协同过滤模型调研报告

Research Report on Collaborative Filtering Models

吴颖馨

2019.10.10

ABSTRACT

本报告由 Matrix Factorization 传统模型入手，至 SIGIR 2019 录用论文 Neural Graph Collaborative Filtering 收尾，对基于协同过滤算法的推荐系统进行了广泛的调研。该论文结合了神经网络模型与协同过滤算法提出了新的推荐系统算法 (NGCF)，并与当代各种主流的推荐算法进行了模型准确度的比较验证。

该论文不仅在数学建模 (如嵌入式传播、高阶连通性、节点丢弃与消息丢弃) 上具有典型性，而且覆盖了 Matrix Factorization(MF), Neural Collaborative Filtering (NCF)、Collaborative Memory Network (CMN)、Graph Convolutional Networks (GCN) 以及 High-order Proximity for Implicit Recommendation (HOP-Rec)、PinSage 等其它主流的推荐系统模型。

本报告通过广泛地研究原始论文以及参考网络上对推荐系统模型的说明与思考，选取 MF、NCF、CMN、NCGF 协同过滤模型进行学习与总结，同时对以及 SGD、BPR 等算法进行了程序实践，最后进行了模型间的比较与说明。

特别地，本报告中忽略了对于模型公式的机理的详细说明，着重于总结模型原理与进行编程实践。限于时间与篇幅，部分模型源代码的研究将留由以后完成。

KEYWORDS

Recommendation, Collaborative Filtering, Matrix Factorization, Memory Network，Graph Neural Network.

1 INTRODUCTION

推荐系统是一种信息过滤系统，它需要已经存在的连接去预测未来的连接，从而找到 user 的个性化需求。协同过滤是推荐系统中最重要的一种算法。协同过滤通常分为两类：基于记忆的协同过滤 (Memory-Based) 和基于模型的协同过滤 (Model-Based)，基于记忆的协同过滤又可以分为基于用户的协同过滤 (User-Based) 和基于物品的协同过滤 (Item-Based)。

普遍来说，要建立一个推荐系统的机器学习模型，首先要进行数据预处理，并把用户的 ID 号这些 categorical features 转换成 numerical features，或者说将用户和项目转换为矢量化表示，这种转换称为一种编码的过程。之后通过矩阵分解等算法、神经网络等模型做特征提取。

利用基于训练集初步建立好的模型，我们能够对测试集进行模型预测。对照模型预测与测试机集的数据，我们能进一步地利用 BPR 等算法得到损失函数，以估量模型的预测值 f(x) 与真实值 Y 的不一致程度。为减小损失函数，采用 SGD 等算法来调节模型的参数，并且可以采取节点丢失或信息丢失来增强模型的鲁棒性 (防止过拟合)，即进一步优化。最后，我们能够得到了一个训练良好的模型，以预测用户喜好或物品的性质。

随着深度学习的推广，我们看到了一些传统模型的局限性，如没有很好地体现局部性特征或忽略了潜在协同信号的提取等。因此，我们应用了一系列手段，如引入神经网络，记忆网络，图神经机制来加强对于用户与物品的特征提取，以在数据预测准确度和实际应用层面达到更好的效果。

2 Matrix Factorization

2.1 Methodology

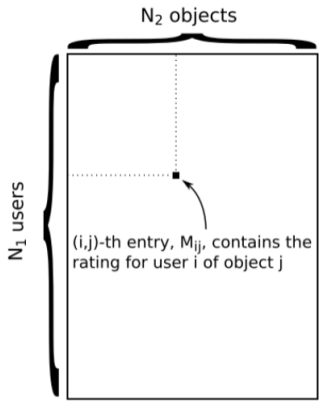

图 1: User-item Matrix

在一个 user-item matrix 中，用横行表示某个特定用户对各种物品的评分或与网页的交互频数，对缺失值一般记为 0，其中用户的序列号一般用 one-hot 编码方式。记上图中 N_1 为 m, N_2 为 n, 则有下述变量说明。

表 1: MF 变量说明

变量	说明
R_{mn}	评分矩阵 (m 个用户对 n 个物品的评分)
P_{mF}	P 的每一行代表某用户对各隐因子的喜欢程度
Q_{Fn}	Q 的每一列代表一个物品在各个隐因子上的概率分布
M_{ij}	已有数据矩阵中用户 i 对物品 j 的评分
r_{ij}	预测结果中用户 i 对物品 j 的评分

MF 总的思路就是先计算用户对各个隐因子的喜好程度 $(p_1, p_2, ..., p_f)$，再计算物品在各个隐因子上的概率分布 $(q_1, q_2, ..., q_f)$，两个向量做内积即得到用户对物品的喜好程度。其假设潜在空间的各个维度相互独立，并以相同的权重线性组合。因此，MF 可视为一个潜在因素的线性模型。

我们的目的为通过矩阵分解使 M_{mn} 的非零值尽可能与 R_{mn} 接近，即得：

$$R_{mn} = P_{mF} \cdot Q_{Fn} \tag{1}$$

$$\hat{r}_{ij} = \sum_{f=1}^{F} P_{if} Q_{fj} \tag{2}$$

即 R 是两个矩阵的乘积，其中 F 是隐因子的个数。

2.2 Optimization

训练的目标是使得对所有的 $r_{ij} \neq 0$，r_{ij} 和 $\hat{r}_{ij}$ 尽可能接近，即

$$min: \quad Loss = \sum_{r_{ij} \neq 0} (r_{ij} - \hat{r}_{ij})^2 \tag{3}$$

为防止过拟合，加个正则项，以对参数进行约束。

$$min: \quad Loss = \sum_{r_{ij} \neq 0} (r_{ij} - \hat{r}_{ij})^2 + \lambda(\sum P_{if}^2 + \sum Q_{fj}^2) = f(P, Q) \tag{4}$$

结合梯度下降法求最优解，在第 $t+1$ 轮迭代中 P 和 Q 的值分别是

$$P^{(t+1)} = P^{(t)} - \alpha \frac{\partial Loss}{\partial P^{(t)}}, Q^{(t+1)} = Q^{(t)} - \alpha \frac{\partial Loss}{\partial Q^{(t)}} \tag{5}$$

$$\frac{\partial Loss}{\partial P^{(t)}} = \begin{bmatrix} \frac{\partial Loss}{\partial P_{11}^{(t)}} & \cdots & \frac{\partial Loss}{\partial P_{1F}^{(t)}} \\ \cdots & \frac{\partial Loss}{\partial P_{if}^{(t)}} & \cdots \\ \frac{\partial Loss}{\partial P_{m1}^{(t)}} & \cdots & \frac{\partial Loss}{\partial P_{mF}^{(t)}} \end{bmatrix} \tag{6}$$

$$\frac{\partial Loss}{\partial P_{if}^{(t)}} = \sum_{j, r_{ij} \neq 0} -2(r_{ij} - \hat{r}_{ij}) Q_{fj}^{(t)} + 2\lambda P_{if}^{(t)} \tag{7}$$

$$\frac{\partial Loss}{\partial Q_{fj}^{(t)}} = \sum_{j, r_{ij} \neq 0} -2(r_{ij} - \hat{r}_{ij}) P_{if}^{(t)} + 2\lambda Q_{fj}^{(t)} \tag{8}$$

