机器学习 Assignment 4

21307265 童齐嘉

实验要求

问题:

选择一个与课程 Lecture 10（PCA）或之后课程内容相关的主题，查阅相关资料并进一步深入研究，完成一份课程报告。可选课程报告的主题具体包括：聚类、表征学习、数据降维、生成模型、异常检测、推荐系统等课程涉及内容。报告需包含如下部分：

1. 研究问题的背景和动机
2. 简要概述当前解决该问题的主要方法
3. 详细阐述你的模型、算法或方法
4. 实验结果及分析
5. 结论

要求:

1. 可以使用现有的深度学习框框，如：Tensorflow, PyTorch, MindSpore 等 （鼓励使用国产华为推出的 MindSpore 深度学习框架，但不强制。）
2. 可以调用现有的软件包
3. 鼓励提出自己的方法或对现有方法的改进
4. 鼓励与其它方法进行比较，可以是性能层面，也可以是方法层面，并对比较的结果进行分析讨论。

一、研究问题的背景和动机

推荐系统早已融入了我的生活，但由于本人从未接触过对于推荐系统的实验或项目，因此，推荐系统的实现在我眼中事实上蒙上了一层神秘的面纱，于是我便有了尝试实现一个推荐系统的想法

在一番搜索后，我得到了著名的影评数据集movielens，于是便想实现基于项的协同滤波算法，以完成对于影评数据集的电影推荐。同时，尝试使用所学知识对推荐系统算法部分进行一定的优化，并对比优化方法的效果

二、解决该问题的主要方法

2.1 推荐系统简介

推荐系统问题旨在用户推荐相关项，项可以是用户未观看过的电影、书籍，未访问过的网站，可以是任何可以购买的产品，实现一种个性化的推荐。

推荐系统可以总结为以下模型：

$$\text{Utility Function: } u : X \times S \to R$$

其中，X 是用户的集合，S 是项的集合，R 是用户对项评分的集合，并且是关于项的有序集。

推荐系统问题主要的问题为：如何为矩阵收集已知的评级，如何从已知的评级中推断未知的评级，如何评估推断的好坏。收集评分可以通过显式收集用户的评分，也可以通过学习用户的行为预测评分；推断未知评分可以使用基于内容、协同相关、基于隐因子（矩阵分解）、基于深度模型的模型甚至混合模型等；评估推断的好坏时可以选择在评分表中划分一块区域用于测试，计算平方根误差（RMSE），Top K 的精确度等。

2.2 协同滤波算法原理及其优化

- **基于项的协同滤波算法**
  - 第一步：读取用户-项的评分矩阵 R。
  - 第二步：跟据评分矩阵计算用户相似度矩阵 S_I，在计算相似度时我们选择皮尔森相关系数。我们可以将计算出的评分矩阵保存在文件中，以免下次重复计算。
  - 第三步：假定我们要预测用户 u 给项 i 的评分。首先找到于目标项最相似的 K 个项 I_{sim}，并且用户 u 对这些项有评分记录，根据以下公式计算预测评分：

    $$r_{u,i} = \frac{\sum_{j \in I_{sim}} s_{i,j} r_{v,i}}{\sum_{j \in I_{sim}} s_{i,j}}$$

    其中，$r_{u,i}$ 指用户 u 对项 i 的预测评分，$s_{i,j}$ 指项 i 和项 j 的相似度。
- **哈利波特效应以及解决方法**
- 哈利波特效应
  - 因为《哈利波特》比较热门，基本购买任何一本书的人似乎都会购买《哈利波特》，这样会使类似《哈利波特》这样的热门物品与很多其他物品之间的相似度都偏高。
- 由于热门物品会与很多物品都有较大的相似度，这样会导致推荐的覆盖率和新颖度都不高。因此，需要适当的对热门物品加大惩罚。我们可利用以下改进公式对热门物品进行惩罚：

$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|^{1-\alpha}|N(j)|^{\alpha}}$$

- 通过提高α就可以惩罚热门物品j，α越大，覆盖率就越高，并且结果的平均热门程度会降低。因此，通过这种方法可以在适当牺牲准确率和召回率的情况下显著提升结果的覆盖率和新颖性。
- **基于项的协同滤波算法的评价**
  - 适用场景：
    - 相对于基于用户的协同滤波算法，更适用于兴趣变化较为稳定的应用，更接近于个性化的推荐，适合物品少用户多，用户兴趣固定持久，物品更新速度不是太快的场合。
  - 协同滤波算法的优点：适用于任何类型的项，不需要特征选择
  - 协同滤波算法的缺点：
    - 冷启动问题：对于基于用户的协同滤波算法，需要积累足够多的用户，并且用户有一定评分时才能找到一个用户的相似用户，而基于项的协同滤波算法没有此问题。
    - 稀疏性问题：项的数目一般很多，一个用户对项的评分往往不会很多，评分矩阵是稀疏的，难以找到对相同的项评分过的用户。
    - 新的项、评分较少的项因为评分较少，难以被推荐。

