Deep Probabilistic Matrix Factorization Framework for Online Collaborative Filtering

高相阳

摘要

随着生活数据的快速增长和演变，传统的机器学习算法在处理新的训练数据时很难在处理新的训练数据时很难更新模型。当新的数据到来时，传统的协同过滤方法必须从头开始训练他们的模型。对他们来说，重新训练模型和更新参数的成本很高。与传统的协同过滤方法相比，在线协同过滤方法是有效的当新数据到来时，可以立即更新模型。但冷启动和数据稀少仍然是在线协同过滤的主要问题。在本文中，我们尝试利用神经网络从用户/项目的侧面信息中提取用户/项目特征来解决这些问题。首先，我们提出了一个概率矩阵分解（PMF）模型，利用神经网络来提取潜伏的用户/物品特征，并在概率矩阵分解中加入偏见，以跟踪用户评级行为和项目流行度。其次，我们对特定用户和特定项目的特征向量进行约束，以进一步以进一步提高 PMF 的性能。第三，我们通过一个在线学习算法更新两个模型。实验结果对数据集（MovieLens100K）进行的广泛实验表明，我们的方法比基线方法有更好的性能。

关键词：深度学习；协同过滤；概率矩阵分解

1 引言

如今，人们在面对各种选择时很难做出合理的决定。为了准确预测客户的需求，需要有效的方法来分析数据。推荐系统就是其中一个解决方案。帮助公司决定应该向不同的用户展示什么以帮助用户快速找到符合他们兴趣的项目。许多公司已经尝试设计不同的模型来应用于推荐系统。使用推荐系统来为用户发现新的歌曲。社会网络使用推荐系统来推荐兴趣相投的朋友。一类成功的方法应用于推荐系统协同过滤（CF）。协同过滤的关键思想是 CF 的关键思想是，如果两个用户在过去有类似的偏好，他们也会喜欢类似的产品。CF 方法不涉及任何物品和用户的信息。用户的信息，而只是通过使用过去的交互式信息来进行推荐，例如用户的明确评级或对商品的隐性评论。CF 方法包含许多机器学习方法。矩阵分解（MF）方法是应用于推荐系统的最流行的方法。系统。另外，概率矩阵分解法（PMF）被广泛地应用于协同过滤。

在本文中，我们尝试利用神经网络从用户/项目的侧面信息中提取用户/项目特征来解决这些问题。首先，我们提出了一个概率矩阵分解（PMF）模型，利用神经网络来提取潜伏的用户/物品特征，并在概率矩阵分解中加入偏见，以跟踪用户评级行为和项目流行度。其次，我们对特定用户和特定项目的特征向量进行约束，以进一步以进一步提高 PMF 的性能。第三，我们通过一个在线学习算法更新两个模型。实验结果对数据集（MovieLens100K）进行的广泛实验表明，我们的方法比基线方法有更好的性能。

2 Motivation

协同过滤推荐（Collaborative Filtering based methods）方法，可以说是现今推荐系统中应用最为广泛和成熟的方法。它利用用户之间共同的偏好或者物品之间共同的持征属性来进行推荐，然后用户可以通过某种合作的机制给予系统相当程度的回应如评分）并记录下来帮助推荐系统更好的预测结果，在协同过滤中有两种主流的方法：一种是基于用户的协同过滤，如图 1 所示；另外一种是基于物品的协同过滤，如图 2 所示。由此不难看出，协同过滤方法并不需要对用户或者物品进行严格的建模，此外由于它可以与别人共用经验，因此能够帮助用户发现自己潜在的偏好。正因为该方法依赖于历史数据信息，和基于内容的推荐一样存在冷启动问题，同时用户的偏好和物品的属性特征都是利用历史数据提取的，很难修改或者捕捉用户和物品特征的时序变化，从而导致这个方法不是特别的灵活。

