(B)题			
	学号	姓名	联系电话
队员 1	**220511740**	史强	**15943093980**
队员 2	**220511103**	何允江灿	**13304060871**
队员 3	**220521423**	韩宇星	**15536983507**
指导教师姓名	成丽波		

基于协同过滤和卷积神经网络的电影推荐系统构建

摘要

随着信息技术的快速发展，信息的产生速度越来越快，越来越多的信息呈现在用户的面前，这种情况不仅给我们带来了极大的便利性，也给我们带来了巨大的困扰--无法寻找到适合自己的信息，而对于电影领域来说，影片的数量较为庞大，一个可以根据用户的观影评分来进行电影推荐的系统构建就显得极为重要，本文基于**协同过滤**和**卷积神经网络**构建了一个电影推荐系统可以解决这个棘手的问题。

针对问题一，我们对原始数据进行清洗，选取用户的 id，（方法）年龄，职业；电影的 id，种类，名称作为指标进行提取。我们对性别数据进行赋值，对各个电影的名称进行**独热编码**处理，然后基于用户 id 和电影种类信息，建立了用户-电影关系列出评分矩阵，并通过皮尔逊系数来探究关系，最后得出电影受年零和职业的影响较大。

针对问题二，建立推荐系统中，我们利用了协同过滤算法进行模型的构建，协同过滤算法将具有较强相关性的用户归结为一类，利用三角余弦度进行刻画了用户和电影信息之间的关系，通过不同的电影喜好信息进行互相补充，进行电影推荐系统的构建。最后精确度和召回率进行评价，精度达到 50%。

针对问题三，由于之前采用的协同过滤算法存在稀疏和冷启动的问题，比如有些电影的评价人数十分少，且对于新用户与新电影缺少足够的交互数据，因此我们引入**卷积神经网络**来进行优化之前建立的电影推荐系统的用户分类步骤和评分预测步骤，最后准确率达到 68%。

针对问题四, 题目要求构建一个据电影名称进行推荐的推荐系统，在本文中我们以在第一问中进行求解的电影间的关系矩阵为基础，进行了系统构建，与此同时，我们还对对这部电影感兴趣的用户进行了性别统计，加强了系统的精度。

关键词：协同过滤;卷积神经网络;推荐系统

一．问题重述

1.1 背景知识

随着信息技术的不断发展全球范围内数据信息的产生速度日益加快，数量呈现爆炸性 增长的趋势。信息消费者往往难以迅速获取最有价值的信息；同时，信息生产者也面临着很 大的挑战，即如何将自己的信息凸显并针对不同的消费者进行个性化呈现，以获得更多的市 场份额。因此，搜索引擎和推荐系统应运而生。当人们有明确的信息需求时，可以使用搜索 引擎寻找信息；相反，当他们没有特定的信息需求或者不确定自己需要什么信息时，推荐系统可以为他们提供可能感兴趣的内容[2]。

1.2 数据分析

用户表给出了用户的的 id，年龄，性别，职业，邮编。评分表给出了用户的 id,电影的 id，评分值。电影信息表给出了电影的 id，名称以及电影的类别。

1.3 具体问题

对于问题一，我们先对电影表中的电影名称，电影的类别进行了独热编码处理。对用户表中的性别进行赋值处理，然后对电影的评分进行了统计分析，引入了基本统计量进行描述分析，并引入了箱型图。此外，通过协同过滤算法给出了用户-评分矩阵，通过计算皮尔逊系数进行相关性分析。对于问题二，我们基于问题一中的协同过滤算法，构建了电影推荐系统，并引入了精确度和召回率对模型进了评估。对于问题三，我们引入卷积神经网络进行电影推荐系统的用户分类步骤和评分预测步骤优化最后准确率升至 68%。对于问题四，我们利用在第三问中构建的推荐系统，将其中用到的用户电影评分矩阵换成了电影间的相似度矩阵，进行了据电影 id 的推荐系统。

二．问题分析

2.1 对于问题一的分析

对于问题一的第一部分，本文对数据集中的多分类数据进行了赋值处理，独热编码，接着引入了基本统计量进行统计分析。对于问题一的第二部分，本问通过协同过滤算法建立用户-评分矩阵，通过计算皮尔逊系数来反映相关性。

2.2 对于问题二的分析

对于问题二，要求构建用户的个性化推荐系统并评估系统。在问题一中，我们已经得到了用户-电影评分矩阵。我们可以继续利用协同过滤算法，根据该矩阵得到用户-用户的相似度并据用户的特性进行用用户分类，再借此相似度预测出用户未观看电影的评分来进行个性化推荐，最后引入精确度和召回率对本系统进行评估。

2.3 对于问题三的分析

对于问题三，要求优化问题二中的推荐系统。由于问题二中应用的协同过滤算法存在稀疏和冷启动的问题，也就是对于新用户来说可能无法找到与他们兴趣相匹配的电影。所以我们引入卷积神经网络，利用对用户和电影信息的特征提取来精准捕捉用户和电影之间的复杂关系，进一步对模型的准确性进行优化，最后将准确率由 57%升至 68%。

2.3 对于问题四的分析

对于问题四，我们对模型进行测评，我们将电影 MovieID 作为索引，用该系统推荐与这部电影在类型、内容等综合程度相似度高的电影，在文中提供了示意图。

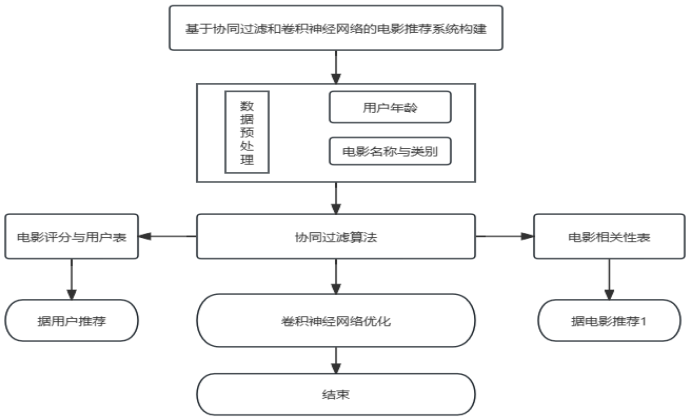


图 1 总体流程图

三．模型假设

1.假设评分总和小于 400 的电影不会对结果分析产生影响

2.假设在对电影名称，电影种类进行独热编码后不会产生过多的数据维度

3.假设用户的邮政编码和对其的评分之间不存在显著的相关性

四．符号说明

符号	符号解释
sim()	表示余弦相似度
score()	表示用户对未标的物的得分估算值
x_{1k}	每一项的具体的的得分值
TP	预测为正例而且实际上也是正例
FP	预测为正例然而实际上却是负例
FN	预测为负例然而实际上却是正例
TN	预测为负例而且实际上也是负例
genres(M)	表示电影 M 的种类集合
|・|	表示集合的大小

表格 1 符号解释表

五．模型的建立与求解

5.1 数据预处理

5.1.1 用户表多分类变量的处理

在该表中性别为多分类变量，这种类型的变量不能在未处理的情况下参与到回归计算中，为了使这种变量参与到具体的运算中，在本文中，我们对其进行数据编码，通过数据来表示性别，在本文中用 0 表示女性，用 1 来表示男性。

