lab4 推荐系统

小组成员

PB18111791 雷雨轩

PB18111793 裴启智

实验目的

- 基于豆瓣电影的评分记录，根据训练数据中的用户评分信息，判断用户偏好，并为测试数据中user-item 对进行评分。
- 结合社交关系信息，建立社交推荐模型，对评分结果进行修正

基于Surprise库的模型

- 我们基于Python中常见的推荐系统库Surprise开展实验。
- Surprise是scikit系列中的一个推荐系统库，其中涵盖了Baseline、基于邻域的协同过滤、矩阵分解（SVD，SVD++，PMF，NMF）、SlopeOne 协同过滤等常用推荐算法。
- Surprise中的评价指标包含**均方根误差（RMSE）、平均绝对误差（MAE）以及一致对的分数（FCP）**。

数据预处理

- 根据surprise模型提供的接口以及数据集的格式，考虑先读入训练集数据然后整合成字典的形式，再转换为pandas dataframe，最后利用Surprise接口`Dataset.load_from_df`将其读入成特定的数据格式。
- ```
  ratings_dict = {'userID': [],
                  'itemID': [],
                  'rating': [],
                  'timestamp': [],
                  'tags':[]}
  with open("./data/training.dat","r",encoding='utf-8') as f:
      for line in f.readlines():
          line_data=line.strip().split(",")
          length = len(line_data)
          ratings_dict['userID'].append(line_data[0])
          ratings_dict['itemID'].append(line_data[1])
          ratings_dict['rating'].append(int(line_data[2]))
          ratings_dict['timestamp'].append(line_data[3])
          tags=[]
          for i in range(4,length):
              tags.append(line_data[i])
          ratings_dict['tags'].append(tags)

  df_train = pd.DataFrame(ratings_dict)
  reader = Reader(rating_scale=(0, 5))
  data_train = Dataset.load_from_df(df_train[['userID', 'itemID', 'rating']],
  reader)
  ```
- 需要训练时，考虑把整个训练集用来训练，使用`data_train.build_full_trainset()`以及 fit方法来训练
  ```

- 预测时，按行读入测试集文件，根据每一行的user-id和item-id，使用predict方法即可得到预测值（float，还需手动转化为int）

SVD思路与原理

在进行矩阵分解时，很多矩阵都是非对称的，矩阵A不是方阵，即维度为m*n，我们可以将它转化为对称的方阵，因为A^TA**和**AA^T**都是对称的方阵**，有

$$AA^T = P\Lambda_1 P^T$$
$$A^TA = Q\Lambda_2 Q^T$$

其中Λ_1和Λ_2为对角矩阵，具有相同非零对角元素σ1,σ2,...,σk，k<=min(m,n)，此时，$\lambda_1 = \sqrt{\sigma_1}, \lambda_2 = \sqrt{\sigma_2}, \ldots, \lambda_k = \sqrt{\sigma_k}$为矩阵A的**奇异值。**

我们可以得到为**奇异值分解**:

$$A = P\Lambda Q^T$$

其中P为左奇异矩阵，m.m维；Q为右奇异矩阵，n.n维；Λ对角线上的非零元素为奇异值λ1,λ2,...,λk。