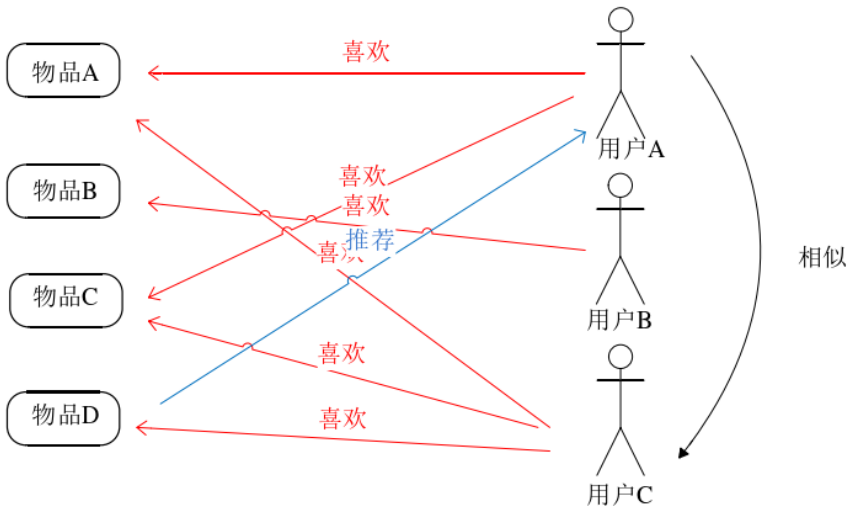

图 1: 基于用户的协同过滤

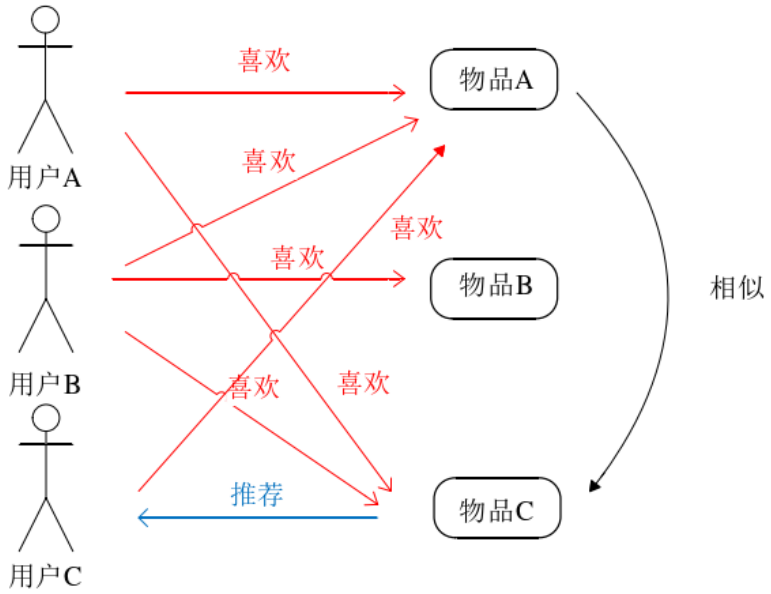

图 2: 基于物品的协同过滤

此时便可以引入混合推荐（Hybrid methods）方法，它的机制包括加权的混合，即用线性公式将集中不同的推荐模型按照一定的权重组合起来，确切的权重值需要在数据集上实验以找到合适的权重，从而达到良好的推荐效果；切换的混合，即对于不同的情况（用户或物品的数量等），推荐策略可能存在很大的差异，这时候应用切换的混合方法，就可以在不同的情况下，选出合适的方式以达到良好的推荐效果：分区混合，即采用多种推荐并存的机制，将不同的推荐结果推送给不同分区的用户，这

样用户就能得到全面的推荐，从而达到良好的推荐效果；分层的混合，即采用多种推荐机制并将一个推荐机制的结果作为下一个推荐机制的输入，从而综合每个推荐机制的优点，来提升推荐效果。

与传统的机器学习方法相比，深度学习中没有复杂的特征工程，能够自动抽取特征，并且随着高性能计算的发展和海量数据的积累，使得近几年来深度学习的应用越来越广泛，包括自然语言处理、语音识别、图像识别等领域，都取得了比传统机器学习方法更好的效果，将深度学习应用到推荐系统中，可以更好的抽象出用户的兴趣偏好，因此把深度学习与推荐系统相结合成为近年来深度学习发展的一个新的热点。

3 相关工作

现如今越来越多的学者开始研究用户间关系对推荐系统造成的影响，工业界在推荐过程中也开始考虑利用用户间的联系来为用户进行更优质的推荐，现如今，借助用户间的信息深入挖掘用户兴趣已经成为工业界工作的新重点。在社交网络时代，可以找到用户信任的朋友并交到自己的朋友，与传统的社交网络相比，用户需要找到与目标用户相似的一组用户，并寻找朋友。年龄、性别、身高以及信息传播等信息可用于改善模型的性能。

