華中科技大学

课 程 实 验 报 告

课程名称： 大数据分析

专业班级： 计科 2003 班

学　　号： U202015374

姓　　名： 张隽翊

指导教师： 崔金华

报告日期： 2022 年 12 月 28 日

计算机科学与技术学院

目录

实验五 推荐系统算法及其实现

1.1 实验目的

1、了解推荐系统的多种推荐算法并理解其原理。
2、实现 **User-User 的协同过滤算法**并对用户进行推荐。
3、实现**基于内容的推荐算法**并对用户进行推荐。
4、对两个算法进行电影预测评分对比
5、在学有余力的情况下，**加入 minhash 算法**对效用矩阵**进行降维处理**

1.2 实验内容

给定 MovieLens 数据集，包含电影评分，电影标签等文件，其中电影评分文件（ratings.csv）分为训练集 train_set 和测试集 test_set 两部分

基础版必做一：**基于用户的协同过滤推荐算法**

对训练集中的评分数据构造用户-电影效用矩阵，使用 **pearson 相似度计算方法计算用户之间的相似度**，也即相似度矩阵。**对单个用户进行推荐时，找到与其最相似的 k 个用户，用这 k 个用户的评分情况对当前用户的所有未评分电影进行评分预测，选取评分最高的 n 个电影进行推荐。**

在测试集中包含 100 条用户-电影评分记录，用于计算推荐算法中预测评分的准确性，对测试集中的每个用户-电影需要**计算其预测评分，再和真实评分进行对比，误差计算使用 SSE 误差平方和**。

选做部分提示：此算法的进阶版采用 minhash 算法对效用矩阵进行降维处理，从而得到相似度矩阵，注意 minhash 采用 jaccard 方法计算相似度，需要对效用矩阵进行 01 处理，也即**将 0.5-2.5 的评分置为 0，3.0-5.0 的评分置为 1**。

基础版必做二：**基于内容的推荐算法**

将数据集 movies.csv 中的电影类别作为特征值，**计算这些特征值的 tf-idf 值**，得到关于电影与特征值的 n（电影个数）*m（特征值个数）的 **tf-idf 特征矩阵**。根据得到的 tf-idf 特征矩阵，**用余弦相似度的计算方法，得到电影之间的相似度矩阵**。

对某个用户-电影进行预测评分时，获取当前用户的已经完成的所有电影的

打分，通过电影相似度矩阵获得已打分电影与当前预测电影的相似度，按照下列方式进行打分计算：

$$\text{score} = \frac{\sum_{i=1}^{n} score'(i) * sim(i)}{\sum_{i=1}^{n} sim(i)}$$

选取相似度大于零的值进行计算，如果已打分电影与当前预测用户-电影相似度大于零，加入计算集合，否则丢弃。（相似度为负数的，强制设置为 0，表示无相关）假设计算集合中一共有 n 个电影，score 为我们预测的计算结果，score'(i)为计算集合中第 i 个电影的分数，sim(i)为第 i 个电影与当前用户-电影的相似度。如果 n 为零，则 score 为该用户所有已打分电影的平均值。

要求能够对指定的 userID 用户进行电影推荐，推荐电影为预测评分排名前 k 的电影。userID 与 k 值可以根据需求做更改。

推荐算法准确值的判断：**对给出的测试集中对应的用户-电影进行预测评分，输出每一条预测评分，并与真实评分进行对比，误差计算使用 SSE 误差平方和**。

选做部分提示：进阶版采用 minhash 算法对特征矩阵进行降维处理，从而得到相似度矩阵，注意 minhash 采用 jaccard 方法计算相似度，特征矩阵应为 01 矩阵。因此进阶版的特征矩阵选取采用方式为，**如果该电影存在某特征值，则特征值为 1，不存在则为 0，从而得到 01 特征矩阵**。

选做（进阶）部分：

本次大作业的进阶部分是在基础版本完成的基础上大家可以尝试做的部分。进阶部分的主要内容是使用**最小哈希（minhash）算法对协同过滤算法和基于内容推荐算法的相似度计算进行降维**。同学可以把最小哈希的模块作为一种近似度的计算方式。

协同过滤算法和基于内容推荐算法都会涉及到相似度的计算，最小哈希算法在牺牲一定准确度的情况下对相似度进行计算，其能够有效的降低维数，尤其是对大规模稀疏 01 矩阵。同学们可以**使用哈希函数或者随机数映射来计算哈希签名**。哈希签名可以计算物品之间的相似度。

