.8]^T$ 就表示这是一部众星云集的外语娱乐巨制。使用向量表示法的优点就是，我们可以直接使用点积（也可以归一化，即计算协方差）来估计一个用户对于一部电影的喜好程度。例如，对于上面提到的用户和电影，该用户对该电影的喜好就是 $r = \mathbf{x}^T \mathbf{y} = -2.06$ ，可以看出该用户其实并不喜欢这部电影。

如果要用矩阵语言来描述，就是评分矩阵 R 可以分解为两个低秩矩阵 U, V 的乘积，即 $R = U^T V$ ，其中 $D \times N$ 矩阵 U 描述了 N 个用户的喜好；而 $D \times M$ 矩阵则描述了 M 部电影的特征。由矩阵秩的性质我们可以推得：

$$\text{rank}(R) \leq \min\{\text{rank}(U), \text{rank}(V)\}$$

但是，由于评分矩阵中 **非常稀疏**，以及 **噪声** 的存在，实际情况下不可能直接得到这种完美的分解，因此我们做如下的处理：

- 我们不对原评分矩阵 R 进行分解，而是对一个 近似矩阵 \hat{R} 进行分解，即作 $\hat{R} = U^T V$ 。
- 我们要求 \hat{R} 在已有的观测值（ratings）上与真实的评分矩阵 R 尽可能地相似。
- 为了防止过拟合，我们还需要对 U 和 V 做出适当的约束。

从贝叶斯观点来看待这个问题的话， R 是我们观测到的值，而 U, V 则是对系统内部特征的描述。

2. 朴素PMF模型

(1) 基本假设

1. 观测噪声（ R 与 \hat{R} 之间的差值）服从 高斯分布；
2. 用户属性 U 和 电影属性 V 均服从 高斯分布，且假定它们相互独立。

假设2存疑，详见自己在Web Page上做的笔记。

利用假设1，我们可以得到真实评分矩阵上的概率密度函数，其中的参数 σ 是**噪声的方差**，是我们人为指定的。

$$p(R | U, V, \sigma^2) = \prod_{i=1}^N \prod_{j=1}^M [\mathcal{N}(R_{ij} | U_i^T V_j, \sigma^2)]^{I_{ij}}$$

- $\mathcal{N}(x | \mu, \sigma^2)$ 是具有均值 μ 以及方差 σ^2 的高斯分布；
- $I_{ij} = \begin{cases} 1 & \text{if user } i \text{ has rated movie } j, \\ 0 & \text{if user } i \text{ has not rated movie } j. \end{cases}$

利用第二个假设，我们也可以写出用户偏好、电影特征的概率密度函数，其中 σ_U 和 σ_V 是先验噪声的方差，人为指定。

$$p(U | \sigma_U^2) = \prod_{i=1}^N \mathcal{N}(U_i | 0, \sigma_U^2 \mathbf{I})$$

$$p(V | \sigma_V^2) = \prod_{j=1}^M \mathcal{N}(V_j | 0, \sigma_V^2 \mathbf{I})$$

综合以上两个概率密度函数，我们可以得到对应的后验概率

$$p(U, V | R, \sigma^2, \sigma_V^2, \sigma_U^2) = p(U, V, R) / p(R) \propto p(U, V, R) = p(R | U, V) p(U) p(V)$$

(2) 求解方法

要通过已有的评分矩阵 R 估计出系统参数 U, V ，我们可以采用最大化上述概率函数的方法。

为了计算上的方便，我们对后验概率取对数

$$\ln p(U, V | R, \sigma^2, \sigma_V^2, \sigma_U^2) = \ln p(R | U, V) + \ln p(U) + \ln p(V)$$