三、基于项的协同滤波算法的实现

- **方法选择**
  - 对于电源推荐，其特征为：物品少用户多，用户兴趣固定持久，物品更新速度不快。
  - 因此，首选基于项的协同滤波算法
- **数据预处理**
  - 为了加速训练过程，我们需要保存中间结果（也即评分矩阵与相似度矩阵）

```
# 将 DataFrame 压缩保存至压缩文件中
def save_matrix_to_pickle(matrix, dir_path, file_name):
    if os.path.exists(dir_path) is False:
        os.mkdir(dir_path)
    file_path = os.path.join(dir_path, file_name + ".pkl")
    print("Save Matrix to", file_path)
    matrix.to_pickle(file_path)

# 读取压缩过的 DataFrame 并返回
def load_matrix_from_pickle(file_path):
    print("Load Matrix from", file_path)
    matrix = pd.read_pickle(file_path)
    return matrix
```

- **读入数据**
  - 此后，将数据读入为矩阵以方便相似度计算：（行为用户，列为项）

```
# 读取 载入用户-项 评分表文件，转化为 DataFrame 输出
def load_data_to_matrix(file_path, step=","):
    print(f"Load Data From {file_path} to matrix")
    data = pd.read_csv(file_path, dtype={"userId": np.int32, "movieId": np.int32, "rating": np.float32},
                       usecols=range(3), sep=step, engine='python')
    rating_matrix = data.pivot_table(index=["userId"], columns="movieId", values="rating")
    print(f"Shape of Rating Matrix (Users, Movies): {rating_matrix.shape}")
    return rating_matrix
```

- **相似度计算**
  - 按照算法原理部分所述，我们使用了两种方法计算相似度，其中，改进方法中的α的值需要手动调整设置

```
# 皮尔森相似度
def compute_similarity(rating_matrix):
    similarity_matrix = rating_matrix.corr(method="pearson")
    return similarity_matrix


# 改进方法
def compute_similarity_alpha(rating_matrix, alpha):
    rating_matrix_sparse = csr_matrix(rating_matrix.fillna(0).values)
    item_count = np.array((rating_matrix_sparse != 0).sum(axis=0))
    # 分母
    denominator = np.power(item_count + 1e-8, 1 - alpha) * np.power(item_count.T + 1e-8, alpha)
    # 分子
    numerator = rating_matrix_sparse.T.dot(rating_matrix_sparse)
    similarity_matrix = pd.DataFrame(numerator / denominator, index=rating_matrix.columns,
                                     columns=rating_matrix.columns)
    return similarity_matrix
```

- **评分预测**
  - 首先挑选出和项 j 相似度大于 0 的项，再挑选出用户 i 评分过的项，二者取交集，再跟据相似度进行排序，选择前 K 个计算预测评分

```
def predict_item_score_for_user(user_id, item_id, rating_matrix, similarity_matrix, k=-1):
    similar_items = similarity_matrix[item_id].drop(item_id).dropna()
    similar_items = similar_items.where(similar_items > 0).dropna()
    if similar_items.empty:
        return None

    user_rated_items = rating_matrix.loc[user_id].dropna()
    user_rated_similar_items = similar_items.loc[list(set(similar_items.index) & set(user_rated_items.index))]
    user_rated_similar_items.sort_values(ascending=False, inplace=True)

    a = 0  # 相似项的评分乘以相似度的累加和
    b = 0  # 相似度的累加和
    c = 0
    for similar_item, similarity in user_rated_similar_items.iteritems():
        a += similarity * rating_matrix.loc[user_id, similar_item]
        b += similarity
        c += 1
        if c == k:
            break
    if b == 0:
        return None

    r = a / b
    if r > 5.0:  # 评分预测值
        r = 5.0
    return r
```

- **整合**
  - 使用CF类实现整个算法与推荐流程

```
class CF:
    """
    初始化评分矩阵和相似度矩阵
    input:
        data_name: String 数据集名称标识
        data_file_path: String 评分数据文件路径
        step: String 评分数据文件的分隔符
        val: Boolean 是否为验证模式，验证模式下空出一块预取的评分来进行验证测试
        val_mask: ((a, b), (c, d)) 验证模式下评分矩阵空出一块的范围：a:b行，c:d列
    """

    def __init__(self, alpha, data_name, data_file_path, step=",", val=True, val_mask=((0, 100), (0, 200))):
        self.alpha = alpha
        self._matrix_path = "./Matrix/"
        self._val = val

        # 载入/计算评分矩阵
        file_name = data_name + "-rating"
        save_file_path = self._matrix_path + file_name + ".pkl"
        start = time.time()
        if os.path.exists(save_file_path):
            self._rating_matrix = load_matrix_from_pickle(save_file_path)
        else:
            self._rating_matrix = load_data_to_matrix(file_path=data_file_path, step=step)
            save_matrix_to_pickle(self._rating_matrix, self._matrix_path, file_name)
        end = time.time()

        if val:
            (u_l, u_r), (i_l, i_r) = val_mask
            self._mask_ground_truth = self._rating_matrix.iloc[u_l:u_r, i_l:i_r].copy()
            self._rating_matrix.iloc[u_l:u_r, i_l:i_r] = np.nan
```