2012 年周涛等人介绍了推荐系统中基于相似性的方法、降维技术、基于传播的方法和社会过滤算法并对个性化推荐方法进行了总结。2017 年 Zhang C 等人采用协同用户嵌入法提取用户对项目隐含、可靠的社会信息并将这些社会信息作为补充评级数据来提高矩阵分解算法的性能。2017 年 Wang M 等人提出了一种新型的融入社交信息的协同过滤模型，该模型假定用户从他们在线的朋友那里得到信息，但用户与朋友之间并不都存在相同的偏好。2019 年 Fan W 等人设计了一种深层社交协同过滤框架，该框架在社交网络更深层次的社交信息来进行推荐。

关于神经网络的个性化推荐算法的研究更加深入。2016 年 Hidasi 等人设计了一种基于循环神经网络的会话推荐方法，该方法能够根据用户短时间内的行为来进行推荐。Zheng Y 等人在 2016 年针对神经协作推荐方法而开发一种协作过滤框架高斯分布范围结合在一起的方法，该模型能够根够挖掘推荐的得分边界。2017 年 Xue HJ 提出了一种新的基于神经网络结构的矩阵分解模型，该模型利用一个深层结构学习框架将用户和项目学习到一个共同的低维空间，基于二元交叉滴的新损失函数进行优化。2018 年黄立威等人展示了目前推荐方法中用到的深度学习模型，介绍了应用卷积神经网络、循环神经网络模型的相关推荐算法并对基于深度学习推荐算法应用领域进行总结。

4 本文方法

4.1 本文方法概述

本文复现的工作首先是使用 python 语言实现论文中描述的 PMF 方法，并在 ML100K 实验集上进行实验，尤其是分析不同活跃度用户上的推荐效果，其中分析推荐效果的指标通过 MAE，RMSE 进行判断。

其次是在 pytorch 深度学习平台上建立多层感知机神经网络模型，实现论文中描述的 PMF 方法，并在 ML100K 实验集上进行离线训练实验，与非深度学习版本的效果进行比较和分析。

4.2 概率矩阵分解 (PMF) 实现

假设有 N 个用户，M 个商品，形成一个 N×M 维的评分矩阵 R，矩阵 R 中的元素 Rij 表示用户 i 对商品 j 的评分。假设潜在特征个数为 D，那么 D×N 维的矩阵 U 表示用户的潜在特征矩阵，Ui 表示用户 i 的潜在特征向量；D×M 维的矩阵 V 表示商品的潜在特征矩阵，Vj 表示商品 j 的潜在特征向量。概率模型图如下图 3 所示：

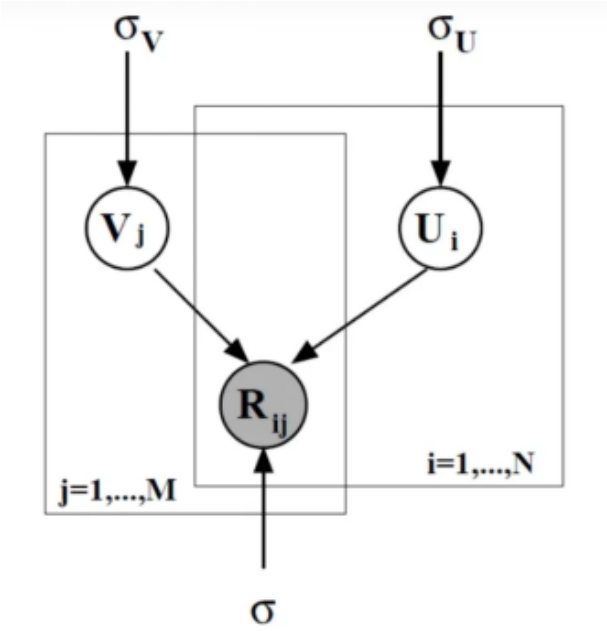

图 3: PMF 的概率模型图

假设关于已知评分数据的条件分布满足高斯分布：

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

再假设用户潜在特征向量和商品潜在特征向量都服从均值为 0 的高斯先验分布，然后计算的后验概率并等式左右两边取对数得：

$$\ln p\left(U, V \mid R, \sigma^2, \sigma_V^2, \sigma_U^2\right)=-\frac{1}{2\sigma^2} \sum_{i=1}^{N} \sum_{i=1}^{M} I_{ij}\left(R_{ij}-U_i^T V_j\right)^2-\frac{1}{2\sigma^2} \sum_{i=1}^{N} U_i^T U_i \frac{1}{2\sigma^2} \sum_{i=1}^{M} V_i^T V_i-\frac{1}{2}\left(\left(\sum_{i=1}^{N} \sum_{i=1}^{M} I_{ij}\right) \ln \sigma^2+N D \ln \sigma_U^2+M D \ln \sigma_V^2\right)+C$$