奇异值分解还可表示为

$$A = \lambda_1 p_1 q_1^T + \lambda_2 p_2 q_2^T + \cdots + \lambda_k p_k q_k^T$$

其中p为左奇异矩阵的特征向量；q为右奇异矩阵的特征向量。

FunkSVD思路与原理

传统SVD解决评分预测问题时，需要补全矩阵再预测，产生的噪音大。事实上，矩阵分解之后的还原，只需要关注与原来矩阵中有值的位置进行对比即可，不需要对所有元素进行对比。FunkSVD算法的思路是：

- 避开稀疏问题，而且只用两个矩阵进行相乘（设置k值，来对矩阵近似求解）：

$$M_{m\times n} \approx P^T_{m\times k} Q_{k\times n}$$

- 损失函数=P和Q矩阵乘积得到的评分，与实际用户评分之差：

$$\sum_{i,j} \left(m_{ij} - q_j^T p_i\right)^2$$

为了防止过拟合，增加正则项：

$$\arg\min_{p_i q_j} \sum_{(i,j)\in K} \left(m_{ij} - q_j^T p_i\right)^2 + \lambda\left(\|p_i\|_2^2 + \|q_j\|_2^2\right)$$

- 让损失函数最小化 => 最优化问题：梯度下降法

BiasSVD思路与原理

BiasSVD算法是在FunkSVD算法的基础上考虑了用户和商品的偏好。用户有自己的偏好(Bias)，比如乐观的用户打分偏高；商品也有自己的偏好(Bias)，比如质量好的商品，打分偏高；BiasSVD算法将与个性化无关的部分，设置为偏好(Bias)部分。

其目标函数表示为：

$$\arg\min_{p_i q_j} \sum_{(i,j)\in K}\left(m_{ij}-\mu-b_i-b_j-q_j^T p_i\right)^2+\lambda\left(\|p_i\|_2^2+\|q_j\|_2^2+\|b_i\|_2^2+\|b_j\|_2^2\right)$$

其中μ为所有记录的整体平均数；bi为用户偏好（自身属性，与商品无关）；bj为商品偏好（自身属性，与用户无关）。

在迭代过程中，bi,bj的初始值可以设置为0向量，然后进行迭代：

$$b_i = b_i+\alpha\left(m_{ij}-\mu-b_i-b_j-q_j^T p_i-\lambda b_i\right)$$
$$b_j = b_j+\alpha\left(m_{ij}-\mu-b_i-b_j-q_j^T p_i-\lambda b_j\right)$$

最终得到P和Q。

SVD++思路与原理

SVD++算法是在BiasSVD算法的基础上考虑了用户的隐式反馈。**隐式反馈**是指没有具体的评分，但可能有点击，浏览等行为。

对于某一个用户ii，假设他的隐式反馈item集合为$I(i)$，用户i对商品j对应的隐式反馈修正值为：cij，用户i所有的隐式反馈修正值之和为：

$$\sum_{s\in N(i)} c_{sj}$$

目标函数表示为：

$$\arg\min_{p_i q_j} \sum_{(i,j)\in K}\left(m_{ij}-\mu-b_i-b_j-q_j^T p_i-q_j^T|I(i)|^{-\frac{1}{2}}\sum_{s\in I(i)} y_s\right)^2+\lambda\left(\|p_i\|_2^2+\|q_j\|_2^2+\|b_i\|_2^2+\|b_j\|_2^2+\sum_{s\in I(i)}\|y_s\|^2\right)$$

最小化目标函数，求解最优化问题，最终得到P和Q。

结果

使用基础的SVD算法,得到结果

裴启智_PB18111793_2.txt	1.5515106389941193

使用SVD++算法,得到结果

裴启智_PB18111793_3.txt	1.5744613575608222

模型分析

- 实际运行中SVD++算法的耗时约是SVD的近百倍
- 但SVD++的效果并没有SVD好，这和我们的预期有些出入。可能是数据量或者数据之间的关联性不足引起的

BaselineOnly

思路与原理

Baseline算法（基于统计的基准预测线打分）假设训练数据为： <user, item, rating>三元组， 其中user为用户id， item为物品id，rating为user对item的投票分数， 其中用户u对物品i的真实投票分数我们记为r_{ui}，基线模型预估分数为b_{ui}，则可建模：