- **推荐**
  - 实现在CF类中，为一个用户推荐 N 个预测评分最高的项

```
    def top_n_recommend(self, user_id, n, k=-1):
        """
        为一个用户推荐 N 个预测评分最高的项
        input:
            user_id: Integer 用户 ID
            n: Integer 返回的推荐项的个数
            k: Integer 预测分数时考虑的相似用户个数
        return:
            predict_result: [] 每一个元素是由 Item ID 和 预测评分组成的元组
        """
        start = time.time()
        pr = predict_all_items_score_for_user(user_id, self._rating_matrix, self._similarity_matrix, cold=10, k=k)
```

```
    predict_result = sorted(pr.items(), key=lambda x: -x[1])
    end = time.time()
    print("Time Cost of Top N Recommend: %.2f s" % (end - start))
    return predict_result[:n]
```

- **评估**
  - 使用了经典的RMSE（均方根误差）进行评估，也实现在CF类中

```
def score_predict_val(self, k):
    """
    验证测试函数
    input:
        k: Integer 预测分数时考虑的相似用户个数
    return:
        res: [] of (User ID, Item ID, Truth Score, Predict Score)
        rmse: Float 均方根误差
    """
    if not self._val:
        raise Exception("Only 'val' mode can call this method.")

    res = []
    rmse = 0
    count = 0
    start = time.time()
    for i in self._mask_ground_truth.index:
        for j in self._mask_ground_truth.columns:
            if self._mask_ground_truth.loc[i, j] > 0:
                truth_score = self._mask_ground_truth.loc[i, j]
                predict_score = predict_item_score_for_user(i, j, self._rating_matrix, self._similarity_matrix, k=k)
                if predict_score is not None:
                    rmse += (truth_score - predict_score) ** 2
                    count += 1
                    res.append((i, j, truth_score, predict_score))
rmse = (rmse / count) ** 0.5
end = time.time()
print("Time Cost of Score Predict Val: %.2f s " % (end - start))
return res, rmse
```