求上式的最大值，等价于最小化目标函数：

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

用 E 对两者分别求导并使用随机梯度下降法更新得：

$$U_i \leftarrow U_i-\eta \frac{\partial E}{\partial U_i}=U_i+\eta\left(\left(R_{ij}-U_i^T V_j\right) V_j-\lambda_U U_i\right)$$

$$V_i \leftarrow V_j-\eta \frac{\partial E}{\partial V_j}=V_j+\eta\left(\left(R_{ij}-U_i^T V_j\right) U_i-\lambda_V V_i\right)$$

4.3 多层感知机 (MLP) 模型实现

多层感知机对比感知机解决了之前无法模拟异或逻辑的缺陷的同时可以学习到非线性映射且完成实时学习。一般来说，我们用 MLP 根据模型训练过程中的监督信息的误差反向传播来学习隐藏参数。在这个过程中，多层感知机将有监督信息的稀疏的高维度特征进行压缩和抽象，高维度特征中的有效特征部分被保留或合并，部分转化为低维分布式表示。应用到神经协同过滤框架中可理解为把用户、项目的原始特征表示抽取出来用隐向量表示这样一个降低推荐系统规模的过程。

由下图 4 所知，模型分为四层，输入层输入是的用户 id 和商品 id 的 one-hot 编码，嵌入层是经过编码层后得出用户潜在特征向量，商品特征向量，第三层由神经协同过滤层，使用乘法层通过基本积 (element-wise) 替换原有传统的基于内积的矩阵分解方式或者使用深层神经网络多层感知机，然后加上激活函数实现线性或非线性建模，层与层之间通过上层 (靠近输入层) 作为下层 (靠近输出层) 的输入实现交互或隐特征的学习及建模。第四层是由输出层，输出为预测评分矩阵。

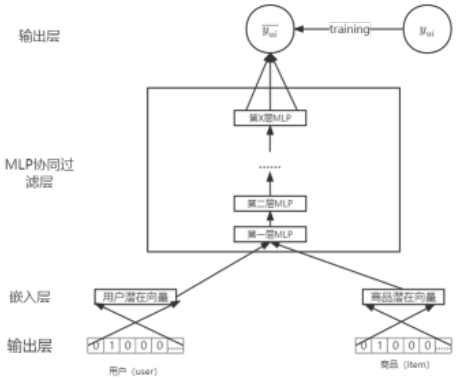

图 4: MLP 的模型示意图

模型每一层 MLP 的输出和模型最后的输出公式表示为如下所示：

$$z_i = \begin{cases} a_i\left(W_L^T Z_{L-1} + b_L\right), i = 2 \ldots L \\ \phi_L\left(p_u, q_i\right) = \begin{pmatrix} p_u \\ q_i \end{pmatrix}, i = 1 \\ \overline{y_{ui}} = \sigma\left(h^T z_L\right) \end{cases}$$

5 复现细节

5.1 与已有开源代码对比

概率矩阵分解 (PMF) 算法部分在原有代码上进行了改进，深度学习方面实现了多层感知机 (MLP) 模型以及预测结果评估。在评估结果生成之后，进行了简单的曲线图绘制便于直观感受区别。

5.2 PMF 算法伪代码

Procedure 1 Probabilistic Matrix Factorization Algorithm.

