



南開大學
Nankai University

网络空间安全学院
大数据计算与应用期末报告

推荐系统的设计与实现

姓名：杨鑫 吴晨宇

学号：2011028 2012023

专业：信息安全 信息安全

2023 年 6 月 8 日

目录

1 概述	2
1.1 背景	2
1.2 常用算法	2
1.2.1 基于流行度的推荐算法	2
1.2.2 基于用户的协同过滤推荐算法	2
1.2.3 基于物品的协同过滤推荐算法	2
1.2.4 基于矩阵分解的推荐算法	3
1.2.5 基于内容的推荐算法	4
1.3 评测指标	4
1.4 实验环境	5
2 数据集分析	5
3 基于物品的协同过滤算法实现	9
4 基于 biasSvd 矩阵分解的 CF 算法实现	11
5 基于内容的推荐算法实现	14
6 实验结果	17
6.1 基于 BiasSVD 矩阵分解推荐算法结果	17
6.2 基于内容的推荐算法结果	20
6.3 实验总结	21
7 附录	22
7.1 小组分工	22
7.2 仓库	22

1 概述

1.1 背景

推荐系统是一种信息过滤技术，旨在根据用户的个人偏好和行为，提供个性化的推荐内容。推荐系统的出现是为了解决信息过载问题和帮助用户在海量信息中发现他们感兴趣的内容。

随着互联网的快速发展，人们面临越来越多的信息，在这个信息爆炸的时代，用户往往感到困惑，很难找到他们真正感兴趣的内容。同时每个人的喜好和兴趣都不同，传统的广播和电视节目无法满足个人的需求，因为它们是广泛的、普遍的内容，无法根据个人的喜好进行定制。另外随着社交媒体、在线购物和数字娱乐的普及，用户的互动和参与度越来越高，用户生成的内容和用户行为数据成为了个性化推荐的重要依据。最后个性化推荐系统不仅能提升用户体验，还能帮助企业提高销售额和利润。通过向用户推荐他们可能感兴趣的产品和服务，企业能够增加销售量，并建立更紧密的客户关系。

基于以上的原因，推荐系统的目标是根据用户的历史行为、个人偏好和与其他用户的相似性等因素，预测用户可能喜欢的内容，并向他们提供个性化的推荐。推荐系统的实现涉及数据分析、机器学习和算法优化等技术领域，目前已广泛应用于电子商务、社交媒体、音乐和视频流媒体等领域。

1.2 常用算法

1.2.1 基于流行度的推荐算法

基于流行度的推荐算法简单粗暴，类似于各大新闻、热榜等，根据访问次数和独立访客以及日均访问量或分享率等数据按某种热度排序来推荐给用户。优点是算法简单，适合刚注册的用户；缺点是无法针对用户提供个性化的推荐。

1.2.2 基于用户的协同过滤推荐算法

该算法是当目标用户需要推荐时，可以先通过兴趣、爱好或行为习惯找到与他相似的其他用户，然后把哪些与目标用户相似的用户喜欢的并且目标用户没有浏览过的物品推荐给目标用户。基于用户的 CF 原理如下：

1. 分析每个用户对物品的评价，通过购买记录、收藏等得到用户的隐性评分。
2. 根据用户对物品的评分计算得到所有用户之间的相似度。
3. 选出与目标用户最相似的 K 个用户；
4. 将这 K 个用户评分最高且目标用户没有浏览过的物品推荐给目标用户。

此算法的优点是较为社会化，给目标用户推荐那些和他有共同兴趣的用户喜欢的物品，适用于物品比用户多、物品时效性比较强的情形。缺点是在很多时候很多用户两两之间的共同评分仅有几个，即用户之间重合度不高，同时仅有的共同评分的物品往往是一些很常见的物品导致相似度计算不准确，同时用户是随时间多变的，用户之间的距离可能变得很快，因为这种离线的算法难以更新推荐结果。

1.2.3 基于物品的协同过滤推荐算法

当一个用户需要个性化推荐时，根据先前已经评价过的物品，然后选择评分最高的物品计算与其他没有买过物品的相似度，然后推荐与之相似度最大的物品。基于物品的 CF 原理如下：