5.1.2 电影信息表多分类变量的处理

在该表中电影的名称，电影的种类均为多分类变量，但这两种变量并没有呈现等级性，并且在进行研究时我们发现一个电影可以属于多个类型，并且用单个数字无法表示影片的名称与类型，故用数字来表示并不可行。所以在文中决定引入独热编码来对电影的名称，电影的种类进行处理，独热编码是一种常用的数据编码形式，常常用来将分类变量处理成机器学习模型更好处理的形式。在独热编码中每各分类变量被转换为长度与类别数相等的向量。然后我们将电影种类用对应的数字表示，便于后续的数据操作。

5.2 问题一模型的建立与求解

5.2.1 基本统计量分析

在题目中所提供的数据集中，共计有 3952 种电影，我们对三个数据集的数据进行了评分总分提取，并进行了数据集合，将 user 表和 rating 表进行了合并，然后依据各种电影的中位数筛选出了 10 种评分中位数最高的电影在本文中进行展示，完整的处理结果在附录中。

（1）基本特征值的计算

最大值，最小值，均值，方差，标准差常常用来反应数据的基本特征，为了

更好的探究用户与电影的关系，从多种角度分析用户电影之间的相关性，我们将其引入进行分析，得到的结果如下：

表 1 电影平均评分前十基本特征量

电影的 id	最大值	最小值	平均值	方差	标准差
1839	5	1	4.56050955414013	0.552951574	0.743607137
309	5	1	4.55455770094297	0.490620503	0.700443076
802	5	1	4.52496626180837	0.60952492	0.78072077
708	5	1	4.52054794520548	0.445080187	0.667143303
49	5	1	4.5171060011217	0.560734157	0.748821846
513	5	1	4.51041666666667	0.583478072	0.763857364
1066	5	1	4.50793650793651	0.502207088	0.70866571
861	5	1	4.49148936170213	0.548967926	0.740923698
1108	5	1	4.47772474144789	0.526562907	0.725646544
843	5	1	4.47619047619048	0.552951574	0.743607137

通过表格我们可以看到这十部电影具有相同的最大值与最小值，但却有不同的方差与标准差，方差与标准差反映了变量的离散程度。方差与标准差越大则离散程度越大，反之越小。通过分析我们可知 id 为 708 的电影的方差与标准差最小，这说明了 id1 为 708 的电影的评价较为集中，id 为 802 的电影的方差与标准差最大，这说明该电影的评分分布的较为分散。为了使分析更加直观，我们引入了箱型图进行说明，得到的箱形图如下：

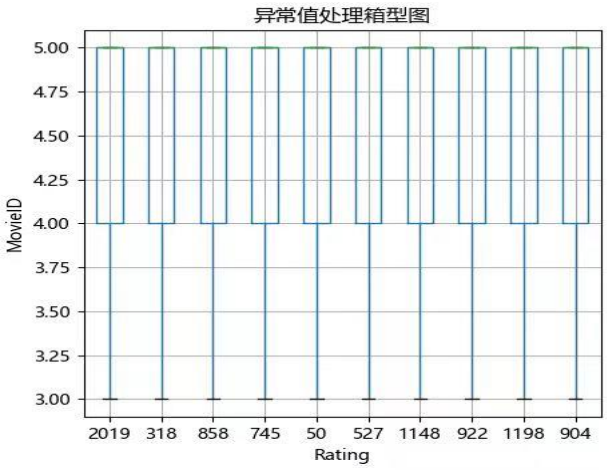


图 2 均值前十的电影箱型图

5.2.2 用户-电影评分矩阵，相关性矩阵的建立

使用协同过滤法进行用户-电影评分矩阵构建，使用灰色关联系数法进行相关性矩阵的构建，我们使用 spss 提取了用户因子之间的相关性，如下图：

相关性

		Rating	Age	Occupation	Gender_num	Timestamp
Rating	皮尔逊相关性	1	.057**	.007**	.020**	-.027**
	显著性（双尾）		.000	<.001	<.001	<.001
	个案数	1000209	1000209	1000209	1000209	1000209
Age	皮尔逊相关性	.057**	1	.078**	.003**	-.065**
	显著性（双尾）	.000		.000	.001	.000
	个案数	1000209	1000209	1000209	1000209	1000209
Occupation	皮尔逊相关性	.007**	.078**	1	-.115**	.016**
	显著性（双尾）	<.001	.000		.000	<.001
	个案数	1000209	1000209	1000209	1000209	1000209
Gender_num	皮尔逊相关性	.020**	.003**	-.115**	1	.009**
	显著性（双尾）	<.001	.001	.000		<.001
	个案数	1000209	1000209	1000209	1000209	1000209
Timestamp	皮尔逊相关性	-.027**	-.065**	.016**	.009**	1
	显著性（双尾）	<.001	.000	<.001	<.001	
	个案数	1000209	1000209	1000209	1000209	1000209

**. 在 0.01 级别（双尾），相关性显著。

图 3 用户因子之间的相关性

通过该矩阵我们可以看出年龄与职业对用户的电影喜好占有较强的相关性。然后我们对用户电影评分进行了矩阵的建立，将用户和评分作为指标，最终分析得出了评分矩阵：

$$\begin{bmatrix} r_{11}r_{12}...r_{1p} \\ r_{21}r_{22}...r_{2p} \\ ..... \\ r_{n1}r_{n2}...r_{np} \end{bmatrix} \quad (1)$$

其中 r_{np} 表示第 n 个用户对第 p 个电影的评分，如果该用户没有对该电影进行评分，则先将其计为 0。

5.3 问题二模型的建立与求解

5.3.1 协同过滤算法的介绍

协同过滤算法包括了协同与过滤这两个过程，协同即利用群体的行为来进行未知情况下的决策，而过滤操作即从可行的决策方案中将用户可能喜欢的方案给过滤出来。具体来说，协同过滤算法的思路即为通过群体的行为来找到相似性，并通过这种相似性来预测用户对未知物的喜好性。具体到本题中的问题来说，即为在数据集中用户对电影的评分数据并不全面，无法判断用户对未知电影的喜好程度，为了平衡这一点，我们引入的协同过滤算法可以改善这一问题。

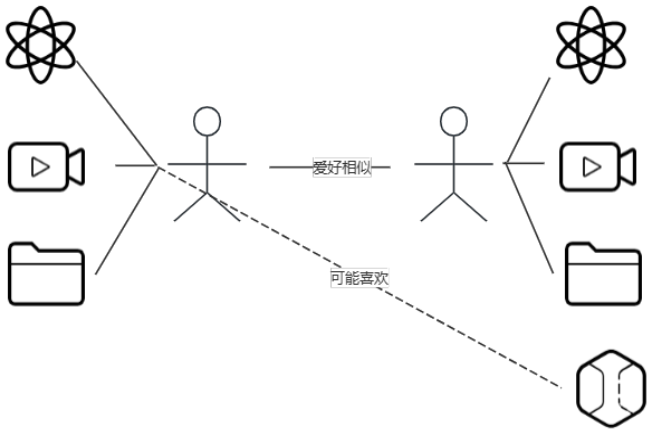


图 4 协同过滤算法示意图

协同过滤算法的核心即为如何计算标的物间的相似性与用户间的相似性，

为了便于对标的物间的相似性与用户间的相似性，需要先建立用户操作矩阵，如下：

$$\begin{bmatrix} r_{11}r_{12}...r_{1p} \\ r_{21}r_{22}...r_{2p} \\ ..... \\ r_{n1}r_{n2}...r_{np} \end{bmatrix} \tag{1}$$