四、实验结果及分析

- **数据集描述**：验证测试采用了以下两个数据集

数据集	用户数目	电影数目	评分数目
ml-latest-small	610	9274	100 836
ml-1m	6040	3900	1 000 209

数据集来源： https://grouplens.org/datasets/movielens/

填问卷和邮箱即可获取

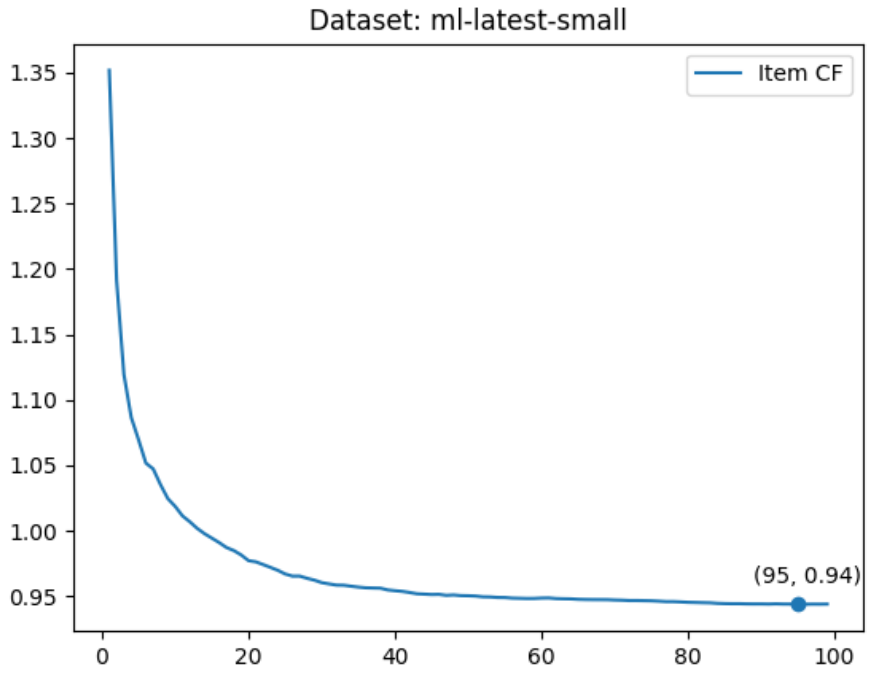
注：在测试时，我们选择前 100 个用户，200 个电影的区域作为验证区域。

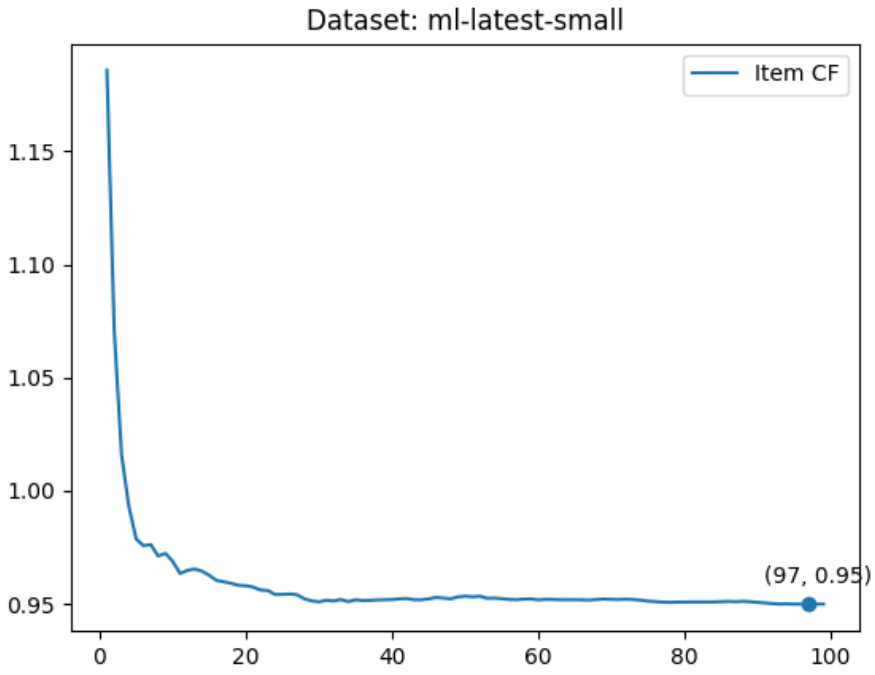
4.1.在数据集ml-latest-small上的结果

注：设置的最大迭代上限步数为100

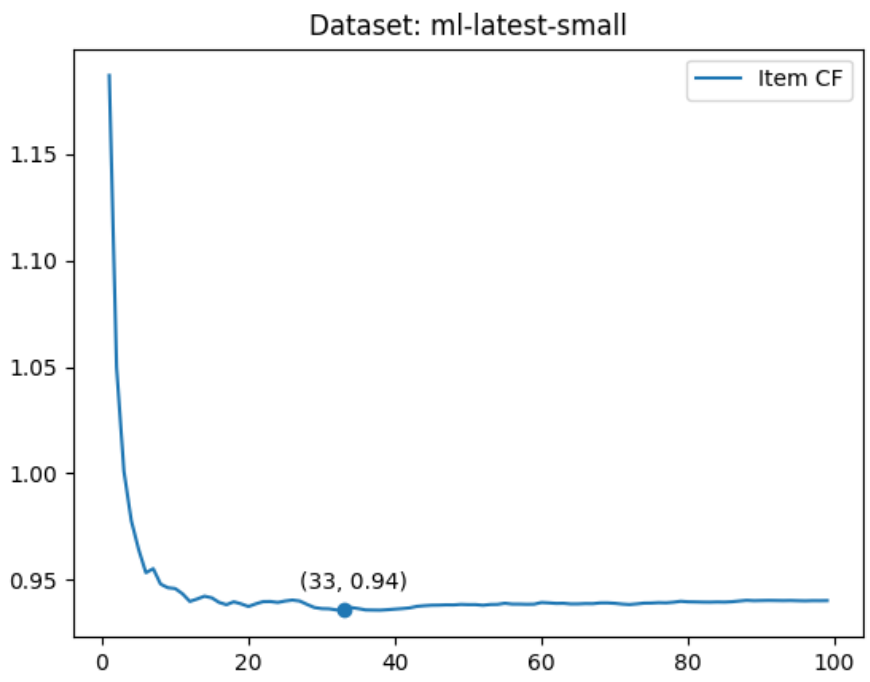
相似度计算方法	最小RMSE	取得最小RMSE时的迭代步数
皮尔森相似度	0.94	95
α=0.6	0.95	97
α=0.7	0.94	33
α=0.8	0.92	21
α=0.9	0.92	26
α=0.99	0.92	45