这里为了清晰起见，忽略掉了原文中该公式后面的“常数项”——剩余的项与我们人为事先指定的各个方差有关，而与训练得到的参数无关。

【高斯分布及其对数形式】

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

$$\ln p(x) = -\ln(\sqrt{2\pi}\sigma) - \frac{(x - \mu)^2}{2\sigma^2}$$

可以看到，取对数以后，后验概率中的每一项都是对数高斯分布；而且方差都是与预设常数，因此只有 **第二项** 与待优化的 U, V 有关。

最大化上述对数后验概率，等价于 **最小化** 下面的能量函数：

$$E(U, V) = \frac{\|R - U^T V\|_2^2}{2\sigma^2} + \frac{(U^T U)}{2\sigma_U^2} + \frac{(V^T V)}{2\sigma_V^2}$$

通过参数替换约掉一个变量：

$$E(U, V) = \frac{1}{2} \|R - U^T V\|_2 + \frac{\lambda_U^2}{2} (U^T U) + \frac{\lambda_V^2}{2} (V^T V)$$

如果系统先验方差 σ_U 和 σ_V **无穷大**，也即无法对系统参数做人为约束，则优化目标就只剩下第一项，此时问题退化为一个 **SVD分解问题**。

而矩阵的概率密度应该等于其entries的概率密度的乘积。**取对数**以后，即等于其元素

$$E(U, V) = \frac{1}{2} \sum_{ij} I_{ij} (R_{ij} - U_i^T V_j)^2 + \frac{\lambda_U^2}{2} \sum_i (U_i^T U_i) + \frac{\lambda_V^2}{2} \sum_j (V_j^T V_j)$$

上式中， R_{ij} 是标量， U_i 和 V_j 是维度为 D 的向量。最后两项相当于约束了内部特征矩阵 U ， V 的范数。而标记 I_{ij} 是用户 i 是否对电影 j 进行评分的 **示性变量**。上式就是从 **向量**的这一“维度”表述我们的 **目标函数**。

如果我们将和项写得清晰一点儿，上面的表达式应该是这样的：

$$E(U, V) = \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^M I_{ij} (R_{ij} - U_i^T V_j)^2 + \frac{\lambda_U}{2} \sum_{i=1}^N \|U_i\|_{Fro}^2 + \frac{\lambda_V}{2} \sum_{j=1}^M \|V_j\|_{Fro}^2$$

最后，为了限制评分的范围，我们对高斯函数的均值施加一个logistic函数

$g(x) = \frac{1}{1+\exp(-x)}$ ，其值域为 $(0, 1)$ 。所以，最终的目标函数（能量函数）就是：

$$E(U, V) = \frac{1}{2} \sum_{ij} I_{ij} (R_{ij} - g(U_i^T V_j))^2 + \frac{\lambda_U}{2} \sum_i U_i^T U_i + \frac{\lambda_V}{2} \sum_j V_j^T V_j$$

至此，我们可以使用 **梯度下降法**，通过求 $\partial E / \partial U_{ik}$ ， $\partial E / \partial V_{jk}$ ，求取 U_i ， V_j 中的每一个entry：

$$U_i = U_i + \alpha \frac{\partial E}{\partial U_i}$$

$$V_j = V_j + \alpha \frac{\partial E}{\partial V_j}$$

我们需要估计的参数数量为 $M \times D + N \times D$ 。

(3) 控制模型复杂度

最简单的控制复杂度的方法就是调整特征的维度，即 D 。但是 D 越大，模型越精确的同时，也越容易造成过拟合；并且， D 应当与用户评过分的电影的数量大致相等。但现实中的问题是：数据往往是**不均衡**的——不同用户评过分的电影差异较大。例如，一个电影爱好者可能动辄评过数十上百部电影的分；而一个普通的用户可能只给寥寥几部电影评过分。

一个比较好的方法是选择一个中等尺度的 D ，然后令 $\lambda_U = \frac{\sigma_u^2}{\sigma_v^2}$ ， $\lambda_V = \frac{\sigma_v^2}{\sigma_u^2}$ 。

二、悬而未决的问题

1. **【logistic function】** 限制评分范围的**logistic函数**在很多实现中都没有进行还原（不添加这个函数确实使得求偏导得到的表达式更加简单清晰了.....），其实我还没有进行很仔细的数学推导，去考虑为什么要通过这个函数限制评分范围？
2. **【低秩分解模型的先验分布】** U 和 V 的先验分布为零均值的高斯分布，这样设置有什么依据吗？

