《数据科学与工程算法》项目报告

报告题目：使用矩阵分解进行推荐

姓　　名：杨茜雅

学　　号：10215501435

完成日期：2023.7.5

摘要 [中文]：

本论文介绍了一种基于矩阵分解的推荐系统模型，旨在通过学习用户和电影的隐向量表示来预测用户对电影分。该模型利用了大量的用户-电影数据，通过将用户评分矩阵分解为两个低维矩阵的乘积，得到了用户和电影的隐向量表示。通过这种方式，推荐系统能够学习到用户和电影之间的潜在关系，从而实现个性化的推荐。

在模型训练过程中，使用梯度下降和随机梯度下降等优化算法来最小化预测评分与实际评分之间的差异，并引入正则化项来防止过拟合。此外，在训练过程中对训练集进行随机打乱，以提高模型的泛化能力。

为了评估模型的性能，采用了损失函数和其他评估指标，如均方根误差（RMSE）和平均绝对误差（MAE）。通过在验证集上进行评估，证明了该基于矩阵分解的推荐系统模型在准确性和稳定性方面的优势。

实验结果表明，该模型能够有效地捕捉用户和电影之间的潜在关系，并能够提供个性化的推荐结果。此外，该模型还具有较好的扩展性和适用性，可以适用于各种规模和类型的推荐系统。

综上所述，基于矩阵分解的推荐系统模型为推荐领域的研究和实践带来了新的思路和方法。未来可以进一步探索更加高效和准确的矩阵分解算法，以及更多的评估指标来评估推荐系统的性能。该模型在实际应用中具有广阔的发展前景，并有望为用户提供更加个性化和优质的推荐服务。

Abstract [English]

This thesis presents a matrix decomposition-based recommendation system model that aims to predict user ratings of movies by learning the implicit vector representations of users and movies. The model utilises a large amount of user-movie data and obtains the implicit vector representation of users and movies by decomposing the user rating matrix into the product of two low-dimensional matrices. In this way, the recommendation system is able to learn the potential relationship between the user and the movie, thus enabling personalised recommendations.

During model training, optimisation algorithms such as gradient descent and stochastic gradient descent are used to minimise the difference between predicted and actual ratings, and regularisation terms are introduced to prevent overfitting. In addition, the training set was randomly scrambled during the training process to improve the generalisation ability of the model.

To evaluate the performance of the model, a loss function and other evaluation metrics such as root mean square error (RMSE) and mean absolute error (MAE) were used. The advantages of this matrix decomposition-based recommender system model in terms of accuracy and stability were demonstrated by evaluating it on the validation set.

The experimental results show that the model is effective in capturing the potential relationship between users and movies, and is able to provide personalised recommendation results. In addition, the model has good scalability and applicability, and can be applied to various sizes and types of recommender systems.

In summary, the matrix decomposition-based recommendation system model brings new ideas

and methods to the research and practice in the field of recommendation. More efficient and accurate matrix decomposition algorithms, as well as more evaluation metrics to assess the performance of recommendation systems, can be further explored in the future. The model has a broad development prospect in practical applications and is expected to provide users with more personalised and quality recommendation services.

一、 项目概述

科学价值：

基于矩阵分解技术的推荐系统在包含用户项目评分数据的情况下，能够通过对数据进行降维和模式发现，提供个性化的推荐结果。这种推荐系统具有以下科学价值：

• 个性化推荐：矩阵分解技术能够挖掘用户和项目之间的潜在关系，将数据转化为低维表示，从而捕捉到用户的偏好和项目的特征。通过分析大规模的评分数据，推荐系统可以准确地预测用户对未评分项目的喜好，为用户提供个性化的推荐结果。

• 数据挖掘与模式发现：矩阵分解技术能够对评分数据进行降维和特征提取，从而揭示数据中的隐藏模式和规律。通过对用户项目评分数据的分析，可以了解用户群体的行为偏好、项目的特征和相关性等信息，进而提供更准确的推荐结果。

• 高效的推荐算法：矩阵分解技术具有良好的计算性能和可扩展性，适用于处理大规模的评分数据。它能够通过并行计算和分布式处理等方法，高效地进行模型训练和推荐结果生成，满足实时推荐系统对快速响应的需求。

• 推荐系统优化与改进：矩阵分解技术为推荐系统的优化和改进提供了理论基础和方法支持。通过不断优化矩阵分解算法，结合其他推荐技术如协同过滤、内容过滤等，可以提高推荐系统的准确性、覆盖率和多样性，进一步提升用户体验和满意度。

• 实践应用与商业化潜力：基于矩阵分解技术的推荐系统已经在众多领域得到广泛应用，如电子商务、社交媒体、音乐和视频推荐等。这些应用不仅提供了个性化的用户体验，还为企业带来了商业价值。推荐系统的科学研究和技术创新可以进一步推动相关领域的发展，并为商业化应用提供支持。

综上所述，基于矩阵分解技术的推荐系统在挖掘用户项目评分数据中的潜在信息、提供个性化推荐、优化推荐算法以及应用于实践中具有重要的科学价值。它不仅推动了推荐系统的研究和发展，还为用户提供了更好的使用体验和选择，帮助他们发现感兴趣的项目和内容。此外，矩阵分解技术还为企业提供了商业化的机会，通过精准的推荐能够增加销售额、提高用户留存率和客户满意度，从而带来商业上的成功和竞争优势。