不停迭代直到算法最终收敛 (达到约定的 step 或 f(P,Q) 小于某设定阈值)

2.3 Stochastic Gradient Descent

SGD 相比于 GD 而言，进行的是单个参数的更新，且更新单个参数时只利用到一个评分，更新后的参数立即可用于更新剩下的参数，所以 SGD 比批量的梯度下降需要更少的迭代次数。具体地，SGD 有以下两个特点：

- 通过单独更新参数 $P_{if}^{(t+1)} = P_{if}^{(t)} - \alpha\frac{\partial Loss}{\partial P_{if}^{(t)}}$，而不是整体更新参数 $P^{(t+1)} = P^{(t)} - \alpha\frac{\partial Loss}{\partial P^{(t)}}$。
- 在 $t+1$ 轮次中计算其他参数的梯度时直接使用 P_{if} 的最新值 $P_{if}^{(t+1)}$ 计算 $\frac{\partial Loss}{\partial P_{if}^{(t)}}$ 时只利用用户 u 对一个物品的评分，而不是利用用户 u 的所有评分，即

$$\frac{\partial Loss}{\partial P_{if}^{(t)}} = -2(r_{ij} - \hat{r}_{ij})Q_{fi}^{(t)} + 2\lambda P_{if}^{(t)} \tag{9}$$

记 $\alpha/2$ 为步长参数，从而

$$P_{if}^{(t+1)} = P_{if}^{(t)} + \alpha[(r_{ij} - \hat{r}_{ij})Q_{fj}^{(t)} - \lambda P_{if}^{(t)}] \tag{10}$$

$$Q_{fj}^{(t+1)} = Q_{fj}^{(t)} + \alpha[(r_{ij} - \hat{r}_{ij})P_{if}^{(t+1)} - \lambda Q_{fj}^{(t)}] \tag{11}$$

值得提的是，训练模型时 $r_{ij} = 0$ 的项并不会对 Loss 的大小产生影响。通过借助隐因子为“中间变量”，最终得到的与 $r_{ij} = 0$ 对应的项 $\hat{r}_{ij}$ 即能预测用户 u 对物品 i 的评分。

2.4 Model Simulation

本小节通过在 Anaconda 环境下撰写 python 代码实现 MF 模型的模拟，代码见附录 A：MF Code Implement. 代码的输入数据节选自 Jester-Datasets, 为 1999 年 73421 个用户对 100 则笑话的超过 410 万个连续评分。

首先通过 random 函数生成指定 F 值 (F 设置为 3) 下的 P、Q 矩阵，设定 α 值为 0.0002，λ 值为 0.01. 按照前述算法多次单独迭代 P_{if} 与 Q_{fj}。每一次迭代后，计算并记录 Loss 值，在 main 函数中对 Loss 的变化进行可视化，根据图像结果可以调整 steps 与 α、λ 值大小。

```
P:
 [[ 3.04892908e+00  2.17396433e+00  1.69194868e+00]
 [-4.48569102e-02  2.45652534e+00  1.83046160e+00]
 [ 4.71956603e-01 -2.17689106e-03 -1.07762764e+00]
 [ 3.01900776e+00 -1.51090963e+00  7.57445717e-01]
 [ 4.87855020e-01 -1.30791640e+00  1.36098464e+00]
 [-2.32415003e+00  3.94636624e-01  6.09720527e-01]]
Q:
 [[ 0.5946162  -0.73130526 -0.12596645 -0.46373968  2.92299788  3.66480407]
 [ 1.13485615  2.84240739  0.66594526  1.42754179  1.75283118 -0.63134093]
 [ 1.74662592  0.81753061 -0.76552104 -1.28362646  2.10431345 -0.54399341]]
M:
 [[ 7.28  5.39  0.    0.    0.    8.93]
 [ 0.    8.35  0.    0.    8.16 -2.82]
 [ 0.   -2.18  2.67  1.8  -0.44  0.  ]
 [ 0.   -6.02  0.    0.    7.82  0.  ]
 [ 1.26  0.   -1.89 -3.98  1.99  1.84]
 [ 0.29  3.11  0.    0.    0.   -9.17]]
R:
 [[ 7.23528081  5.33281422 -0.23154382 -0.48231455 16.28299602  8.88080609]
 [ 5.95826191  8.51170825  0.24030501  1.17796556  8.02662251 -2.71105582]
 [-1.60404978 -1.23232554  0.76404624  1.16129874 -0.89195391  2.31722517]
 [ 1.40346016 -5.88320189 -1.96631742 -4.52919768  7.77008698 11.60592553]
 [ 1.18292058 -2.96175557 -1.97431647 -3.84033894  1.99738085  1.87326753]
 [ 0.13083222  3.31984639  0.08881742  0.85850746 -4.81871111 -9.09838868]]
Loss:
 11.242709971138158
```

