

《大数据挖掘》实验报告3

题 目 ：音乐推荐算法优化

专 业 数据科学与大数据技术

学 号

姓 名

日 期 2025年4月18日

**目 录**

[第1章 预处理与基本统计 2](#_Toc195891585)

[1.1数据预处理 2](#_Toc195891586)

[1.1.1统计数据不规范的类型 2](#_Toc195891587)

[1.1.2根据artist\_alias.txt的映射关系更新user\_artist\_data.txt 2](#_Toc195891588)

[1.1.3将user\_artist\_data.txt和artist\_data.txt转换为csv格式 3](#_Toc195891589)

[1.2艺术家播放量排名 3](#_Toc195891590)

[第2章 基本UV分解方法 5](#_Toc195891591)

[2.1基本UV分解方法概述 5](#_Toc195891592)

[2.1.1交替最小二乘法ALS 5](#_Toc195891593)

[2.1.2梯度下降 6](#_Toc195891594)

[2.2程序流程图 6](#_Toc195891595)

[2.3 实验结果分析 8](#_Toc195891596)

[2.3.1基于ALS的矩阵分解结果 8](#_Toc195891597)

[2.3.2基于梯度下降的矩阵分解结果 9](#_Toc195891598)

[第3章 算法优化 11](#_Toc195891599)

[3.1 基于梯度下降法的算法优化 11](#_Toc195891600)

[3.2 编程实现中的优化 11](#_Toc195891601)

[3.3 实验结果对比分析 13](#_Toc195891602)

[3.4 改进思路 13](#_Toc195891603)

# 第1章 预处理与基本统计

## 1.1数据预处理

### 1.1.1统计数据不规范的类型

找出artist\_data.txt和artist\_alias.txt文件中数据不规范的类型，并分别统计不同类型的记录数量。经过检查，发现artist\_data.txt和artist\_alias.txt中有以下几种数据不规范的类型。

（1）缺失ID：artist\_data.txt中部分样本缺少艺术家ID，artist\_alias.txt中部分样本只有一个ID。

（2）缺少艺术家姓名：artist\_data.txt中有部分样本缺少艺术家姓名。

（3）艺术家姓名出现了乱码：在 artist\_data.txt 文件中，部分样本的艺术家姓名存在乱码问题。可以使用 chardet 库的 detect 方法来检测字符编码，该方法接收一个字节序列作为输入，自动判断其可能的编码格式（如 UTF-8、GBK 等），并返回一个包含编码类型和置信度的字典。将艺术家姓名转换为字节序列后传入该方法，根据返回的置信度进行判断：若置信度低于 50%，则认为该艺术家姓名为乱码。

数据不规范类型的统计如图1所示。

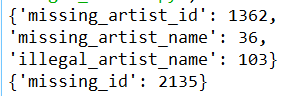


图1 artist\_data.txt和artist\_alias.txt数据不规范情况统计

在后续处理时，需要将这些数据不规范的样本删除，从而实现对数据的清洗。

### 1.1.2根据artist\_alias.txt的映射关系更新user\_artist\_data.txt

artist\_alias.txt是拼写错误的艺术家ID到艺术家规范ID的映射，格式为:（拼写等错误的艺术家ID 艺术家正确ID）。user\_artist\_data.txt是用户播放艺术家歌曲次数，格式为：(用户ID 艺术家ID 播放次数)。根据artist\_alias.txt更新user\_artist\_data.txt的具体流程如下。

（1）首先读取 artist\_alias.txt 文件，解析其中的错误 ID 和正确 ID，将它们存储在一个字典 alias\_map 中，键是错误 ID，值是正确的 ID。

（2）接下来遍历 user\_artist\_data.txt 中的每个样本，对于每个 artist\_id，调用 alias\_map.get(artist\_id) 来检查是否有映射关系，如果没有，则使用原来的 artist\_id，否则返回映射的正确 ID。