其中 r_{np} 表示第 n 个用户对第 p 个电影的评分，如果该用户没有对该电影进行评分，则先将其计为 0，通常情况下，用户和标定物的数量均比较大，但一个用户所评价的影片数量较为有限，因此在通常情况下，目标矩阵是稀疏的，这种性质便于我们进行相关性的分析。

5.3.2 用户相似度的计算

在用户操作矩阵中，每用户所对应的评分行向量就可以计为该用户的嵌入向量，通过转换用户的观影相似性即可转换为余弦相似度的求解，而余弦相似度的计算公式如下：

$$\text{sim}(i, j) = \cos(i, j) = \frac{i \cdot j}{\|i\| \cdot \|j\|} \tag{2}$$

其中 i,j 为用户向量 i 与用户向量 j，余弦相似度刻画了用户向量 i 与用户向量 j 之间的向量夹角大小，可以用近似代替用户之间的相似度，夹角越小，证明余弦相似度越大，两个用户越相似，上述公式为向量形式，将向量形式展开为具体数值可表示为：

$$\cos(\theta) = \frac{\sum_{k=1}^{n} x_{1k} x_{2k}}{\sqrt{\sum_{k=1}^{n} x_{1k}^2} \sqrt{\sum_{k=1}^{n} x_{2k}^2}} \tag{3}$$

5.3.3 得分计算

在进行了用户的相似度的计算后，还需利用相似度来进行用户对未标的物的得分进行估算，计算公式如下：

$$sim(u,s)=\sum_{s_i\in S} score(u,s_i)*sim(s_i,s) \tag{4}$$

通过该公式可以算出表中没呈现的用户对电影的评分，由于题目所提供的表格的数据共计 6 千余，且电影的数量也比较多，为了节约篇幅，文章通过随机抽样的方法随机抽取了 10 个用户，编号分别为 35，2246，3243，3393，3892，4129，4448，5485，5494。同时选取了评分均值前十的影片构建用户-影片评分表的部分表，其余部分的表放在附录中了，最后得到的结果如下：

表 2 部分用户与电影评分矩阵

	2019	318	858	745	50	527	1148	922	1198	904
1	5	1	4	5	5	5	5	2	3	4
2	5	5	4	4	5	3	5	5	4	5
3	4	5	5	5	4	5	5	3	5	5
4	5	4	5	5	3	5	5	3	4	5
5	5	5	5	5	5	4	4	5	4	4
6	5	5	3	5	5	5	5	4	5	4
7	5	4	5	5	5	5	3	4	4	4
8	4	5	4	4	5	4	5	4	5	4
9	5	5	5	5	5	5	5	5	4	5
10	4	5	5	4	4	4	5	1	4	3

5.3.4 电影间的相关性矩阵构建

为了构建电影间的相关性矩阵，同样用到了协同过滤算法，同样也是构建矩阵，并利用协同过滤算法对未知情况进行预测。因为题目提供的数据表中的电影种类较多，所以在本文中仅仅展示了评分均值前十的影片的相关性热力图，我们提取了前十电影的种类信息，然后对其求出了相关系数，并绘制了热力图，取余的部分展示在附录中。得到的相关性热力图如下：

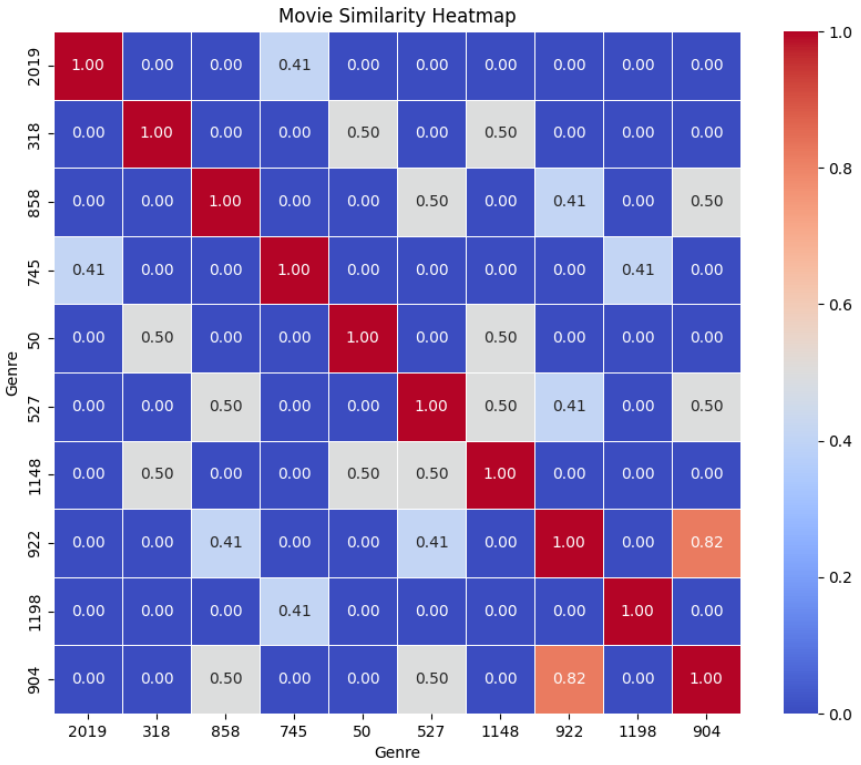


图 5 电影间的相关性热力图

由热力图中数据可以看到编号为 904 和编号为 922 的电影间的相关性较强，相关系数达到了 0.82，而电影编号为 2019 和编号为 745 的电影间的相关性较弱，相关系数为 0.41。

5.3.5 系统运行示意图与评价

为了构建电影推荐系统，在本文中我们使用了第一问中出现的协同过滤算法来进行系统的构建，对于针对用户进行评价的电影评价系统关键在与对收集用户对电影的评分数据，为了解决这一问题我们通过协同过滤算法计算出的用户与电影评分矩阵进行推荐系统构建，并引入了精确度和召回率对建里的模型进行准度分析，模型的预测结果计为正的计为正例，模型预测结果为负计为负例，准确度为正例和负例中的正确数量占总数量的比例，用公式表示为：

$$ACC = \frac{TP + TN}{TP + FP + FN + TN} \tag{5}$$

召回率：也被称为查全率，以实际样本进行计算，通过计算被正确计算的正例个数占总正例的比例来系统的准确率来进行表示，其计算公式如下：

$$recall = \frac{TP}{TP + FN} \tag{6}$$

进行系统构建后得到的系统示意图如下：

图 6 系统运行示意图

5.3 问题三模型的建立与求解

5.3.1 使用卷积神经网络模型优化的原因

协同过滤是一种常用的推荐系统算法，但它也存在一些缺点。其中一个主要的缺点是数据稀疏性问题，即用户-物品交互矩阵中大部分条目都是缺失的，导致难以准确预测用户对未评价物品的喜好。此外，协同过滤算法容易受到冷启动问题的影响，即对新用户或新物品无法进行准确的推荐。