在科学研究方面，基于矩阵分解的推荐系统还涉及到推荐算法的改进和优化，包括降低计算复杂度、提高推荐准确性、解决数据稀疏性等问题。这些研究不仅推动了推荐系统领域的进步，还对机器学习、数据挖掘和人工智能等领域的发展产生了积极的影响。

此外，基于矩阵分解技术的推荐系统还具有社会和经济影响。它可以促进信息的流动和共享，推动用户之间的互动和交流。同时，它也为企业提供了更好的商业模式和营销策略，能够更精确地理解用户需求和行为，为用户提供个性化的服务和推荐，进而提高用户忠诚度和购买意愿。

总的来说，基于矩阵分解技术的推荐系统在科学研究、商业应用和社会影响方面都具有重要的价值。它不仅提供了个性化的推荐服务，满足用户的需求，还为企业带来商业利益。同时，它也为推动相关领域的研究和技术创新做出了贡献，促进了社会的信息共享和交流。

相关研究工作：

构建基于矩阵分解技术的推荐系统涉及一系列相关研究工作，以下是其中的一些主要方面：

• 数据预处理：在构建推荐系统之前，需要对原始的用户项目评分数据进行预处理。这包括处理缺失值、处理异常值、数据规范化和标准化等操作。此外，还需要进行数据的分割，将数据集划分为训练集和测试集，以评估推荐系统的性能。

• 矩阵分解模型选择：选择合适的矩阵分解模型是构建推荐系统的关键。常见的矩阵分解模型包括基于奇异值分解（SVD）的模型、潜在语义索引（LSI）模型、非负矩阵分解（NMF）模型等。不同的模型具有不同的假设和优化算法，研究工作涉及对不同模型的性能比较和选择。

• 模型训练与参数调优：在选择矩阵分解模型后，需要对模型进行训练和参数调优。这涉及到优化算法的选择，如梯度下降法、交替最小二乘法等，以及超参数的选择和调整。研究工作可以探索不同的训练策略和参数调优方法，以提高模型的准确性和泛化能力。

• 评估指标定义与评估方法：为了评估推荐系统的性能，需要定义适当的评估指标和评估方法。常用的评估指标包括准确率、召回率、覆盖率和多样性等。研究工作可以通过比较不同的评估指标和方法，提出更合理和全面的评估体系，以评估推荐系统的质量和效果。

• 处理数据稀疏性：在用户项目评分数据中，通常存在数据稀疏性的问题，即大部分用户-项目组合都没有评分记录。研究工作可以探索如何有效地处理数据稀疏性，以提高推荐系统的预测能力。例如，可以引入正则化项、使用隐式反馈数据、结合内容信息等方法来解决数据稀疏性问题。

• 预测算法扩展与改进：除了基本的矩阵分解模型，研究工作还可以探索如何扩展和改进预测算法。例如，可以引入上下文信息、社交网络信息或时间序列信息等来增强推荐系统的性能。此外，可以探索如何结合其他推荐技术，如协同过滤、内容过滤等，与矩阵分解技术进行融合，以提高推荐系统的准确性和多样性。

• 用户群体建模与分析：研究工作可以针对不同的用户群体进行建模和分析，以提供更精准的个性化推荐。可以利用聚类分析、分类模型等方法，将用户划分为不同的群体，并针对每个群体设计针对性的推荐策略。

• 实时推荐与系统架构：在构建基于矩阵分解的推荐系统时，还需要考虑实时推荐和系统架构的设计。研究工作可以探索如何有效地处理实时的推荐请求，如基于流数据的推荐算法、分布式计算和存储等技术，以提高推荐系统的实时性和可扩展性。

• 用户反馈和个性化解释：为了提升用户对推荐结果的满意度，研究工作可以考虑用户反馈和个性化解释的问题。可以通过用户反馈数据来不断优化推荐结果，并提供解释和理由，使用户更好地理解和接受推荐结果。

• 隐私保护和公平性：在构建基于用户项目评分数据的推荐系统时，隐私保护和公平性也是重要的研究方向。研究工作可以探索如何在推荐过程中保护用户的隐私，如差分隐私技术的应用。同时，还可以关注推荐结果的公平性，避免对某些用户或项目存在偏见。

综上所述，构建基于矩阵分解技术的推荐系统涉及多个方面的研究工作，包括数据预处理、模型选择与训练、评估指标定义与评估方法、数据稀疏性处理、预测算法扩展与改进、用户群体建模与分析、实时推荐与系统架构、用户反馈和

个性化解释，以及隐私保护和公平性等。这些工作的开展将有助于提升推荐系统的性能和用户体验，推动推荐系统领域的进一步发展。

二、 问题定义

给定一个用户评分矩阵 R∈$R^{m\times n}$，其中 m 为用户（user）的数量，n 为物品（item）的数量。矩阵元素γ_{ij}∈R 表示用户u_i为物品v_j的评分值。任务目标有两个：