最终降维后的维数等于我们定义映射函数的数量，我们设置的映射函数越少，整体计算量就越少，但是准确率就越低。**大家可以分析不同映射函数数量下，最终结果的准确率有什么差别**。

对基于用户的协同过滤推荐算法和基于内容的推荐算法进行推荐效果对比和分析，选做的完成后再进行一次对比分析。

1.3 实验过程

1.3.1 编程思路

推荐系统可以从两种思路进行分析：以用户为中心和以内容为中心，分别称作基于用户的推荐系统和基于内容的推荐系统。在本次实验中，基于用户的协同过滤推荐系统以用户为中心，查找相似的用户，对目标用户未打分的电影进行预测评分，将预测分数高的电影推荐给目标用户；基于内容的推荐系统则以内容为中心，以电影类型作为标签查找电影之间的相似关系，对目标用户未打分的电影进行预测评分，将预测分数高的电影推荐给目标用户。

（1）基于用户的协同过滤推荐系统（基础版）

整体思路如图 1.1 所示。

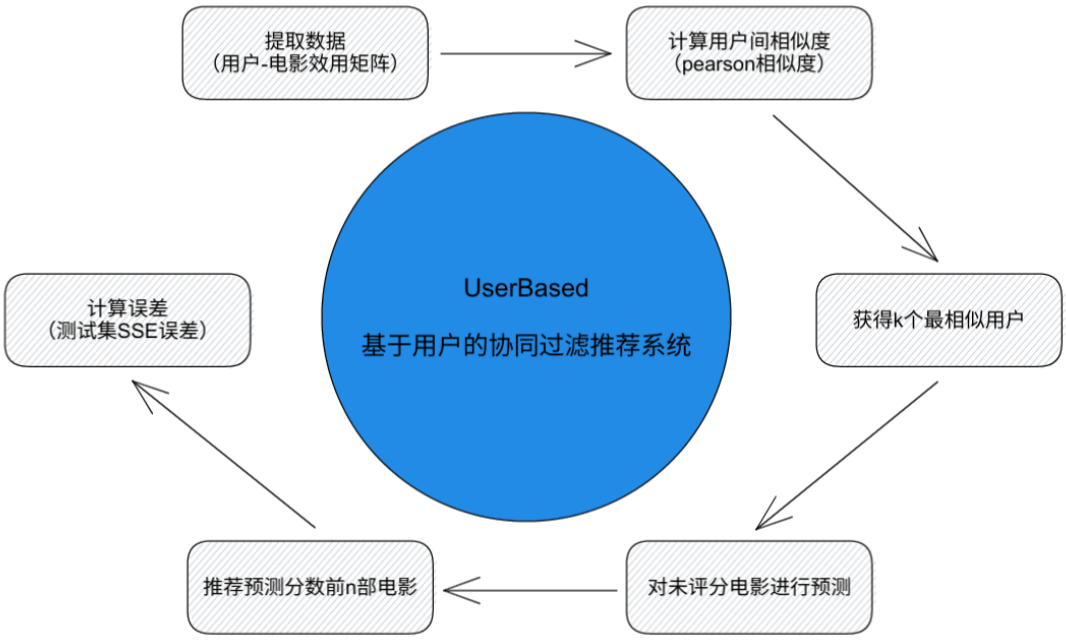

图 1.1 基于用户的协同过滤推荐系统流程

提取数据生成用户-电影效用矩阵的函数如下，该函数后续还会用到。

```python
def get_utility_mat() -> np.ndarray:
    """计算效用矩阵

    Returns:
        np.ndarray: 效用矩阵
    """
    # ? 获取用户-电影评分效用矩阵
    train_file = open(get_filename('train_set.csv'), 'r', encoding='utf-8')
    # * train_file: 存放每位用户评论的电影和评分
    # * train_file 是一个嵌套字典
    train_data = {}
    for line in train_file.readlines()[1:]:
        line = line.strip().split(',')
        # line[0] 为用户 id, line[1] 为电影 id, line[2] 为评分
        if line[0] not in train_data.keys():
            train_data[line[0]] = {line[1]: line[2]}
        else:
            train_data[line[0]][line[1]] = line[2]
    # * 效用矩阵
    utility_mat = pd.DataFrame(train_data).fillna(0).astype(float)
    return utility_mat
```

