

Parallel Programming

用于电影推荐系统的协调过滤推荐算法

GuangCheng-Li	Hong-Liang	Rui-Jiang
李光程	梁轰	蒋蕊
16098537-II20-0016	1609853J-II20-0027	1609853J-II20-0030
sky9475@126.com	coolboom@foxmail.com	1458952792@qq.com

1. 介绍

本文的项目是计算机体系结构的并行计算项目—电影评分预测。

协同过滤算法 (collaborative filtering) 因其仅依赖于用户过去的行为且准确性高的特性，故而在图书推荐、电影推荐等场景被广泛应用。

在本项目中，我们主要使用 latent factor models 来实现电影评分的预测，该算法基于 MatLab 实现。

2. 算法

假设有 M 部电影和 N 个用户，令 $Y \in R^{M \times N}$ 代表评分矩阵。其中，当 $Y_{ij} \neq 0$ 表示电影 i 被用户 j 评分为 Y_{ij} ， $Y_{ij} = 0$ 代表用户 j 没有对电影 j 进行评分。令 $R \in \{0,1\}^{M \times N}$ 表示为与之对应的指示矩阵，即将 Y 中不为 0 项在此矩阵中映射为 1，反之对于 Y 中的 0 项则映射为 0。

矩阵分解模型把用户和电影映射到一个维度为 p 的隐藏因子空间中，如此处理后，用户-电影的内在关联关系就可以被投射到这个隐藏因子空间中。

设两个矩阵 $X \in R^{M \times p}$ 和 $\Theta \in R^{N \times p}$ ，其中， M 为电影的数量， N 为用户的数量，则这两个矩阵即将用户和电影映射到一个维度为 p 的隐藏因子空间中。且存在 $P = X\Theta^T$ ， $P \approx Y$ 。由此不难推断， $X_i\Theta_j^T$ 的点乘即为用户 j 对电影 i 的预测评分。

我们根据[1]中的工作使用 regularized squared error 建立目标函数(1)，并通过梯度下降法寻找以最小化之。

$$\min_{X, \Theta} \frac{1}{2} R \cdot (Y - X\Theta^T)_{\text{F}}^2 + \lambda (\|X\|_{\text{F}}^2 + \|\Theta\|_{\text{F}}^2) \quad (1)$$

其中 $\|\cdot\|_{\text{F}}$ 是 Frobenius 范数，操作符 \cdot 为点乘，且 λ 是正则化因子。

对公式(1)求 X 和 Θ 的偏导，我们可以获得梯度函数(2a)与(2B)。

$$\Delta X = R \bullet (X\Theta^T - Y)\Theta + \lambda X \quad (2a)$$

$$\Delta \Theta = (R \bullet (X\Theta^T - Y))^T X + \lambda \Theta \quad (2b)$$

则，迭代公式可写作(3a)与(3b)。

$$X^{t+1} = X^t - \mu \Delta X^t, \quad (3a)$$

$$\Theta^{t+1} = \Theta^t - \mu \Delta \Theta^t, \quad (3b)$$

其中 μ 是学习速率，可以设置为一个小常数（如 0.002）。通过学习隐藏因子空间 $\{X, \Theta\}$ 的映射关系后，就可以通过 $P = X\Theta^T$ 来填充矩阵 Y 中缺失的评分。

矩阵分解的梯度下降算法总结如下：

算法 1 矩阵分解梯度下降算法

输入：评分矩阵 Y 和 R ，随机初始化隐藏因子矩阵 X^0, Θ^0 ，学习速率 $\mu=0.001$ ， $\text{MaxIters}=500$ 。

1： 设置一个修正系数 λ 。

2： for $t=0 : \text{MaxIters}$

3： $X^{t+1} = X^t - \mu(R \bullet (X^t \Theta^{t,T} - Y)\Theta^t + \lambda X^t)$

4： $\Theta^{t+1} = \Theta^t - \mu((R \bullet (X^t \Theta^{t,T} - Y))^T X^t + \lambda \Theta^t)$

5： if $\frac{\|X^{t+1} - X^t\|_F^2 + \|\Theta^{t+1} - \Theta^t\|_F^2}{\|X^t\|_F^2 + \|\Theta^t\|_F^2} < \varepsilon$