- 通过矩阵分解和凸优化技术，获得每个用户u_i和物品v_j的隐式向量，分别记作u_i∈R^k和v_i∈R^k，其中 k 为向量维度；所有用户和物品分别组成矩阵 P∈$R^{m\times k}$和 Q∈$R^{n\times k}$；
- 根据获得的用户和物品向量，预测该用户对某个物品的偏好程度$\hat{r}_{ij}=u_iv_i^T$；

因为在实际应用中，这个用户评分矩阵式稀疏的。例如某电影网站一共有 100 k 用户和 10k 部电影，有一些用户可能只看不超过 10 部电影，或者有些的电影可能被观影的人数很少。换句话说，这个用户评分矩阵存在大量的缺失值，因此通过矩阵分解可以先挖掘出已存在的用户行为，再预测这些缺失值。

实验步骤：

- 数据预处理:收集用户-物品评分数据，并将其转换为评分矩阵。矩阵分解:用 MF 或者 NMF 将评分矩阵分解为两个低维矩阵，即用户矩阵和物品矩阵
- 模型训练:通过优化目标函数来训练模型。优化目标通常包括最小化预测评分与实际评分之间的差异，以及控制矩阵维度和正则化等
- 预测评分:通过对用户矩阵和物品矩阵进行乘法运算，得到预测评分矩阵。这个预测评分矩阵可以用于推荐系统的个性化推荐。
- 评估模型:使用评估指标(例如 RMSE、MAE 等)来评估模型的准确性和性能。
- 应用推荐:使用预测评分矩阵为每个用户推荐与他们最有可能喜欢的物品。

三、 方法

数据集介绍：

选择包含用户项目评分数据的电影评分数据集，包括约 20k 用户和 10k 部电影，总共有超过 240k 评分数据。根据评分数据划分了训练集（237k 评分数据）、验证集（10k 评分数据）和测试集（50k 评分数据）。

训练集、验证集样例，每一条数据表示为(u_i，v_j，r_{ij})三元组

	userId	movieId	rating
2	16696	35	4
3	7297	719	2
4	17618	2439	3
5	9522	44709	4.5
6	11085	4886	3.5
7	19444	377	4
8	1319	156	4

测试集样例，用于评测，其中只有用户和商品（u_i，v_j），我们需要预测出他们的评分

ID	userId	movieId
0	7314	553
1	18570	1485
2	10484	191
3	1841	2269
4	1234	1073
5	11514	5391
6	15364	539
7	6870	329
8	16730	2616

变量表：

1、lr：学习率，用于控制模型参数的更新步长。
2、batch_size：批量大小，用于指定每次训练中使用的样本数量。
3、reg_p：P 矩阵的正则化系数，用于控制 P 矩阵的正则化项的影响。
4、reg_q：Q 矩阵的正则化系数，用于控制 Q 矩阵的正则化项的影响。
5、hidden_size：隐向量的维度，表示用户和物品的隐向量的长度。
6、epoch：最大迭代次数，表示训练过程中数据将被遍历的次数。
7、columns：数据集的列名，包括用户 ID、物品 ID 和评分等列的名称。
8、metric：评估指标，用于衡量模型在开发集上的性能。
9、loss_history：记录每一步训练的损失值的列表。
10、metric_history：记录每一步训练的评估指标值的列表。
11、train_dataset：训练集数据，以 DataFrame 形式存储。
12、dev_dataset：开发集数据，以 DataFrame 形式存储。
13、users_ratings：按用户分组的训练集数据，包含每个用户对应的物品和评分。
14、items_ratings：按物品分组的训练集数据，包含每个物品对应的用户和评分。
15、globalMean：所有评分的均值，用于在用户或物品不在训练集中时作为默认评分。
16、P：用户隐向量矩阵，以字典形式存储，键为用户 ID，值为对应的隐向量。
17、Q：物品隐向量矩阵，以字典形式存储，键为物品 ID，值为对应的隐向量。
18、best_metric_result：最好的评估指标结果，用于记录当前最佳的模型性能。
19、best_P：最佳的用户隐向量矩阵。
20、best_Q：最佳的物品隐向量矩阵。
21、starttime：训练开始时间。
22、endtime：训练结束时间。
23、test_data：测试集数据，以 DataFrame 形式存储。
24、save_results：保存预测结果的列表。
这些变量用于存储和处理数据，控制模型训练和预测过程中的参数和状态，以及

