HW2 个性化推荐系统

本此作业的代码位于 `RecSys.py` 中，运行代码时有以下参数：推荐算法类型`--method`，`cf` 为协同过滤，`lf` 为矩阵分解；`-k`，`-l` 为矩阵分解算法中的超参数 k, λ；`--lr` 为梯度下降过程中的学习率；`--epoch` 为矩阵分解算法训练过程的最大轮数。运行程序的示例命令为：

```
python3 RecSys.py -k 10 -l 0.001 --lr 1e-4 --epoch 1000 --method lf
```

1 数据预处理

在数据预处理阶段，主要完成的任务是将原始的格式化数据整理为维度为 $user \times item$ 的矩阵 X，在本次作业中**使用 Netflix 数据集的全量数据**进行个性化推荐系统的构建，即用户数目及电影数目都为 10000，对训练和测试集通过下面的方式分别构建得到两个 10000×10000 的矩阵 X_{train} 和 X_{test}，其中每行代表一个用户，每列代表一部电影，X_{ij} 表示用户 i 对电影 j 的评分。

观察数据集中的数据特点，电影的 id 为 $[1, 10000]$ 间的整数，而用户的 id 是离散的，所以在构建行为矩阵 (Utility Matrix) 的过程中，首先通过用户列表对用户的 id 进行映射，将离散的用户 id 映射到区间 $[0, 9999]$ 中，对应矩阵的行索引，对于电影的 id 则通过原始 $id - 1$ 的方式映射到区间 $[0, 9999]$ 中，对应矩阵的列索引。

由于在完整的矩阵中已被打分的元素处于少数，所以构建得到的训练和测试矩阵 X 为稀疏矩阵，为了减少非必要的存储，使用按行压缩的稀疏矩阵存储方式 (csr_matrix)，使用 python 中的`scipy.sparse.csr_matrix`实现，对于矩阵中分数未知的项全定为 0。

2 协同过滤

使用协同过滤推荐算法的代码位于`collaborative_filtering()`中。

a) 矩阵形式的协同过滤

在本次作业中实现的是基于用户的协同过滤算法，也就是根据其他相似用户对一部电影的评分，预测目标用户对这部电影的评分。给定一个用户 i 以及一部电影 j，希望预测用户 i 对电影 j 的评分，对应到 Utility Matrix 中也就是 X_{ij} 的值。预测的核心在于如何评价用户之间的相似度，考虑用户对所有电影的打分，如果两个用户对大部分电影的打分是相似的，则认为这两个用户是相似的。记 $X(i)$ 为用户 i 对所有电影的打分，对应矩阵 X 中第 i 行的行向量，$sim(X(i), X(k))$ 表示用户 i 和 k 的相似度。在本次作业中使用余弦相似度进行评价，计算两个用户向量的余弦值：

$$cos(X(i), X(k)) = \frac{X(i) \cdot X(k)}{|X(i)| \cdot |X(k)|}$$

这样得出的用户相似度为 $[-1,1]$ 内的一个实数，1 表示两个用户的打分行为完全一致，越靠近 -1 说明两个用户的偏好相似度越小。余弦相似度使用 `sklearn.metrics.pairwise` 中的 `cosine_similarity()` 计算。该函数可以直接接收矩阵 X 作为输入，对任意两行间计算向量的余弦值，输出尺寸为 $user \times user$ 的矩阵 $user\_sim$，$user\_sim_{ij}$ 为用户 i 与 j 的相似度。

得到用户相似度后，将相似度作为权重，对打分进行加权平均，用于对用户新评分的预测。传统形式的评分预测公式为：

$$score(i,j) = \frac{\sum_k sim(X(i),X(k))score(k,j)}{\sum_k (X(i),X(k))} \tag{1}$$

其中历史评分 $score(k,j) = X_{kj}$，由于训练过程中使用的 X_train 为一个稀疏矩阵，使用 for 循环进行加权平均的计算会效率很慢，公式1的分子可以很容易地对应到余弦相似度向量 $user\_sim(i)$ 与 X_train 中第 j 列向量的内积。在常规的算法中 k 值的含义是 k 个相似度最高的用户参与加权平均，在考虑矩阵计算的情况下，筛选出前所有用户相似度最高的 k 个用户会带来较高的额外成本，所以在每次加权平均计算分数的过程中使用所有的用户数据，即 $k = 10000$。于是便可以得到矩阵形式下的协同过滤推荐算法：