图 1.2 计算效用矩阵的函数

采用 pearson 相似度对用户相似度进行评价，pearson 相似度的计算方法如下。

$$\rho_{X,Y} = \frac{\sum XY - \frac{\sum X * \sum Y}{N}}{(\sum X^2 - \frac{(\sum X)^2}{N}) * (\sum Y^2 - \frac{(\sum Y)^2}{N})}$$

图 1.3 pearson 相似度计算方法

编写一个 Python 类 UserBasedRecommendSystem 实现上述功能。

UserBasedRecommendSystem 类具有以下字段：

- utility_mat：用户-电影效用矩阵，通过初始化参数传递；
- user_sim_mat：pearson 相关系数矩阵。

UserBasedRecommendSystem 类具有以下方法：

- get_corr_mat：计算 pearson 相关系数矩阵；
- predict：根据 k 个最相似的用户预测指定用户对指定电影的评分；
- recommend：根据 k 个最相似的用户为指定用户推荐 n 部电影。

基于用户的协同过滤推荐算法的过程为：

①由给定的数据集 train_set.csv 计算用户-电影效用矩阵；

②用效用矩阵初始化 UserBasedRecommendSystem 类，调用 get_corr_mat 方法计算 pearson 相关系数矩阵；

③读取测试数据集 test_set.csv，调用 predict 方法进行预测，记录预测评分；

④使用预测评分和标准评分计算 SSE；

⑤调用 recommend 方法为某个用户推荐电影。

（2）基于内容的推荐系统（基础版）

整体思路如图 1.4 所示。

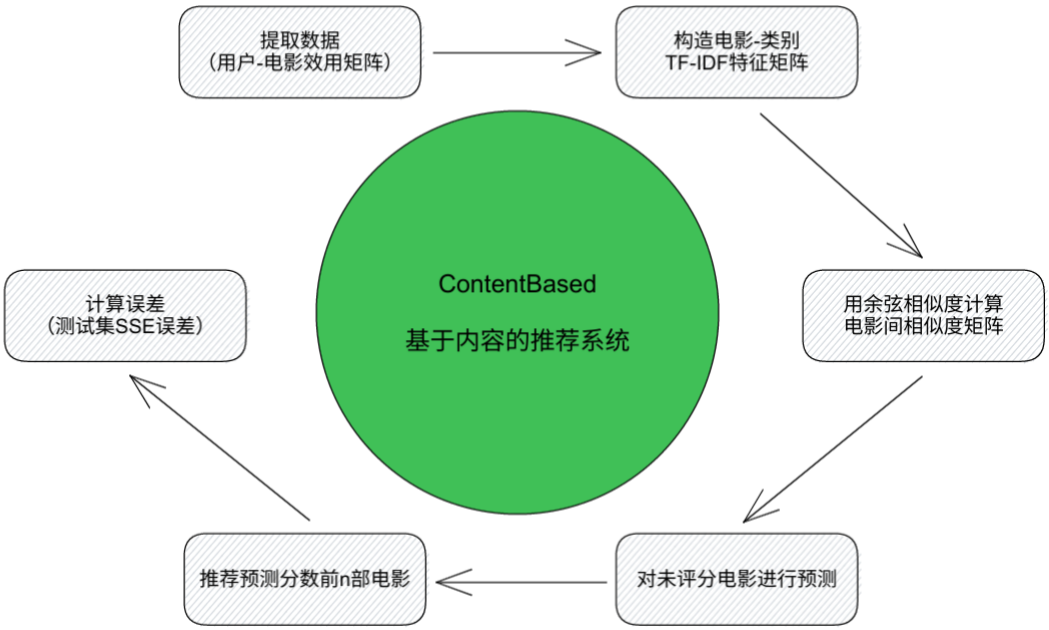

图 1.4 基于内容的推荐系统流程

基于内容的推荐系统使用电影的类别标签作为考察的特征对象，计算 tf-idf 特征矩阵，得到电影间的相似度矩阵。TF-IDF 是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。

在本实验中，对于电影的种类标签 t 有平滑后的 tf-idf 计算公式：

$$tf-idf(t) = tf(t,d) * (idf(t,d)+1)$$
$$= tf(t,d) * (log\frac{n_d+1}{1+df(t,d)})$$