文中有给出对应的[link 2](#) [3](#)，可进行延伸阅读。

三、代码实现

这部分详见本次作业上传的PMF.py及代码中的注释说明。下面仅给出参数设置以及说明所使用的数据集。

（一）所使用的数据集

参见第四大点[对应小节](#)的说明。

（二）参数设置

参数名称	值
D (number of latent factors)	10
学习率	0.005
U的正则化系数： λ_u	0.01
V的正则化系数： λ_v	0.001
训练最大迭代次数	50

四、结果分析

（一）原文的实验方法以及结果分析

1. 数据集

所采用的数据集的规模如下表所示：

数据集	评分数据	用户数量	电影数量
Netflix Train	100,480K	480K	17K
Netflix Valid	1,408K	/	/
Netflix Test	2,817K	/	/

2. 训练方法

为了提高训练速度，使用了 **mini-batch** 方法：每100K个评分数据就更新一次待求得的参数。

learning rate设置为0.005；momentum设置为0.9⁴。

Momentum的引入可以避免GD过程中得到的轨迹曲线过于曲折（不光滑）。

3. 结果分析

- 文中 D 设置为10与30。
- 参与对比的模型有SVD model、两个固定先验的PMF model（正则化项不同，后面表格会提到）、两个具有自适应先验的PMF model。

其中SVD model的训练目标是“minimize the sum-squared distance only to **the observed entries** of the target matrix”。并且其中的特征向量 **没有进行正则化**。

- 关于训练过程中RMSE这一度量上的比较，参照下面的图表：

由于没有复现adaptive-prior PMF，所以它们没有列在下面的表格中。

模型	U上的正则化项系数	V上的正则化项系数	RMSE
SVD	/	/	overfits
PMF1	0.01	0.001	slightly greater than 0.94
PMF2	0.001	0.0001	slightly smaller than 0.93

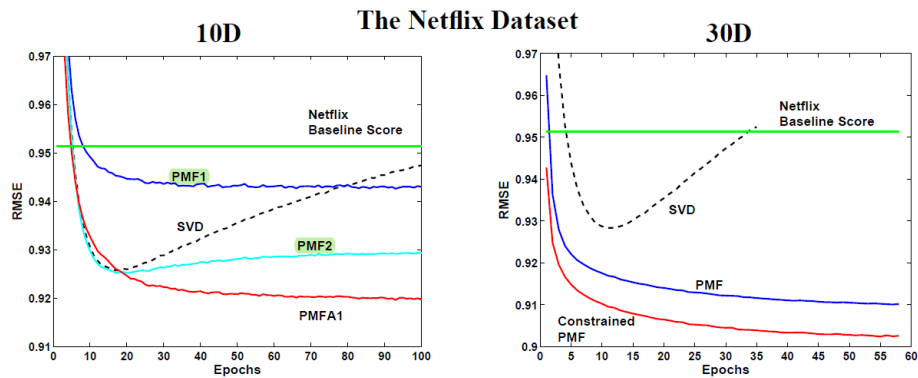


Figure 2: Left panel: Performance of SVD, PMF and PMF with adaptive priors, using 10D feature vectors, on the full Netflix validation data. Right panel: Performance of SVD, Probabilistic Matrix Factorization (PMF) and constrained PMF, using 30D feature vectors, on the validation data. The y-axis displays RMSE (root mean squared error), and the x-axis shows the number of epochs, or passes, through the entire training dataset.