1. 分析各个用户对物品的评分；
2. 根据评分等分析得到所有物品之间的相似度；
3. 对于目标用户评价高的物品，找出与之相似度最高的 K 个物品；
4. 将这 K 个物品中目标用户没有浏览过的物品推荐给目标用户。

此算法的优点是较为个性，为用户推荐那些和他之前喜欢的物品类似的物品，推荐的物品一般都满足目标用户的独特兴趣，同时物品之间的距离往往能保持稳定，可以进行离线计算。缺点是不同领域的最热门物品之间经常具有较高的相似度，而这种推荐结果可能并不是用户想要的，同时在物品冷启动、数据稀疏时的效果不佳。

1.2.4 基于矩阵分解的推荐算法

矩阵分解是指将一个矩阵分解成两个或者多个矩阵的乘积。对于用户-商品矩阵 (评分矩阵)，记为 R_{m*n} 可以将其分解成两个或者多个矩阵的乘积，假设分解成两个矩阵 P_{m*k} 和 Q_{k*n} ，我们要使得这两个矩阵的乘积能够还原原始的矩阵 R_{m*n} ：

$$R_{m*n} \approx P_{m*k} \times Q_{k*n} = \hat{R}_{m*n}$$

其中，矩阵 P_{m*k} 表示的是 m 个用户与 k 个主题之间的关系，而矩阵 Q_{k*n} 表示的是 k 个主题与 n 个商品之间的关系。

使用原始的评分矩阵 R_{m*n} 和重新构建的评分矩阵 \hat{R}_{m*n} 之间的误差的平均作为损失函数 (这样做的好处是可以方法较大的误差)，即：

$$e_{ij}^2 = (r_{i,j} - \hat{r}_{i,j})^2 = (r_{i,j} - \sum_{k=1}^K p_{i,k} q_{k,j})^2$$

加入 L2 正则化 (惩罚那些比较活跃的用户和物品)：

$$e_{i,j}^2 = (r_{i,j} - \sum_{k=1}^K p_{i,k} q_{k,j})^2 + \frac{\beta}{2} \sum_{k=1}^K (p_{i,k}^2 + q_{k,j}^2)$$

最后需要求解所有的非 '-' 项 (即评分不为空的地方) 的损失之和的最小值：

$$\min \quad loss = \sum_{r_{i,j} \neq -} e_{i,j}^2$$

然后利用梯度下降方法求解损失函数的负梯度：

$$\begin{aligned} \frac{\partial}{\partial p_{i,k}} E_{i,j}^2 &= -2 \left(r_{i,j} - \sum_{k=1}^K p_{i,k} q_{k,j} \right) q_{k,j} + \beta p_{i,k} = -2e_{i,j} q_{k,j} + \beta p_{i,k} \\ \frac{\partial}{\partial q_{k,j}} E_{i,j}^2 &= -2 \left(r_{i,j} - \sum_{k=1}^K p_{i,k} q_{k,j} \right) p_{i,k} + \beta q_{k,j} = -2e_{i,j} p_{i,k} + \beta q_{k,j} \end{aligned}$$

根据负梯度的方向更新变量：

$$p_{i,k}' = p_{i,k} - \alpha \left(\frac{\partial}{\partial p_{i,k}} e_{i,j}^2 + \beta p_{i,k} \right) = p_{i,k} + \alpha (2e_{i,j}q_{k,j} - \beta p_{i,k})$$

$$q_{k,j}' = q_{k,j} - \alpha \left(\frac{\partial}{\partial q_{k,j}} e_{i,j}^2 + \beta q_{k,j} \right) = q_{k,j} + \alpha (2e_{i,j}p_{i,k} - \beta q_{k,j})$$

然后利用上述过程可以得到矩阵 $P_{m \times k}$ 和 $Q_{k \times n}$ ，这样便可以为用户 i 和商品 j 进行打分：

$$\sum_{k=1}^K p_{i,k} q_{k,j}$$

同时矩阵分解还有改进的矩阵分解方法：

- traditional SVD
- FunkSVD(LFM)
- BiasSVD
- SVD++

本次实验的第二部分内容就是基于 BiasSVD 算法实现的推荐算法系统。具体的算法过程在下部分会进行说明。

使用矩阵的分解的推荐算法的优点是能够处理稀疏的用户-物品矩阵，能够填充缺失值并预测用户对未评价项目的兴趣，同时矩阵分解算法能够处理隐性反馈数据，即只有用户与项目之间的交互信息，如评分或点击次数，这使得算法能够推荐那些没有明确评分或反馈的项目。缺点是：矩阵分解算法在面对新用户或新项目时存在冷启动问题。由于缺乏新用户或项目的历史交互数据，算法无法准确预测其兴趣或推荐合适的项目，同时如果用户-项目矩阵中存在严重的数据偏斜，即某些项目或用户的交互次数明显多于其他项目或用户，矩阵分解算法可能会受到偏斜数据的影响，导致推荐结果不均衡。