### 1.1.3将user\_artist\_data.txt和artist\_data.txt转换为csv格式

为了方便后续的统计与建模，本文将 user\_artist\_data.txt 和artist\_data.txt转换为 CSV 格式。

在实现过程中，对于user\_artist\_data.txt，需要遍历 user\_artist\_data.txt 的每一行，过滤掉字段数量少于 3 个的异常样本，同时根据 artist\_alias.txt 中的映射关系，对 artist\_id 进行标准化处理。最终将清洗后的数据保存至 data/clean/user\_artist\_data.csv 文件中，生成的 CSV 文件包含三列：user\_id、artist\_id 和 count。

对于artist\_data.txt，遍历artist\_data.txt的每一行，以第一个空白符作为分隔符提取出艺术家ID和艺术家姓名，过滤掉缺失艺术家ID、缺失艺术家姓名、艺术家姓名为乱码的异常样本。清洗后的数据保存至data/clean/artist\_data.csv文件中，生产的CSV文件包含两列：artist\_id、artist\_name。

## 1.2艺术家播放量排名

统计播放量最高的30名艺术家ID以及其被播放总次数，要求按照播放次数降序输出，显示格式为：序号，艺术家名字，总播放次数。具体实现流程如下。

（1）读取user\_artist\_data.csv，按artist\_id分组并聚集计算每个artist\_id的总count值，即计算每位艺术家的总播放量。选取总播放量最高的前30个艺术家。

（2）从artist\_data.csv中读取艺术家信息，通过艺术家ID与前一步的统计结果进行关联合并，得到包含艺术家姓名以及播放量的列表。根据播放量对结果进行降序排列，将这前30名艺术家的信息写入top\_30\_artists.csv文件中，结果如图2所示。

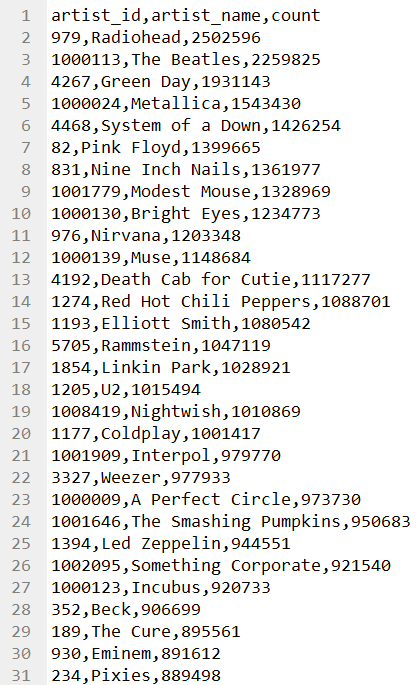


图2 播放量最高的前30名艺术家信息

# 第2章 基本UV分解方法

## 2.1基本UV分解方法概述

设效用矩阵为,其中表示用户对物品的评分。UV分解就是从效用矩阵中提取隐含因子，将效用矩阵分解为用户矩阵和物品矩阵，即，其中m是用户数量，n是物品数量，k是潜在因子维度。UV分解的常见方法有两种：交替最小二乘法ALS和梯度下降方法，这两种方法都是通过最小化效用矩阵与预测评分矩阵之间的误差来迭代优化矩阵和。