$$b_{ui} = \mu + b_u + b_i$$

其中μ为所有已知评分数据的均值，b_u为用户的打分相对于平均值的偏差（如果某用户比较苛刻，打分都相对偏低， 则b_u会为负值；相反，如果某用户经常对很多片都打正分， 则b_u为正值）； b_i为该item被打分时，相对于平均值的偏差，可反映电影受欢迎程度。 该模型虽然简单， 但其中其实已经包含了用户个性化和item的个性化信息， 而且特别简单。

基线模型中，μ可以通过统计得到，为了估计b_u和b_i，我们的目标函数可以写成：

$$\min_{b_*} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_1 (\sum_u b_u^2 + \sum_i b_i^2) .$$

使用ALS优化该目标函数步骤：

- Step1，固定b_u，优化b_i
- Step2，固定b_i，优化b_u
- 重复Step1和2，直到收敛

$$b_i = \frac{\sum_{u\in \mathrm{R}(i)} (r_{ui} - \mu)}{\lambda_2 + |\mathrm{R}(i)|} .$$

$$b_u = \frac{\sum_{i\in \mathrm{R}(u)} (r_{ui} - \mu - b_i)}{\lambda_3 + |\mathrm{R}(u)|} .$$

结果

裴启智_PB18111793_5.txt	1.4761186658367007

模型分析

- BaselineOnly的运行时间快且结果算是比较好的

CoClustering

思路与原理

基于协同聚类的协同过滤算法。首先为用户和项目分配了一些集群 C_u，C_i，以及一些共同团体C_{ui}

预测$\hat{r}_{ui}$为$\hat{r}_{ui} = \overline{C_{ui}} + (\mu_u - \overline{C_u}) + (\mu_i - \overline{C_i})$

其中:

$\overline{C_{ui}}$是用户u和item i所在的联合类中的所有成员的分数均值，类中的成员以(u,i)的形式存在

$\overline{C_u}$是用户u所在的用户类中的所有成员的分数均值

$\overline{C_i}$是item i所在的item 类中的所有成员的分数均值

μ_u是用户u历史上打过的所有的分值的均值

μ_i是item i得到的所有分数的均值

Surprise包在实现该算法时，逻辑如下：

- 初始化所有item和user所属的类
- 计算$\overline{C_{ui}}$$\overline{C_u}$和$\overline{C_i}$
- 更新每一个item和user所属的类
- 跳转到第二步循环进行

结果

裴启智_PB18111793_6.txt	1.5340670480672136

模型分析

- 该模型运行时间和BaselineOnly近似，但效果没有BaselineOnly好

最近邻算法+基于内存的协同过滤算法

注：以下提及的公式里变量含义参考https://surprise.readthedocs.io/en/stable/notation_standards.html

思路

- 以基于用户的协同过滤方法为例：有相似偏好的用户，过去与未来的评分行为相似。

  所以寻找最近邻用户（从用户评分的相似趋势，不是绝对分值，即用户对共同物品的评分趋势相似）

  得到用户的相似性后，把相似性作为加权（选取k个最近邻用户）来根据历史上其他用户对该物品的评分来预测物品的评分

相似度计算方式

- 由surprise库提供计算方式，`sim_options={name, user_based, min_support ,shrinkage}`
- 其中name有`cosine`，`msd`，`pearson`，`pearson_baseline`四种，本次实验用到了后两种
- Pearson similarity

$$pearson\_sim(u,v)=\frac{\sum_{i\in I_{uv}}(r_{ui}-\mu_u)(r_{vi}-\mu_v)}{\sqrt{\sum_{i\in I_{uv}}(r_{ui}-\mu_u)^2}\sqrt{\sum_{i\in I_{uv}}(r_{vi}-\mu_v)^2}}$$