$$score = \frac{SX_{train}}{SX_{bool}} \tag{2}$$

其中 S 为用户相似度矩阵 $user\_sim$，X_{bool} 为指示矩阵，其中 $A_{ij} = 1$ 表示 X_{ij} 值已知，反之则为未知值，由于加权平均仅使用已知的打分数据，所以通过指示矩阵使分母的计算中不含未打分用户的相似度。SX_{train} 的结果为 10000×10000 的矩阵，SX_{bool} 的结果为 10000×1 的矩阵，按广播除法的机制，现将分母中矩阵的值按行进行广播，得到 10000×10000 每行中元素相同的矩阵，再进行**逐元素除法**，最终得到的矩阵 $score$ 中的每个元素的计算对应公式1。在具体的实现过程中使用推导得到的矩阵形式的协同过滤算法2。

b) 协同过滤推荐算法的实现

基于 python 实现上述的推荐算法，矩阵计算使用 `scipy.sparse` 和 `numpy` 中提供的操作进行，算法整体流程如下：

Algorithm 1: Collaborative Filtering

Input: X_train,X_test

user_sim = cosine_similarity(X_train) # 计算用户间相似度

X_pred = user_sim.dot(X_train) # 计算分子

sum = user_sim.dot(X_bool) # 计算分母

pred = X_pred/sum # 得到预测评分

calc_RMSE(pred,X_test) # 计算 RMSE 值

采用均方根误差 (RMSE) 作为最终的指标对预测的结果进行评价，RMSE 的计算公式为：

$$RMSE = \sqrt{\frac{1}{|Test|}(\sum_{<i,j>\in Test}(X_{ij} - \widetilde{X_{ij}})^2)}$$

由于测试集 X_test 仍为一个稀疏矩阵，其中未知值会被处理为 0 影响 RMSE 的计算，所以仍用一个指示矩阵从预测评分的结果中筛选出对应位置的评分，再将二者在测试集覆盖的数据上计算 RMSE。

在 Netflix 数据集上运行该算法得到 RMSE 的值为 **1.0183690394072502**，相比全部猜测打分为 3 的情况下的 RMSE 值 1.1715111196739934 有较大的提升，说明这种基于协同过滤的推荐算法是有效的。但这种算法存在的缺点就是在相似度计算和评分预测的过程中要进行较大量的复杂大矩阵运算，所以算法的运行时间较长，程序运行一次的时间为 **405.1821644306183** 秒，但这种矩阵形式的算法相比 for 循环已经高效很多了。在最初尝试通过找出 k 个最相似用户进行预测的过程中，进行单条评分的预测大概就需要 3 秒左右，这也是协同过滤这种方法一直以来比较明显的问题之一。

3 基于梯度下降的矩阵分解算法

基于梯度下降的矩阵分解推荐算法的代码位于`latent_factor()`中。

a) 基于梯度下降的矩阵分解推荐算法

基于矩阵分解的推荐算法核心在于将行为矩阵分解为两个矩阵的乘积，得到用户及电影在隐空间内的特征矩阵 U, V：

$$X_{m*n} \approx U_{m*k} {V_{n*k}}^{\top}$$

其中 k 为隐空间的维度，可以理解 U, V 为用户和电影在隐空间的特征表达，一个用户对电影的评分由他们在隐空间的相关特征决定，因为对于评分的预测可以使用这两个矩阵的成绩预测矩阵 X 中的未知部分。

为了得到符合训练集特征的矩阵 U 和 V，使用梯度下降的方法使 $UV^{\top}$ 逼近矩阵 X 中训练值的已知部分，梯度下降法优化的目标函数为：

$$J = \frac{1}{2}||A \odot (X - UV^{\top})||_F^2 + \lambda||U||_F^2 + \lambda||V||_F^2$$

目标函数中的第一项表示 U 和 V 的乘积得到的逼近值与实际训练集 X 间的误差，后面两项为正则项，防止在梯度下降的过程中陷入过拟合，将这个目标函数作为训练过程中的`loss`，训练的目标是使 loss 下降，提升预测结果在测试集上的 RMSE 表现。根据以上的目标函数可以求得目标函数对两个矩阵的梯度为：

$$\begin{aligned} \frac{\partial J}{\partial U} &= (A \odot (UV^{\top} - X))V + 2\lambda U \\ \frac{\partial J}{\partial V} &= (A \odot (UV^{\top} - X))^{\top}U + 2\lambda V \end{aligned} \tag{3}$$

梯度下降的部分在 `gd(X_train,U,V,lr)` 中实现，在每一轮迭代的过程中，按公式3计算出优化目标`loss`对矩阵 U 和 V 的梯度，按设定好的学习率 (lr) 对两个矩阵沿梯度的反方向进行更新，然后进行下一轮的迭代。

Algorithm 2: Gradient Descent

`gd(`U, V, lr`)`

Calculate $\frac{\partial loss}{\partial U}, \frac{\partial loss}{\partial V}$

Update $U = U - lr\frac{\partial loss}{\partial U}, V = V - lr\frac{\partial loss}{\partial V}$

foreach *epoch* **do**

loss = loss_func(X,U,V)