记录训练过程中的损失和评估指标。

实验过程

1、 读取包含训练集、验证集和测试集的 CSV 文件，并将其存储在适当的数据结构中，以便后续的数据处理和分析。

首先，Read_Dataset 类接收 label1、label2 和 scores 作为参数，通过调用 return_dataset 函数读取数据集，并将训练集、验证集和测试集保存在实例变量中。然后，get_set 方法根据传入的参数 train 返回对应的数据集。return_dataset 是一个辅助函数，它通过 pandas 库的 read_csv 函数读取训练集、验证集和测试集的数据，并指定列的名称和数据类型。

```python
import pandas as pd
import numpy as np
class Read_Dataset():

    def __init__(self,label1,label2,scores):
        # 初始化函数，接收三个参数：label1、label2、scores
        # 通过调用 return_dataset 函数读取数据集，并将训练集、验证集和测试集保存在实例变量中
        self.train_dataset,self.dev_dataset,self.test_dataset = return_dataset(label1,label2,scores)

    def get_set(self,train):
        # 根据传入的参数 train，返回对应的数据集（训练集和验证集或测试集）
        if train == "train":
            return self.train_dataset,self.dev_dataset
        elif train == "test":
            return self.test_dataset

def return_dataset(label1,label2,scores):
    # 辅助函数，用于读取数据集并返回训练集、验证集和测试集
    dtype = [(label1, np.int32), (label2, np.int32), (scores, np.float32)]
    # 定义一个 dtype，指定各列的名称和数据类型
    print("正在读取数据......")
    print("正在读取训练集数据......")
    train_dataset = pd.read_csv("data/train.csv", usecols=range(3), dtype=dict(dtype))
    # 使用 pandas 库的 read_csv 函数读取训练集数据，仅使用前三列，并指定数据类型为 dtype
    print("正在读取验证集数据......")
    dev_dataset = pd.read_csv("data/dev.csv", usecols=range(3), dtype=dict(dtype))
    # 使用 pandas 库的 read_csv 函数读取验证集数据，仅使用前三列，并指定数据类型为 dtype
    print("正在读取测试集数据......")
    dtype = [("userId", np.int32), ("movieId", np.int32)]
    test_dataset = pd.read_csv("data/test.csv", usecols=range(1,3), dtype=dict(dtype))
    # 使用 pandas 库的 read_csv 函数读取测试集数据，仅使用第二列和第三列，并指定数据类型为 dtype
    print("数据载入完毕")
    return train_dataset,dev_dataset,test_dataset
    # 返回读取的训练集、验证集和测试集数据
```

对于读入的数据，其记录了某个用户对于某部电影的评分。

依据此数据分组可以分别获取到一位用户其评价了的电影以及对应的评分，一场电影评价了它的所有用户以及对应的所有评分，在用户中只有 19694 个用户对于至少一部电影进行的评分，电影中只有 10330 部电影有至少一位用户对其进行了评分，说明对于以上的用户和电影中并没有每一位用户都对于所以电影的电影进行过评分，而我们需要做的就是将上述的缺失的评分矩阵（19694*10330）进行预测补全。

2、load_dataset 方法

加载训练和验证数据集，并计算一些统计信息，例如用户对物品的评分均值。

```python
def load_dataset(self,train_dataset,dev_dataset):
    self.train_dataset = pd.DataFrame(train_dataset)
    self.dev_dataset = pd.DataFrame(dev_dataset)
    # for i in range (10):
    #     print("iter_test:")
    #     print(self.train_dataset[i*10:(i+1)*10])

    self.users_ratings = train_dataset.groupby(self.columns[0]).agg([list])[[self.columns[1], self.columns[2]]]
    self.items_ratings = train_dataset.groupby(self.columns[1]).agg([list])[[self.columns[0], self.columns[2]]]

    ## 一位用户其评价了的电影以及对应的评分
    ## 一场电影评价了它的所有用户以及对应的所有评分

    ## 所有评分的均值
    self.globalMean = self.train_dataset[self.columns[2]].mean()
```

3、__init__方法，模型初始化

创建一个 MatrixRecSysModel 对象，在初始化方法中，我们设置了模型的各种超参数，例如学习率（lr）、批处理大小（batch_size）、P 矩阵的正则化系数（reg_p）、Q 矩阵的正则化系数（reg_q）、隐向量维度（hidden_size）、最大迭代次数（epoch）、数据集的列名（columns）和评估指标（metric）。同时，我们还定义了一些实例变量来存储训练过程中的数据和结果，例如训练集（train_dataset）、开发集（dev_dataset）、用户评分数据（users_ratings）、物品评分数据（items_ratings）、全局均值评分（globalMean）、用户隐向量矩阵（P）、物品隐向量矩阵（Q）、损失历史（loss_history）和评估指标历史（metric_history）。

```python
def __init__(self,lr,
             batch_size,
             reg_p,
             reg_q,
             hidden_size=10,
             epoch=10,
             columns=["uid", "iid", "rating"],
             metric=None,):
    self.lr = lr # 学习率
    self.batch_size = batch_size
    self.reg_p = reg_p     # P矩阵正则系数
    self.reg_q = reg_q     # Q矩阵正则系数
    self.hidden_size = hidden_size  # 隐向量维度
    self.epoch = epoch     # 最大迭代次数
    self.columns = columns # 数据集的列名
    self.metric = metric # 评估指标
    self.loss_history = [] # 记录每一步的损失
    self.metric_history = []# 记录每一步的评估指标
```

4、init_matrix 方法

利用上一步分组得到的数据生成用户矩阵 P(19694 x hidden_size),电影矩阵 Q(10330 x hidden_size)，矩阵的初始化可以有多种方法，例如正态分布，全 0 矩阵，全 1 矩阵，本次实验中采用矩阵归一化生成，即矩阵满足正态分布。

```
def init_matrix(self):

    P = dict(zip(
        self.users_ratings.index,
        np.random.randn(len(self.users_ratings), self.hidden_size).astype(np.float32)
    ))
    # Item-LF
    Q = dict(zip(
        self.items_ratings.index,
        np.random.randn(len(self.items_ratings), self.hidden_size).astype(np.float32)
    ))
    return P, Q
```