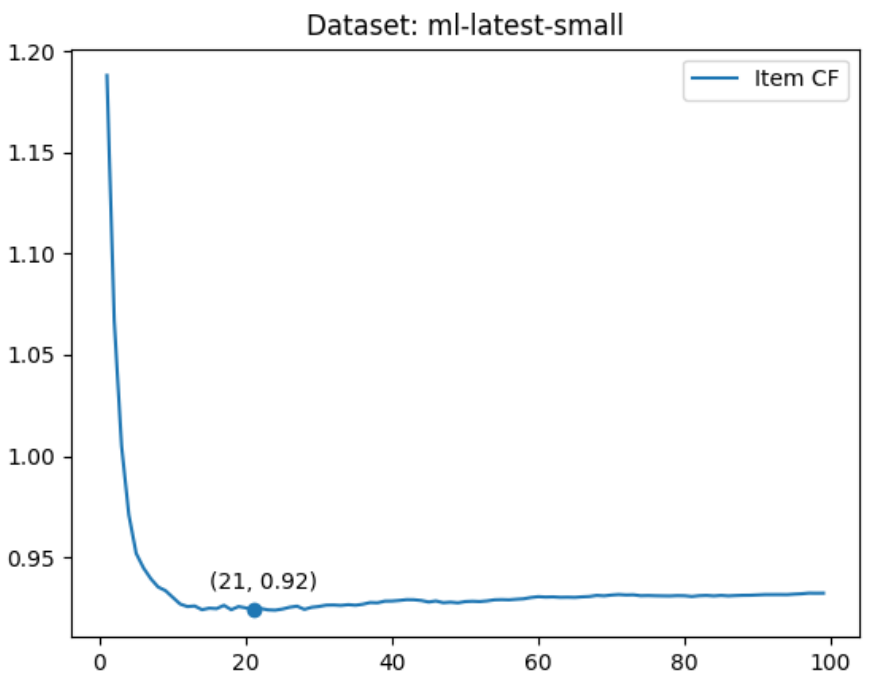
- 皮尔森相似度（左）、α=0.6（右）



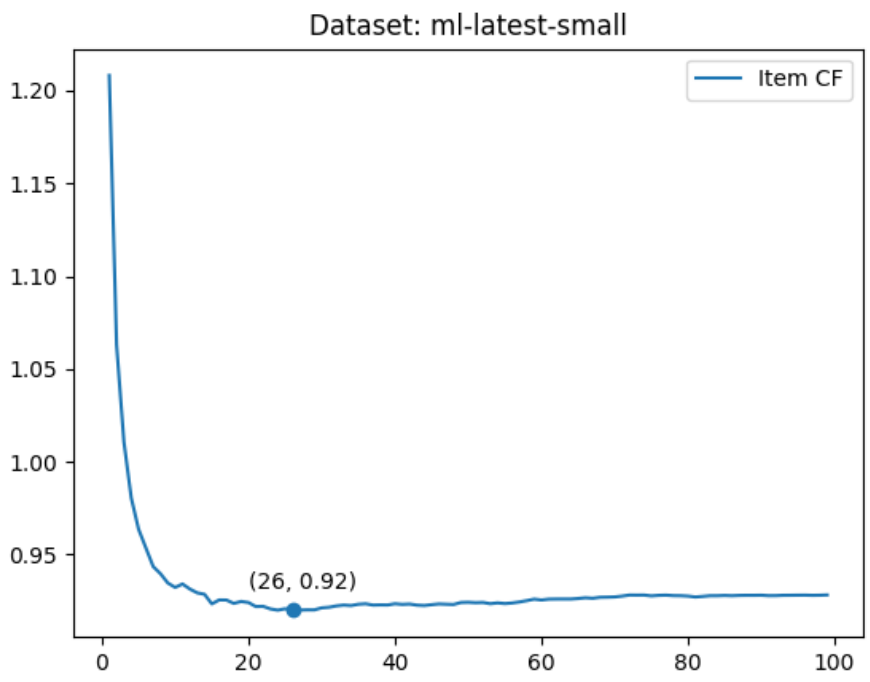


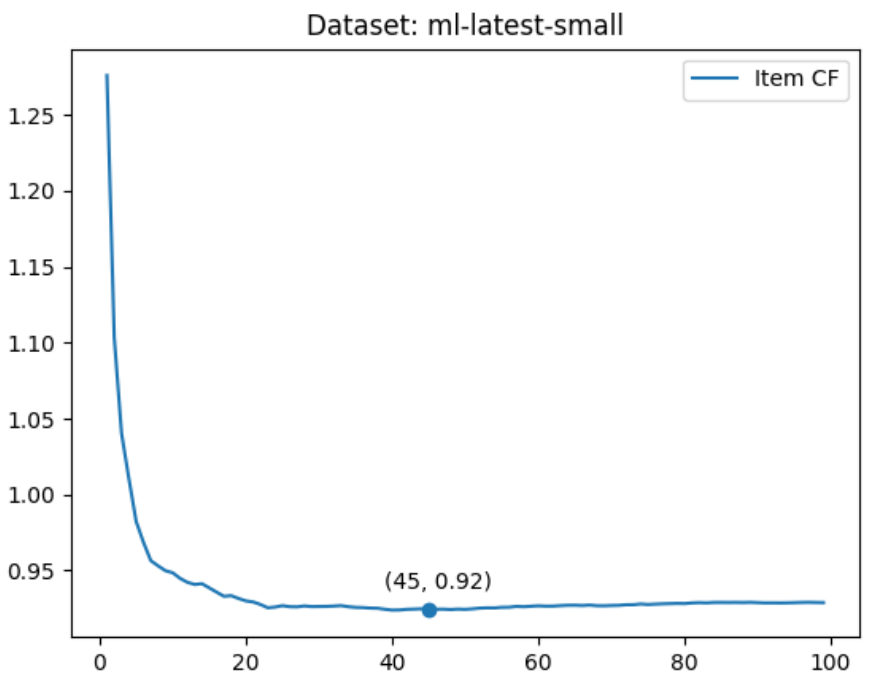
- α=0.7（左）、α=0.8（右）