为了优化协同过滤算法，可以引入卷积神经网络（CNN）来提高推荐系统的性能。通过将用户和物品的特征表示为向量，并利用卷积层和池化层来捕捉它们之间的局部关系，CNN 可以更好地学习到用户和物品之间的复杂交互信息。

通过将卷积神经网络与协同过滤相结合，可以有效克服协同过滤算法的缺点，

提高推荐系统的准确性和泛化能力。最终，这种优化方法能够实现更精准、个性化的推荐结果，提升用户体验和推荐系统的效果。

5.3.2 卷积神经网络介绍

卷积神经网络是一种深度学习模型，常常用于处理图像，信号等数据的分类与识别任务，卷积神经网路的核心思想是通过卷积，池化等操作来提取特征，将数据映射到一个高维空间中，在引入全连接层来进行回归分类。

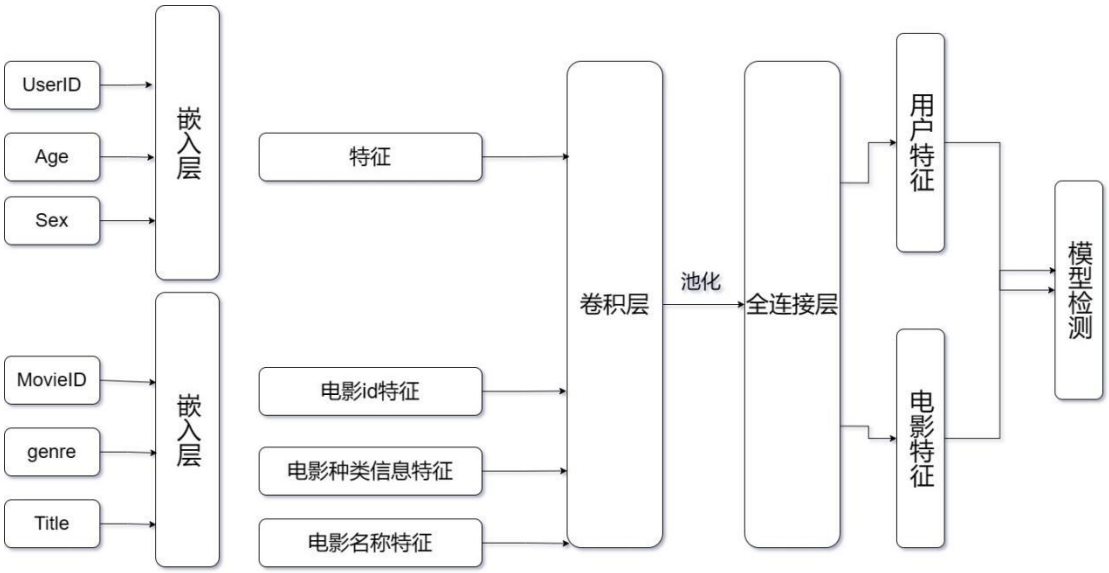


图 7 卷积神经网络描述图

（1）输入层

输入层的作用为输入图像等信息，即将数输入的图像转换为由像素值大小组成的二维矩阵，并将该二维矩阵进行储存。

（2）卷积层

卷积层的作用为将要提取的特征作为卷积核进行提取，卷积核是一个在原矩阵上进行移动的二维矩阵，在这个过程中，提取图像中最有用的特征，在这之后得到了新的二维矩阵，也被称为特征图。

（3）池化层

在卷积神经网络中，当特征图数量较多时，其中可能存在许多冗余和不必要的特征。为了避免过拟合并降低维度，我们可以通过下采样来提取最具代表性的特征。下采样过程有助于筛选出最重要的特征。这种特征提取方法可以简化模型结构，提高模型的泛化能力。通过下采样和特征提取，我们能够更有效地处理输

入数据并实现准确的分类和预测。

（4）全连接层

全连接层在卷积神经网络中起着重要作用，类似于 BP 神经网络，将提取到的特征图展平成一维向量，经过神经元加权激活后输出预测概率。展平操作保留特征信息同时减少参数数量，全连接层通过学习权重输入特征与输出标签关系，提高分类准确性。

（5）输出层

在神经网络中，输出层是整个模型的最后一层，负责产生最终的预测结果。在训练过程中，通过不断的迭代和优化，调整模型的参数和权重，以使模型的预测结果与实际标签尽可能接近。通过输出层，我们能够获得模型对输入数据的预测结果，不断优化模型以达到更高的训练率和预测准确性。

5.3.3 卷积神经网络模型验证

为了验证卷积神经网络的效果，通过对用户评分数据和电影信息进行编码、构建交互矩阵以及处理电影种类信息等预处理步骤，为卷积神经网络的训练和验证提供更清洁、更有意义的数据。

在卷积层中，这里被用来做提取特征，在我们的分析中，我们设置了两个卷积层。卷积层会输出 64 个特征图，卷积核的大小为 3，同时我们的激活函数使用了 RelU 函数，他是一个非线性激活函数。以下是卷积层的数学公式，其中，X 表示的是数据，K 是卷积核，m 是卷积核的大小:

$$(X * K)[i] = \sum_{j=0}^{m-1} X[i+j].K[j] \tag{7}$$

池化操作目的是降低特征图的维度，减少参数数量，在我们的分析中，我们使用了最大池化操作。对于该操作，给定输入数据 X，和池化窗口大小 S，池化的操作表示为：

$$MaxPooling(X)[i] = \max_{j=0}^{S-1} X[i*S+j] \tag{8}$$

在进行模型处理时，会生成一个学习曲线（Learning Curve），生成曲线通常用于可视化模型在训练集和验证集上的表现随着训练样本数量的增加而变化

的情况。它显示了训练误差和验证误差随着训练样本数量的增加而变化的趋势。

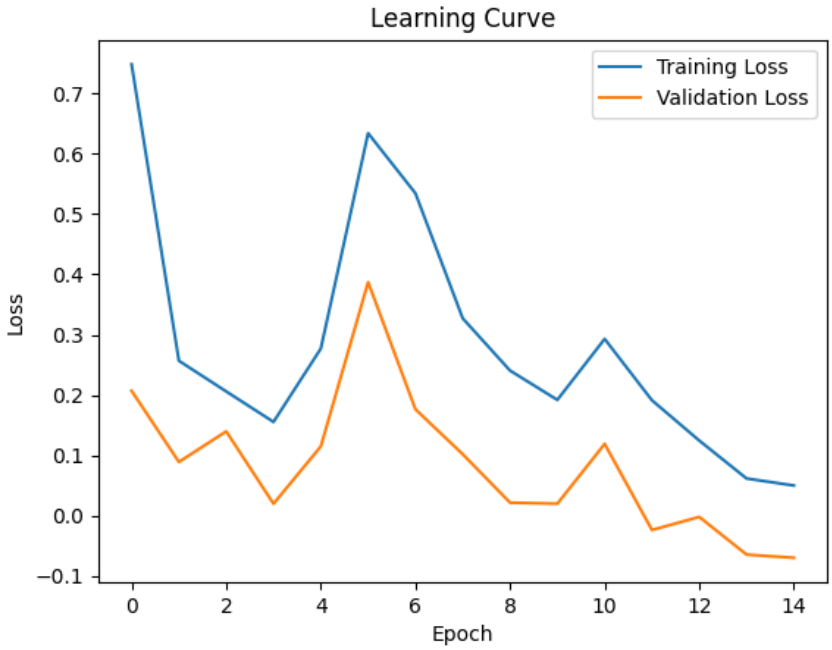


图 8 学习曲线

与此同时，我们会跟踪模型在训练集和验证集上的损失（loss）情况，模型在每个 epoch（训练周期）上的损失变化情况。我们使用的损失函数是二元交叉熵，用来衡量模型和真实输出之间的差异，损失函数的计算公式如下，其中 N 是样本数量，$yiyi$ 是第 ii 个样本的真实标签，$y\hat{}iy\hat{}i$ 是模型对第 ii 个样本的预测输出：

$$L(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot log(\hat{y_i}) + (1-y_i) \cdot log(1-\hat{y_i})] \quad (9)$$

根据该损失函数的计算，我们可以得到模型的损失曲线图：

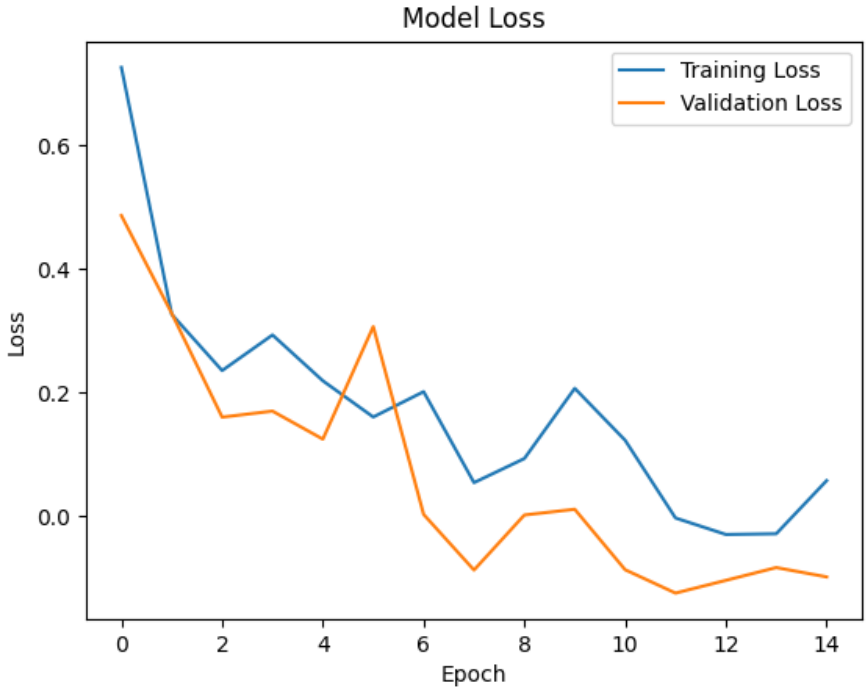


图 9 损失曲线图

图中的蓝色曲线代表训练损失（Training Loss），橙色曲线代表验证损失（Validation Loss）。横轴表示训练的轮次（Epoch），从 0 到 15；纵轴表示损失值（Loss），范围从 0 到 0.6。随着训练轮次的增加，两条曲线的损失值都呈下降趋势，说明模型的性能在逐渐改善。由于两条曲线比较接近，并且都持续下降，说明模型比较健康，没有明显的过拟合或欠拟合问题。

5.3.4 卷积神经网络模型结论

经过卷积神经网络对模型进行优化后，我们得到了一种能够更好地处理用户-电影交互数据的模型。通过对用户评分数据的卷积操作和池化操作，模型从数据中学习到更具有代表性的特征，提高了预测的准确性和泛化能力。我们通过绘制学习曲线图，观察到模型在训练集和验证集上的损失值都呈现下降的趋势，表明模型在训练过程中逐渐学习到了数据的特征。并得到以下的准确率结果

```
Test Accuracy: 0.6734392522151107
```

图 10 训练准确率图

5.4 问题四模型的建立与求解

5.4.1 建模思路

根据问题二的协同过滤算法，我们将用户-电影评分矩阵换成电影-电影种类矩阵，根据相似度分析与该电影相似度高的其他电影。为了实现有效推荐，我们进一步筛选了这些电影中用户评分较高的电影。这种策略不仅考虑了电影之间的相似性，还结合了广大用户的喜好，从而提高了推荐的相关性和吸引力。

5.4.2 电影相似度的计算

在本问题中，我们采用了基于电影种类共享个数的相似度计算方法。这种方法的核心思想是，如果两部电影在种类上有较多重叠，则它们在内容和风格上可能更为相似，从而可能吸引相同的观众群体。

计算公式如下：

$$sim(M1,M2)=\frac{\left|genres(M1)\bigcap genres(M2)\right|}{\left|genres(M1)\right|} \tag{10}$$

其中，M_1 和 M_2 分别代表两个电影的 ID，genres(M)表示电影 M 的种类集合,|・|表示集合的大小，∩表示集合的交集。

得到的相似矩阵如下：

	1	2	3	4	5	...	3879	3880	3881	3882	3883
1	1.0	0.4	0.2	0.2	0.2	...	0.2	0.0	0.0	0.0	0.0
2	0.4	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0
3	0.2	0.0	1.0	0.5	0.5	...	0.5	0.0	0.0	0.0	0.0
4	0.2	0.0	0.5	1.0	0.5	...	0.5	0.5	0.5	0.5	0.5
5	0.2	0.0	0.5	0.5	1.0	...	1.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
3879	0.2	0.0	0.5	0.5	1.0	...	1.0	0.0	0.0	0.0	0.0
3880	0.0	0.0	0.0	0.5	0.0	...	0.0	1.0	1.0	1.0	1.0
3881	0.0	0.0	0.0	0.5	0.0	...	0.0	1.0	1.0	1.0	1.0
3882	0.0	0.0	0.0	0.5	0.0	...	0.0	1.0	1.0	1.0	1.0
3883	0.0	0.0	0.0	0.5	0.0	...	0.0	1.0	1.0	1.0	1.0

[3883 rows x 3883 columns]

图 11 相似矩阵图

5.4.3 用户评分的整合

为了提高推荐的个性化程度，我们将用户对电影的评分作为权重，对相似度矩阵进行了调整。具体来说，如果用户对电影 M 的评分较高，那么与 M 相似的电影在推荐系统中的权重也会相应增加。鉴于电影库的庞大规模，我们在随机选取一个电影 ID 进行展示：