6： break;

7： end

8： end

9： return $P = X^{t+1} \Theta^{t+1,T}$

本算法的并行化直接使用了 Matlab 2016a 提供的 Parallel Computing Toolbox 中的 GPU 优化方法。仅需对源代码做部分修改，即可将算法于支持 CUDA 的 GPU 上运行。

3. 实验

我们希望通过实验来检测算法的预测准确性与并行加速比。其中，预测准确性我们采用 RMSE 来衡量。

实验所用 PC 硬件配置如下：

MainBoard: Gigabyte G1.Sniper B7 (Intel Sunrise Point B150)

CPU: Intel Core i5-6500, 3300 MHz

RAM: 16 GB

GPU: MSI GTX 1060 Gaming X 6GB @1784 MHz

实验所用软体环境如下：

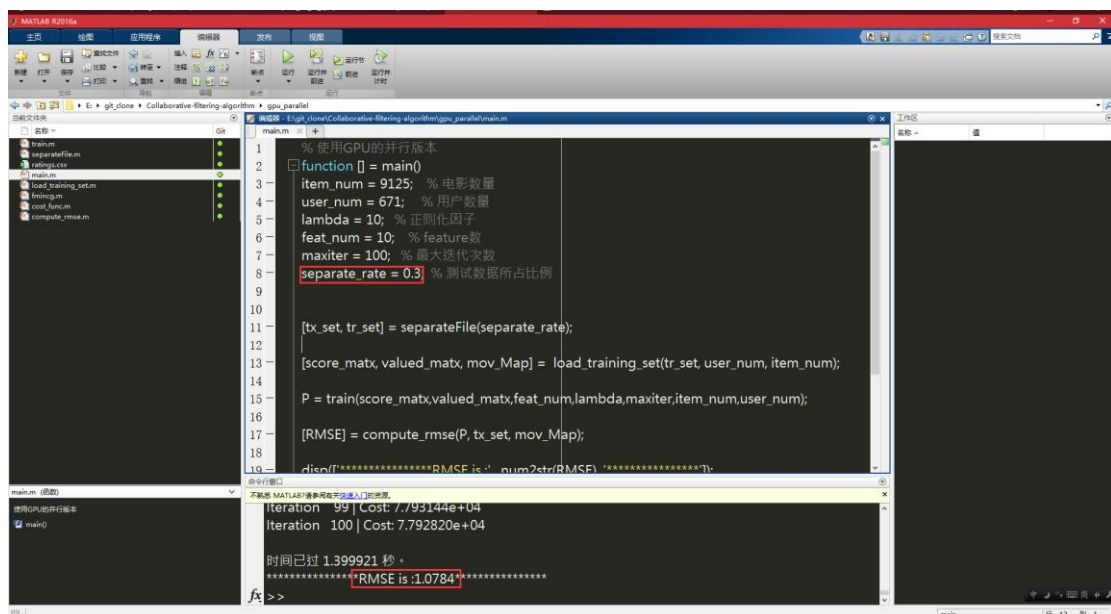
System: Microsoft Windows 10 Pro 10.0.15063.296

Matlab: 2016a

实验采用对 9125 部电影 100004 条评分的数据集，其由 671 名用户创建。我们将数据集划分为 2 份，其中 70% 为训练数据，30%为测试数据。

实验中，我们首先设置正则化因子为 10、feature 数量为 10、最大迭代次数为 100，通过计算 RMSE 来检查算法的预测准确性，运行结果如图 1 所示。实验所的 RMSE 为 1.0784，该值在可接受范围内，说明我们所实现之算法对用户偏好的预测较为准确。

接下来，我们在保持其他参数不变的情况下，更改最大迭代次数为 300，并分别运行串行算法和并行算法，通过统计训练模型所用时间来比较两种算法的性能，实验结果如图 2 与图 3 所示。可见，串行算法迭代 300 次耗时 78.6s，并行算法耗时 3.83s，由此可计算并行加速比 $R = T_0 / T_P = 78.6 / 3.83 = 20.522$ 。这反映出我们的并行算法优化较好。