- Pearson baseline similarity：
  - 使用baseline b_{ui}取代mean
  - 当评分矩阵十分稀疏时，考虑用`shrinkage`参数计算以减少过拟合
  - $pearson\_baseline\_shrink\_sim(u,v)=\frac{|I_{uv}|-1}{|I_{uv}|-1+shrinkage}pearson\_baseline\_sim(u,v)$

协同过滤算法

KNNWithMeans

- 考虑用户/物品评分偏好的协同过滤，即需要减去每个相似用户自己评分的均值后再来对预测用户产生影响。
- $$\hat{r}_{ui}=\mu_u+\frac{\sum_{v\in N^k_{i(u)}}sim(u,v)\times(r_{vi}-\mu_v)}{\sum_{v\in N^k_{i(u)}}sim(u,v)}$$

KNNWithZScore

- 考虑用户/物品评分偏好，并对用户/物品进行`z-score`归一化
- $$\hat{r}_{ui}=\mu_u+\sigma_u\frac{\sum_{v\in N^k_{i(u)}}sim(u,v)\times(r_{vi}-\mu_v)\div\sigma_v}{\sum_{v\in N^k_{i(u)}}sim(u,v)}$$

KNNBaseline

- 基于baseline的协同过滤

  要评估一个策略的好坏，就需要建立一个对比基线，以便后续观察算法效果的提升。此处我们可以简单地对推荐算法进行建模作为基线。假设我们的训练数据为： <user, item, rating>三元组， 其中用户u对物品i的真实投票分数我们记为r_{ui}，基线(baseline)模型预估分数为b_{ui}，则可建模如下：

  $$b_{ui}=\mu+b_u+b_i$$

  其中μ为所有已知投票数据中投票的均值，b_u为用户的打分相对于平均值的偏差（如果某用户比较苛刻，打分都相对偏低， 则b_u会为负值；相反，如果某用户经常对很多片都打正分， 则b_u为正值）； b_i为该item被打分时，相对于平均值得偏差，可反映电影受欢迎程度。b_{ui}则为基线模型对用户u给物品i打分的预估值。该模型虽然简单， 但其中其实已经包含了用户个性化和item的个性化信息。

- $$\hat{r}_{ui}=b_{ui}+\frac{\sum_{v\in N^k_{i(u)}}sim(u,v)\times(r_{vi}-b_{vi})}{\sum_{v\in N^k_{i(u)}}sim(u,v)}$$

结果

- 下图从上往下依次用的方法为
  - KNNBaseline + pearson_baseline + als
  - KNNWithZScore + pearson_baseline
  - KNNWithMeans + pearson
  - 并且均使用的是基于用户的协同过滤（尝试基于物品的协同过滤时，一方面因为item id高达58431，自己电脑没法存储 58431*58431的矩阵 ，而尝试在服务器上跑时，得到的结果反而没有基于用户的协同过滤效果好）
雷雨轩_PB18111791_4.txt	1.4851261270638125
雷雨轩_PB18111791_5.txt	1.5041610158952559
雷雨轩_PB18111791_6.txt	1.4896385734422959

模型分析

- 得到的三种结果在效果上并没有明显的区别，因为本质上相似度计算公式 以及 评分计算公式 的差异并不大，只是可能确实会因为体现用户偏好的不同归一化（mean，z-score, baseline）而导致对于本次实验数据集有不同的契合度。
- 在复杂度方面，三者的运行时间类似，计算复杂度为同一量级

集成学习

- 为了减少取整时四舍五入带来的舍入误差，得到更好的结果，我们对得到的浮点结果进行了一般化的集成，即先对所有的浮点结果求平均再进行集成，可以看到结果有了明显的提高

裴启智_PB18111793_12.txt	1.4564075992808276

社交推荐模型

- 阅读论文`SoRec: Social Recommendation Using Probabilistic Matrix Factorization`，在矩阵分解的基础上加入社会网络的邻接矩阵分解，以期达到利用用户社交网络信息的目的。
