

实验（三）—— 评分预测

童云林 PB17071444 杨朝辉 PB17071433

Date: 2020-02-01

Platform: Python 3.8.5, PyTorch 1.6.0

实验算法

基于模型的评分预测（SVD）

- 模型表示

利用矩阵分解方式（称为SVD）求解使得训练数据上评分的 RMSE 最小的“用户因子”和“商品因子”矩阵等隐变量，同时考虑到商品固有的“品质”和用户固有的“打分偏好”以及网站数据固有的“平均水平”，采用加入偏置项的矩阵分解表示，

$$r_{mn} = \sum_f p_{mf} q_{nf} + b_m + c_n + \mu$$

其中 $P \in R^{M \times F}$, $Q \in R^{N \times F}$ 分别为用户因子和商品因子“隐属性矩阵”， $b \in R^M$, $c \in R^N$ 为用户和商品“偏置项”， μ 为该网站（豆瓣）的均分值（整个训练集中所有评分的平均），这样也能够避免“参数矩阵和参数向量元素都是非负”假定的限制。

- 目标函数

为避免所训练出的参数矩阵元素值不均一而造成过拟合，采用正则化平方损失函数，

$$L = \sum_{mn \in \text{TrainSet}} (r_{mn} - \hat{r}_{mn})^2 + \lambda (\sum_m \|p_m\|^2 + \sum_n \|q_n\|^2)$$

由于训练数据集是一个稀疏矩阵，下标 m, n 均限于训练集中存在的评分值，正则化采用 L-2 范数的平方和。

- 优化方法

实际计算中由于矩阵维度较大（ $M = 2173$, $N = 58431$ ），采用批量损失和批量梯度下降的方法做优化，每个用户的可用数据作为一个 batch， $N(m)$ 表示 m 用户评分数据的个数，损失函数变为

$$L = \frac{\sum_{m, N(m)} (r_m - \hat{r}_m)^2}{N(m)} + \lambda \frac{(\sum_m \|p_m\|^2 + \sum_n \|q_n\|^2)}{N(m)}$$

- 计算

参数初始化时矩阵 P 、 Q 元素按照均匀分布或正态分布随机初始化，并且根据经验，元素数值大小正比于隐属性个数 F 的平方根的倒数； μ 即为整个训练集中的均分，是一个标量值； b 、 c 分别按照正比于用户均分、商品均分的大小作初始化。

实际上，原始数据中评分值所属集合为 $\{0, 1, 2, 3, 4, 5\}$ 整数集合，而计算过程损失函数的计算是基于 $5 * \text{Sigmoid}$ 函数归一化的矩阵计算值和标签值，而 RMSE 基于进一步取整的预测值与标签值。

采用 PyTorch 的自动微分机制，避免了手推梯度即反向传播和更新参数的繁琐过程，具体如下

为避免过拟合和稳定的训练出的参数结果，根据经验，默认选取 $F = 10$ 、 $\lambda = 0.001$ ， $\text{lr} = 0.005$ ；训练集和验证集合按照 7/3 比例划分。

基于内存的评分预测（协同过滤）

按照课件中的算法实现。

结果与优化

基于模型的预测

根据多次训练过程，最终选定使得结果比较好的对 P 、 Q 矩阵按照正态分布初始化的方式；同时使用 PyTorch 内置 Adam 优化器代替原本简单的梯度下降方式，loss 下降更快；为避免过拟合，在每个训练 epoch 都对数据作随机的 7/3 划分，相当于随机批量梯度下降的效果。

训练集和验证集上的效果如下。经历大约 25 个 epoch，验证集上 RMSE 下降至 1.07 左右。

```
epoch: 3, loss on training set: 2.1006, RMSE on validation set: 1.4004
epoch: 4, batch [400/2173], train loss: 1.8589, valid rmse: 1.3113
epoch: 4, batch [800/2173], train loss: 1.8929, valid rmse: 1.3199
epoch: 4, batch [1200/2173], train loss: 1.8548, valid rmse: 1.3051
epoch: 4, batch [1600/2173], train loss: 1.8806, valid rmse: 1.3105
epoch: 4, batch [2000/2173], train loss: 1.8494, valid rmse: 1.2993
epoch: 4, loss on training set: 1.8403, RMSE on validation set: 1.2959
```

得到的矩阵和向量元素值如下，可见其绝对值较小，直觉来看也比较合理。

```

In [14]: print(P)
[[ 1.55602944 -0.63107067  1.50508165 ... -0.22547099  0.78490579
  -0.21556023]
 [-0.11481296  0.11499012 -0.1529175 ... -0.12873651 -0.1928089
  0.04664184]
 [ 0.05000493  0.37948179 -0.4440113 ... -0.07269394 -0.86615193
  -0.2642256 ]
 ...
 [-0.07167871  0.29142129  0.08265466 ...  0.00940289 -0.28386712
  0.27436817]
 [-0.1049418  0.3706181  -0.34901616 ... -0.29970464 -0.35366398
  -0.04379062]
 [-0.22782558  0.25414202  0.12233388 ...  0.1286543  -0.21150637
  0.15176831]]

In [15]: print(Q)
[[ 1.24081329e-04  4.61098389e-05 -8.54408427e-07 ...  2.74799077e-05
  -3.74062296e-07 -1.16948414e-07]
 [ 1.60503387e-02 -2.52510190e-01  6.50200963e-01 ...  5.29703557e-01
  -5.55160604e-02 -3.98403287e-01]
 [ 1.22898266e-01 -1.27829626e-01  3.22199240e-02 ... -1.35921960e-04
  2.77199864e-01 -8.46355967e-03]
 ...
 [ 1.15824668e-02  1.15211960e-02  3.65545333e-04 ... -4.16053347e-02
  -1.80893168e-02 -6.95183128e-03]
 [ 2.68142164e-01 -5.87873042e-01  7.74873734e-01 ...  5.12135684e-01
  3.29910845e-01  4.34324235e-01]
 [-1.35867611e-01 -1.48506090e-01 -9.00861174e-02 ...  1.50001228e-01
  1.48640394e-01  1.45610169e-01]]

In [16]: print(b)
[-1.08144784  1.71043599  2.05152011 ...  1.06713092  1.00226545
  0.74286109]

In [17]: print(c)
[ 3.94739246 -3.5012486  -3.5242424 ... -2.66122127 -2.48614573
  -2.83972859]

```

在线网站上的测试结果并不太好，即便调节了 F、lam、epoch 等参数，RMSE 也都在 2.2 的水平，表面准确率相当不好。

Filename	RMSE
杨朝辉-PB17071433-7.txt	2.241770511325892

基于内存的预测

不愧为基于“内存”的预测方法，即便使用了 python 中 numba 即时编译方式，PC 上仍然跑不出来代码，故而此部分只有程序没有结果。

其他结果

以下为某几个用户评分的均分随时间的变化，但变化并不明显，故而此次实验中未采用考虑了时间属性的“三维”的矩阵分解。