图 2: 输出数据

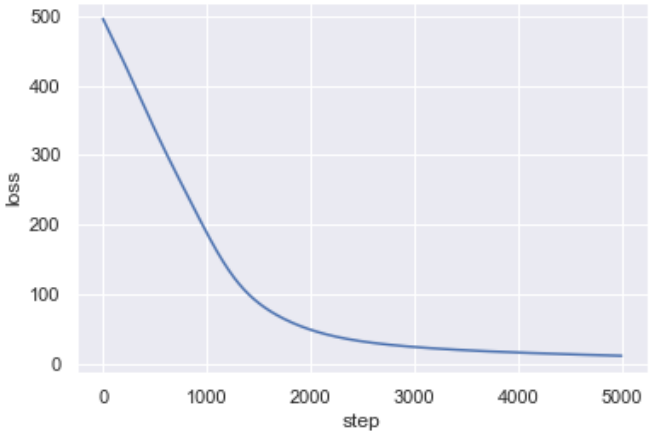

图 3: loss 值随 step 变化图

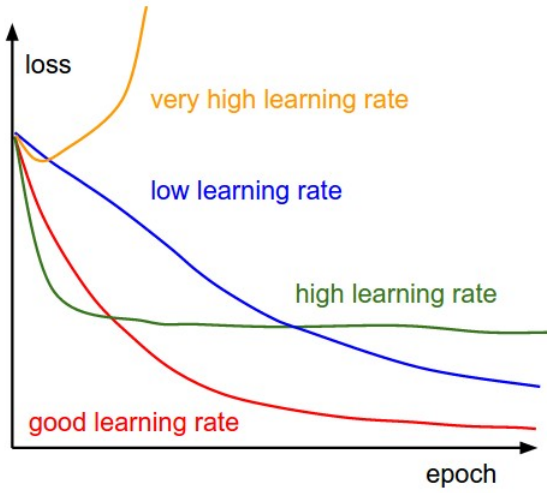

图 4: loss 值随 α(learning rates) 变化图

对于此次模拟有两个发现：

- 从输出数据中可以看出，M 矩阵中的非零值与 R 矩阵中相应的元素值很接近，符合预期。
- 调节参数时 α 值设置过大或 λ 设置过小易产生过拟合现象，同时 λ 参数增大使最终的 Loss 值有较大程度的增大，且数据集越大时，该影响越明显。

对于非零值而言，则为 Matrix Factorization 模型下对未知数据的预测值。

3 Neural Collaborative Filtering

3.1 Model Comparison

MF 是一个线性模型，它不能学习 user-item 交互中潜在的非线性特征。特别是在低维空间，非线性特征的缺失将带来较大的排名损失。

为解决该问题，NCF 将用户和物品的 one-hot encodings 作为输入，然后通过神经网络传播，即使用多层感知机来学习 user-item 交互函数，使模型具有非线性表达能力。

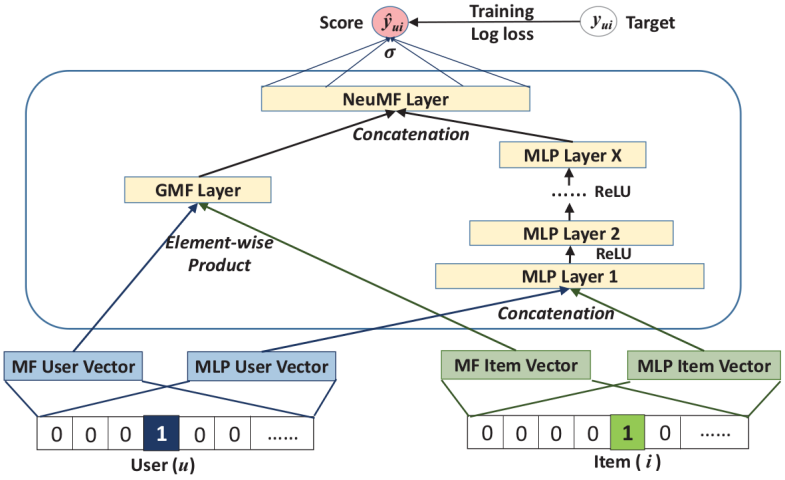

图 5: NCF Model

3.2 Methodology

NCF 的通用框架总结为以下三个部分

- Generalisiere Matrix Factorization (GMF):
  说得简单点，就是在 MF 的基础上以一层神经网络代替 $(p_1, p_2, ..., p_f)$ 与 $(q_1, q_2, ..., q_f)$ 的形成点积运算。特别地，将神经网络的边权矩阵 h 设为全为 1 的矩阵，将激活函数 a_{out} 设为恒等函数，刚好就是 MF 的算法。

$$\phi^{GMF} = p_u^G \bigodot q_i^G \tag{12}$$

$$\hat{y}_{ui} = a_{out}(h^T \phi^{GMF}) \tag{13}$$

  原作者采用 Sigmoid 作为激活函数，可以捕捉传统 MF 所不能学习的非线性特征。

- Multi-Layer Perceptron (MLP):
  多层感知机相比于 GMF 有更多层神经网络，有助于学习更复杂的 user-item 用交互特征。

$$\phi^{MLP} = a_L(W_L^T(a_{L-1}(...a_2(W_2^T \begin{bmatrix} p_u^M \\ q_i^M \end{bmatrix} + b_2)...)) + b_L) \tag{14}$$

$$\hat{y}_{ui} = \sigma(h^T \phi^{MLP}) \tag{15}$$

其中 W_k,b_k,a_k 分别表示 MPL 中第 k-1 层感知器的边权矩阵、偏置向量和激活函数。

- Neural Matrix Factorization (NeuMF):
  经过单独使用 GMF 和 MLP 进行训练后可以获得最后感知机层的权重 h 和偏置项等参数，再将二者的输出做 Concatenation，最后通过一层神经网络来得到最终的评分预测。

$$\hat{y}_{ui} = \sigma(\begin{bmatrix} \alpha h^{GMF} \\ (1-\alpha) h^{MLP} \end{bmatrix}^T \begin{bmatrix} p_u^M \\ q_i^M \end{bmatrix}) \tag{16}$$

  其中 α 为权重控制量。因为 GMF 是对 MF 的扩展，因此可以比较好的捕捉用户、物品之间的线性关系，而 MLP 由于多层神经网络的贡献，可以学习更多的用户、物品之间非线性关系，而 NeuMF 通过对二者的结合，既能很好的学习线性关系，也能很好的学习非线性关系，从而做出更准确的评分预测。

3.3 Optimization&Adam

原论文中对 GMF 与 MLP 层采用 Adaptive Moment Estimation(Adam) 做优化, 而对于 NeuMF 层则采用 SGD 优化。原因是 Adam 需要保存动量信息 (momentum information) 来正确更新参数，而 NeuMF 层是用 GMF 和 MLP 进行训练后得到的参数进行初始化的，因此不需要对动量信息进行保存。SGD 优化算法在 MF section 中已经进行了说明，下面只对 Adam 进行总结。