- 我们首先基于Surprise的各个结果，根据用户的社交关系，对最终得分进行一定的修正，设用户本身被预测的分值为i, 用户关注的其他用户被预测的分值为$j_1, j_2 \dots j_n$，则用户最终得分为$0.8 * i + 0.2 * average(j)$，得到的结果和直接对用户进行预测相差不多
- 下面我们基于论文进行了更深一步的研究

原理

- 显而易见，好友们在偏好与行为上十分相似，即用户已关注的人对某部电影的评分会显式或隐式的影响用户对该电影的评分，基于此项考虑，我门尝试构建用户关系网的邻接矩阵G，G_{ij}=1如果user_i关注了user_j, 否则为0
-

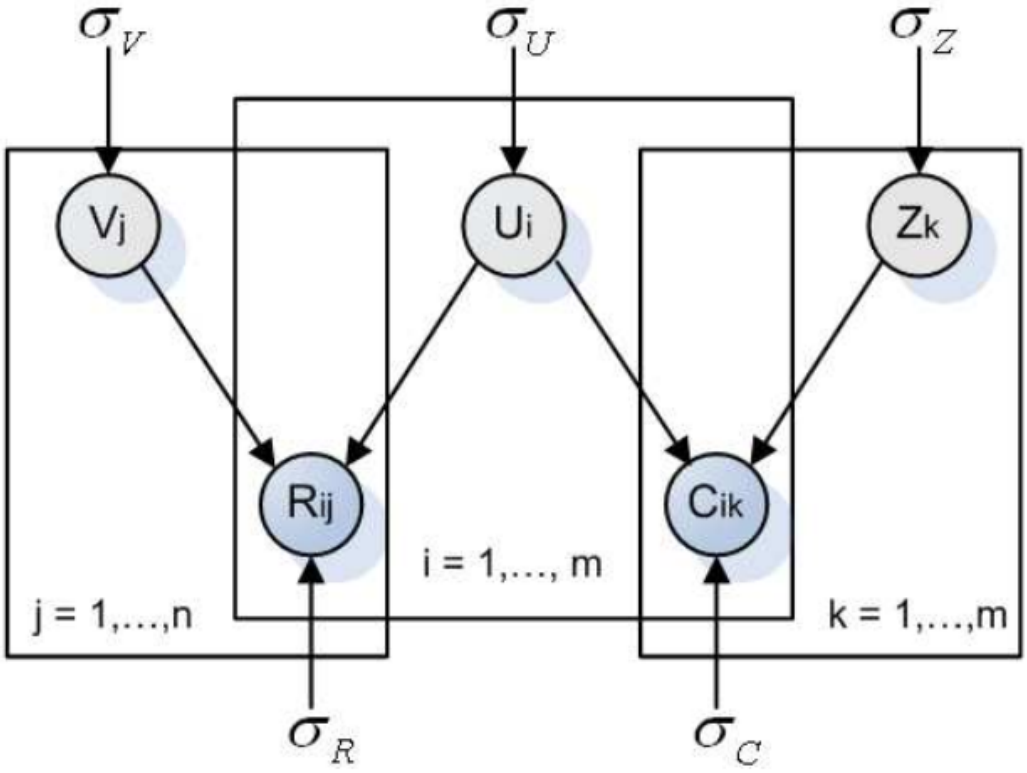


Figure 2: Graphical Model for Social Recommendation

- 利用基于概率矩阵分解（PMF）的因子分析，我们首先认为 评分矩阵R可以分解为物品和用户的潜在特征矩阵，其次，社交网络矩阵也可以分解为同样的用户潜在特征矩阵以及factor-specific潜在特征矩阵
- 考虑到信任网络的的影响关系，对原本的邻接矩阵根据节点的入度、出度做了如下修正

$$\sum_{i=1}^{} \sum_{j=1}^{}$$

$$c_{ik}^{*} = \sqrt{\frac{d^{-}(v_k)}{d^{+}(v_i) + d^{-}(v_k)}} \times c_{ik}, \tag{4}$$

where $d^{+}(v_i)$ represents the outdegree of node v_i, while $d^{-}(v_k)$ indicates the indegree of node v_k.

- 此外，利用sigmoid函数对矩阵乘积进行归一化，并把评分归一化（根据本实验[0,5]的评分范围，利用函数f(x)=x/5 即可）

- 最后，优化目标为（基于条件分布，并同时结合了评分矩阵和社交网络矩阵）

quadratic regularization terms:

最终是用梯度下降法最小化这个式子从而求出U, V, Z三个矩阵

$$\mathcal{L}(R, C, U, V, Z) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n} I_{ij}^{R}(r_{ij} - g(U_i^T V_j))^2 + \frac{\lambda_C}{2}\sum_{i=1}^{m}\sum_{k=1}^{m} I_{ik}^{C}(c_{ik}^{*} - g(U_i^T Z_k))^2 + \frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_V}{2}\|V\|_F^2 + \frac{\lambda_Z}{2}\|Z\|_F^2, \quad (9)$$

where $\lambda_C = \sigma_R^2/\sigma_C^2$, $\lambda_U = \sigma_R^2/\sigma_U^2$, $\lambda_V = \sigma_R^2/\sigma_V^2$, $\lambda_Z = \sigma_R^2/\sigma_Z^2$, and $\|\cdot\|_F^2$ denotes the Frobenius norm. A local minimum of the objective function given by Eq.(9) can be found by performing gradient descent in U_i, V_j and Z_k,

需利用梯度下降法求解。

数据预处理

- 根据数据统计，user-id共2185种，item-id共58431种
- 根据代码接口需要，把训练集读入并利用`scipy.sparse`提供的`coo_matrix`,`csr_matrix`来生成矩阵
  - `def get_trust_data()`

    构建的社交网络矩阵为(2185,2185)
  - `def get_ratings_data()`

    构建的评分矩阵为(2185,58431)