### 2.1.1交替最小二乘法ALS

ALS的核心思想是通过交替优化用户矩阵和物品矩阵，以最小化效用矩阵与预测矩阵之间的误差。ALS的目标是最小化以下的损失函数：

|  |  |
| --- | --- |
|  | （1） |

是效用矩阵中已知评分的位置集合，是正则化系数。损失函数的第一项是效用矩阵与预测矩阵之间的平方误差，第二项是L2正则项，用于降低模型复杂度，防止过拟合。

ALS算法的具体流程如下。

（1）初始化用户矩阵和物品矩阵。

（2）交替更新。每次固定其中一个矩阵，然后优化另一个矩阵。以固定物品矩阵，优化用户矩阵为例。对于每个用户，固定物品矩阵，求解以下优化问题：

|  |  |
| --- | --- |
|  | （2） |

表示用户评过分的物品的集合。对求导：

|  |  |
| --- | --- |
|  | （3） |

令导数为0，可得：

|  |  |
| --- | --- |
|  | （4） |

简写为：

|  |  |
| --- | --- |
|  | （5） |

同理，若固定，更新，则对每个物品，更新公式为：

|  |  |
| --- | --- |
|  | （6） |

其中，是对物品评过分的用户集合。

### 2.1.2梯度下降

梯度下降的核心思想是按照梯度方向减小损失函数，每一次都需要计算损失函数关于用户因子和物品因子的梯度，根据这些梯度逐步更新用户矩阵和物品矩阵。

为了防止模型过于复杂，避免过拟合，在损失函数中加入L2正则项。梯度下降的损失函数为：