5、loss 方法

对于初始化的矩阵将其带入迭代训练，将估计的矩阵与真实值的均方误差作为损失函数来进行目标函数优化。

```
def loss(self,ground_truth, prediction):
    truth = np.array(ground_truth)
    pred = np.array(prediction)
    tmp = truth-pred
    loss = tmp.dot(tmp.T)
    return loss
```

6、loss_norm 方法

对于损失函数，加入正则项(本项目中采用 L2 正则项)，以对参数进行约束

损失函数 Loss = $E_{ij}^2 = \left(\gamma_{ij} - \sum_{k=1}^{k} p_{ik} q_{kj}\right)^2$

加入正则项 Loss = $E_{ij}^2 = \left(\gamma_{ij} - \sum_{k=1}^{k} p_{ik} q_{kj}\right)^2 + \frac{\beta}{2} \sum_{k=1}^{k} \left(p_{ik}^2 + q_{kj}^2\right)$

```
def loss_norm(self,ground_truth, prediction, P, Q, train_step_dataset):
    truth = np.array(ground_truth)
    pred = np.array(prediction)
    tmp = truth-pred
    loss = tmp.dot(tmp.T)
    ## 正则化项
    bias = 0
    for uid, iid, real_rating in train_step_dataset.itertuples(index=False):
        if uid not in self.users_ratings.index or iid not in self.items_ratings.index:
            bias += 0
            continue
        p_u = P[uid]
        q_i = Q[iid]
        bias += np.dot(p_u,p_u.T)
        bias += np.dot(q_i,q_i.T)
    loss += (self.reg_p+self.reg_q)/2*bias
    return loss
```

7、bgd 方法

基于上述损失函数的表达式，得到损失函数的梯度

$$\nabla_{p_{ik}} = -2\left(r_{ij} - \sum_{k=1}^{k} p_{ik}q_{kj}\right)q_{kj} + \beta p_{ik}$$

$$\nabla_{q_{kj}} = -2\left(r_{ij} - \sum_{k=1}^{k} p_{ik}q_{kj}\right)p_{ik} + \beta q_{kj}$$

利用上市得到首先得到批量梯度矩阵下降的算法

使用批量梯度下降（BGD）算法进行训练。它按批次选择样本进行训练，并更新 P 和 Q 矩阵。

```python
def bgd(self, P, Q, batch_size):
    '''
    **********************************
    基本分：请实现【批量梯度下降】优化
    加分项：进一步优化如下
    - 考虑偏置项
    - 考虑正则化
    - 考虑协同过滤
    **********************************
    '''
    num_train = len(self.train_dataset)
    iterations_per_epoch = max(num_train // batch_size, 1)
    for i in range(iterations_per_epoch):
        ## 按照顺序取出batch_size个样本对P,Q进行优化
        # train_step_dataset = self.train_dataset[i*batch_size:(i+1)*batch_size]
        train_step_dataset = self.train_dataset.sample(batch_size,replace=False)
        # print("train_step_dataset: ",train_step_dataset)
        train_loss = 0.
        prediction, ground_truth = list(), list()
        for uid, iid, real_rating in train_step_dataset.itertuples(index=False):
            prediction_rating = self.predict_user_item_rating(uid, iid, P, Q)
            # dev_loss += abs(prediction_rating - real_rating)
            prediction.append(prediction_rating)
            ground_truth.append(real_rating)

        ## 计算这batch_size样本的损失
        # print("prediction: ",prediction)
        # print("ground_truth: ",ground_truth)

        train_loss = self.loss_norm(ground_truth,prediction,P,Q,train_step_dataset)
        print("batch_size: {}/{}, loss : {}".format((i+1)*batch_size, num_train, train_loss))
        # print("batch_size: {}/{}".format((i+1)*batch_size, num_train))
        self.loss_history.append(train_loss)

        ## 利用batch_size样本进行梯度更新
        P,Q = self.grad_norm(P, Q, train_step_dataset)
    # print(train_loss)
    return P, Q
```

8、grad_norm 方法

计算梯度更新时的正则化项。根据训练数据集中的每个用户-物品对，计算 P 和 Q 矩阵的梯度。

```
def grad_norm(self, P, Q, train_step_dataset):
    P_tmp = P
    Q_tmp = Q
    for uid, iid, real_rating in train_step_dataset.itertuples(index=False):
        if uid not in self.users_ratings.index or iid not in self.items_ratings.index:
            continue
        p_u = P[uid]
        q_i = Q[iid]
        p_u_tmp = P_tmp[uid]
        q_i_tmp = Q_tmp[iid]
        r_ui = np.dot(p_u,q_i.T)
        dp_u = self.reg_p*p_u-2*(real_rating-r_ui)*q_i
        dq_i = self.reg_q*q_i-2*(real_rating-r_ui)*p_u
    # return dp_u,dq_i
        P_tmp[uid] = p_u_tmp-self.reg_p*dp_u
        Q_tmp[iid] = q_i_tmp-self.reg_q*dq_i
    P = P_tmp
    Q = Q_tmp
    return P, Q
```