Adam 自适应矩估计同样适用于稀疏的矩阵。与普通的 SGD 相比，能够计算每个参数的自适应学习率 (无需自己调整学习速率)，并获得更快的收敛速度。普通地，对于给定的一个向量 P(x,y,z...), 我们更新的方式是：

$$P = P - \alpha dP \tag{17}$$

其中 α 为学习速率，dP 为向量梯度。
考虑一个小球在山体滚动的物理情形，即梯度只是影响速度 (动量)，然后速度 (动量) 再影响位置：

$$v = \mu v - \alpha dP \tag{18}$$

$$P = P + v \tag{19}$$

即对动量更新后再与位置信息融合，其中 μ 理解为物理上的摩擦系数。通过动量更新，参数向量会在任何有持续梯度的方向上增加速度。

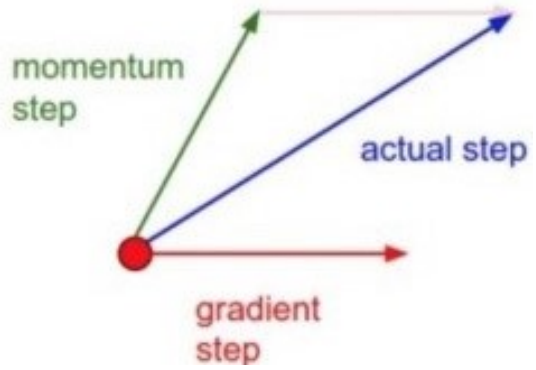

图 6: Momentum Update

另也有 Nesterov 动量更新方法，即考虑用下一个位置的梯度代替小球此刻位置处的梯度，从而加快梯度收敛速率，即

$$P_{next} = P + \mu v \tag{20}$$

$$v = \mu v - \alpha dP_{next} \tag{21}$$

$$P = P + v \tag{22}$$

为了使梯度更新更加平滑，用梯度 m 取代原始梯度 dP。

$$m = \beta_1 m + (1 - \beta_1) dP \tag{23}$$

$$v = \beta_2 v + (1 - \beta_2)(dP)^T dP \tag{24}$$

$$P = P - \alpha \cdot \frac{m}{v^2 + eps} \tag{25}$$

一般取 $eps = 10^{-8}$, $\beta_1 = 0.9$,$\beta_2 = 0.999$。简而言之，模型的梯度是一个随机变量，一阶矩 m 控制模型更新的方向，二阶矩 v 控制学习速率。

4 Collaborative Memory Network

4.1 Overview

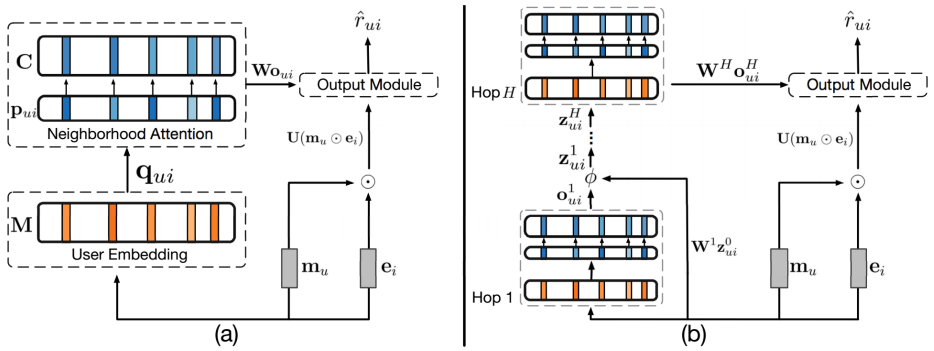

Figure 1: Proposed architecture of Collaborative Memory Network (CMN) with a single hop (a) and with multiple hops (b).

图 7: CMN Model

一个完善的推荐系统模型，不仅仅要考虑到全局的，基于整个评分矩阵的特征，还要考虑到局部的，即基于相似性较高的用户或物品之间的特征。在 MF 模型中，通过隐藏因子捕捉交互的全局特征; 在 NCF 模型中，多层神经网络机制通过非线性的表达在一定程度上能提炼出数据的局部性特征。另外还有前文未提到的基于近邻的相似度模型，其能够捕捉交互的局部特征。

CMN 将隐藏因子模型与相似度模型进行统一，既充分利用整个评分矩阵，又考虑到用户与用户和物品与物品之间的相似性，对相似性高的用户或者物品给予更高的权重，并根据注意力机制和记忆模块刻画复杂的 user-item 交互关系。

4.2 Local Feature Presentation

表 2: CMN 变量说明

变量	说明
M^{PXd}	User 记忆矩阵，P 为 user 数目
E^{QXd}	Item 记忆矩阵，Q 为 item 数目
m_u^{1Xd}	M 中的行向量，编码特定 user 的偏好
e_i^{1Xd}	E 中的行向量，编码特定 item 的性质
$N(i)$	与 item i 有关联的 user 集合
q_{uiv}	表征 user u 与 user v 有关 Item i 的相似性
c_v	user v 的外部记忆，在 C 中的列数由 M 矩阵中 v 相应的行数决定

q_{uiv} 由两部分组成，分别是用于衡量 m_u 与 m_v 相似程度以及衡量 m_v 对 item i 推荐程度的点积运算，并将二者直接相加。个人认为可能用两部分相乘更好，可能是因为前者在数据预测上准确度更高，因此作者采取的相加的方式。得到的式子如下：

$$q_{uiv} = m_u^T m_v + e_i^T m_v \tag{26}$$

有了一系列的 q_{uiv}，能够进一步得到和为 1 的权重向量，用于衡量每个 user 对社区的贡献：

$$p_{uiv} = \frac{exp(q_{uiv})}{\sum_{k \in N(i)} exp(q_{uik})} \tag{27}$$

因此权重更高的用户将对用户 u 在 item i 的推荐上产生更高的决定权。

接下来用 p_{uiv} 和 c_v 的组合相加作为输出，外部记忆允许存储长期信息，这些信息与每个用户在社区中的角色相关。换句话说，关联寻址方案通过注意机制识别邻近区域内的相似用户，作为 key 来对存储在记忆矩阵 C 中的相关值进行加权。

$$o_{ui} = \sum_{v \in N(i)} p_{uiv} c_v \tag{28}$$

至此我们得到了一个加权和 o_{ui}, 表达了特定的 user, item 和整个社区之间的局部特性。

4.3 Global Feature Presentation

最终的预测输出表示为：

$$\hat{r}_{ui} = v^T \phi(U(m_u \bigodot e_i) + W_{o_{ui}} + b) \tag{29}$$

$$\phi(x) = max(0, x) \tag{30}$$

其中 W,U$\in R^{dXd}$, v,b$\in R^d$ 为实验中待设定的参数。括号中的第一项是用户的记忆 m_u 和物品的记忆 e_i 进行的 element-wise 相乘操作，这相当于矩阵分解的思想，即考虑了全局的信息。

之后将 CMN 拓展为多层，保持记忆不变，改变权重向量，根据局部信息与全局信息之间的非线性变换计算排名分数，得到最后的输出。详细内容从略。

4.3.1 Bayesian Personalized Ranking

值得一提的是，CMN 网络采取的是一个 pair-wise 的方式，即训练时每输入一对正负样本相应的有正负两组评分。为了使正样本的得分远大于负样本的得分，CMN 应用了一个常见的算法—BPR.