|  |  |
| --- | --- |
|  | （7） |

其中，和分别为用户和物品的因子向量。

梯度下降方法的具体流程如下。

（1）初始化用户矩阵和物品矩阵。

（2）计算损失函数关于用户因子向量和物品因子向量的梯度：

|  |  |
| --- | --- |
|  | （8） |
|  | （9） |

（3）根据上一步计算的梯度，更新用户因子向量和物品因子向量：

|  |  |
| --- | --- |
|  | （10） |

其中是学习率，它决定更新时步长的大小。除此之外，梯度下降方法中还需要设置最大迭代次数，通过多次迭代更新和实现矩阵分解。

## 2.2程序流程图

基于ALS方法的矩阵分解的程序流程如图3所示。

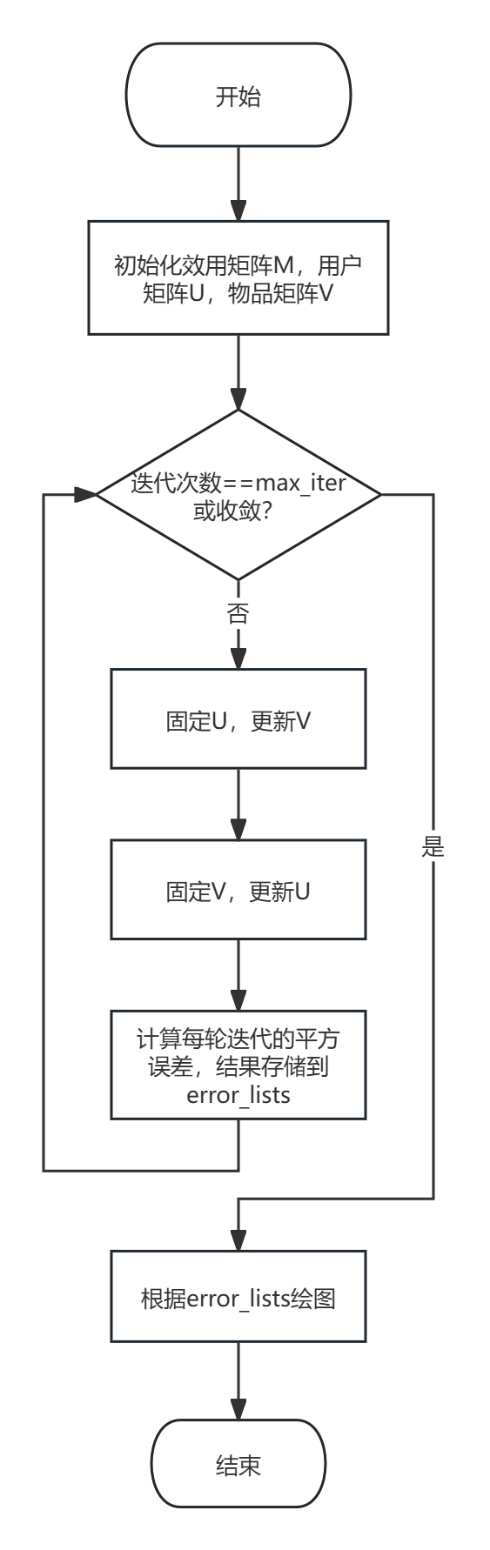


图3 基于ALS方法的矩阵分解的程序流程图

基于梯度下降方法的矩阵分解的程序流程如图4所示。

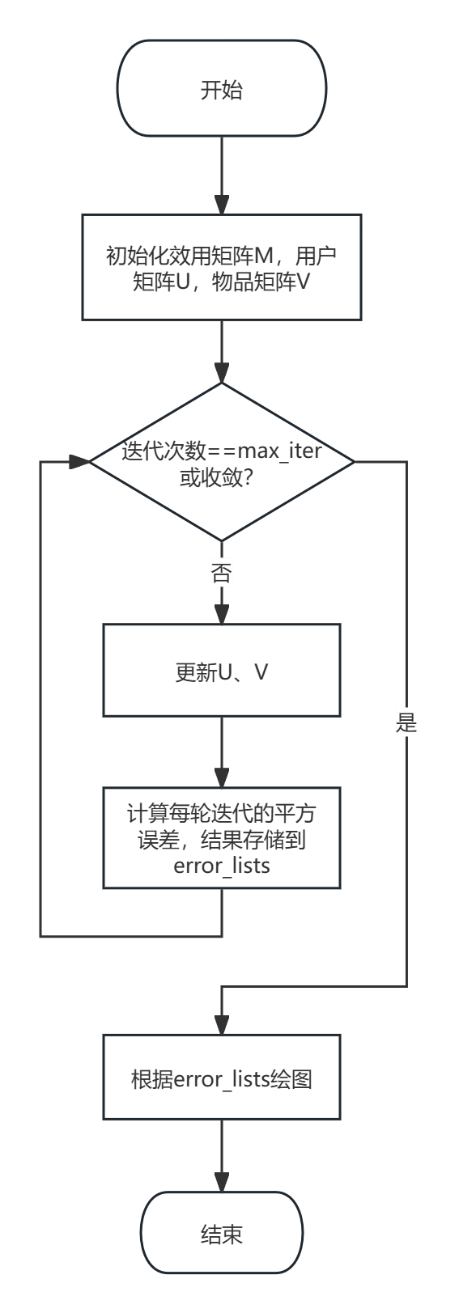


图4 基于梯度下降方法的矩阵分解的程序流程图

## 2.3 实验结果分析

### 2.3.1基于ALS的矩阵分解结果

（1）迭代20次，抽取20%的艺术家，样本共有236, 0126个，总样本数的10%左右。每次迭代的误差平方和曲线如图5所示。

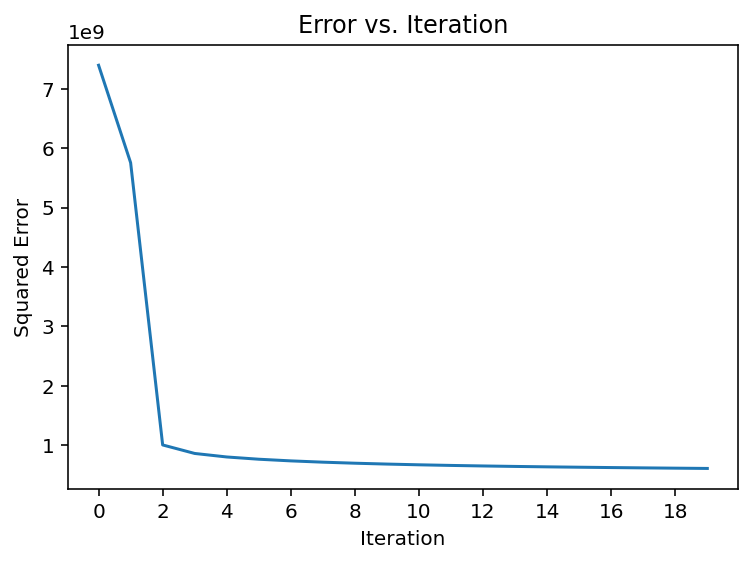


图5 基于ALS的矩阵分解的误差平方和曲线（10%样本）