  - 这里需要注意到，根据数据集显示，用户id和物品id都存在不连续值，也就是2185个user-id的值有>2185的情况。所以为了在构建矩阵时让id与矩阵的行、列数对应，考虑边读数据边构建字典 user_id_map和item_id_map来把id映射到[0,2184] 以及 [0,58430]
  - 同样，在预测时，也只需要根据字典映射后找到矩阵对应位置即可

模型构建

- 由于本次实验的要求，所以复现论文模型并不是我们的要求，所以我们在https://blog.csdn.net/q1072118803/article/details/104665790找到了实现好的模型代码并加以修改来契合本次实验。
- 具体思路
  - 设置超参数：

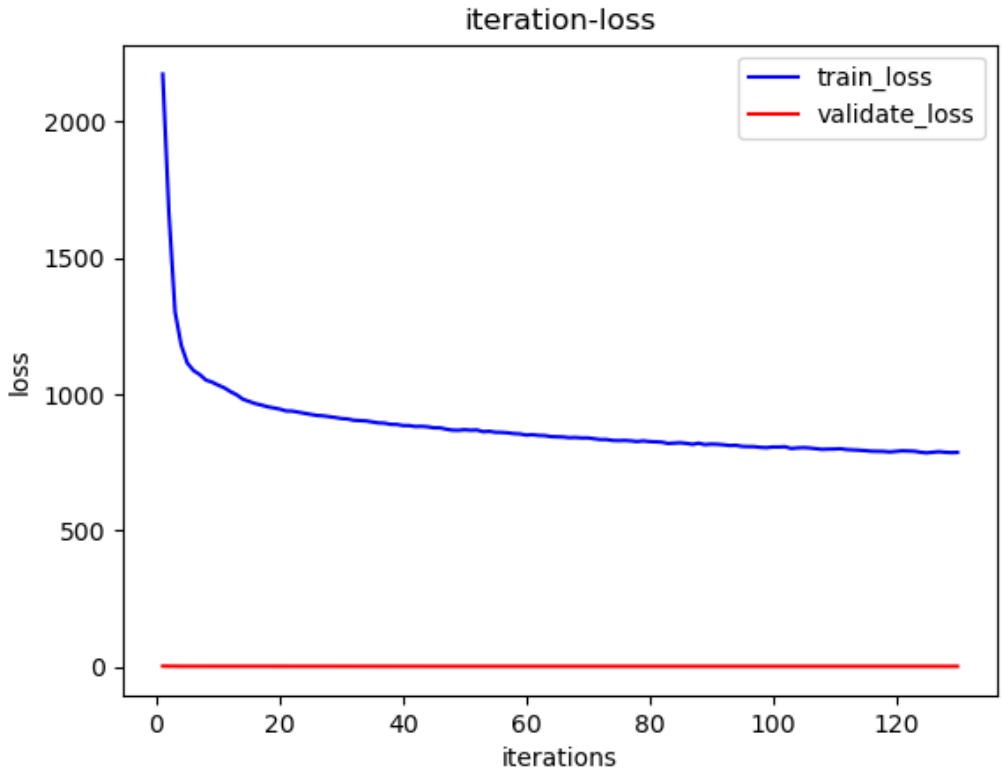
```
lr=0.01    学习率
momentum=0.5 梯度计算因子
latent_size=10   矩阵分解后的潜在特征维度
iters=130    迭代次数
lambda_c=10, lambda_u=0.001, lambda_v=0.001, lambda_z=0.001 均为论文里推荐的取值
```

  - 此外，训练过程把原始数据集training.dat数据随机打乱后按99:1分成训练集和验证集， 最后算得的结果取的是迭代过程里验证集上MAE指标最小的那一组。
  - 预测结果：只需要根据得到的用户潜在特征矩阵U以及物品潜在特征矩阵V乘起来即为预测的评分矩阵。

结果

- 迭代过程示意

```
iter:1, last_train_loss:2175.2604597625236, validate_loss:1.5996388423757355, timecost:38.803784132003784, have run:38.803784132003784
iter:2, last_train_loss:1667.6175951578853, validate_loss:1.2611648290337147, timecost:35.36511301994324, have run:74.76113653182983
iter:3, last_train_loss:1302.3850682955713, validate_loss:1.1445124642730609, timecost:35.891000270843506, have run:111.27399230003357
iter:4, last_train_loss:1180.3085256994423, validate_loss:1.080136360309632, timecost:36.36603093147278, have run:148.2050232887268
iter:5, last_train_loss:1114.0316258972787, validate_loss:1.0576381821402305, timecost:36.109556436538696, have run:184.8725459575653
iter:6, last_train_loss:1086.8281196490855, validate_loss:1.047670229846387, timecost:36.420143604278564, have run:221.84664797782898
iter:7, last_train_loss:1072.2785388495365, validate_loss:1.0259122413934485, timecost:37.75058603286743, have run:260.216237783432
iter:8, last_train_loss:1052.0971901929474, validate_loss:1.0177970188988865, timecost:35.828545570373535, have run:296.62778306007385
iter:9, last_train_loss:1044.3843075337727, validate_loss:1.0085849819286856, timecost:35.277000427246094, have run:332.465749502182
iter:10, last_train_loss:1033.3347233215163, validate_loss:1.0001418905939023, timecost:38.56258177757263, have run:371.5873317718506
