大规模数据处理技术 Assignment1

1 实验介绍

1.1 数据集介绍：MovieLens-1M

MovieLens-1M是一个广泛用于推荐系统研究的数据集，它包含了从一百万个匿名用户收集的电影评分数据。这个数据集由三个文件组成：用户数据、电影数据和评分数据。我们的数据主要来源于评分数据集（ratings.csv）其主要包含以下字段：

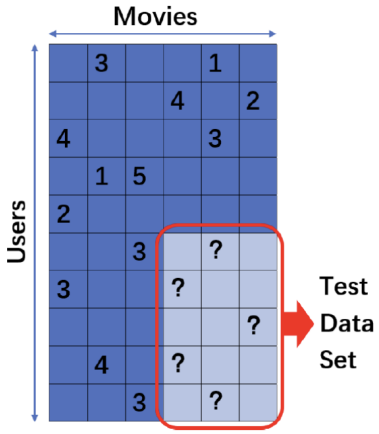
- UserID：用户ID
- MovieID：电影ID
- Rating：用户对电影的评分，从1到5（5表示最高评分）
- Timestamp：评分时间戳

数据集中的电影涵盖了不同年代、不同类型的电影，用户的年龄和职业也多种多样，因此适合用于推荐系统的研究和评估。

1.2 数据结构

数据集以矩阵表示，文件名为 col_matrix.csv

- 行标：表示用户的ID，从0开始编号，共6040个用户（行）
- 列标：表示影片的ID，从0开始编号，共3952部影片（列）
- 矩阵的元素：表示用户对该影片的评分，以0-5分表示，0分为缺失，分数均为整数
- 测试集部分为 col_matrix[4100:, 2700:]（如下图淡蓝色的区域）



2 算法介绍

2.1 皮尔逊积差相关系数（Pearson product-moment correlation coefficients）

皮尔逊相关系数是两个变量协方差除以它们标准差的乘积。

$$r = \frac{cov(X,Y)}{\sigma_X \sigma_Y} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

使用皮尔逊相关系数来计算两个对象之间的相关度，同时可以消除整体偏置带来的影响。

2.2 协同过滤

- **基于用户的协同过滤 (User-Based CF)**
  - **特点**: 根据用户的历史行为找出兴趣相似的用户群，用这群用户的行为预测目标用户的喜好。
  - **优点**: 简单直观，易于实现。
  - **缺点**: 用户数目庞大时计算复杂度高，对新用户和长尾物品推荐效果不佳。
- **基于物品的协同过滤 (Item-Based CF)**
  - **特点**: 根据物品之间的相似度进行推荐，适合用户喜欢某物品时推荐相似物品。
  - **优点**: 计算复杂度相对较低，适合大规模数据。
  - **缺点**: 物品数目庞大时计算复杂度高，不太适用于新物品的推荐。

3 算法实现

3.1 基于用户的协同过滤 (User-Based CF)

```
class UserCF:
    def __init__(self, data):
        self.data = data
        self.norm_user_rate = data - data.mean(axis=1,keepdims=True)
        self.user_similarity_matrix = np.corrcoef(data)

    def predict(self, user_id, item_id, k=10):
        # remove users who haven't rated the item
        non_zero_users = np.nonzero(self.data[:,item_id])[0]
        # remove the user itself
        non_zero_users = non_zero_users[non_zero_users != user_id]

        sim_scores = self.user_similarity_matrix[user_id][non_zero_users]

        if len(sim_scores) == 0:
            return 0
        elif len(sim_scores) < k:
            similar_users = non_zero_users
        else:
            similar_users = non_zero_users[np.argsort(sim_scores)[-k:]]
        prediction = self.data[similar_users,
item_id].dot(self.user_similarity_matrix[user_id][similar_users]) /
np.sum(self.user_similarity_matrix[user_id][similar_users])
        return prediction
```

给定一个用户与一个物品，算法逻辑如下：

- 首先，找出对该物品有评分的用户（排除需要预测的用户）。
- 然后，根据用户相似度矩阵，获得目标用户与这些用户的相似度得分。
- 如果没有相似用户，说明其他用户对该物品没有评分，返回0。如果相似用户数目小于k，直接使用这些用户进行预测。否则，选取相似度最高的k个用户进行预测。

- 最后，使用这些相似用户的评分和相似度得分，通过加权平均的方式计算出对物品的预测评分。

3.2 基于物品的协同过滤 (Item-Based CF)

```
class ItemCF:
    def __init__(self, data):
        self.data = data
        self.norm_user_rate = data - data.mean(axis=1,keepdims=True)
        self.item_similarity_matrix = np.corrcoef(data.T)

    def predict(self, user_id, item_id, k=10):
        # remove items that the user hasn't rated
        non_zero_items = np.nonzero(self.data[user_id])[0]
        sim_scores = self.item_similarity_matrix[item_id][non_zero_items]

        if len(sim_scores) == 0:
            return 0
        elif len(sim_scores) < k:
            similar_items = non_zero_items
        else:
            similar_items = non_zero_items[np.argsort(sim_scores)[-k:]]
        prediction = self.data[user_id,
similar_items].dot(self.item_similarity_matrix[item_id][similar_items]) /
np.sum(self.item_similarity_matrix[item_id][similar_items])
        return prediction
```

给定一个用户与一个物品，算法逻辑如下：

- 首先，找出用户评分过的物品（排除需要预测的物品）。
- 然后，根据物品相似度矩阵，获得目标物品与这些物品的相似度得分。
- 如果没有相似物品，说明其他物品没有被评分，返回0。如果相似物品数目小于k，直接使用这些物品进行预测。否则，选取相似度最高的k个物品进行预测。
- 最后，使用这些相似物品的评分和相似度得分，通过加权平均的方式计算出对物品的预测评分。

4 实验结果

4.1 运行环境

- CPU: Intel(R) Xeon(R) Silver 4210R CPU
- OS: Ubuntu 20.04.6 LTS Linux 5.4.0-177-generic
- Python: Python 3.9.7
- pandas: 1.3.4
- numpy: 1.26.4

4.2 时间开销

- User-Based CF
  - 相似度矩阵计算: 1.61 s ± 64.5 ms
  - 预测: 45.5 µs per prediction

- Item-Based CF
  - 相似度矩阵计算: 1.03 s ± 47.4 ms
  - 预测: 31.9 µs per prediction

4.3 评估指标

选取左上角1000 × 1000的数据作为测试数据进行评分预测，预测结果与实际结果的损失如下表：

- L1 Loss

	k=10	**k=100**
User-Based CF	0.78	0.76
Item-Based CF	0.51	0.69

- RMSE Loss

	k=10	**k=100**
User-Based CF	0.98	0.96
Item-Based CF	0.66	0.87