U,V = gd(U,V,lr)

X_pred=U.dot(V.T)

calc_RMSE()

end

考虑 U 和 V 中的值应该为用户/电影对一系列隐空间中的特征的相关系数，且选择的 k 个特征应该为比较显著的特征，与特征的相关系数体现为极端值概率低，中间值概率大，考虑可以近似为正态分布进行初始化，所以对初值采用正态随机的方式 `np.random.normal()` 进行初始化，均值为 0，方差与隐空间的维度相关为 $\frac{1}{k}$。

b) 实验参数的选取

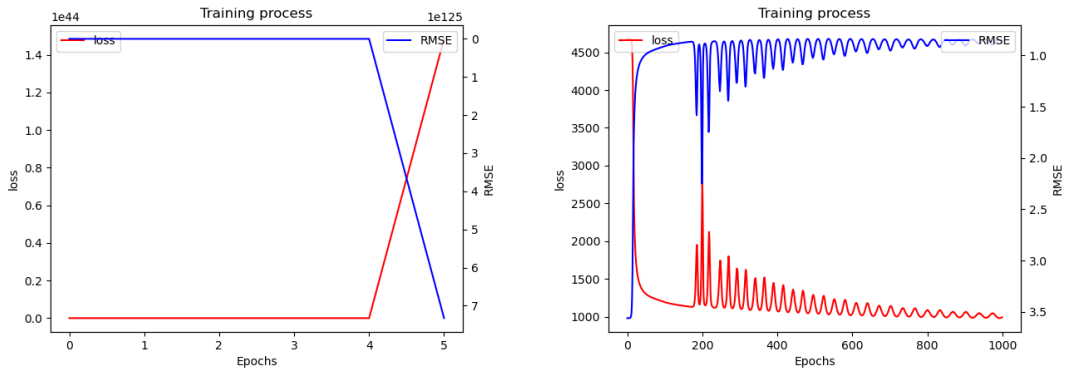
在基于矩阵分解的推荐算法中，梯度下降过程中的参数 k, λ，学习率 lr 为超参数，前两个参数在实验的过程中调整观察效果。对于学习率 lr，影响梯度下降过程中矩阵更新的步长，若学习率过大，可能导致在优化过程中直接跨过最优解，目标函数值震荡不收敛；学习率过小则会导致目标函数值下降过慢，长时间不下降到期望的解附近，所以选择一个合适的学习率对于训练过程很重要。固定 $k = 50, \lambda = 0.01$，训练轮数为 100，尝试不同的学习率，得到效果如下：

Table 1: 不同学习率下的结果

lr	RMSE	epoch
1e-1	不收敛	
1e-2	不收敛	
1e-3	不收敛	
1e-4	**0.910388058680946**	100
1e-5	3.484665851596683	100

可以看到，学习率过大时会出现不收敛的现象，在很少个 epoch 内 loss 值不降反而一直增大，导致模型无法收敛，如图1(a)所示。学习率为 $1e-5$ 时与 $1e-4$ 相比，在相同个 epoch 下收敛的更慢，所以综合考虑模型效果以及训练速度，固定最终的学习率为 $1e-4$。

但在学习率为 $1e-4$ 的训练过程中，由于学习率相对稍大，优化随着接近最优解会发生幅度较大的震荡收敛，如图1(b)所示，其中 RMSE 对应的 y 轴为翻转坐标，为了更直观地能看出 RMSE 效果的提升：



(a) lr=1e-2 (b) lr=1e-4

Figure 1: 不同学习率下的训练曲线

测试集上的 RMSE 会出现增大再下降的过程，对比震荡前的 RMSE 为 0.8608966455385907，训练最终的 RMSE 为 0.8378561099477627。为了更有效地训练与防止过拟合，采用**早停 (early stop)** 机制，在训练过程中，当 RMSE 值小于 0.9(认为达到可接受的局部最优解附近) 且 RMSE 值出现增大时停止训练，经测试这样可以得到较好的效果。

设定学习率后，最终算法的停止条件为两次迭代间 loss 的变化量小于阈值 $1e-3$ 或达到最大迭代次数`num_epochs`时，停止训练。

c) 实验结果

给定 $k = 50, \lambda = 0.01$，迭代过程中的目标函数值以及 RMSE 的变化如图2所示，最终的 RMSE 为 **0.8666673192427726**，