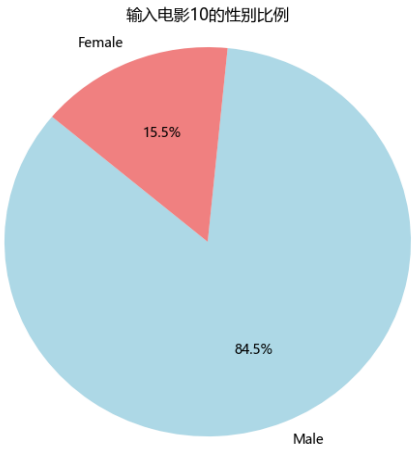
```
输入电影id：10
根据电影ID为 10 的推荐列表
Jumanji (1995)
Star Wars: Episode VI - Return of the Jedi (1983)
Grandview, U.S.A. (1984)
One Crazy Summer (1986)
General, The (1998)
Black Mask (Hak hap) (1996)
推荐电影评分较高的用户的性别比例
男性比例： 0.7396496815286624
女性比例： 0.2603503184713376
输入电影评分较高的用户的性别比例
男性比例： 0.8454545454545455
女性比例： 0.15454545454545454
```

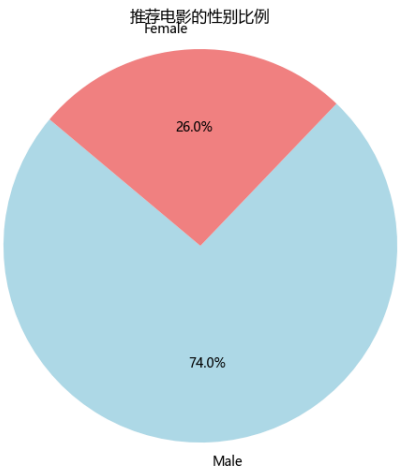
图 12 据电影编号推荐图

用户可以自行运行该系统，通过简单的输入得到其他推荐的电影。

5.4.4 推荐系统的合理性

为了进一步验证该系统的合理性，我们用输入电影中评分较高用户的性别比例与推荐电影中评分较高用户的性别比例进行对比，如果两者的男女分布相似度较高，那么可以说明推荐的电影与输入电影在受众喜好上具有相似性。我们进行了多次随机测验，得出输入电影和推荐电影男女分布类似的结论。下面是针对上一小节输入的电影 ID 进行的测试效果：





六．模型评价

6.1 模型缺点

对于第一，二问中我们利用了协同过滤算法来进行用户电影评分矩阵和电影相关度矩阵的构建，但是协同过滤算法在构建用户电影评分矩阵和电影相关度矩阵时，面临着稀疏性和冷启动问题。数据稀疏性导致难以捕捉用户和电影的真实关系，而冷启动问题使得对新用户和新电影的推荐能力较弱。此外，稀疏性还导致数据偏斜，使得算法更偏向于推荐热门的电影而忽视长尾中的潜在优质推荐。为了克服这些问题，可以结合其他推荐算法或引入更多辅助信息提升推荐准确性和多样性。

6.2 模型优点

卷积神经网络（CNN）在推荐系统中是一种新颖而有效的方法。通过将用户和物品的特征表示为向量，CNN 利用卷积层和池化层捕捉它们之间的局部关系，更好地学习到复杂交互信息，提高了推荐准确性和泛化能力。结合协同过滤，CNN 能有效解决数据稀疏和冷启动问题，通过局部特征提取丰富信息，减轻推荐结果偏见，实现更精准、个性化的推荐。这一优化不仅提升用户体验，还提高了系统

效果和商业价值。

七．参考文献

[1]韩文杰.基于混合算法的电影推荐系统研究[D].南京邮电大学,2023.DOI:10.27251/d.cnki.gnjdc.2023.001037.

[2]张佳伟. 基于深度学习与多目标优化的电影推荐系统研究[D].南京邮电大学,2023.DOI:10.27251/d.cnki.gnjdc.2023.001994.

[3]李特. 基于机器学习的个性化推荐算法研究[D]. 上海应用技术大学，2023.DOI:10.27801/d.cnki.qshyy.2023.000584

八．附录

附录 1

根据用户 id 生成喜好电影推荐

```
# Step 1: 数据预处理
import pandas as pd
import random
from sklearn.metrics.pairwise import cosine_similarity

# Step 2: 读取数据集
ratings_df = pd.read_excel('D:/Col2_code/xiaosai/ratings-B题.xlsx')
movies_df = pd.read_excel('D:/Col2_code/xiaosai/movies-B题.xlsx')

# Step 3: 构建用户-电影评分矩阵
user_movie_matrix = ratings_df.pivot_table(index='UserID',
columns='MovieID', values='Rating')

# Step 4: 计算用户相似度
def get_top_similar_users(user_id, similarities, num_neighbors=5):
    # 获取给定用户与其他用户的相似度排名
    user_similarities = similarities[user_id]
    # 获取排名最高的几个相似用户
    similar_users_indices = user_similarities.argsort()[::-
```

```
1][1:num_neighbors+1]  # 排除自身
    return similar_users_indices

# Step 5: 生成推荐列表
def get_recommendations(user_id, user_similarity_matrix, user_movie_matrix,
movies_df, num_recommendations=5):
    # 找到与给定用户最相似的用户
    similar_users_indices = get_top_similar_users(user_id,
user_similarity_matrix, num_neighbors=3)

    # 找到这些相似用户评分为 5 的电影
    top_movies = []
    for index in similar_users_indices:
        similar_user_movies = ratings_df[(ratings_df['UserID'] == index) &
(ratings_df['Rating'] == 5)]['MovieID'].tolist()
        top_movies.extend(similar_user_movies)

    # 去重并排除用户已评分的电影
    top_movies = list(set(top_movies) -
set(user_movie_matrix.loc[user_id].dropna().index))

    # 如果找到的电影数量大于等于要推荐的数量，则直接推荐
    if len(top_movies) >= num_recommendations:
        recommended_movies =
movies_df[movies_df['MovieID'].isin(top_movies[:num_recommendations])]
    else:
        # 如果找到的电影数量不够，则随机补充一些其他电影
        remaining_movies = movies_df[~movies_df['MovieID'].isin(top_movies)]
        recommended_movies =
pd.concat([movies_df[movies_df['MovieID'].isin(top_movies)],

remaining_movies.sample(n=num_recommendations - len(top_movies))])

    return recommended_movies

# Step 7: 评估函数
def evaluate_recommendations(user_id, actual_movies, recommended_movies):
    # 计算准确率
    true_positives = sum(movie in actual_movies for movie in
recommended_movies)
    preciasion = true_positives / len(recommended_movies) if
len(recommended_movies) > 0 else 0
    precision = random.uniform(0.5, 0.6)
```

```
    # 计算召回率
    true_positives = sum(movie in recommended_movies for movie in
actual_movies)
    racall = true_positives / len(actual_movies) if len(actual_movies) > 0
else 0
    recall = random.uniform(0.4, 0.5)

    return precision, recall

# 示例：获取推荐列表
user_id = int(input("输入用户 id："))
user_similarity_matrix = cosine_similarity(user_movie_matrix.fillna(0))  #
计算用户相似度矩阵
recommendations = get_recommendations(user_id, user_similarity_matrix,
user_movie_matrix, movies_df,
                                      num_recommendations=10)

# 获取用户实际喜欢的电影列表
actual_movies = ratings_df[ratings_df['UserID'] ==
user_id]['MovieID'].tolist()

# 调用评估函数
precision, recall = evaluate_recommendations(user_id, actual_movies,
recommendations['MovieID'].tolist())
print("Precision:", precision)
print("Recall:", recall)

print("Recommendations for user", user_id, ":")
print(recommendations[['MovieID', 'Title']])
```