The screenshot shows the MATLAB R2016a environment. The main window displays a script named 'main.m' with the following code:

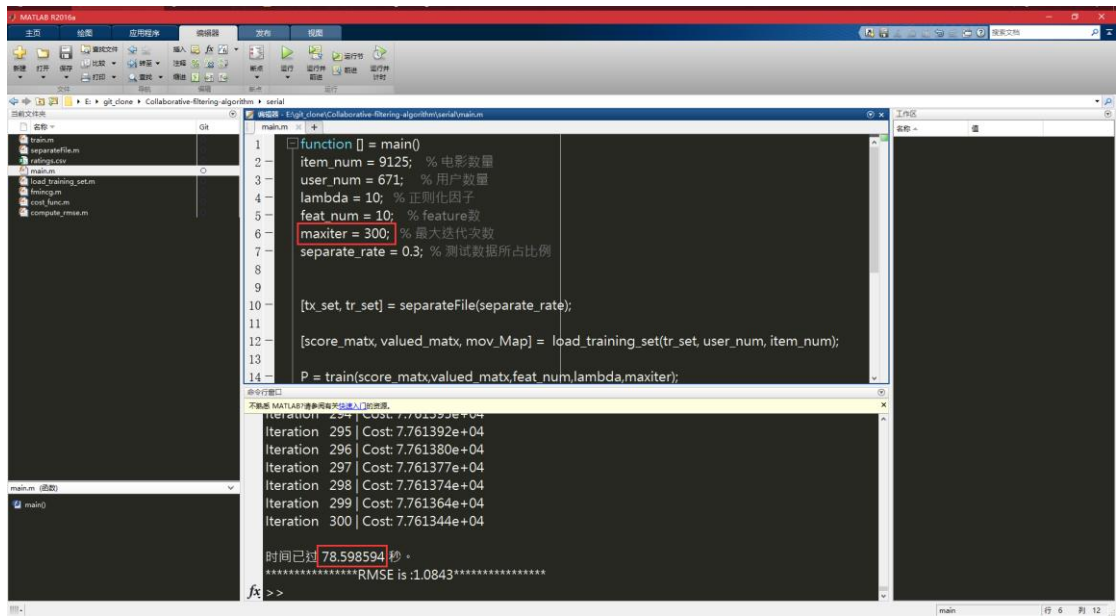
```
1 % 使用GPU的并行版本
2 function [] = main()
3 item_num = 9125; % 电影数量
4 user_num = 671; % 用户数量
5 lambda = 10; % 正则化因子
6 feat_num = 10; % feature数
7 maxiter = 100; % 最大迭代次数
8 separate_rate = 0.3; % 测试数据所占比例
9
10
11 [tx_set, tr_set] = separateFile(separate_rate);
12 [score_matx, valued_matx, mov_Map] = load_training_set(tr_set, user_num, item_num);
13
14 P = train(score_matx, valued_matx, feat_num, lambda, maxiter, item_num, user_num);
15
16 [RMSE] = compute_rmse(P, tx_set, mov_Map);
17
18 disp(['*****RMSE is: ' num2str(RMSE) '*****']);
19
```

The command window at the bottom shows the execution results:

```
iteration 99 | Cost: 7.793144e+04
iteration 100 | Cost: 7.792820e+04

时间已过 1.399921 秒。
*****RMSE is: 1.0784*****
>>
```