得到 tf-idf 特征矩阵后，对所有电影求余弦相似度，计算公式为：

$$cos(X,Y) = \frac{X \bullet Y}{|X| \bullet |Y|}$$

编写一个 Python 类 ContentBasedRecommendSystem 实现上述功能。

ContentBasedRecommendSystem 类具有以下字段：

- utility_mat：用户-电影效用矩阵，通过初始化参数传递；
- movies：电影列表，通过初始化参数传递；
- tfidf_mat：tf-idf 特征矩阵，通过初始化参数传递；
- movie_sim_mat：电影相似度矩阵；
- index_to_id：索引-电影序号映射；
- id_to_index：电影序号-索引映射。

ContentBasedRecommendSystem 类具有以下方法：

- get_movie_sim_mat：利用余弦相似度计算电影间相似度矩阵；
- get_predict_score：计算预测值；
- predict：预测指定用户对指定电影的评分；
- recommend：为指定用户推荐 n 部电影。

基于内容的推荐算法过程为：

①读取电影数据集，获得电影数据，计算 tfidf_matrix 特征矩阵；

②由给定的数据集 train_set.csv 计算用户-电影效用矩阵；

③用上述数据初始化 ContentBasedRecommendSystem 类，利用余弦相似度计算电影之间的相似度矩阵；

④读取测试数据集 test_set.csv，调用 predict 方法进行预测，记录预测评分；

⑤使用预测评分和标准评分计算 SSE；

⑥调用 recommend 方法为某个用户推荐电影。

（3）基于用户的协同过滤推荐系统（进阶版）

推荐系统中需要频繁比较两个集合的相似度，当集合中的元素个数较多的时候，传统方法计算过程的时间开销较大，故引入 MinHash 算法对效用矩阵进行降维，将 0.5~2.5 的评分置为 0，将 3.0~5.0 的评分置为 1，得到 01 特征矩阵。

采用随机数模拟的方式生成哈希函数，通过特征矩阵生成签名矩阵，利用签名矩阵计算得到相似度矩阵。在极大似然估计情况下，两个集合经随机排列转换后得到的两个最小哈希值相等的概率与两个集合的 Jaccard 相似度相等。

编写一个 Python 类 MinHashUserRecommendSystem 实现上述功能，该类继承自 UserBasedRecommendSystem 类。与基类相比新增的字段有：

- signature_mat：签名矩阵；
- minhash_sim_mat：MinHash 相似矩阵。

新增的方法有：

- get_signature_mat：生成签名矩阵；
- cal_minhash_sim：计算 MinHash 相似度；
- get_minhash_sim_mat：计算 MinHash 相似矩阵。

其余方法仅调整了部分参数的类型，功能与基类一致。

（4）基于内容的推荐系统（进阶版）

本模块中依然使用 MinHash 方法对特征矩阵进行降维，选取方式为如果该电影存在某特征值，则特征值为 1，否则为 0，从而得到 01 特征矩阵。

编写一个 Python 类 MinHashContentRecommendSystem 实现上述功能，该类继承自 ContentBasedRecommendSystem 类。与基类相比新增的字段有：

- signature_mat：签名矩阵；
- minhash_sim_mat：MinHash 相似矩阵。

新增的方法有：

- get_signature_mat：生成签名矩阵；
- cal_minhash_sim：计算 MinHash 相似度；
- get_minhash_sim_mat：计算 MinHash 相似矩阵。

其余方法仅调整了部分参数的类型，功能与基类一致。

1.3.2 遇到的问题及解决方式

（1）MinHash 法基于用户的协同过滤算法中未评分电影的处理

在进阶版基于用户的协同过滤算法实现中，任务书仅给出已评分电影的 01 化方式，对未评分电影没有详细说明。实验中我将这里以 0 处理，是考虑到 MinHash 过程中，我们在每一列中寻找“第一个 1”的序号，也就是用标签 1 区分特征，故未评分的电影可以认为不是特征，将其视作 0 是合理的。

（2）MinHash 法基于内容的推荐系统计算速度

在进阶版基于内容的推荐算法实现中，由于电影集合的大小在 10^4 量级，相似矩阵的规模较大，传统循环方式两两之间计算相似度的循环次数在 10^8 量级，耗时很长。在此我利用了 numpy 库的 broadcast 机制，计算相似度时以列为基本单位，每次选中一列计算它与其余列的相似度。复制原来的矩阵，利用 broadcast 机制将矩阵的每一列减去选中的这一列，然后在纵轴上求和（即 axis=0）。关键代码如图 1.5 所示。

```python
def cal_minhash_sim(self, idx: int) -> np.ndarray:
    """计算 MinHash 相似度

    Args:
        idx (int): MinHash 矩阵行序号

    Returns:
        np.ndarray: MinHash 相似矩阵的 idx 行
    """
    tmp_mat = self.signature_mat - \
        self.signature_mat[:, idx].reshape(-1, 1)
    res_mat = 1 - np.count_nonzero(tmp_mat, axis=0) / tmp_mat.shape[0]
    res_mat[res_mat == 1] = 0
    return res_mat
```