1.2.5 基于内容的推荐算法

基于内容的推荐方法是非常直接的，它以物品的内容描述信息为依据来做出的推荐，本质上是基于对物品和用户自身的特征或属性的直接分析和计算。例如，假设已知电影 A 是一部喜剧，而恰巧我们得知某个用户喜欢看喜剧电影，那么我们基于这样的已知信息，就可以将电影 A 推荐给该用户。基于内容的推荐算法原理：

1. 根据 PGC/UGC 内容构建物品画像和用户画像；
2. 根据用户行为记录生成用户画像；
3. 根据用户画像从物品中寻找最匹配的 TOP-N 物品进行推荐。

本次实验的第三部分就是基于此算法进行实现的，具体的算法过程在该部分会进行详细介绍。

1.3 评测指标

对于评分预测常用的准确性评测指标是均方根误差 RMSE 和平均绝对误差 MAE。

RMSE: 均方根误差（对大的偏差更敏感）

$$RMSE = \sqrt{\frac{1}{|R|} \sum_{r_{ui} \in R} (r_{ui} - \hat{r}_{ui})^2}$$

MAE：平均绝对值误差

$$MAE = \frac{1}{|R|} \sum_{r_{ui} \in R} |r_{ui} - \hat{r}_{ui}|$$

注意：|R| 表示数据集的长度。在本次实验中使用了 RMSE 作为实验结果的评测指标。

1.4 实验环境

本次实验使用的语言及版本是 **python 3.7.5**, 使用远程 jupyter 环境进行开发, 然后转存为 python 文件进行结果的运行。项目实现过程中使用到的库函数为：

```

1  import csv                # 生成和处理 csv
2  import pandas             # 数据处理
3  import numpy              # 数据处理
4  import matplotlib        # 图像绘制
5  import timeit             # 计算耗时
6  import math               # 数据处理
7  from scipy.sparse import coo_matrix, csr_matrix # 数据处理
8  import random             # 初始矩阵

```

2 数据集分析

首先对给的文件进行统计分析，如图2.1所示

```

1  用户总数：19835
2  训练集商品总数：455705
3  属性集商品总数：507172
4  最大商品id：624960
5  无属性商品总数：40846
6  训练集评分总数：5001507
7  测试集评分总数：119010
8  训练集评分平均：49.50458011955197
9  商品属性1的平均：288394.8737647189
10 商品属性2的平均：272492.80033203727
11

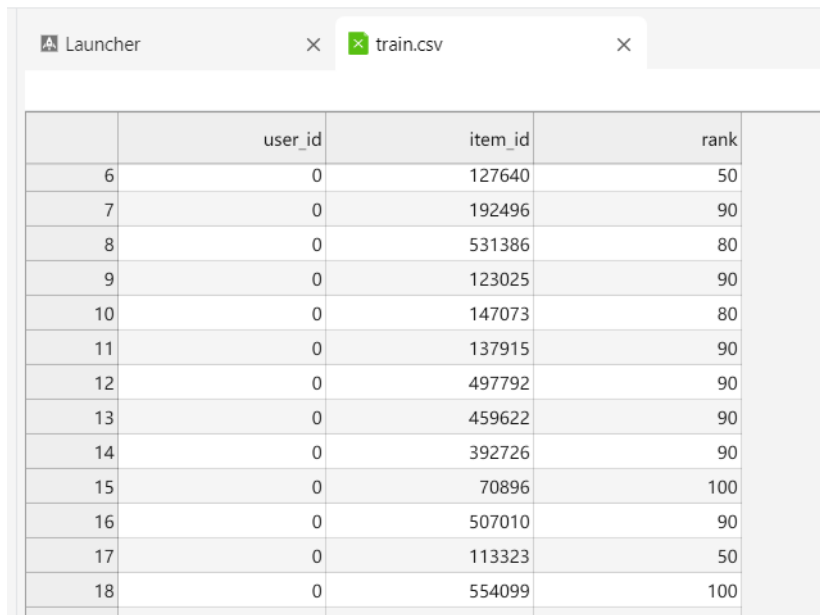
```