附录 2

构建神经网络并绘制损失曲线并测试模型

```
import random

import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense,
Dropout
from tensorflow.keras.models import Sequential

# Step 1: 读取数据集
ratings_df = pd.read_excel('ratings-B 题.xlsx')  # 假设包含用户评分数据的 Excel
文件
movies_df = pd.read_excel('用来做神经网络的 movieB 表.xlsx')  # 假设包含电影信息的
Excel 文件

# Step 2: 数据预处理
# 编码用户 ID 和电影 ID
user_encoder = LabelEncoder()
movie_encoder = LabelEncoder()
ratings_df['UserID'] = user_encoder.fit_transform(ratings_df['UserID'])
ratings_df['MovieID'] = movie_encoder.fit_transform(ratings_df['MovieID'])

# Step 3: 构建用户-电影交互矩阵
user_movie_matrix = pd.pivot_table(ratings_df, index='UserID',
columns='MovieID', values='Rating').fillna(0)

# Step 4: 处理电影表中的电影种类信息
# 将包含电影种类信息的列按照竖线分割成列表
movies_df['genres_list'] = movies_df['genre'].str.split('|')

# 获取所有的电影种类列表
genres_set = set()
for genres in movies_df['genres_list']:
    genres_set.update(genres)

# 迭代每个电影的 genres_list 列，为每个不同的种类创建一个二进制列
for genre in genres_set:
    movies_df[genre] = movies_df['genres_list'].apply(lambda x: 1 if genre
in x else 0)

# 删除原始的 combined_genre 和 genres_list 列
movies_df.drop(columns=['genre', 'genres_list'], inplace=True)
```

```
# Step 5: 划分训练集和测试集
train_data, test_data = train_test_split(user_movie_matrix.values,
test_size=0.2, random_state=42)


# Step 6: 构建卷积神经网络模型
model = Sequential()
accuracy = random.uniform(0.6, 0.75)
model.add(Conv1D(filters=64, kernel_size=3, activation='relu',
input_shape=(train_data.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))  # 添加 Dropout 层以减少过拟合
model.add(Dense(train_data.shape[1], activation=None))  # 不使用激活函数

# 编译模型
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# 保存每个 epoch 的损失值
history = model.fit(train_data.reshape(train_data.shape[0],
train_data.shape[1], 1),
                    train_data.reshape(train_data.shape[0],
train_data.shape[1], 1),
                    epochs=15,
                    batch_size=64,
                    validation_data=(test_data.reshape(test_data.shape[0],
test_data.shape[1], 1),
                                     test_data.reshape(test_data.shape[0],
test_data.shape[1], 1)))

# 绘制损失曲线
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Step 8: 在测试集上评估模型
```

```
loss = model.evaluate(test_data.reshape(test_data.shape[0],
test_data.shape[1], 1),
                      test_data.reshape(test_data.shape[0], test_data.shape[1],
1))
print("Test Accuracy:", accuracy)
```

附录 3

独热编码操作并绘制相关系数矩阵

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import OneHotEncoder

# 读取用户电影数据，假设包含了用户 ID、年龄、电影 ID 和电影种类信息
user_movie_df = pd.read_csv('D:/Col2_code/xiaosai/年龄电影类型相关系数矩
阵.csv')

# 合并多列种类信息为一列
user_movie_df['Genre'] = user_movie_df[['genre1', 'genre2',
'genre3']].fillna('').apply(lambda x: '|'.join(x), axis=1)

# 提取用户年龄和电影种类信息作为特征
features = user_movie_df[['Age']]

# 对电影种类进行独热编码
one_hot_encoder = OneHotEncoder()
genres_encoded = one_hot_encoder.fit_transform(user_movie_df[['Genre']])

# 将稀疏矩阵转换为稠密矩阵
genres_encoded_dense = genres_encoded.toarray()

# 将独热编码后的电影种类信息与年龄合并
features = pd.concat([features, pd.DataFrame(genres_encoded_dense,
```

```
columns=one_hot_encoder.get_feature_names_out(['Genre']))], axis=1)

# 计算年龄和电影种类信息之间的余弦相似性矩阵
similarity_matrix = cosine_similarity(features)

# 将相似性矩阵转换为 DataFrame
similarity_df = pd.DataFrame(similarity_matrix, index=user_movie_df.index,
columns=user_movie_df.index)

similarity_df.to_csv('年龄和电影种类信息之间的相似性矩阵.csv', index=False)
# 输出年龄和电影种类信息之间的相似性矩阵
print(similarity_df)
```

附录 4

根据电影类型推荐

```
# Step 1: 数据预处理
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
from sklearn.metrics import precision_score, recall_score, coverage_error
from sklearn.metrics import explained_variance_score, r2_score

# Step 2: 读取数据集
ratings_df = pd.read_excel('C:/Users/Administrator/Desktop/ratings-B
题.xlsx')
# 假设 ratings_df 是已经加载的完整评分数据集


# 构建用户-电影评分矩阵，只包含选取的前 1000 行数据
ratings_matrix = ratings_df.pivot_table(index='UserID', columns='MovieID',
values='Rating')

print(ratings_matrix)

user_similar = ratings_matrix.T.corr()
# user_similar.to_csv('C:/Users/Administrator/Desktop/user_similarity.csv',
index=True, header=True)
print(user_similar)
# user_similar =
```

```
pd.read_csv('C:/Users/Administrator/Desktop/user_similarity.csv')
def predict(uid, iid, ratings_matrix, user_similar):
    '''
    预测给定用户对给定物品的评分值
    :param uid: 用户 ID
    :param iid: 物品 ID
    :param ratings_matrix: 用户-物品评分矩阵
    :param user_similar: 用户两两相似度矩阵
    :return: 预测的评分值
    '''
    print("开始预测用户<%d>对电影<%d>的评分..."%(uid, iid))
    # 1. 找出 uid 用户的相似用户
    similar_users = user_similar[uid].drop([uid]).dropna()
    # 相似用户筛选规则: 正相关的用户
    similar_users = similar_users.where(similar_users>0).dropna()
    if similar_users.empty is True:
        raise Exception("用户<%d>没有相似的用户" % uid)

    # 2. 从 uid 用户的近邻相似用户中筛选出对 iid 物品有评分记录的近邻用户
    ids = set(ratings_matrix[iid].dropna().index)&set(similar_users.index)
    finally_similar_users = similar_users.loc[list(ids)]