- α=0.9（左）、α=0.99（右）





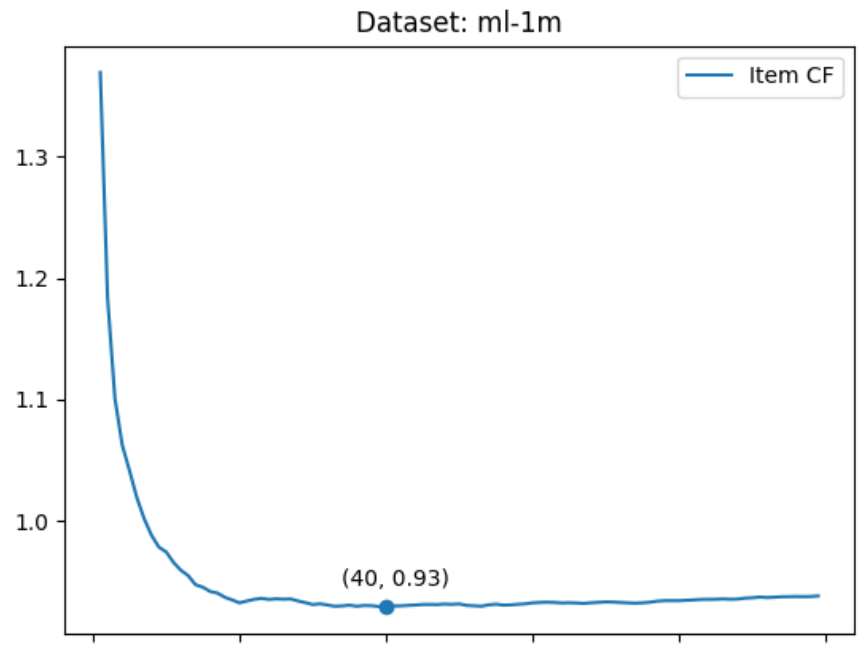
4.2.在数据集ml-1m上的结果

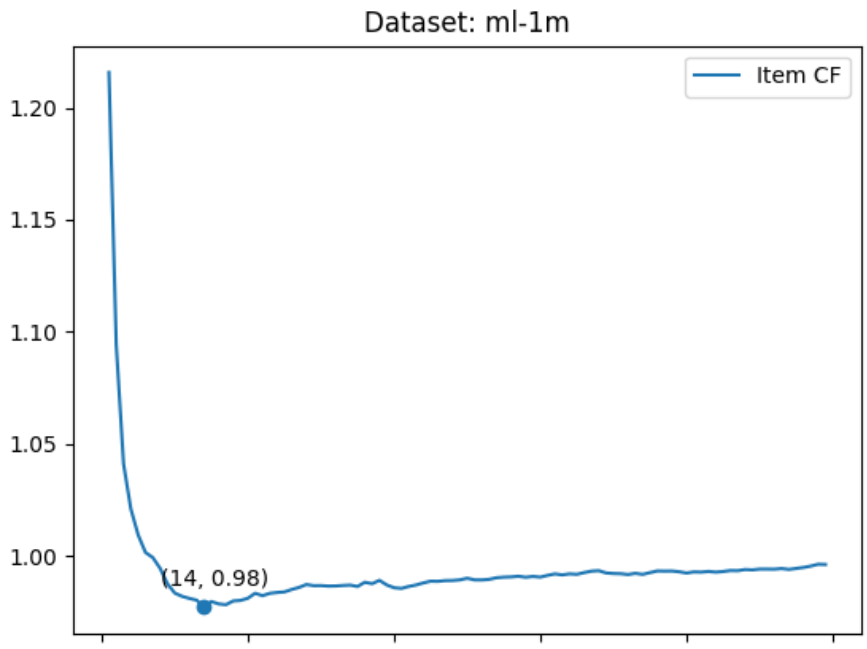
注：设置的最大迭代上限步数为100

相似度计算方法	最小RMSE	取得最小RMSE时的迭代步数