类似地，实现随机梯度下降的算法批量梯度下降

9、sgd 方法

- 批量梯度下降：
  - 在更新参数的过程中使用所有的样本
- 随机梯度下降：
  - 在更新参数的过程中仅选取一个样本

使用随机梯度下降（SGD）算法进行训练。它随机选择一个样本进行训练，并更新 P 和 Q 矩阵。

```
def sgd(self, P, Q):
    '''
    ********************************
    基本分：请实现【批量梯度下降】优化
    加分项：进一步优化如下
    - 考虑偏置项
    - 考虑正则化
    - 考虑协同过滤
    ********************************
    '''
    num_train = len(self.train_dataset)
    # mask = np.random.randint(0,num_train-1)
    # # print("mask",mask)
    # # print("self.train_dataset[0]",self.train_dataset[mask:mask+1])
    # # exit(0)
    # train_step_dataset = self.train_dataset[mask:mask+1]
    train_step_dataset = self.train_dataset.sample(1)
    # print("self.train_dataset\n",train_step_dataset)
    # exit(0)
    train_loss = 0.
    prediction, ground_truth = list(), list()
    for uid, iid, real_rating in train_step_dataset.itertuples(index=False):
        prediction_rating = self.predict_user_item_rating(uid, iid, P, Q)
            # dev_loss += abs(prediction_rating - real_rating)
        prediction.append(prediction_rating)
        ground_truth.append(real_rating)
    train_loss = self.loss_norm(ground_truth,prediction,P,Q,train_step_dataset)
    # print("batch_size: {}/{}, loss : {}".format((i+1)*batch_size, num_train, train_loss))
    # print("batch_size: {}/{}".format((i+1)*batch_size, num_train))
    self.loss_history.append(train_loss)
    ## 利用batch_size样本进行梯度更新
    P,Q = self.grad_norm(P, Q, train_step_dataset)
    return P, Q
```

10、train 方法

在训练数据集上训练模型。根据指定的优化器类型（SGD 或 BGD），使用梯度下降算法更新 P 和 Q 矩阵。

```
def train(self,optimizer_type: str):
    starttime = datetime.datetime.now()
    P,Q = self.init_matrix()
    best_metric_result = None
    best_P, best_Q = P, Q
    print("touch train")

    for i in range(self.epoch):
        print("Epoch time : ",i)
        if optimizer_type == "SGD":
            P,Q = self.sgd(P,Q)
        elif optimizer_type == "BGD":
            print("BGD")
            P,Q = self.bgd(P,Q,self.batch_size)
        else:
            raise NotImplementedError("Please choose one of SGD and BGD.")

        print("epoch:")
        print("epoch: {}/{} ,loss: {}".format(i,self.epoch,self.loss_history[-1]))
        metric_result = self.eval(P,Q)
        print("Current dev metric result: {}".format(metric_result))
        self.metric_history.append(metric_result)
        # print("Current dev loss result: {}".format(loss_test))
        if best_metric_result is None or metric_result <= best_metric_result:
            best_metric_result = metric_result
            best_P, best_Q = P, Q
            print("Best dev metric result: {}".format(best_metric_result))
    endtime = datetime.datetime.now()
    print("training durtion: ",(endtime-starttime).seconds)
    x_value = [i for i in range(self.epoch)]
    y_value = self.metric_history
    plt.plot(x_value,y_value)
    plt.show()

    np.savez("best_pq.npz", P=best_P, Q=best_Q)
    return
```

11、predict_user_item_rating 方法

预测用户对物品的评分。给定用户 ID、物品 ID 以及 P 和 Q 矩阵，它计算用户和物品的特征向量的点积，并返回预测的评分值。

```
def predict_user_item_rating(self, uid, iid, P, Q):
    # 如果uid或iid不在，我们使用全剧平均分作为预测结果返回
    if uid not in self.users_ratings.index or iid not in self.items_ratings.index:
        return self.globalMean

    p_u = P[uid]
    q_i = Q[iid]

    return np.dot(p_u, q_i)
```

12、eval 方法

评估模型在验证集上的性能。它根据当前的 P 和 Q 矩阵，计算预测评分与真实评分之间的差异，并返回评估指标的值。

```
def eval(self, P, Q):
    # 根据当前的P和Q，在dev上进行验证，挑选最好的P和Q向量
    dev_loss = 0.
    prediction, ground_truth = list(), list()
    for uid, iid, real_rating in self.dev_dataset.itertuples(index=False):
        prediction_rating = self.predict_user_item_rating(uid, iid, P, Q)
        # dev_loss += abs(prediction_rating - real_rating)
        prediction.append(prediction_rating)
        ground_truth.append(real_rating)

    # metric_result = self.loss(ground_truth, prediction)
    # dev_loss = self.loss_norm(ground_truth, prediction, P, Q, self.dev_dataset)
    metric_result = self.metric(ground_truth, prediction)
    # dev_loss = self.loss(ground_truth, prediction)
    return metric_result
```