图 2.1: 统计结果文件

可以发现现在 train.txt、test.txt、itemAttribute.txt 记录的用户数为 19835 名，属性集商品数为 507172，在训练集中的评分总数为 5001507，在测试集评分总数为 119010，在训练集的平均评分为 49.50，商品属性 1 的平均为 288394，商品属性 2 的平均为 272492。由于属性集的最大商品 id 是 624960，因此在属性集中缺失一部分商品的属性数据。同时可得矩阵的稀疏性如下：

$$sparsity = 1 - 5001507 / (19835 * 624960) = 99.96\%$$

可以看到我们的数据集是极其稀疏的，同时为了方便我们进行数据处理生成 train.csv、test.csv、itemAttribute.csv，这里我们使用 csv 工具包，将我们原始数据集分别转为 csv 进行存储。比如 train.txt 的数据集如下图2.2所示：



	user_id	item_id	rank
6	0	127640	50
7	0	192496	90
8	0	531386	80
9	0	123025	90
10	0	147073	80
11	0	137915	90
12	0	497792	90
13	0	459622	90
14	0	392726	90
15	0	70896	100
16	0	507010	90
17	0	113323	50
18	0	554099	100

图 2.2: train.csv 文件部分数据

然后使用 pandas 工具包将数据导入到环境中：

```

1 import pandas as pd
2 import numpy as np
3
4 train_dataset = pd.read_csv(filepath_or_buffer=train_csvfile_path,
5                             sep=',', header=None, names=['user_id', 'item_id',
6                                                         'rank'],
7                             skiprows=1)
8 test_dataset = pd.read_csv(filepath_or_buffer=test_csvfile_path,
9                             sep=',', header=None, names=['user_id', 'item_id',
10                                                         'rank'],
11                             skiprows=1)
12 attr_dataset = pd.read_csv(filepath_or_buffer=attr_csvfile_path,
13                             sep=',', header=None, names=['item_id', 'attribute_1',
14                                                         'attribute_2'],
15                             skiprows=1)

```

查看 train_dataset 数据集的信息，如图2.3所示：

```
[5]: train_dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001507 entries, 0 to 5001506
Data columns (total 3 columns):
#   Column  Dtype
---  -
0    user_id  int64
1   item_id  int64
2    rank    int64
dtypes: int64(3)
memory usage: 114.5 MB
```