皮尔森相似度	0.93	40
α=0.6	0.98	14
α=0.7	0.94	17

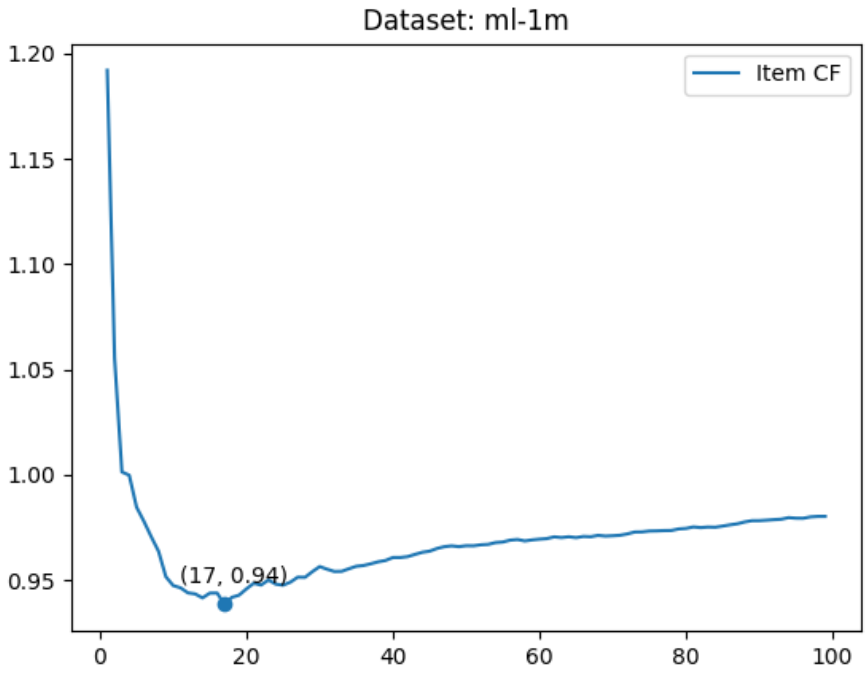
相似度计算方法	最小RMSE	取得最小RMSE时的迭代步数
α=0.8	0.92	16
α=0.9	0.92	17
α=0.99	0.92	27