13、test 方法

在测试集上进行评分预测。它加载训练好的 P 和 Q 矩阵，并对每个用户-物品对进行评分预测，并将结果保存到 CSV 文件中。

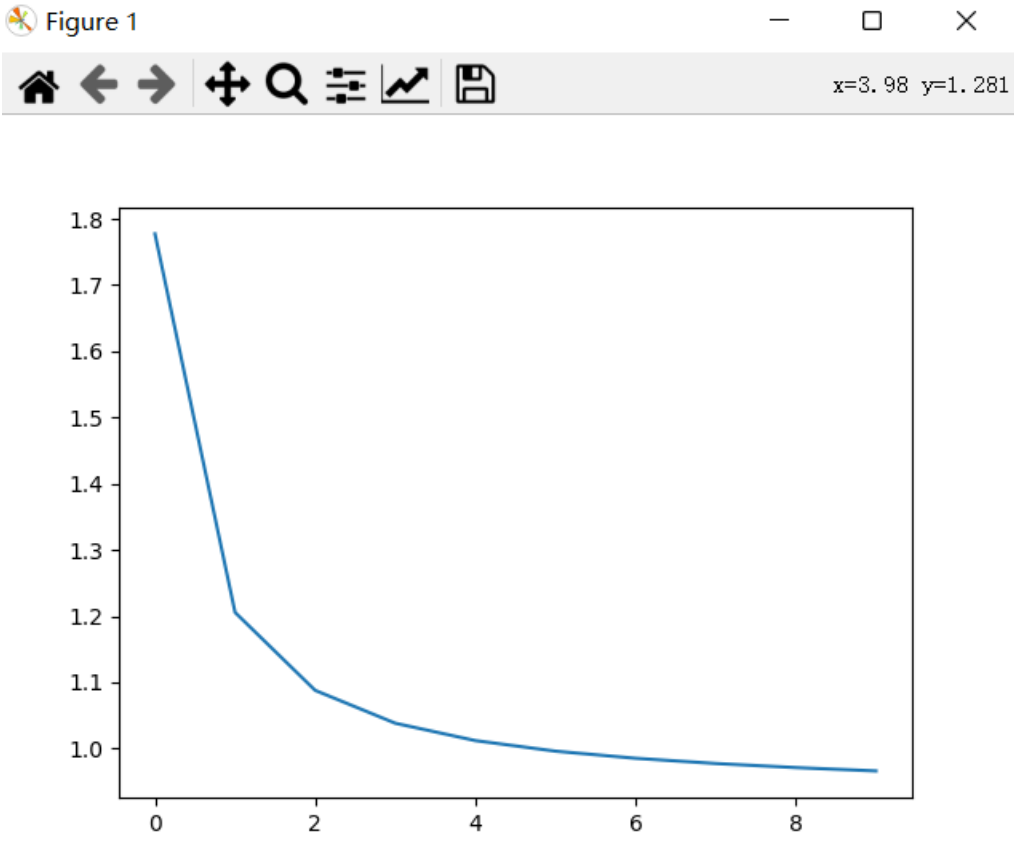
```
def test(self, test_data):
    test_data = pd.DataFrame(test_data)
    # print(test_data[:10])
    # exit(0)
    # 加载训练好的P和Q
    best_pq = np.load("best_pq.npz", allow_pickle=True)
    P, Q = best_pq["P"][()], best_pq["Q"][()]

    save_results = list()
    for uid, iid in test_data.itertuples(index=False):
        pred_rating = self.predict_user_item_rating(uid, iid, P, Q)
        save_results.append(pred_rating)

    log_path = "submit_results.csv"
    if os.path.exists(log_path):
        os.remove(log_path)
    file = open(log_path, 'a+', encoding='utf-8', newline='')
    csv_writer = csv.writer(file)
    csv_writer.writerow([f'ID', 'rating'])
    for ei, rating in enumerate(save_results):
        csv_writer.writerow([ei, rating])
    file.close()
```

四、 实验结果

1、 首先验证批量梯度下降（BGD）的训练结果以及训练时常记录（秒）

```
batch_size: 2376512/2376558, loss : 21.214930609962824
batch_size: 2376520/2376558, loss : 43.42950998590624
batch_size: 2376528/2376558, loss : 24.665821756335795
batch_size: 2376536/2376558, loss : 23.593764338052168
batch_size: 2376544/2376558, loss : 15.855130243029977
batch_size: 2376552/2376558, loss : 24.65249166492226
epoch:
epoch: 9/10 ,loss: 24.65249166492226
Current dev metric result: 0.9661021561428896
Best dev metric result: 0.9661021561428896
training durtion:  12178
Finish training.
```