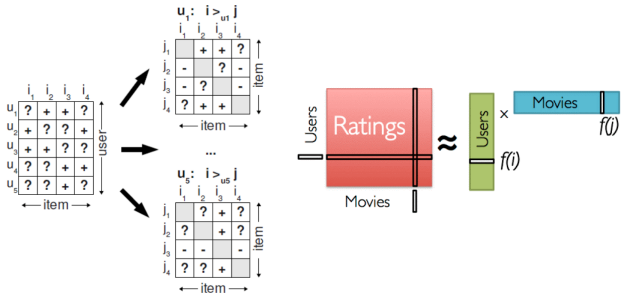

图 8: BPR algorithm illustrations

贝叶斯个性化排序模型同样是将评分矩阵分解为两个矩阵相乘，与 MF 不同的是 BPR 引入了一个核心假设，即某用户对他有过反馈的物品的偏好程度一定比没有反馈过的物品高，未反馈的物品包括真正的负例以及缺失值。而我们的目的为使得正例与负例的差距越大越好。通过采用贝叶斯最大后验概率，结合独立分布原则，则最终的损失函数为

$$Loss(\theta) = \sum_{(u,i,j)\in D} ln(\sigma(\bar{x}_{ui} - \bar{x}_{uj}) + \lambda||\theta||^2 \tag{31}$$

其中 θ 表示两个参数矩阵，σ 为 logistic 函数，$\bar{x}_{ui}$ 与 $\bar{x}_{uj}$ 分别表示 user 对 i 与 j 的评分，λ 为正则化常量。得到 Loss function 后即可对参数进行梯度下降并最小化 Loss 值，具体过程从略。

以下通过运行 Pinard Liu 在 Github 上利用 tensorflow 实现 BPR 算法的代码 (代码详见附录 B BPR code implement, 略有改动)，以更透彻地学习该算法。取矩阵分解的隐因子维度为 20，迭代次数为 10，正则化常量为 0.001. 部分迭代过程的输出结果为：

```
epoch:  1
bpr_loss:  0.7242649589974681
_train_op
test_loss:  0.7715164 test_auc:  0.5018897025389419

epoch:  2
bpr_loss:  0.7236284205355056
_train_op
test_loss:  0.77018845 test_auc:  0.5018144878073648

epoch:  3
bpr_loss:  0.7229894958130955
_train_op
test_loss:  0.76886815 test_auc:  0.5017805319744353

epoch:  4
bpr_loss:  0.7224048877220245
_train_op
test_loss:  0.76762176 test_auc:  0.5017963629250523

epoch:  5
bpr_loss:  0.7217969976203111
_train_op
test_loss:  0.7663763 test_auc:  0.501844701317928

epoch:  6
bpr_loss:  0.7212364097718454
_train_op
test_loss:  0.76523125 test_auc:  0.5018434931836678

epoch:  7
bpr_loss:  0.7206455108808932
_train_op
test_loss:  0.76408434 test_auc:  0.5018668832879095

epoch:  8
bpr_loss:  0.7201505888221407
_train_op
test_loss:  0.7629419 test_auc:  0.501927678846366
```

图 9: BPR iteration process

可以看到 *bpr_loss* 与 *test_loss* 随 epoch 而减小，*test_auc* 在下降后

有震荡现象。可能需要对隐因子维度作进一步调整。接下来我们可以使用得出的 W,H 矩阵，对任意一个 user 求出对所有电影的评分，取评分最高的前五部电影作为推荐。第一列为物品序列号，第二列为 (相对) 评分大小。

```
以下是给用户0的推荐：
39 0.14739878
882 0.115820915
1567 0.12509236
1596 0.13213354
1648 0.12138675
```

图 10: BPR result

5 NGCF

5.1 Overview

对于以上所有模型的建立过程，NGCF 论文指出，诸多模型均未对 user-item 交互时产生的协作信号进行编码。该协作信号潜伏在 user-item 的交互中，以揭示用 users(items) 之间的行为相似性。

NGCF 设计了一种在图上递归传播的神经网络方法。通过在神经网络中插入一个嵌入传播层，从而在嵌入空间中构造信息流。更深入地，叠加多重嵌入传播层，我们可以通过在用户-项交互图上传播嵌入来细化嵌入，以捕获协作信号的高阶连通性。其显式地将协作信号写入模型表示中，保证了模型的准确度。

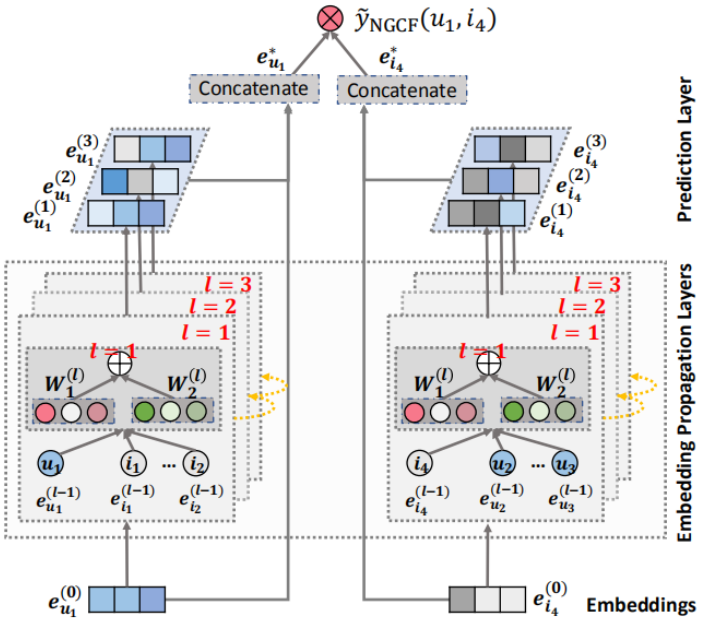

图 11: NGCF overview

该模型不仅解决了现实生活中数据规模大的问题，同时在对三个数据集 (Yelp201、Amazon-book、Gowalla) 预测的性能上优于只使用描述性特性 (例如 id 和属性) 构建嵌入功能的方法。

5.2 Neural Graph

JMedia 有一篇论文叫“为什么图网络是 AI 的未来”，里面深刻地说明了图网络作为一个重要的数学模型的优点。并且图网络在图像和文本的处理方面，已经取得了巨大的成功。