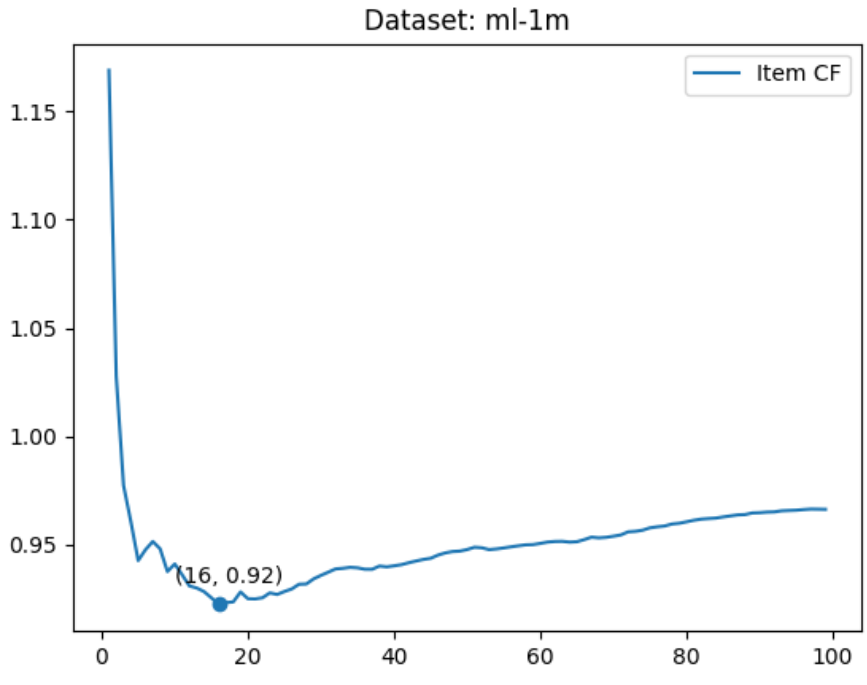
- 皮尔森相似度（左）、α=0.6（右）



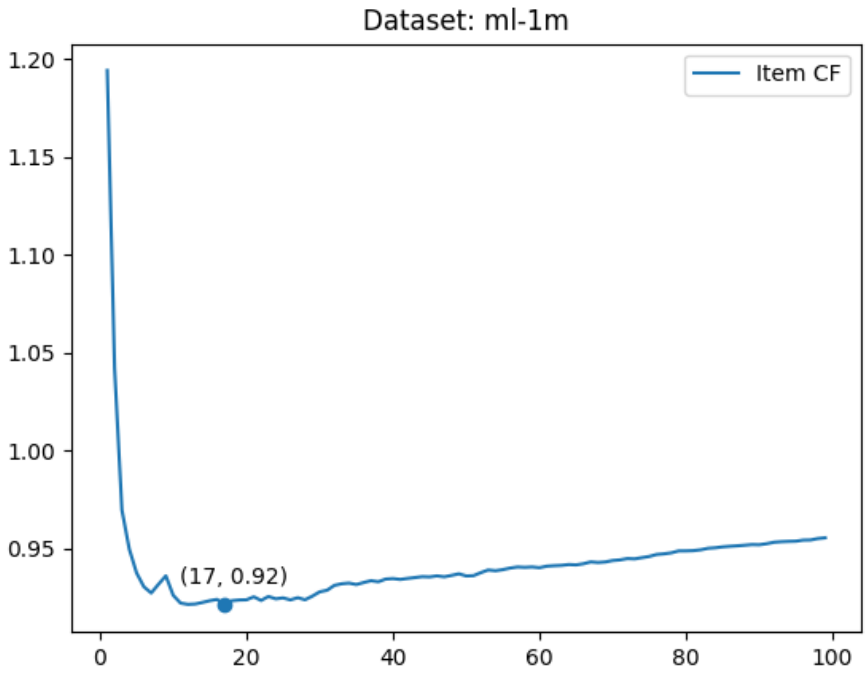


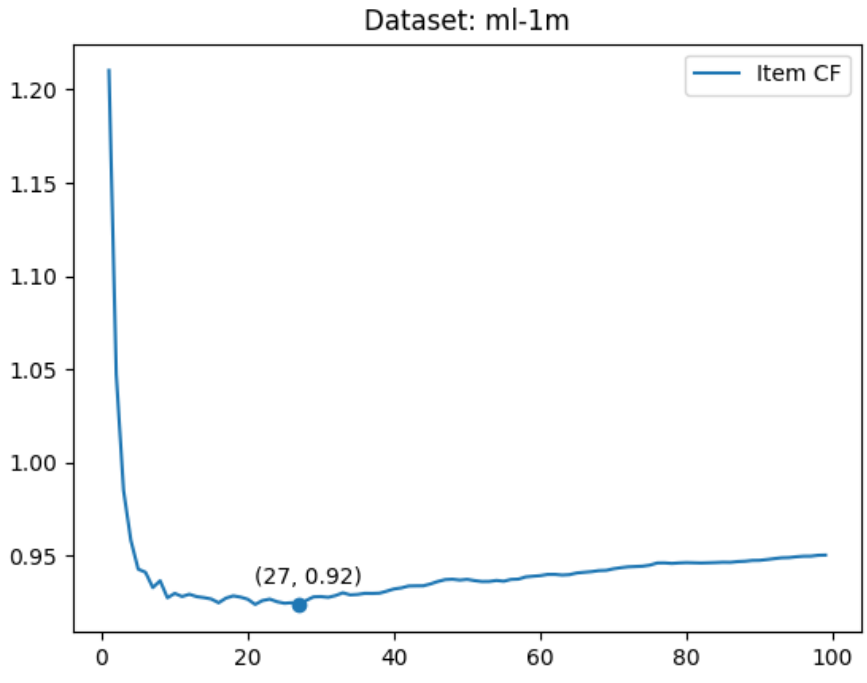
- α=0.7（左）、α=0.8（右）





- α=0.9（左）、α=0.99（右）





4.3.结果分析

1. 当α=0.8得到最优，在得到了更高准确率的同时，提升了一倍以上的迭代速度。实际应用时，如果为了提升效率，可以将迭代步数限制为20
2. 虽然**ml-latest-small**数据集的评价样本比**ml-1m**少了十倍，但由于其项的数量是ml-1m的两倍以上，所以对于基于项的协同过滤算法而言，实际计算量比**ml-1m**更大，耗时也更久，符合预期

五、结论

本实验实现了基于项的协同过滤算法，完成了对影评集的推荐系统的构建，并通过实际调参得出了当改进算法的α=0.8可以取得较优值，迭代步数设置为20有助于更高效地获取结果等结论