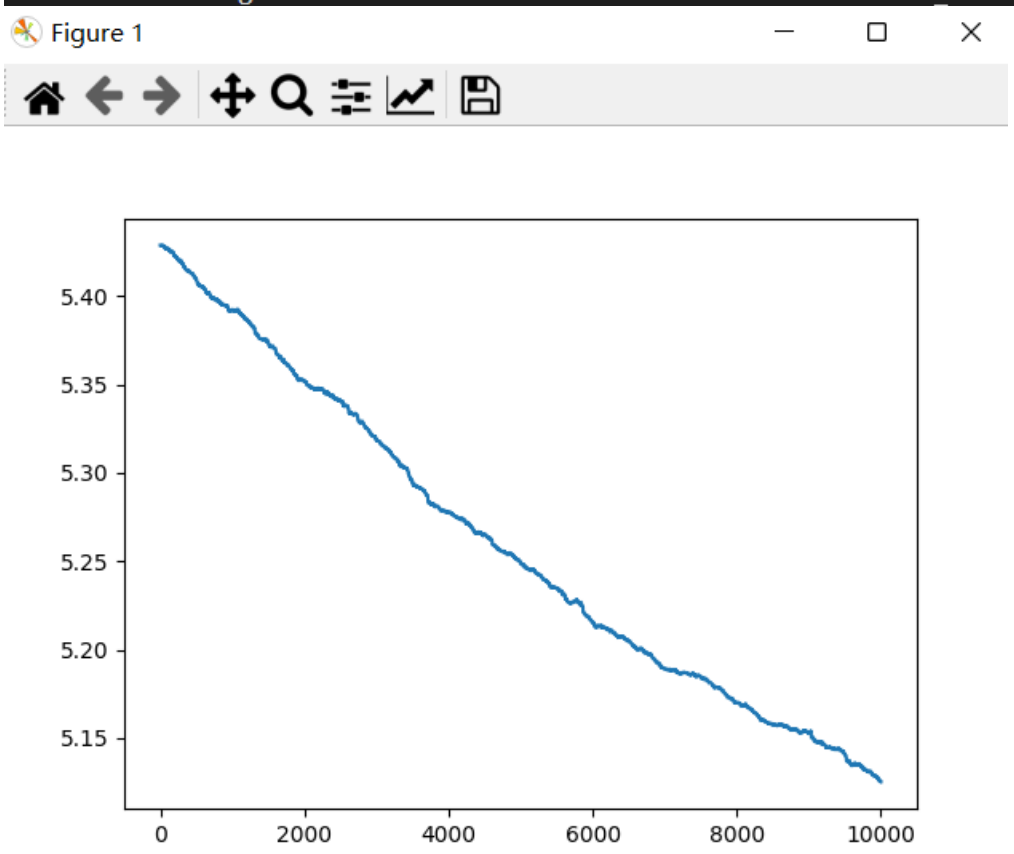
2、 修改结果输出代码可视化在迭代过程中预测结果 RMSE 的变化（备注：因为 BGD 训练用时太长了所以减少了迭代的次数为 10 次）



3、 再次更改参数训练随机梯度下降（SGD），因为在随机梯度下降中每次只是取

出一个样本对矩阵 P、Q 进行梯度更新，因此加大迭代次数使得模型有更高的预测结果

```
Epoch time :  9998
epoch:
epoch: 9998/10000 ,loss: 20.05163037339139
Current dev metric result: 5.125120507693309
Best dev metric result: 5.125120507693309
Epoch time :  9999
epoch:
epoch: 9999/10000 ,loss: 2.223840497198771
Current dev metric result: 5.125110088097629
Best dev metric result: 5.125110088097629
training durtion:  2547
Finish training.
```



五、 结论

推荐系统是信息过滤系统的子集，系统任务便是预测用户接下来更可能想要查看的内容，也就是将特定的信息过滤出来提交给用户。

在本次的实验过程中我们利用矩阵分解的方法利用当前已有的用户对于电影的评分补全其他电影对应的评分从而进一步实现推荐推荐系统 本模型中将预测结果与实际结果的均方误差作为优化目标的损失函数，在优化的过程中分别采用了批量梯度下降方法和随机梯度下降方法进行模型的训练，对比两种不同的优化算法可以看出：

• 对于批量梯度下降：在每次迭代更新参数的过程中需要使用所有的样本，历遍所有的训练样本使得每次的迭代用时相较于 SGD 要高出很多，对于大样本量的训练集来说训练的用时太高，但是因为每次训练考虑的都是全部的样本，所

以在迭代的过程中函数可以在较小的迭代次数内收敛

• 对于批量梯度下降：在每次迭代更新参数的过程中只需要随机抽取一个样本，少量的样本采用使得每次的迭代用时相较于 BGD 要少很多，但是对于大样本量的训练集来说，应为每次训练只从中采取一个样本，使得每次迭代后损失值的下降不够明显，所以在迭代的过程中函数需要多次迭代来达到收敛

• 结合上述两种梯度下降的优缺点，我们可以采取另一种梯度下降的方式 min-batch GD，每次迭代过程中随机从训练集中选取 batch_size 数量的样本用于更新矩阵 P，Q，下面附上代码

```
def mbgd(self, P, Q, batch_size):

    train_step_dataset = self.train_dataset.sample(batch_size)
        # print("train_step_dataset: ",train_step_dataset)
train_loss = 0.
prediction, ground_truth = list(), list()
for uid, iid, real_rating in train_step_dataset.itertuples(index=False):
    prediction_rating = self.predict_user_item_rating(uid, iid, P, Q)
    # dev_loss += abs(prediction_rating - real_rating)
    prediction.append(prediction_rating)
    ground_truth.append(real_rating)

## 计算这batch_size样本的损失
# print("prediction: ",prediction)
# print("ground_truth: ",ground_truth)

train_loss = self.loss_norm(ground_truth,prediction,P,Q,train_step_dataset)
# print("batch_size: {}/{}, loss : {}".format((i+1)*batch_size, num_train,
train_loss))
# print("batch_size: {}/{}".format((i+1)*batch_size, num_train))
self.loss_history.append(train_loss)
## 利用batch_size样本进行梯度更新
P,Q = self.grad_norm(P, Q, train_step_dataset)
# print(train_loss)
return P, Q
```