（二）复现代码的结果分析

1. 数据集

>

数据集使用movielens的1m数据集⁵。取其中的ratings.dat文件作为训练数据，关于该数据集的描述如下表所示：

数据文件	数据规模	用户数量	电影数量	数据在文件中的格式
ratings.dat	1,000,209	6,040	3,952	UserID::MovieID::Rating::Timestamp

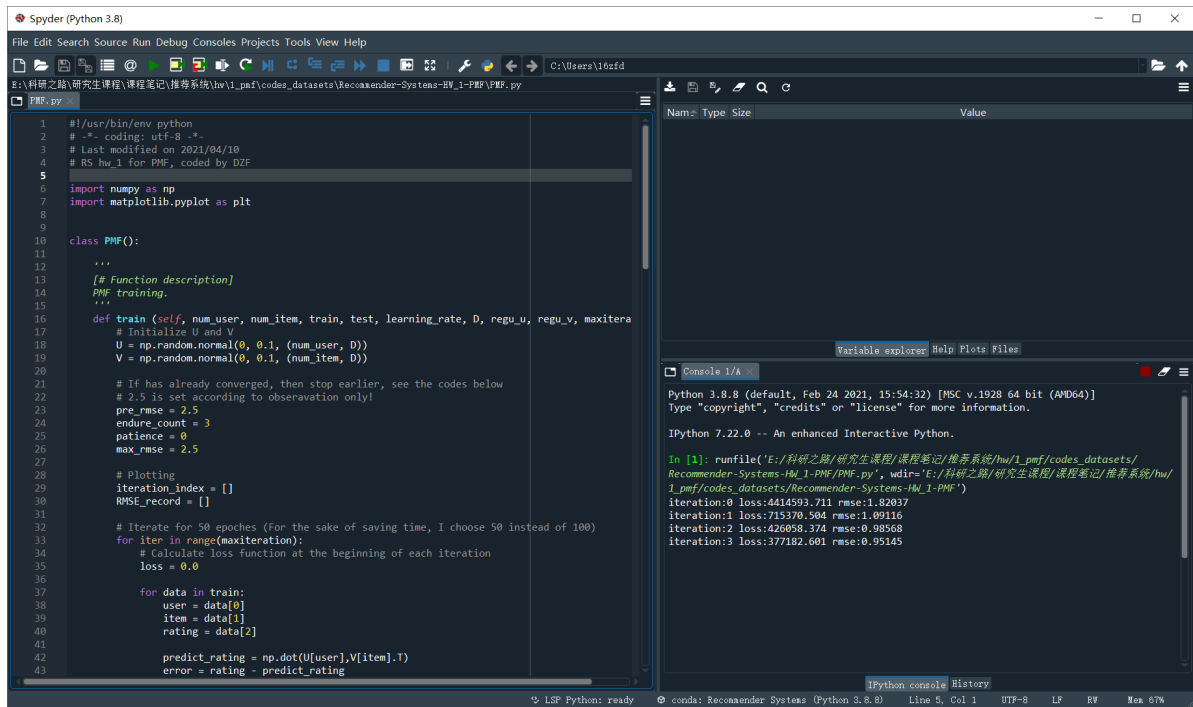
- 评分是5-star scale。
- 每个用户至少都对20部电影作出了评价。
- 训练过程中用到的只有UserID，MovieID以及Rating。