（2）迭代20次，使用全部样本。每次迭代的误差平方和曲线如图6所示。

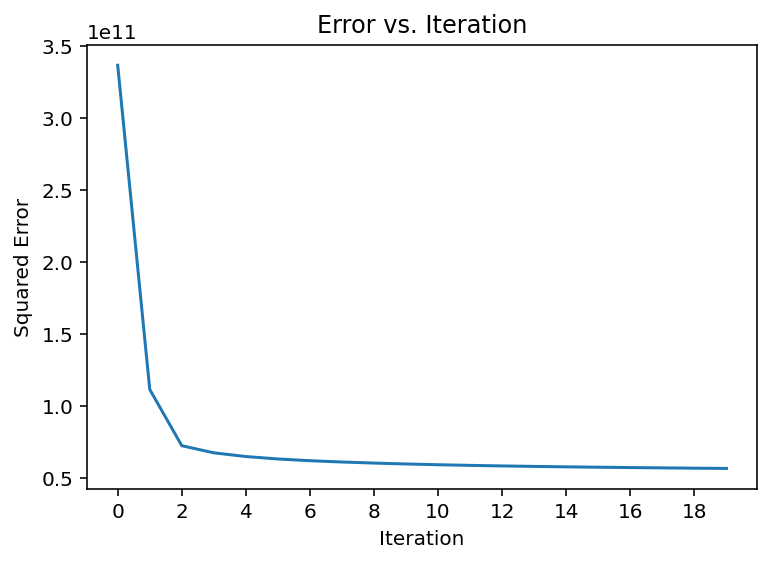


图6 基于ALS的矩阵分解的误差平方和曲线（全部样本）

使用全部样本运行ALS算法，每轮迭代时间大致为1min，这是一次迭代更新U、V矩阵和计算误差平方和的总时间。迭代20次的总时长为20min。

由图5，图6可知，ALS算法的收敛速度非常快，前两次迭代误差下降的非常大，从第3次迭代就已经接近收敛。这是因为ALS每轮迭代都是精确最优解（闭式解），即每一次对 U 或 V 的更新，都是“在当前固定另一边情况下的最优解”，不是一步步试出来的，而是一次就算出最优。目标函数是一个二次凸函数，每轮迭代都会精确地逼近局部极小点，下降路径是稳定的。

### 2.3.2基于梯度下降的矩阵分解结果

迭代30次，使用全部样本，学习率为1e-3，正则化系数为0.1。每次迭代的误差平方和曲线如图7所示。

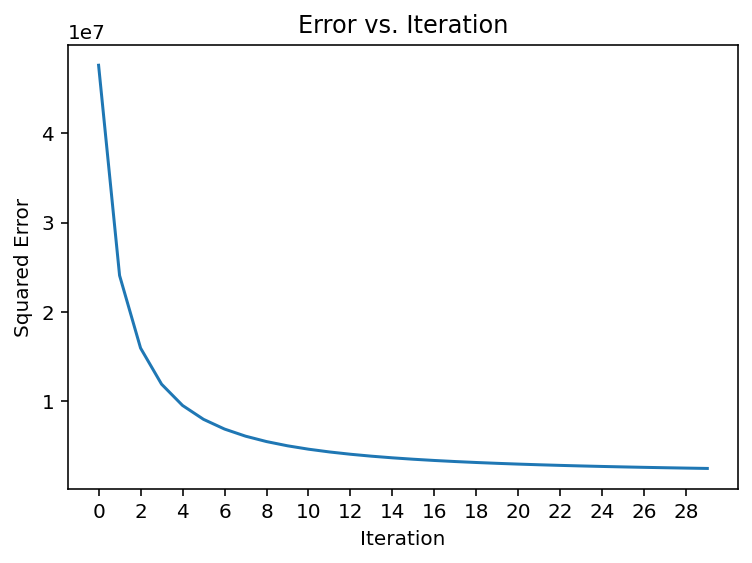


图7 基于梯度下降的矩阵分解的误差平方和曲线

使用全部样本运行梯度下降方法，每轮迭代时间大致为4min，这是一次迭代更新U、V和计算误差平方和的总时间。迭代30次的总时长为2h。

由图7可知，从第5次迭代之后，曲线逐渐变得平滑，误差下降变得缓慢，这说明模型逐步接近最优解。在接近第 30 次迭代时，误差趋于稳定，没有剧烈波动或震荡，说明训练过程稳定，梯度下降效果良好。