“图像由横平竖直的像素矩阵组成。如果换一个角度，把每个像素视为图谱中的一个点，每个像素点与它周边的 8 个相邻像素之间都有边，而且每条边都等长。通过这个视角，重新审视图像，图像是广义图谱的一个特例。”，同样地在文本处理方面，图网络也有其独特的处理方法，“如果把文本中的每个文字或词汇，当成图谱中的一个点，同时把词与词之间的语法语义关系，当成图谱中的一条边，那么语句和段落，就等同于行走在文本图谱中的一条行进路径。如果能够给每个文字和词汇，都赋予一个贴切的数值张量，那么语句和段落对应的行进路径，多半是最短路径。”

在推荐系统方面，图网络也有其用武之地。将端到端学习与归纳推理相结合，在一定程度上能解决深度学习无法进行关系推理的问题。通过对图像的不断抽象，即从原始的矩阵中，抽象出线段，节点。从首尾相连的相邻线段中，抽象出实体的轮廓。从轮廓抽象出实体，从实体抽象出语义。

5.3 Core Concept

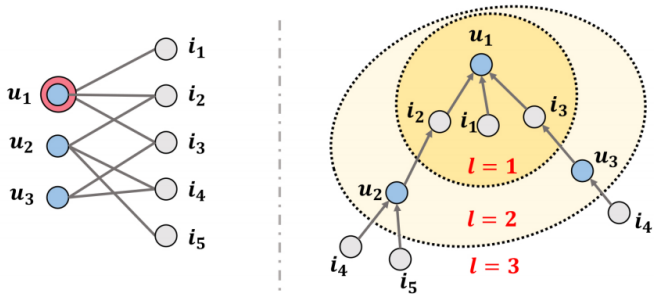

图 12: high-order connectivity

通过在图上表示每个 u 与 i 之间的交互，我们可以清晰地看到推荐系统中存在的高阶连通性。如图所示，通过 u_2,u_3, 可以在 u_1 与 i_4,i_5 间建立联系，这是 MF 中简单的内积形式所不能体现的。

5.4 Methodology

- Embedding Layer
  输入用户与项目交互的数据，建立参数矩阵, 作为初始化。

$$E = [e_{u_1}, ..., e_{u_N}, e_{i_1}, ..., e_{i_N}] \tag{32}$$

- Embedding Propagation Layers

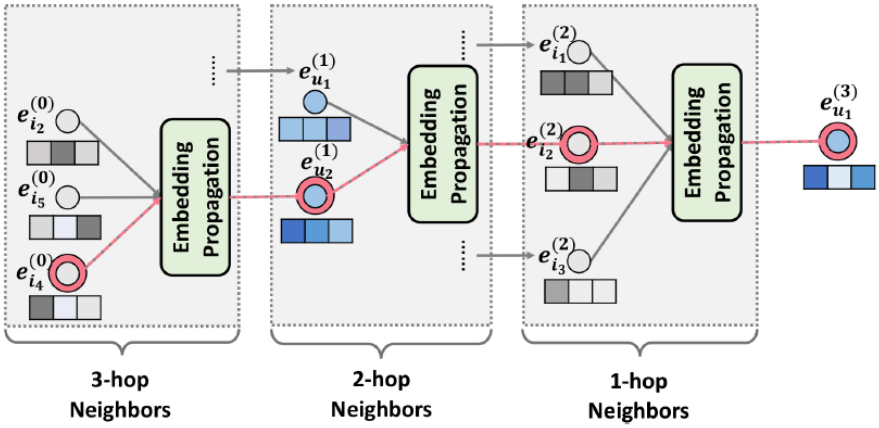

图 13: NGCF illustration

对于一个连接的用户-项目对 (u,i)，将 i 给 u 的消息定义为:

$$m_{u\leftarrow i}^{(l)} = p_{ui}(W_1^{(l)} e_i^{(l-1)} + W_2^{(l)}(e_i^{(l-1)} \bigodot e_u^{(i-1)})) \tag{33}$$

其中 $W_1^{(l)}$,$W_2^{(l)}$ 为参数矩阵，l 表示消息传播的层数，p_{ui} 反映历史项目对用户偏好的贡献程度。从消息传递的角度来看, 其可以被解释为一个折扣因子，因为所传播的消息应该随着路径长度而衰减。

之后的阶段，我们聚合来自 u 的邻居的信息来改进 u 的表示，即

$$e_u^{(l)} = LeakyReLU(m_{u\leftarrow u}^{(l)} + \sum_{i\in\mathcal{N}_u} m_{u\leftarrow i}^{(l)}) \tag{34}$$

其中 $\mathcal{N}_u$ 表示 u 的邻居集合。特别地，公式第一项中考虑了 u 的自我传播，即 $m_{u\leftarrow u}^{(l)}$=$W_1^{(l)} e_u^{(l-1)}$ 以保持 u 的原有特征。由此递推公式出发，相应的有消息传播的矩阵形式，在此省略。

- Model Prediction
  经过 L 层的传播，我们得到了用户 u 的多个表示, 即$e_u^{(1)}$,...,$e_u^{(L)}$, 同样对 i 有类似的表示。考虑到每一种表示在反映 user 的特性上有不同的贡献，我们将其连接作为 u 的最终嵌入表示。

$$e_u^* = e_u^{(0)}||...||e_u^{(L)} \tag{35}$$

最后用点乘的形式得到 u 对于 i 的喜爱程度。

$$\hat{y}_{NGCF} = e_u^{*^T} e_i^* \tag{36}$$

6 MODEL COMPARISION

Table 2: Overall Performance Comparison.