Input: the number of latent factor K, the learning rata eta, regularization parameters lambda1, lambda2, the max iteration Step,and the rating matrix R

Initialization:Initialize a random matrix for user matrix U and item matrix V

```
for t = 1, 2,...Step do
    for (u,i,r) in R do
        make prediction pr=Ui^T * Vj
        error e=r - pr
        update Ui and Vj
        the algorithm suffers a loss (Ui, Vj, r)
    end
end
```

5.3 核心代码截图

```python
def train(self):
    P = np.random.normal(0, 0.1, (self.N, self.K))
    Q = np.random.normal(0, 0.1, (self.M, self.K))

    train_mat = sequence2mat(sequence_=_self.train_list, N_=_self.N, M_=_self.M)
    test_mat = sequence2mat(sequence_=_self.test_list, N_=_self.N, M_=_self.M)

    records_list = []
    for step in range(self.max_iteration):
        los=0.0
        for data in self.train_list:
            u,i,r = data
            P[u],Q[i],ls = self.update(P[u], Q[i], r=r,
                                       learning_rate=self.learning_rate,
                                       lamda_regularizer=self.lamda_regularizer)
            los += ls
        pred_mat = self.prediction(P,Q)
        mae, rmse, recall, precision = evaluation(pred_mat, train_mat, test_mat)
        records_list.append(np.array([los, mae, rmse, recall, precision]))

        if step % 10 ==0:
            print(' step:%d \n loss:%.4f,mae:%.4f,rmse:%.4f,recall:%.4f,precision:%.4f'
                  %(step,los,mae,rmse,recall,precision))

    print(' end. \n loss:%.4f,mae:%.4f,rmse:%.4f,recall:%.4f,precision:%.4f'
          %(records_list[-1][0],records_list[-1][1],records_list[-1][2],records_list[-1][3],records_list[-1][4]))
    return P, Q, np.array(records_list)
```

图 5: PMF 核心代码截图

```python
def training(model, trainData,predData, batch_size,num_epochs=20, learning_rate=0.5):
    train_dataset=PMFDATASET(trainData[:,0],trainData[:,1],predData)
    train_loader=DataLoader(train_dataset,batch_size=batch_size)
    #使用定义优化器
    optimizer=optim.Adam(model.parameters(),lr=learning_rate)
    #定义损失函数 交叉熵代价函数
    mess_loss=nn.MSELoss()
    train_ls, valid_ls = [], []
    rmse=[]
    for epoch in range(num_epochs):
        total_loss, total_len = 0.0, 0
        total_loss2=0.0
        for user_id,item_id,rating in train_loader:
            #rating = rating.to(torch.int64)
            y_pred=model(user_id,item_id)
            l=mess_loss(y_pred,rating)
            optimizer.zero_grad()  #梯度清零
            l.backward()
            optimizer.step()
            total_loss+=l.item()
            total_loss2+=l.item()
            total_len+=len(y_pred)

        train_ls.append(total_loss/total_len)
        rmse.append(math.sqrt(total_loss2/total_len))
        print('epoch %d, train mse %f, train rmse %f' % (epoch + 1, train_ls[-1],rmse[-1]))
    return train_ls
```

图 6: MLP 核心代码截图

5.4 创新点

在原有 PMF 模型的基础上，增加了 n 层的 MLP 层，多层感知机对比感知机解决了之前无法模拟异或逻辑的缺陷的同时可以学习到非线性映射且完成实时学习，应用到神经协同过滤框架中可理解为把用户、项目的原始特征表示抽取出来用隐向量表示这样一个降低推荐系统规模的过程。

6 实验结果分析

6.1 PMF 算法结果

```
step:0
loss:548724.8577,mae:3.5219,rmse:3.6967,recall:0.0217,precision:0.0461
step:10
loss:65449.6996,mae:0.7813,rmse:0.9942,recall:0.0323,precision:0.0684
step:20
loss:62628.7700,mae:0.7546,rmse:0.9566,recall:0.0323,precision:0.0685
step:30
loss:61452.3035,mae:0.7422,rmse:0.9405,recall:0.0333,precision:0.0707
step:40
loss:60660.6761,mae:0.7354,rmse:0.9317,recall:0.0332,precision:0.0704
step:50
loss:60020.7035,mae:0.7310,rmse:0.9263,recall:0.0333,precision:0.0707
step:60
loss:59509.4126,mae:0.7282,rmse:0.9231,recall:0.0323,precision:0.0686
step:70
loss:59112.2272,mae:0.7266,rmse:0.9215,recall:0.0291,precision:0.0617
step:80
loss:58807.6870,mae:0.7256,rmse:0.9209,recall:0.0271,precision:0.0574
step:90
loss:58574.9342,mae:0.7251,rmse:0.9207,recall:0.0271,precision:0.0574
end.
loss:58412.2211,mae:0.7250,rmse:0.9209,recall:0.0260,precision:0.0551
MAE:0.7250;RMSE:0.9209;Recall:0.0260;Precision:0.0551
```