# 第3章 算法优化

## 3.1 基于梯度下降法的算法优化

相较于ALS，在理论上，梯度下降法在矩阵分解中有以下优化。

（1）梯度下降的计算效率更高。ALS的计算涉及解线性方程组，需要进行矩阵求逆，当维度较高或者数据稀疏时计算量较大。梯度下降的每次迭代只需要简单的矩阵操作，如矩阵的加减乘除，计算更高效。

（2）梯度下降更适合于稀疏矩阵。ALS在更新一行或一列时，需要考虑所有观测值对应的元素，包括零值，导致冗余计算。梯度下降法可以仅基于观测值的非零元素更新参数，跳过效用矩阵中的零值，避免无用计算。

## 3.2 编程实现中的优化

由于user\_artist\_data数据集中的样本数量极大，这为初始化效用矩阵，运算效率以及存储上带来了很大的挑战。本文使用了多种手段提高计算效率，并使用高效的数据结构存储效用矩阵，对梯度下降方法尝试了多种初始化方法，提高梯度下降结果的质量。以下是一些优化的方法。

（1）**效用矩阵的创建和存储**：首先读取user\_artist\_data数据，将用户ID与艺术家ID分别映射为连续索引，基于用户索引（行）、艺术家索引（列）及播放次数，调用scipy库中的csr\_matrix方法构建CSR格式的稀疏效用矩阵M，CSR格式是行访问更快的稀疏矩阵类型。这既节省了存储效用矩阵的空间，又能保证按行计算时能快速访问效用矩阵M的一行。

（2）**本文通过以下方法优化计算效率**：首先，充分利用Python的矩阵运算库进行向量化计算，避免逐元素操作，提升整体运算速度。其次，在ALS算法迭代过程中，针对固定物品矩阵V更新用户矩阵U的需求，将原始行优先存储的CSR格式稀疏矩阵转换为列优先的CSC格式稀疏矩阵。由于CSC格式支持高效的列切片操作，可显著减少固定V时列数据访问的时间开销。为避免迭代中重复转换带来的性能损耗，该格式转换操作在ALS迭代开始前完成，确保算法执行期间直接调用预处理的列优化数据结构。

（3）**为提高梯度下降的质量，对效用矩阵M进行归一化**：这是因为用户的播放行为差异很大，归一化后可以消除这偏差，让模型专注于“用户对艺术家的偏好差异”上。归一化能够将数据缩放到一个统一范围内，使得损失函数对每条样本的梯度贡献更加平衡，有利于更快收敛。此外，归一化能防止梯度爆炸情况，因为它避免了极端数值的影响，而不会导致计算过程中出现溢出情况。

（4）**为提高梯度下降的收敛质量，本文尝试了多种 U、V 矩阵的初始化方式**。初始化方式对前几次迭代过程具有显著影响。若 U、V 初始化不当，容易导致梯度下降初期的搜索方向偏离最优路径，从而造成误差下降缓慢，甚至陷入局部最优，影响整体训练效果。

经实验研究，将 U、V 初始化为全1矩阵可以取得更好的误差下降效果。特别是在算法前几轮迭代中，该方式表现出更快的误差收敛趋势。这可能是由于常数初始化（尤其是全1）提供了一个平衡而稳定的起点，避免了初始梯度过大或过小的情况，从而有助于模型更稳定地调整参数方向。