	Gowalla		Yelp2018		Amazon-Book	
	recall	ndcg	recall	ndcg	recall	ndcg
MF	0.1291	0.1878	0.0317	0.0617	0.0250	0.0518
NeuMF	0.1326	0.1985	0.0331	0.0840	0.0253	0.0535
CMN	**0.1404**	**0.2129**	0.0364	0.0745	0.0267	0.0516
HOP-Rec	0.1399	0.2128	**0.0388**	**0.0857**	**0.0309**	**0.0606**
GC-MC	0.1395	0.1960	0.0365	0.0812	0.0288	0.0551
PinSage	0.1380	0.1947	0.0372	0.0803	0.0283	0.0545
NGCF	**0.1547***	**0.2237***	**0.0438***	**0.0926***	**0.0344***	**0.0630***
%Improv.	10.18%	5.07%	12.88%	8.05%	11.32%	3.96%
p-value	1.01e-4	5.38e-3	4.05e-3	2.00e-4	4.34e-2	7.26e-3

图 14: Model Comparision

MF 利用隐因子的思想在建立了一个新的维度，先计算用户对各个隐因子的喜好程度，再计算物品在各个隐因子上的概率分布，并做内积即得到用户对物品的喜好程度。虽然内积可以使用户和物品嵌入的观察到的相互作用接近，但它的线性使它不足以揭示用户和物品之间复杂的非线性关系。

NCF 在 MF 的基础上采用神经网络、多层感知机等形式捕获用户与物品之中的非线性关系。从数据上表明，NCF 相比于 MF 在模型准确度上有明显提升，从而证明了用户物品系统中非线性因素的重要性。

CMN 在前面两种模型的基础上，显式地加强了对用户物品系统中局部特征的体现，即对相似性高的用户或者物品给予更高的权重，有利于刻画更复杂的关系。

NGCF 利用图神经网络，将用户-物品交互抽象化为节点，线段，嵌入层的概念，从而捕获协作信号的高阶连通性，显式地将协作信号写入模型表示中，高效地体现了用户-物品间的高维度复杂关系。

7 ATTAINMENT EXPERIENCE

通过本次较大任务量的调研，收获颇丰。具体来说有以下几点：

- 对于一个模型的建立，往往需要经历发现问题–作出优化–发现问题的循环过程。并且进行优化的角度是多样的，如从局部性特征、全局性特征、高维复杂度等角度出发，能达到不同的效果。建立模型的关键在于继承与创新，并且模型的效果不是绝对的，而是由数据集的大小，具体的数据特征决定。
- 深度学习中的神经网络、图网络都是一些很好的基础模型，通过深刻地理解其中的机制，我们才能顺利地将其应用以解决其他问题。
- 数据，或者说是事物之间的联系往往具有很高层的复杂性。我们的目的就是要把这种联系，通过不同手段抽象出来。而这种抽象靠的是对问题的准确而深刻地认知，以发掘出事物的本质关联。
- 对一个模型，知道其思想与算法远远不够，还应该在代码层面进行仔细研究，才能真正理解这个模型。由于时间有限，本人对本报告中涉及的 CMN,NGCF 模型没能彻底地理解其代码，之后将会继续完成。

8 REFERNENCE

[1]Matrix Factorization[DB/OL].[2018-07-31]. $https://www.jianshu.com/p/af49053832c5$

[2] 推荐系统之矩阵分解 (MF) 及其 python 实现 [DB/OL].[2019-07-23]. $https://hpu-yz.github.io/2019/07/23/$ 推荐系统之矩阵分解 (MF) 及其 python 实现/

[3]Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua.2017. Neural Collaborative Filtering.In WWW.173–182.

[4] 推荐系统论文阅读—Neural Collaborative Filtering[DB/OL].[2018-03-04]. $https://blog.csdn.net/stalbo/article/details/79431662$

[5]Neural Collaborative Filtering NCF 模型的理解以及 python 代码 [DB/OL]. [2019-04-09].$https://blog.csdn.net/qq\_40006058/article/details/89074172$

[6] Travis Ebesu,Bin Shen,and Yi Fang.2018. Collaborative Memory Net work for Recommendation Systems.In SIGIR.515–524.

[7] 推荐系统遇上深度学习 (二十九)–协同记忆网络理论及实践 [DB/OL].
[2019-01-15]. *https*://*www.jianshu.com*/*p*/3*e*80*d*8426*f*7*f*
[8]Collaborative Memory Network for Recommendation Systems 推荐系统之协同记忆网络 CMN[DB/OL].[2019-05-13].
https://*blog.csdn.net*/*qq*$_4$0006058/*article*/*details*/90018931
[9] 贝叶斯个性化排序 (BPR) 算法小结 [DB/OL].[2018-06-03].
https://*www.cnblogs.com*/*pinard*/*p*/9128682.*html*
[10] 用 tensorflow 学习贝叶斯个性化排序 [DB/OL].[2018-06-10].
https://*www.cnblogs.com*/*pinard*/*p*/9163481.*html*
[11] 推荐算法之贝叶斯个性化排序 BPR[DB/OL].[2019-07-23].
https://*www.biaodianfu.com*/*bpr.html*
[12] 矩阵分解之 SVD 和 SVD++[DB/OL].[2018-08-26].
https://*www.jianshu.com*/*p*/*f*06860717*c*9*e*
[13]]Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng and Tat-Seng Chua.2019. Neural Graph Collaborative Filtering.In SIGIR.
[14]Neural Graph Collaborative Filtering–总结.[2019-07-04].
https://*www.jianshu.com*/*p*/16*c*8973*ef*8*ff*
[15] 如何理解 Adam 算法 [DB/OL].[2019-08-16].
https://www.zhihu.com/question/323747423/answer/790457991

A MF Code Implement

```
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import seaborn as sns
import xlrd
import pprint
from sklearn import preprocessing

#M为原始数据矩阵, alpha/2为步长, lamb(lamba)为正则参数
def MF_training(M, P, Q, alpha = 0.0002, lamb = 0.01, steps = 5000):

  Loss = []
  for step in range(steps):
    for i in range(len(M)):
      for j in range(len(M[0])):
```

```
            eij=M[i][j]-np.dot(P[i,:],Q[:,j])
          for f in range(len(Q)):
            if M[i][j] != 0:#对于不为零的项进行迭代
              P[i][f] += alpha * (eij * Q[f][j] - lamb * P[i][f])
              Q[f][j] += alpha * (eij * P[i][f] - lamb * Q[f][j])

      e = 0
      for i in range(len(M)):#计算每个step执行后的Loss
        for j in range(len(M[0])):
          if M[i][j] != 0 :
            e += pow(M[i][j] - np.dot(P[i,:], Q[:,j]), 2)
          for f in range(len(Q)):
            e += lamb * (pow(P[i][f], 2) + pow(Q[f][j], 2))

      Loss.append(e)
      if e < 0.001:
        break
      return P, Q, Loss


def excel_to_matrix(path):

  table = xlrd.open_workbook(path).sheets()[0]
  row = table.nrows  # 行数
  col = table.ncols  # 列数
  #生成一个nrows行ncols列，且元素均为0的初始矩阵
  matrix = np.zeros((row, col))

  for i in range(col):
    cols = np.matrix(table.col_values(i))
    matrix[:, i] = cols
  return matrix

if __name__ == '__main__':   #主函数

  datafile = r'C:\Users\ASUS\Desktop\MF_test.xlsx'

  M = excel_to_matrix(datafile)
  m = len(M)
  n = len(M[0])
  F = 3 #可调整

  P = np.random.rand(m,F) #随机生成一个 m行 F列的矩阵
  Q = np.random.rand(F,n) #随机生成一个 F行 n列的矩阵

  P_, Q_, Loss = MF_training(M,P,Q)
```

```
  R = np.dot(P_,Q_)

  print('P:\n',P)
  print('Q:\n',Q)
  print('M:\n',M)
  print('R:\n',R)
  print('Loss:\n',Loss[-1])
  plot1 = plt.plot(range(len(Loss)),Loss)
  plt.xlabel("time")
  plt.ylabel("loss")
  plt.show()
  fig.savefig('MF_1.png')
```