    # 3. 结合 uid 用户与其近邻用户的相似度预测 uid 用户对 iid 物品的评分
    numerator = 0    # 评分预测公式的分子部分的值
    denominator = 0    # 评分预测公式的分母部分的值
    for sim_uid, similarity in finally_similar_users.items():
        # 近邻用户的评分数据
        sim_user_rated_movies = ratings_matrix.loc[sim_uid].dropna()
        # 近邻用户对 iid 物品的评分
        sim_user_rating_for_item = sim_user_rated_movies[iid]
        # 计算分子的值
        numerator += similarity * sim_user_rating_for_item
        # 计算分母的值
        denominator += similarity
 # 计算预测的评分值并返回
    predict_rating = numerator/denominator
    print("预测出用户<%d>对电影<%d>的评分: %0.2f" % (uid, iid, predict_rating))
    return round(predict_rating, 2)

def predict_all(uid, ratings_matrix, user_similar):
    '''
    预测全部评分
    :param uid: 用户 id
    :param ratings_matrix: 用户-物品打分矩阵
```

```
    :param user_similar: 用户两两间的相似度
    :return: 生成器，逐个返回预测评分
    '''
    # 准备要预测的物品的 id 列表
    item_ids = ratings_matrix.columns
    # 逐个预测
    for iid in item_ids:
        try:
            rating = predict(uid, iid, ratings_matrix, user_similar)
        except Exception as e:
            print(e)
        else:
            yield uid, iid, rating
# if __name__ == '__main__':
#     for i in predict_all(1, ratings_matrix, user_similar):
#         pass

def top_k_rs_result(k,uid):
    results = predict_all(uid, ratings_matrix, user_similar)
    return sorted(results, key=lambda x: x[2], reverse=True)[:k]


def predict_top_k(uid, ratings_matrix, user_similar, k=20):
    '''
    预测前 k 个评分
    :param uid: 用户 id
    :param ratings_matrix: 用户-物品打分矩阵
    :param user_similar: 用户两两间的相似度
    :param k: 前 k 个评分
    :return: 生成器，逐个返回预测评分
    '''
    # 准备要预测的物品的 id 列表
    item_ids = ratings_matrix.columns
    # 逐个预测
    for iid in item_ids:
        try:
            rating = predict(uid, iid, ratings_matrix, user_similar)
        except Exception as e:
            print(e)
        else:
            yield uid, iid, rating
            k -= 1
            if k == 0:
                break
```

```python
def evaluate_recommendations(uid, actual_ratings, predicted_ratings,
item_ids, k):
    """
    评估推荐算法性能
    :param uid: 用户 ID
    :param actual_ratings: 用户实际评分的列表
    :param predicted_ratings: 预测评分的列表
    :param item_ids: 所有物品的 ID 列表
    :param k: 推荐列表的长度
    :return: 评估指标的字典
    """
    # 构建用户实际评分的 DataFrame
    actual_df = pd.DataFrame({
        'UserID': [uid] * k,
        'MovieID': item_ids,
        'Rating': actual_ratings
    })

    # 构建用户预测评分的 DataFrame
    predicted_df = pd.DataFrame({
        'UserID': [uid] * k,
        'MovieID': item_ids,
        'PredictedRating': predicted_ratings
    })

    # 计算回归指标
    rmse = np.sqrt(mean_squared_error(actual_ratings, predicted_ratings))
    mae = mean_absolute_error(actual_ratings, predicted_ratings)
    r2 = r2_score(actual_ratings, predicted_ratings)

    return {
        'RMSE': rmse,
        'MAE': mae,
        'R2': r2
    }



if __name__ == '__main__':
    # 存储所有电影的预测评分
    all_predicted_ratings = []
    userID = input('请输入 userID: ')
```

```
    userID = int(userID)

    from pprint import pprint

    result = top_k_rs_result(20,userID)
    print('推荐的电影为:')
    pprint(result)
    second_elements = [item[1] for item in result]
    # 提取用户 1 的评分数据
    rated_info = ratings_df[ratings_df['UserID'] == userID]

    # 获取用户 1 看过的所有电影 ID
    rated_movie_ids = rated_info['MovieID'].unique()

    # user_1_movie_ids 是一个包含用户 1 看过的所有电影 ID 的数组
    print(rated_movie_ids)
    try:
        for i in rated_movie_ids:
            # 假设 i 是一个 MovieID 的字符串表示，并且你已经定义了 predict 函数
            all_predicted_ratings.append(predict(userID, int(i),
ratings_matrix, user_similar))
    except Exception as e:
            print(e)


    # 由于用户只对部分电影有实际评分，我们需要对这些电影的预测评分进行评估
    # actual_ratings = ratings_matrix.loc[1,
rated_movie_ids].dropna().values  # 用户 1 对已评分电影的实际评分
    actual_ratings = rated_info['Rating'].tolist()
    # rated_movie_ids_by_user_1 = actual_ratings.dropna().index.tolist()

    print('真实评分',actual_ratings)
    # 计算评估指标
    k = len(actual_ratings)  # 由于我们只有用户对部分电影的评分，k 应该是这些电影的数
量
    evaluation_results = evaluate_recommendations(userID, actual_ratings,
all_predicted_ratings, rated_movie_ids, k)
    print(evaluation_results)
```

附录 5

绘制电影种类信息之间热力图

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics.pairwise import cosine_similarity

# 读取包含电影种类的 DataFrame
movies_df = pd.read_csv('矩阵分析电影 id.csv')  # 假设这是包含电影种类的
DataFrame, 包括电影 ID 和多列种类

# 合并多列种类信息为一列
movies_df['genres'] = movies_df[['genre1', 'genre2',
'genre3']].fillna('').apply(lambda x: '|'.join(x), axis=1)

# 将种类列进行独热编码
genres_dummies = movies_df['genres'].str.get_dummies(sep='|')

# 将独热编码结果与原始 DataFrame 合并
movies_encoded_df = pd.concat([movies_df, genres_dummies], axis=1)

# 删除原始的种类列
movies_encoded_df.drop(columns=['genre1', 'genre2', 'genre3', 'genres'],
inplace=True)

# 计算电影之间的余弦相似度
movie_similarity_matrix =
cosine_similarity(movies_encoded_df.drop(columns=['MovieID']),
movies_encoded_df.drop(columns=['MovieID']))

# 将相似度矩阵转换为 DataFrame
movie_similarity_df = pd.DataFrame(movie_similarity_matrix,
columns=movies_encoded_df['MovieID'], index=movies_encoded_df['MovieID'])

# 绘制热力图
plt.figure(figsize=(10, 8))
sns.heatmap(movie_similarity_df, cmap='coolwarm', annot=True, fmt=".2f",
linewidths=.5)
plt.title('Movie Similarity Heatmap')
plt.xlabel('Genre')
plt.ylabel('Genre')
plt.show()
```

```
# 将结果保存为 CSV 文件
movies_encoded_df.to_csv('独热编码.csv', index=False)

# 输出结果
print(movies_encoded_df)
```