效用矩阵未归一化，U、V初始化不佳的梯度下降结果如图8所示。

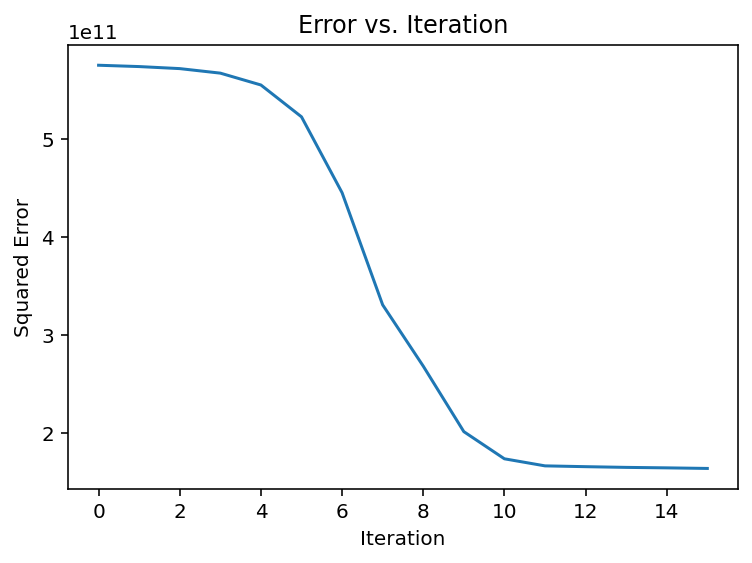


图8 M未归一化，U、V初始化不佳的梯度下降结果

效用矩阵归一化，U、V初始化为全1矩阵的梯度下降结果如图9所示。

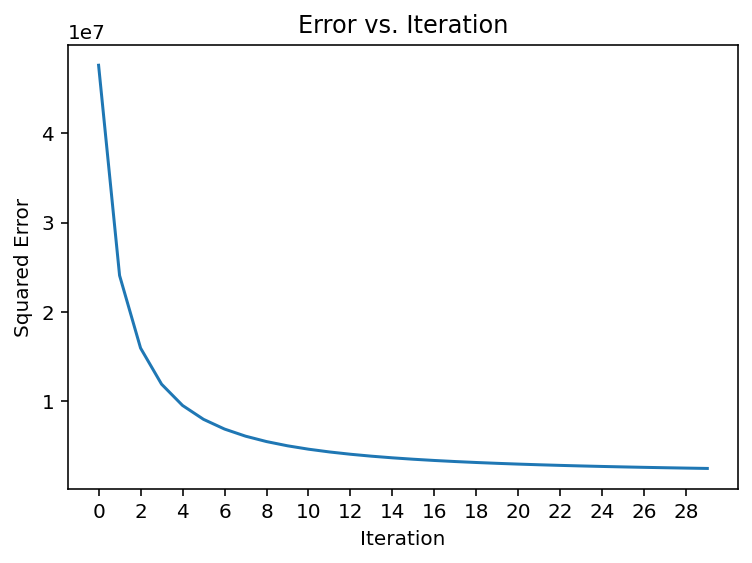


图9 M归一化，U、V初始化为全1矩阵的梯度下降结果

可以看到，在前5次迭代过程中，效用矩阵未归一化，U、V矩阵初始化不佳的误差下降非常缓慢，这是因为在初始阶段模型没有找到好的梯度下降方向，导致误差下降幅度很小。同时，由于效用矩阵未归一化，矩阵中部分数值过大，模型训练时学习率必须极小，如1e-5、1e-6，否则会很容易出现梯度爆炸的情况。

## 3.3 实验结果对比分析

在运行速度方面，虽然理论上梯度下降应该快于ALS方法，但实际上发现：在整个user\_artist\_data数据集上，梯度下降的每次迭代的时间为4min，而ALS方法的每次迭代时间为1min，二者相差较大。这可能是因为Python提供了高效的矩阵求逆和线性方差求解的工具，使ALS的算法效率显著优于手动实现的梯度下降。

在收敛稳定性方面，ALS的收敛速度快于梯度下降，这是因为ALS每轮迭代都是精确最优解（闭式解），即每一次对U或V的更新，都是“在当前固定另一边情况下的最优解”，不是一步步试出来的，而是一次就算出最优。目标函数是一个二次凸函数，每轮迭代都会精确地逼近局部极小点，下降路径是稳定的。而梯度下降依赖学习率和梯度方向逐步逼近最优解，容易受到初始值和学习率的影响，需要更多轮次迭代才能收敛。

## 3.4 改进思路

本文中梯度下降的运行速度较慢，考虑到梯度下降方法适合于并行计算，可以使用小批量并行计算的方式来优化梯度下降运行速度。此外，还可以考虑使用分布式计算框架，如Spark，来提高ALS和梯度下降的算法效率。