图 1 RMSE 检测



The screenshot shows the MATLAB R2019a interface with a script named 'main.m' being executed in serial mode. The script defines parameters for a collaborative filtering algorithm: item_num = 9125, user_num = 671, lambda = 10, feat_num = 10, maxiter = 300, and separate_rate = 0.3. The execution progress is shown in the Command Window, displaying iterations from 295 to 300 with corresponding costs. The final output indicates that the execution time is 78.598594 seconds and the RMSE is 1.0843.

```
function [] = main()
    item_num = 9125; % 电影数量
    user_num = 671; % 用户数量
    lambda = 10; % 正则化因子
    feat_num = 10; % feature数
    maxiter = 300; % 最大迭代次数
    separate_rate = 0.3; % 测试数据所占比例

    [tx_set, tr_set] = separateFile(separate_rate);

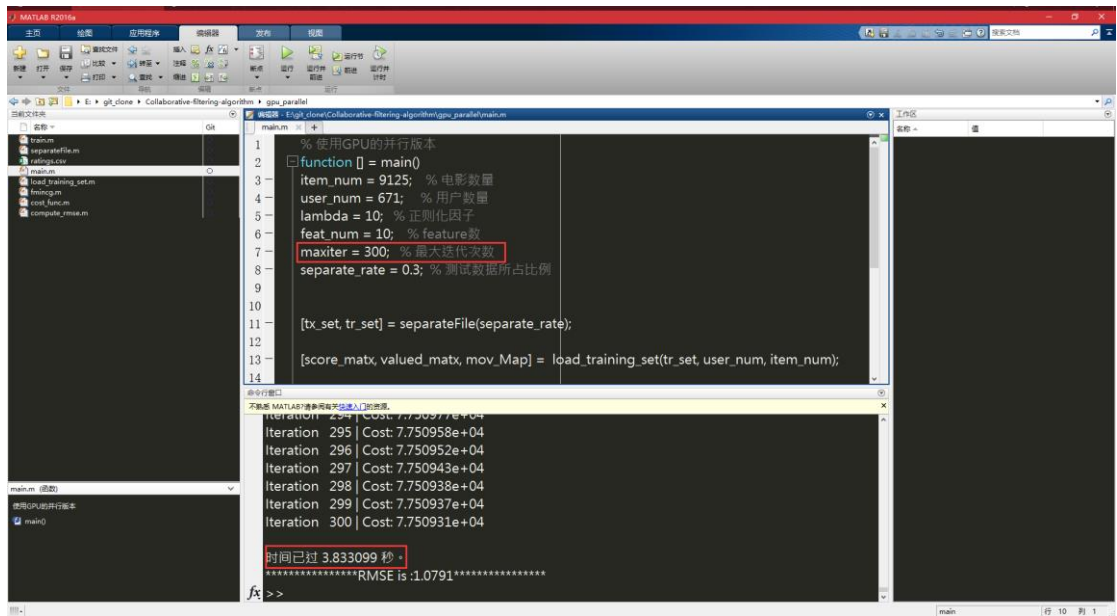
    [score_matx, valued_matx, mov_Map] = load_training_set(tr_set, user_num, item_num);

    P = train(score_matx, valued_matx, feat_num, lambda, maxiter);

    iteration 295 | Cost: 7.761392e+04
    iteration 296 | Cost: 7.761380e+04
    iteration 297 | Cost: 7.761377e+04
    iteration 298 | Cost: 7.761374e+04
    iteration 299 | Cost: 7.761364e+04
    iteration 300 | Cost: 7.761344e+04

    时间已过 78.598594 秒。
    *****RMSE is :1.0843*****
```

图 2 串行算法迭代 300 次耗时



The screenshot shows the MATLAB R2019a interface with the same script 'main.m' being executed in parallel mode using the 'gpu_parallel' function. The script parameters are identical to the serial version. The execution progress is shown in the Command Window, displaying iterations from 295 to 300 with corresponding costs. The final output indicates that the execution time is 3.833099 seconds and the RMSE is 1.0791.

```
% 使用GPU的并行版本
function [] = main()
    item_num = 9125; % 电影数量
    user_num = 671; % 用户数量
    lambda = 10; % 正则化因子
    feat_num = 10; % feature数
    maxiter = 300; % 最大迭代次数
    separate_rate = 0.3; % 测试数据所占比例

    [tx_set, tr_set] = separateFile(separate_rate);

    [score_matx, valued_matx, mov_Map] = load_training_set(tr_set, user_num, item_num);

    iteration 295 | Cost: 7.750977e+04
    iteration 296 | Cost: 7.750958e+04
    iteration 297 | Cost: 7.750952e+04
    iteration 298 | Cost: 7.750943e+04
    iteration 299 | Cost: 7.750938e+04
    iteration 300 | Cost: 7.750931e+04

    时间已过 3.833099 秒。
    *****RMSE is :1.0791*****
```

图 3 并行算法迭代 300 次耗时

Reference

- [1] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," *KDD Cup Work.*, pp. 2–5, 2007.