iter:11, last_train_loss:1023.3756595313025, validate_loss:0.9875485665789377, timecost:36.067002296447754, have run:408.2343351840973
iter:12, last_train_loss:1009.2463970592356, validate_loss:0.9790009879269035, timecost:35.936057567596436, have run:444.7404239177704
iter:13, last_train_loss:997.863285998902, validate_loss:0.9633967898925889, timecost:36.43627047538757, have run:481.7446959018707
iter:14, last_train_loss:981.6243946981589, validate_loss:0.9562054046391016, timecost:36.12800359725952, have run:518.4490072727203
iter:15, last_train_loss:973.6952630324121, validate_loss:0.9515920556382483, timecost:35.45354723930359, have run:554.4575514793396
iter:16, last_train_loss:965.601735076343, validate_loss:0.94534164913314, timecost:37.7354199886322, have run:592.7599747180939
iter:17, last_train_loss:960.2122367699362, validate_loss:0.9415193605158543, timecost:35.591986656188965, have run:628.9149930477142
iter:18, last_train_loss:954.0063654858673, validate_loss:0.9366532512352801, timecost:37.585997581481934, have run:667.0969746112823
```



-

雷雨轩_PB18111791_10.txt	1.4870294805736384
雷雨轩_PB18111791_11.txt	1.4920552675191596
雷雨轩_PB18111791_12.txt	1.5098238353887279

- 如上图所示，多次调参后，得到的3次结果

模型分析

- 结果相比用surprise库提供的协同过滤模型没有显著提升，考虑原因有以下几点
  - 模型超参数设置可能与本实验数据集存在偏差
  - 社交关系的数据矩阵过于稀疏，导致联合矩阵分解效果不佳
  - 可能本次实验的评分数据集和社会关系数据集本身联系不大，通过社交推荐模型没法很好的找到被关注者对关注者的影响。

实验总结

- 了解了推荐系统的常用算法，并使用Python的Surprise库对测试集中的电影评分进行了预测

- 对基于内存和模型的协同过滤推荐算法有了更加深刻的认识

参考

https://alice1214.github.io/%E6%8E%A8%E8%8D%90%E7%AE%97%E6%B3%95/2020/07/08/%E6%8E%A8%E8%8D%90%E7%AE%97%E6%B3%95%E5%AD%A6%E4%B9%A0-%E4%BA%94-%E7%9F%A9%E9%98%B5%E5%88%86%E8%A7%A3/

https://surprise.readthedocs.io/en/stable/getting_started.html

https://blog.csdn.net/q1072118803/article/details/104665790

https://surprise.readthedocs.io/en/stable/notation_standards.html