图 2.3: train_dataset 数据集

同时我们还是用 matplotlib 工具包，对实验数据进行进一步的可视化统计处理。用户评分的分布情况如图2.4所示：

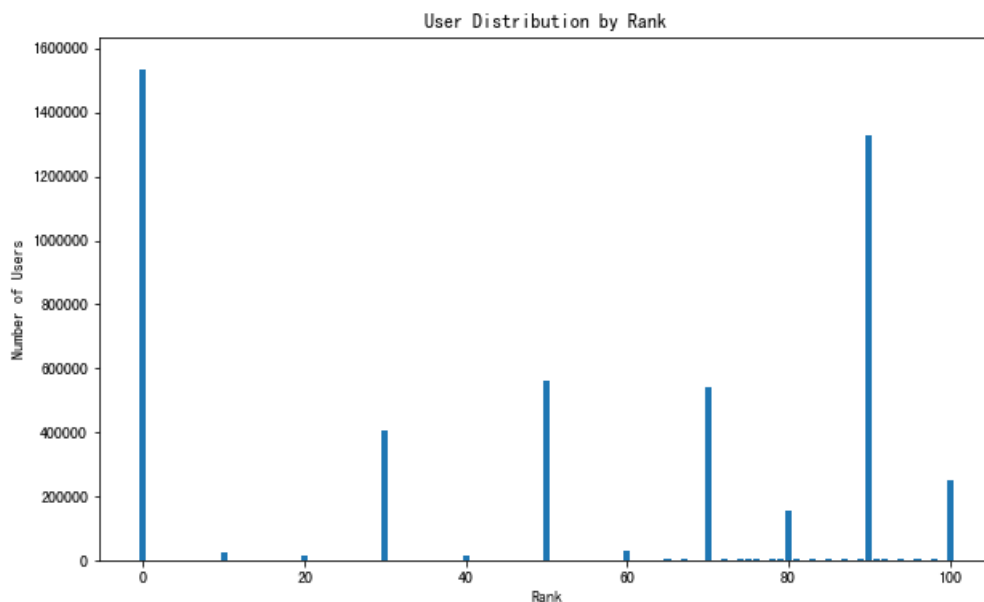


图 2.4: 用户评分打分情况

然后根据 train_dataset 分析得到用户打分平均值分布，可以发现平均值主要分布在 80 到 90 之间，意味着用户的打分总体来说是比较宽容的，如图2.5所示：

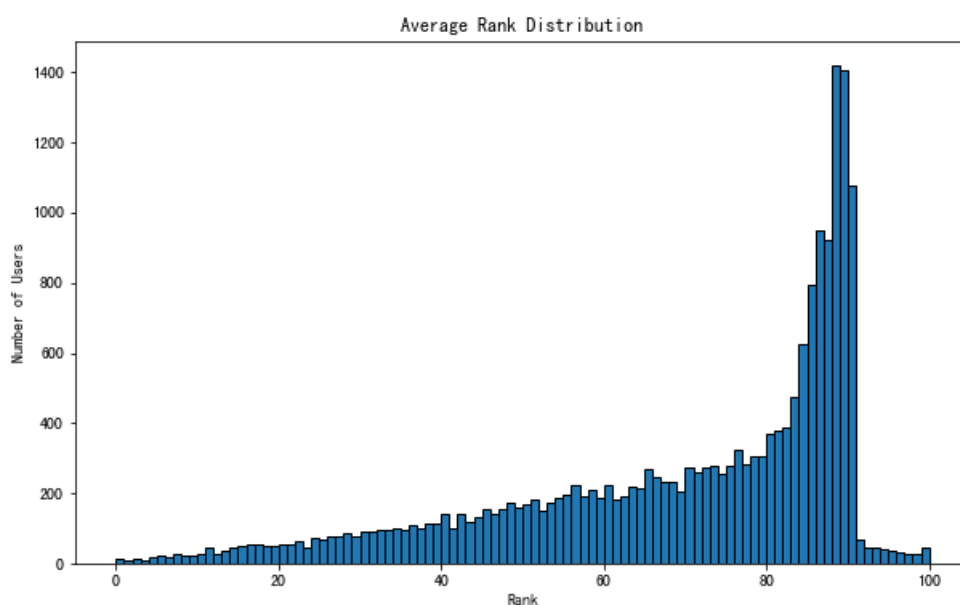


图 2.5: 用户平均打分分布情况

然后根据 `train_dataset` 分析得到物品被用户打分的平均分分布情况, 发现物品均分 (不包括 0 分) 主要分布于在 20-80 之间, 说明物品之间存在差异性, 这也使得我们实现推荐算法有了必要。如图2.6所示:

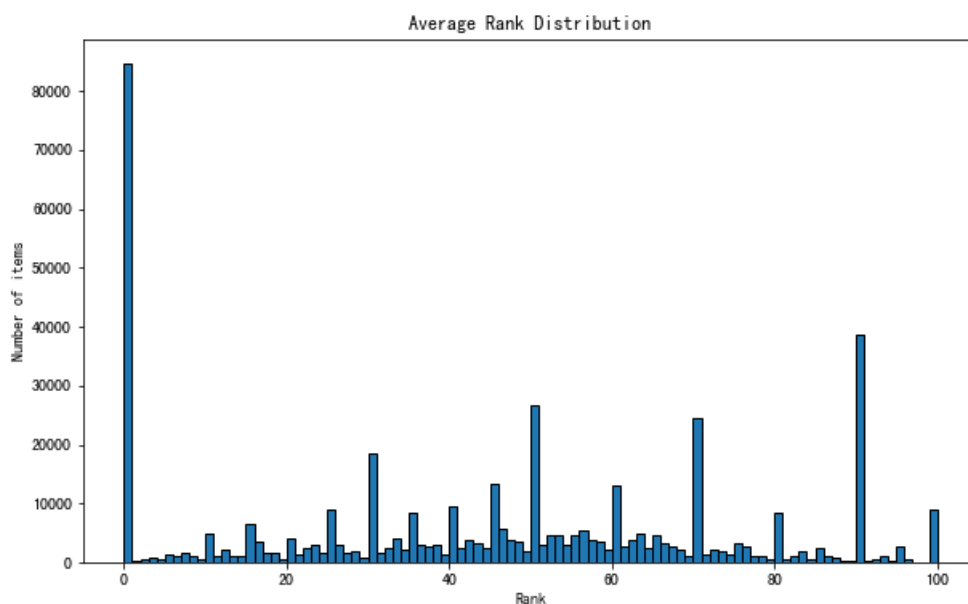


图 2.6: 物品平均被打分分布情况

3 基于物品的协同过滤算法实现

协同过滤 (collaborative filtering) 算法是最经典、最常用的推荐算法。其基本思想是收集用户偏好，找到相似的用户或物品，然后计算并推荐。基于物品的协同过滤算法的核心思想就是：给用户推荐那些和他们之前喜欢的物品相似的物品。主要可分为四步：

1. 分析各个用户对物品的评分；
2. 根据评分等分析得到所有物品之间的相似度；
3. 对于目标用户评价高的物品，找出与之相似度最高的 K 个物品；
4. 将这 K 个物品中目标用户没有浏览过的物品推荐给目标用户。

另外本次实验使用的相似度计算使用的是**同现相似度**，其公式如下：

$$i_j = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)||N(j)|}}$$

这个公式惩罚了物品 j 的权重，因此减轻了热门物品会和很多物品相似的可能性。另外为减小活跃用户对结果的影响，考虑 IUF 即用户活跃度对数的倒数的参数，认为活跃用户对物品相似度的贡献应该小于不活跃的用户。

$$w_{ij} = \frac{\sum_{u \in N(i) \cap N(j)} \frac{1}{\log 1 + |N(u)|}}{\sqrt{|N(i)||N(j)|}}$$

为便于计算，还需要进一步将相似度矩阵归一化 $w'_{ij} = \frac{w_{ij}}{\max_j w_{ij}}$

在进行相似度计算的时候需要进行数据的预处理工作以方便我们的计算，这里因为是超级稀疏的矩阵，因此我们这里需要将其转为 dok_matrix 稀疏矩阵的存储方式进行存储，同时我们需要定义一些在过程中需要用到的变量，

```

1  class ItemBasedCF():
2      def __init__(self):
3          self.n_sim_item = 20 #相似物品数
4          self.n_rec_item = 10 #推荐物品数
5          self.item_sim_matrix = dok_matrix((624961, 624961), dtype=np.float32)
6              #相似矩阵
7          self.item_popular = {} #每个物品被评价次数
8          self.item_count = 0 #物品数
9
10         def get_dataset(self, train_dataset):
11             self.user_group =
12                 train_dataset.groupby('user_id').agg([list])[['item_id', 'rank']]
13             self.user_ids = train_dataset['user_id'].astype(int)
14             self.item_ids = train_dataset['item_id'].astype(int)
15             self.ranks = train_dataset['rank'].astype(int)
16             self.user_items = train_dataset.groupby('user_id')['item_id'].agg(list)
17             print('Load dataset success!')