图 1.5 numpy 库 broadcast 机制加速代码

这样做的根据是两列如果存在相同的元素，相减之后会得到 0，对最终的和没有贡献，因此“0”元素在和中所占的比例就是两列之间的相似度。通过这种方式，将原本的双重循环降为了单重循环，并且全部的计算过程都通过 numpy 库函数完成，效率相比之前提高了不少。

同样也可以在基于用户的推荐系统中采用这种算法，但由于本身相似矩阵的

数量级不大，最终优化效果不太明显。

1.3.3 实验测试与结果分析

（1）基于用户的协同过滤推荐系统

运行 user_based.py，输出结果如图 1.6 所示。

```
基础版 SSE =  72.48753164275061
总时间: 6.312 s.
对用户 300 推荐如下电影:
Movie   Score
---------------
318     4.694
858     4.526
1221    4.524
1197    4.485
50      4.476
1136    4.463
260     4.451
1196    4.447
4993    4.411
527     4.387
```

```
MinHash 版 SSE =  82.84549629665152
总时间: 0.990 s.
对用户 300 推荐如下电影:
Movie   Score
---------------
318     4.623
50      4.414
2571    4.329
7153    4.315
608     4.288
527     4.276
1270    4.264
589     4.250
2028    4.250
1210    4.240
```

图 1.6 基于用户的协同过滤推荐系统结果

可以看出，采用 MinHash 算法大大加速了计算，但也牺牲了一部分准确性。

对于基于用户的协同过滤算法，选择前多少个最相似的用户进行电影评分，会明显影响到最终预测结果的 SSE 值。选择不同 k 值进行预测结果的比较，一个较为合适的区间为 $[50, 300]$，得到如图 1.7 所示的结果，绘图结果如图 1.8 所示。

```
sse_array:
[82.4171908  78.5788054  77.61413611 75.90954068 74.9244316  73.90166181
 72.48753164 73.84637532 73.79534756 72.61389856 72.99254633 73.04325417
 72.6137007  73.95746697 74.09599844 75.14806047 75.2527888  75.20588612
 75.26249006 76.18021252 76.30079161 76.47764707 76.31957285 77.15668753
 77.35866318 77.31053329]
```

图 1.7 不同 k 值下的计算得到的 SSE 值

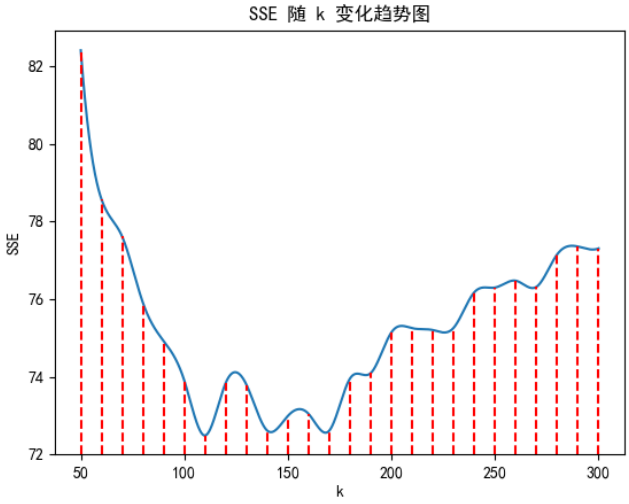

图 1.8 SSE 随 k 变化趋势图

从 SSE 随 k 的变化趋势可以看出，k 值选择过小是不合适的，因为这样具有较大的随机性，很可能有电影没有被选择的人评分过而被预测为 0 分；k 值选择过大也不合适，因为从某个阶段开始预测结果受到相似度不高用户评分结果影响带来偏差会逐渐增大。从顶点来看，实验中选取的默认值 k=110 是一个比较合理的数值。