B BPR Code Implement

```
import numpy
import tensorflow as tf
import os
import random
import pprint
from collections import defaultdict

def load_data(data_path):
  user_ratings = defaultdict(set)
  max_u_id = -1
  max_i_id = -1
  with open(data_path, 'r') as f:
    for line in f.readlines():
      u, i, _, _ = line.split("\t")
      u = int(u)
      i = int(i)
      user_ratings[u].add(i)
      max_u_id = max(u, max_u_id)
      max_i_id = max(i, max_i_id)
  print ("max_u_id:", max_u_id)
  print ("max_i_id:", max_i_id)
  return max_u_id, max_i_id, user_ratings


data_path = os.path.join(r'F:\Data_Science\ml-100k', 'u.data')
user_count, item_count, user_ratings = load_data(data_path)
```

```
#randomly pick i for each u
def generate_test(user_ratings):
  user_test = dict()
  for u, i_list in user_ratings.items():
    user_test[u] = random.sample(user_ratings[u], 1)[0]
  return user_test

user_ratings_test = generate_test(user_ratings)

#generate (u, i, j) for training
def generate_train_batch(user_ratings, user_ratings_test, item_count,
                                    batch_size=512):
  t = []
  for b in range(batch_size):
    u = random.sample(user_ratings.keys(), 1)[0]
    i = random.sample(user_ratings[u], 1)[0]
    while i == user_ratings_test[u]:
      i = random.sample(user_ratings[u], 1)[0]

    j = random.randint(1, item_count)
    while j in user_ratings[u]:
      j = random.randint(1, item_count)
    t.append([u, i, j])
  return numpy.asarray(t)

#for i picked up with respect to u in the function <generate_test> get (
                                    u, i, j)
#where j is a negative case for u
def generate_test_batch(user_ratings, user_ratings_test, item_count):
  for u in user_ratings.keys():
    t = []
    i = user_ratings_test[u]
    for j in range(1, item_count+1):
      if not (j in user_ratings[u]):
        t.append([u, i, j])
  yield numpy.asarray(t)

def bpr_mf(user_count, item_count, hidden_dim):
  u = tf.placeholder(tf.int32, [None])
  i = tf.placeholder(tf.int32, [None])
  j = tf.placeholder(tf.int32, [None])

with tf.device("/cpu:0"):#Get Matrix W, H
  user_emb_w = tf.get_variable("user_emb_w", [user_count+1, hidden_dim],
  initializer=tf.random_normal_initializer(0, 0.1))
```

```
  item_emb_w = tf.get_variable("item_emb_w", [item_count+1, hidden_dim],
  initializer=tf.random_normal_initializer(0, 0.1))

  u_emb = tf.nn.embedding_lookup(user_emb_w, u)
  i_emb = tf.nn.embedding_lookup(item_emb_w, i)
  j_emb = tf.nn.embedding_lookup(item_emb_w, j)

# MF predict: u_i > u_j
#compute 1th term:  difference of xui and xuj
x = tf.reduce_sum(tf.multiply(u_emb, (i_emb - j_emb)), 1, keep_dims=True
                                    )

# AUC for one user:
# reasonable iff all (u,i,j) pairs are from the same user
# average AUC = mean( auc for each user in test set)
mf_auc = tf.reduce_mean(tf.to_float(x > 0))

#computer 2th term
l2_norm = tf.add_n([
  tf.reduce_sum(tf.multiply(u_emb, u_emb)),
  tf.reduce_sum(tf.multiply(i_emb, i_emb)),
  tf.reduce_sum(tf.multiply(j_emb, j_emb))
  ])

regulation_rate = 0.0001
#computte loss
bprloss = regulation_rate * l2_norm - tf.reduce_mean(tf.log(tf.sigmoid(x
                                    )))
#gradient decent
train_op = tf.train.GradientDescentOptimizer(0.01).minimize(bprloss)
return u, i, j, mf_auc, bprloss, train_op


if __name__ == "__main__":

  with tf.Graph().as_default(), tf.Session() as session:
    u, i, j, mf_auc, bprloss, train_op = bpr_mf(user_count, item_count,
                                    20)
    session.run(tf.initialize_all_variables())
    for epoch in range(1, 10):
      _batch_bprloss = 0
      for k in range(1, 5000): # uniform samples from training set
        uij = generate_train_batch(user_ratings, user_ratings_test,
                                    item_count)

        _bprloss, _train_op = session.run([bprloss, train_op],
```

```
            feed_dict={u:uij[:,0], i:uij[:,1], j:uij[:,2]})
          _batch_bprloss += _bprloss

        print ("epoch: ", epoch)
        print ("bpr_loss: ", _batch_bprloss / k)
        print ("_train_op")

        user_count = 0
        _auc_sum = 0.0

        # each batch will return only one user's auc
        for t_uij in generate_test_batch(user_ratings, user_ratings_test,
                                         item_count):

          _auc, _test_bprloss = session.run([mf_auc, bprloss],
            feed_dict={u:t_uij[:,0], i:t_uij[:,1], j:t_uij[:,2]}
            )
          user_count += 1
          _auc_sum += _auc
        print ("test_loss: ", _test_bprloss, "test_auc: ", _auc_sum/
                                         user_count)
        print ("")
      variable_names = [v.name for v in tf.trainable_variables()]
      values = session.run(variable_names)
      for k,v in zip(variable_names, values):
        print("Variable: ", k)
        print("Shape: ", v.shape)
        print(v)

session1 = tf.Session()
u1_dim = tf.expand_dims(values[0][0], 0)
u1_all = tf.matmul(u1_dim, values[1],transpose_b=True)
result_1 = session1.run(u1_all)
print (result_1)

print("以下是给用户0的推荐: ")
p = numpy.squeeze(result_1)
p[numpy.argsort(p)[:-5]] = 0
for index in range(len(p)):
  if p[index] != 0:
    print (index, p[index])
```