2. 结果展示

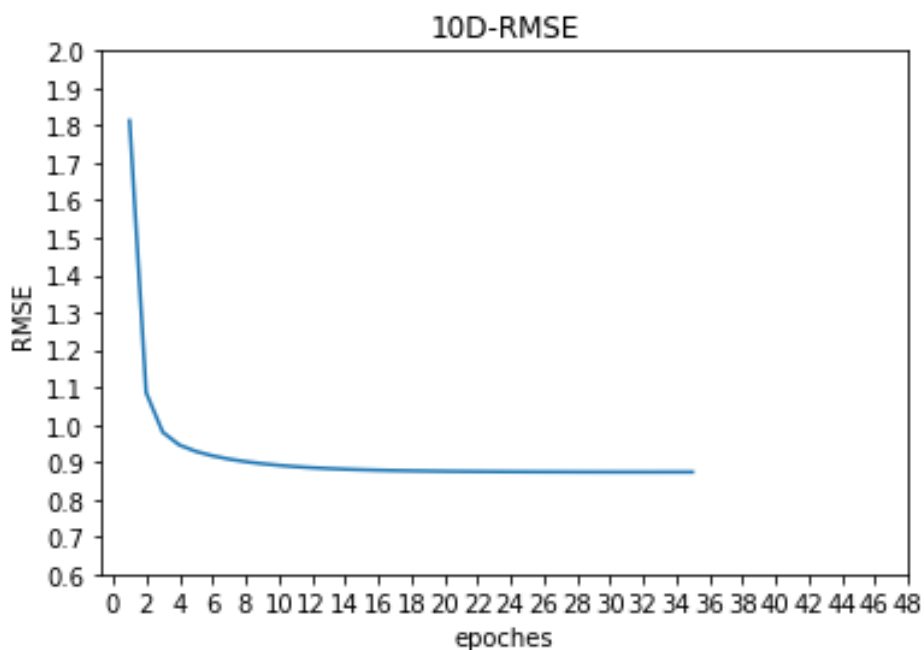
(1) 运行截图

```
iteration:16 loss:290851.834 rmse:0.87698
iteration:17 loss:288361.369 rmse:0.87623
iteration:18 loss:286101.719 rmse:0.87562
iteration:19 loss:284046.681 rmse:0.87512
iteration:20 loss:282173.138 rmse:0.87470
iteration:21 loss:280460.755 rmse:0.87436
iteration:22 loss:278891.707 rmse:0.87407
iteration:23 loss:277450.408 rmse:0.87383
iteration:24 loss:276123.243 rmse:0.87363
iteration:25 loss:274898.325 rmse:0.87346
iteration:26 loss:273765.252 rmse:0.87332
iteration:27 loss:272714.910 rmse:0.87322
iteration:28 loss:271739.286 rmse:0.87313
iteration:29 loss:270831.316 rmse:0.87307
iteration:30 loss:269984.753 rmse:0.87303
iteration:31 loss:269194.055 rmse:0.87302
iteration:32 loss:268454.296 rmse:0.87301
iteration:33 loss:267761.084 rmse:0.87303
iteration:34 loss:267110.491 rmse:0.87306
iteration:35 loss:266499.002 rmse:0.87311
```

图示训练到第35次迭代就根据收敛test提前终止了，可见最终的RMSE大致收敛到0.873附近。



这是运行界面。



这是RMSE随着训练轮数的变化曲线图。

(2) 结果分析

对于RMSE低于原文PMF1的结果有以下几种猜想：

1. 原文的数据规模更大（较movielens 1m高出2个量级），有可能是数据不平衡问题更加突出，又或者是本身PMF的RMSE是关于数据规模（大到一定程度）的正相关函数？

2. 原文所使用的mini-batch训练方法会不会是一种精度-效率的trade-off方法？其背后的数学原理是什么？

受限于当前的知识贫瘠程度，还未能寻得其中原因，但值得在后续学习过程中进行探究。

五、总结与收获

1. 快速复现论文工作的能力是研究生阶段需要培养的重要能力！
2. 对movielens数据集有了一定的了解（通过可视化、阅读相关描述文档等）。
3. 更加深入地思考关于机器学习的收敛定理以及GD相关理论等。

参考资料

-
1. Probabilistic Matrix Factorization, Ruslan Salakhutdinov, Andriy Mnih, 2007: http://www.utstat.toronto.edu/~rsalakhu/papers/nips_draft2.pdf ↩
 2. Delbert Dueck and Brendan Frey. Probabilistic sparse matrix factorization. Technical Report PSI TR 2004-023, Dept. of Computer Science, University of Toronto, 2004. ↩
 3. Michael E. Tipping and Christopher M. Bishop. Probabilistic principal component analysis. Technical Report NCRG/97/010, Neural Computing Research Group, Aston University, September 1997. ↩
 4. Momentum and Learning Rate Adaptation: <http://www.willamette.edu/~gorr/classes/cs449/momrate.html> ↩

5. MovieLens 1m: <http://files.grouplens.org/datasets/movielens/ml-1m-README.txt> ↵