（2）基于内容的推荐系统

运行 content_based.py，输出结果如图 1.9 所示。

```
基础版 SSE =  67.16953413803508
总时间: 0.361 s.
对用户 83 推荐如下电影:
Movie       Title                                          Score
-------------------------------------------------------------
37          Across the Sea of Time (1995)                  4.516
1797        Everest (1998)                                 4.516
4453        Michael Jordan to the Max (2000)               4.516
4458        Africa: The Serengeti (1994)                   4.516
4459        Alaska: Spirit of the Wild (1997)              4.516
4861        Mission to Mir (1997)                          4.516
383         Wyatt Earp (1994)                              4.455
416         Bad Girls (1994)                               4.455
1181        Shooter, The (1997)                            4.455
2401        Pale Rider (1985)                              4.455
```

```
Minhash 版 SSE =  68.22759108140042
总时间: 3.854 s.
对用户 83 推荐如下电影:
Movie       Title                                          Score
-------------------------------------------------------------
383         Wyatt Earp (1994)                              4.600
416         Bad Girls (1994)                               4.600
1181        Shooter, The (1997)                            4.600
2401        Pale Rider (1985)                              4.600
2921        High Plains Drifter (1973)                     4.600
3025        Rough Night in Jericho (1967)                  4.600
3074        Jeremiah Johnson (1972)                        4.600
3487        El Dorado (1966)                               4.600
3806        Mackenna's Gold (1969)                         4.600
4045        Breakheart Pass (1975)                         4.600
```

图 1.9 基于内容的推荐系统结果

由于 MinHash 版本进行了较多预处理计算，在时间上的优势不太明显。

从 SSE 值可以看出基于内容的推荐系统在给定数据集上整体优于基于用户的协同过滤推荐系统。这可能是因为在本数据集中，电影数量（9000+）远多于用户数量（300+），而且每个用户评分的电影数量在 10~20 左右，采用基于用户推荐时会有很多没有评分的电影由于缺少样本评分而误差较大，而基于内容的推荐则没有这个问题。并且对于电影推荐，在区分电影种类标签的情况下基于内容推荐可能更加符合常理，人们往往更多按照自己的喜好对某些类型的电影有倾向性。

选择不同哈希函数的数量对于 MinHash 算法的准确性有一定影响。这里选

择进阶版基于内容的推荐算法进行分析，选取不同哈希函数的个数 n 进行测试，得到如图 1.10 所示的结果，绘图结果如图 1.11 所示。

```
sse_array:
[70.11560002 69.05031909 68.57611876 68.05179927 67.80487567 68.22759108
 68.33371527 68.0970423  68.00583757 68.25537641 68.18385807]
```

图 1.10 不同 n 值下计算得到的 SSE 值

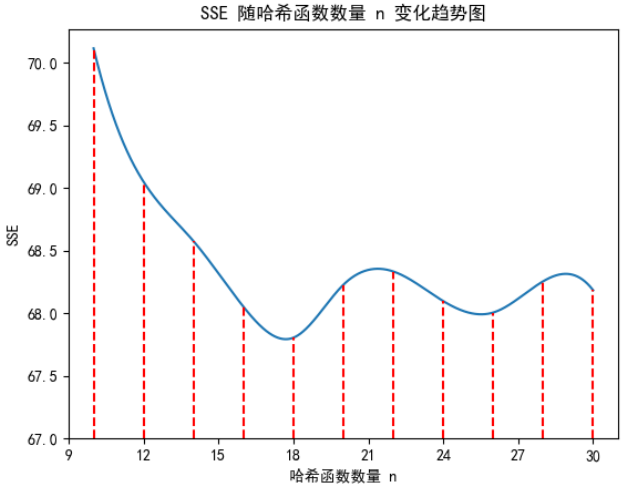

图 1.11 SSE 随哈希函数数量 n 变化趋势图

从 SSE 随 n 的变化趋势可以看出，整体上 SSE 值随着哈希函数数量的增加而呈现下降的趋势，这符合极大似然估计的理论。

1.4 实验总结

本次实验实现了基于用户的协同过滤推荐算法和基于内容的推荐算法，并在此基础上采用 MinHash 算法对特征矩阵进行降维，简化了计算过程。

在 pearson 相似度和余弦相似度计算过程中，调用第三方库函数能够极大加快程序的运行速度，同时能够简化程序的编写。在计算 Jaccard 相似度中，充分利用 numpy 的机制也能够极大加速程序运行，这可能是因为这部分代码被解释为更接近机器级别的语言运行。

总的来说，通过这次实验我进一步感受到了数据挖掘算法的强大，对数据挖掘、数据分析和推荐系统有了更加深入的认识。