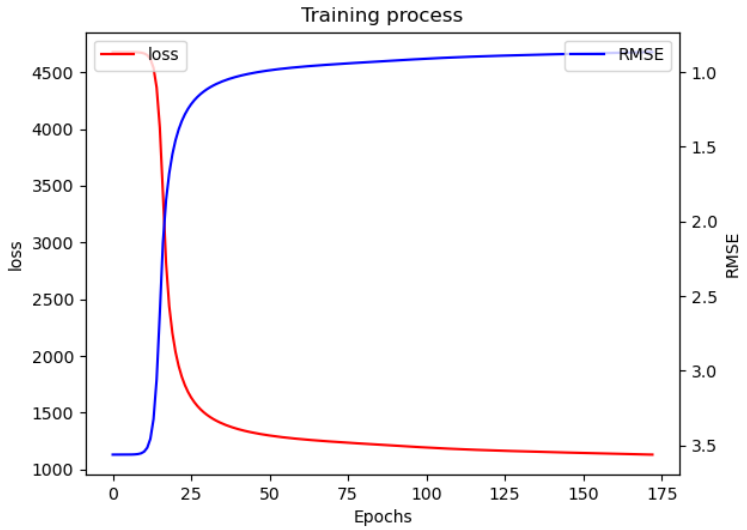


Figure 2: $k = 50, \lambda = 0.01$

可以观察到在训练的过程中，随着梯度下降的进行，目标函数与 RMSE 的值在稳定下降，逐渐收敛，在第 173 个 epoch 结束训练。从最终的结果可以看出，基于矩阵分解的方法得到的结果

相比协同过滤的方法有了很大的提升，且随着 loss 的下降测试集上的 RMSE 表现逐渐提升，未出现明显的过拟合现象，说明正则项和早停机制产生了效果。

调整矩阵分解过程中的超参数 k, λ 的值，训练得到最终的 RMSE 结果如下：

Table 2: 不同参数下的实验结果

k/λ	0.001	0.01	0.1	1
10	0.8690803838971598	0.8616859040996944	0.8654264363299821	0.8666673192427726
50	**0.8608966455385907**	0.8666673192427726	0.8680405246603344	0.8659369378122349
100	0.8701988342132885	0.8697599657504359	0.8687624844659261	0.8699783619491335

对比不同参数组合训练的得到的最终 RMSE 的结果，最优的参数组合为 $k = 50, \lambda = 0.001$，选取值位于中间的隐空间维度数可以有利于包含大部分隐空间中的特征，同时避免了一些对于区分用户及电影作用很小的特征的影响。取较小的 λ 的值在保留正则项防止过拟合的同时，使目标函数的优化主要还是朝着对原始矩阵 X 拟合的方向进行，防止正则项权重过大影响矩阵的拟合效果。

4 两种方法的比较

对比两种不同推荐算法的预测结果：

Table 3: 两种不同方法的效果对比

	RMSE	Execution Time
Collaborative Filtering	1.0183690394072502	405.1821644306183
Latent Factor Model	0.8608966455385907	487.90074586868286

从结果可以明显地看出，在最终的预测效果上基于矩阵分解的方法是明显优于基于协同过滤的方法的，在算法运行时间上矩阵分解的方法略大于协同过滤。

对于协同过滤的方法，其优点在于基于用户与物品间交互行为的相似度进行推荐，无需选取物品的特征，所以可以适配任何种类的物品推荐场景中；同时算法的原理较为简单，由相似用户的历史打分对未知数据进行预测，可解释性好。其缺点在于推荐的过程中需要计算所有用户两两间的相似度，这一过程计算成本很高导致运行较慢，计算量受用户和物品数量增加的影响很大；协同过滤的算法要求找出和用户相似的对物品已打分的用户，传统方法中对于稀疏矩阵这一点是很困难的，对于矩阵形式的算法又只能使用全量的用户数据，无法高效地筛选“topk”用户；对于新用户或新物品的冷启动问题，由于没有相似的历史打分信息，所以无法给出推荐；用户行为的变化比较频繁，所以可能需要经常重新计算用户相似度矩阵，对于系统成本较高。

对于矩阵分解的方法，其优点在运算过程都可以使用高效的矩阵运算，所以训练过程中的每轮计算速度较快；可以更深层次地学到用户和物品的特征，效果要明显好于协同过滤的方法；对于用户行为的变化，可以在之前矩阵分解的基础上继续进行梯度下降微调，对于有数据更新的学习更友好。其缺点在于超参数较多，需要根据实际训练情况进行调整；矩阵分解的效果依赖于隐空间中的 k 个特征，但不知道具体是什么相关特征，可解释性较弱；对于比较复杂的情况，模型的收敛可能需要较多个 epoch，模型初期的训练时间较长。

综上，从整体上考虑基于矩阵分解的算法优于基于协同过滤的方法，对于数据规模较小，且物品特征不明显的场景下可以使用协同过滤的方法；对于数据规模大，需要更高效地计算和更好效果的情况下可以使用基于矩阵分解的方法。