图 7: PMF 程序运行结果图

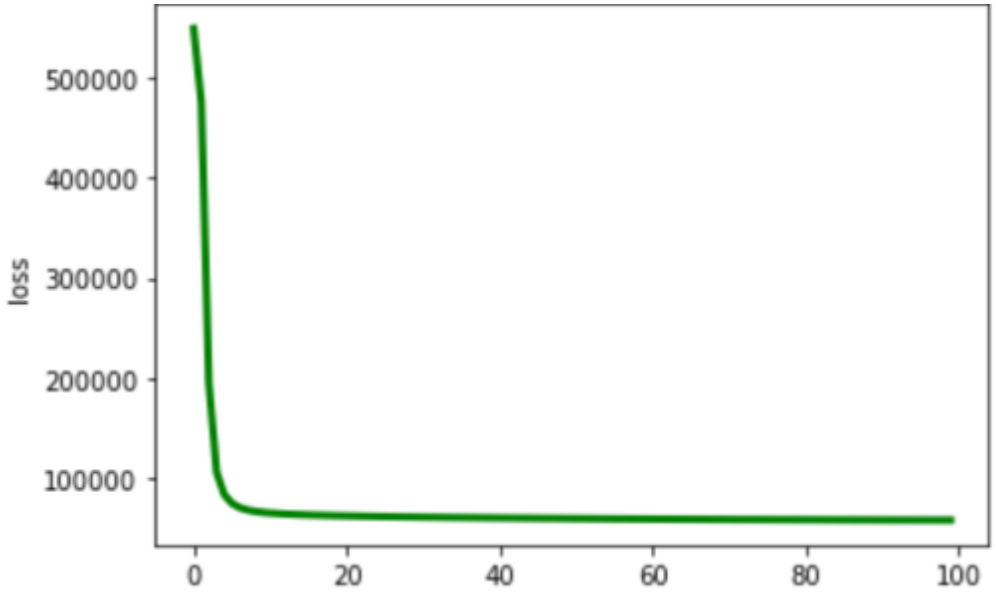

图 8: loss 变化图

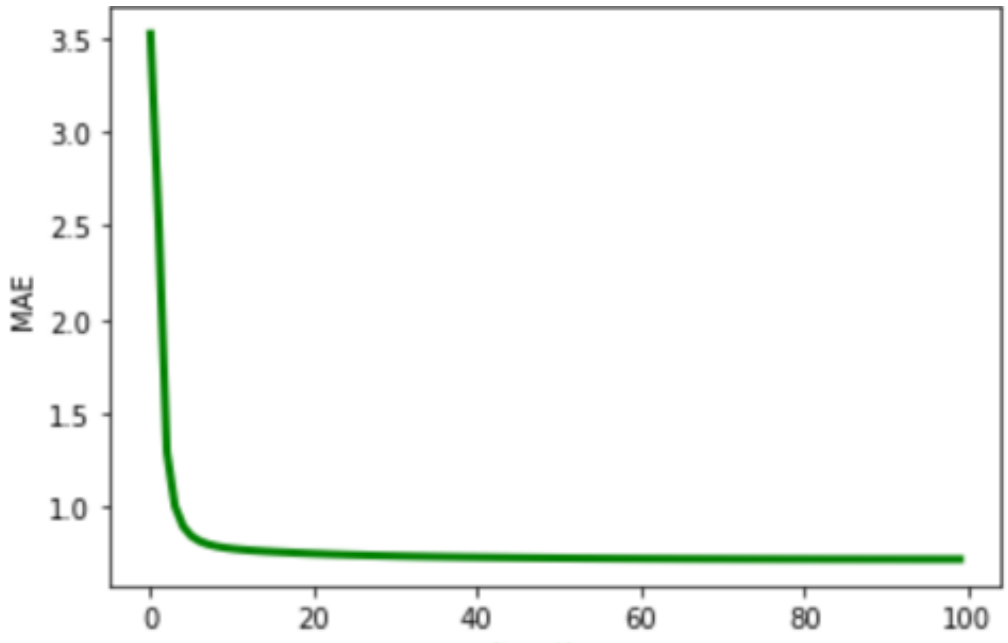

图 9: MAE 变化图

由图 5 可知，loss，MAE 随着迭代次数的增加，明显减少，由图 6 和图 7 可知 loss 和 MAE 在迭代次数 15 前快速下降，在迭代次数大于 20 之后逐渐收敛。

6.2 不同活跃度用户的推荐结果

图 10 为不同活跃程度用户在 step100 的 MAE，LOSS，RMSE,PRECISION 的结果，其中 U1，U2 的训练集和测试集是相邻近的，即训练集和测试集的活跃程度是相同的，UA 和 UB 的训练集和测试集是不相临近的，也就是说 UA 和 UB 的训练集和测试集的活跃程度是不相同的，从下表可知虽然在 UA 和 UB 上进行训练得出的 loss 明显大于在 U1 和 U2 上进行训练，但是 UA 和 UB 数据集上的 MAE 和 RMSE 接近与 U1 和 U2 的 MAE 和 RSME，说明了在不同活跃度上的数据集上进行训练收敛速度小于在相同活跃度的数据集上训练，但是最后模型表现出来的推荐效果并没有产生显著的区别，这也表现出 PMF 模型在解决这种比较稀疏的数据集时能表现出比较好的效果。

活跃度	LOSS	MAE	RMSE	PRECISION
U1	58203.8049	0.7311	0.9241	0.0544
U2	58449.8097	0.7280	0.9237	0.0593
UA	66412.9378	0.7244	0.9169	0.0353
UB	66403.4418	0.7229	0.9153	0.0260

图 10: 不同活跃度用户推荐结果图

6.3 深度学习版本 PMF 结果

在关键参数相同的情况下，如图，深度学习版本相较于非深度学习版本，模型效果一定程度得到了提升，其深度学习的 MAE 和 RMSE 的值优于非深度学习版本。

活跃度	U1	U2	UA	UB
MAE	0.7311	0.7280	0.7244	0.7229
RMSE	0.9241	0.9237	0.9169	0.9153

图 11: 非深度学习版本结果图

活跃度	U1	U2	UA	UB
MAE	0.4744	0.4705	0.4853	0.4723
RMSE	0.6888	0.6859	0.6967	0.6872

图 12: 深度学习版本结果图

7 总结与展望

本次复现实现了论文中的描述的 PMF 方法，并在 ML100K 实验集上进行实验，尤其是分析不同活跃度用户上的推荐效果，其中分析推荐效果的指标通过 MAE，RMSE 进行判断。然后在 pytorch 深度学习平台上建立多层感知机神经网络模型，实现论文中描述的 PMF 方法，并在 ML100K 实验集上进行离线训练实验，与非深度学习版本的效果进行比较和分析。

但是关于论文中描述的，当有效的新数据到来时，可以立即更新模型并对模型进行在线训练以实现在线协同过滤，由于实验条件以及时间等限制暂时只能实现离线训练部分，希望将来还有机会继续完成在线训练部分。

参考文献

[1] Li K, Zhou X, Lin F, et al. Deep probabilistic matrix factorization framework for online collaborative filtering[J]. IEEE Access, 2019, 7: 56117-56128.

[2] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In International Conference on Learning and Intelligent Optimization, pages 507–523, 2011.

[3] Neil Lawrence. Probabilistic non-linear principal component analysis with Gaussian process latent variable models. JMLR, 6:1783–1816, 2005.

[4] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. Selecting classification algorithms with active testing. In International workshop on machine learning and data mining in pattern recognition, pages 117–131, 2012.

[5] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hy-perband: A novel bandit-based approach to hyperparameter optimization. arXiv:1603.06560, 2016.

[6] Yuri Malitsky and Barry O’ Sullivan. Latent features for algorithm selection. In Seventh Annual Symposium on Combinatorial Search, 2014.

[7] Mustafa Mısır and Michèle Sebag. ALORS: An algorithm recommender system. Artificial Intelligence, 244:291–314, 2017.

[8] Matthias Reif, Faisal Shafait, and Andreas Dengel. Meta-learning for evolutionary parameter optimization of classifiers. Machine learning, 87:357-380, 2012.

[9] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In NIPS, pages 2951–2959, 2012.

[10] Christopher KI Williams and Carl Edward Rasmussen. Gaussian processes for machine learning. The MIT Press, 2006.