```

接下来就是计算物品之间的相似度，其主要步骤如下：

1. 循环读取每个用户及评价过的物品，并统计每个物品被看过的次数，以及物品的总数；

2. 计算矩阵 C , $C[i][j]$ 表示同时喜欢看物品 i 和 j 的用户数, 并考虑对活跃用户的惩罚;
3. 根据同现相似度计算物品的相似性;
4. 进行归一化处理。

```

1  def calc_item_sim(self):
2      start = timeit.default_timer()
3      for iid in self.item_ids:
4          if iid not in self.item_popular:
5              self.item_popular[iid] = 0
6              self.item_popular[iid] += 1
7      self.item_count = len(self.item_popular)
8      print('item_popular success!')
9      print(self.item_count)
10
11     # sim_vector = self.item_sim_matrix[i1]
12
13     for user, items in self.user_items.iteritems():
14         # 计算项目数量的倒数的对数值
15         log_item_count = 1 / math.log(1 + len(items))
16         for i1 in items:
17             for i2 in items:
18                 if i1 == i2:
19                     continue
20                 self.item_sim_matrix[i1, i2] += log_item_count
21     print('Build co-rated users matrix success!')
22
23     # 获取稀疏矩阵的元素迭代器
24     item_iterator = self.item_sim_matrix.itemset()
25     for (i1, i2), value in item_iterator:
26         # 计算相似度值
27         similarity = value / math.sqrt(self.item_popular[i1] *
28                                         self.item_popular[i2])
29         # 更新稀疏矩阵的元素
30         self.item_sim_matrix[i1, i2] = similarity
31     print('Calculate item similarity matrix success!')
32
33     # 初始化最大值
34     max_w = 0
35     # 遍历稀疏矩阵的元素
36     for (i1, i2), value in self.item_sim_matrix.itemset():
37         # 更新最大值
38         if value > max_w:
39             max_w = value
40         # 计算相似度值
41         similarity = value / math.sqrt(self.item_popular[i1] *
42                                         self.item_popular[i2])
43         # 更新稀疏矩阵的元素
44         self.item_sim_matrix[i1, i2] = similarity

```

```

43     # 归一化矩阵
44     self.item_sim_matrix /= max_w
45     all_time = end - start
46     print('Time cost is : %fs' % all_time)

```

然后得到相似度矩阵后，我们可以针对目标用户 U，找到 K 个相似的物品，并推荐 N 个物品，如果用户评价过该物品则不推荐：

```

1  def recommend(self, user):
2      K = self.n_sim_item
3      N = self.n_rec_item
4      rank = {}
5      used_items = self.user_items[user]
6      for item, rating in used_items.items():
7          for related_item, w in sorted(self.item_sim_matrix[item].items(),
8                                         key=itemgetter(1), reverse=True)[:K]:
9              if related_item in used_items:
10                 continue
11             rank.setdefault(related_item, 0)
12             rank[related_item] += w*float(rating)
13     return sorted(rank.items(), key=itemgetter(1), reverse=True)[0:N]

```

然后就可以进行训练了，但是由于数据的规模较大，这里在 8G8 核的 GPU 上无法整个进行训练，因此无法进行结果验证。

4 基于 biasSvd 矩阵分解的 CF 算法实现

在前面我们提到了 SVD 算法的基本原理，FunkSVD(LFM) 算法是在 SVD 的基础上不再将矩阵分解为 3 个矩阵，而是分解为 2 个矩阵（用户-隐含特征矩阵，物品-隐含特征矩阵），而 BiasSVD 是 FunkSVD 一个相对成功的变形版本，即带有偏置项的 SVD 分解。

$$\arg \min_{p_i, q_j} \sum_{i,j} (m_{ij} - \mu - b_i - b_j - q_j^T p_i)^2 + \lambda (\|p_i\|_2^2 + \|q_j\|_2^2 + \|b_i\|_2^2 + \|b_j\|_2^2)$$

利用 BiasSvd 预测用户对物品的评分，k 表示隐含特征数量：

$$\hat{r}_{ui} = \mu + b_u + b_i + p_{uk} \cdot q_{ki} = \mu + b_u + b_i + \sum_{k=1}^k p_{uk} q_{ik}$$

最优化函数都可以通过梯度下降或者随机梯度下降法来寻求最优解。BiasSVD 代码的实现如下：

```

1  class BiasSvd(object):
2      def __init__(self, alpha, reg_p, reg_q, reg_bu, reg_bi,
3                  number_LatentFactors=10, number_epochs=10, columns=["user_id", "item_id",
4                  "rank"]):
5          """
6          初始化
7          """
8          self.alpha = alpha # 学习率

```

```

7         self.reg_p = reg_p
8         self.reg_q = reg_q
9         self.reg_bu = reg_bu
10        self.reg_bi = reg_bi
11        self.number_LatentFactors = number_LatentFactors # 隐式类别数量
12        self.number_epochs = number_epochs
13        self.columns = columns
14        self.P = None
15        self.Q = None
16
17    def train(self, dataset):
18        self.dataset = pd.DataFrame(dataset)
19        self.users_ratings =
20            dataset.groupby(self.columns[0]).agg([list])[self.columns[1],
21            self.columns[2]]
22        self.items_ratings =
23            dataset.groupby(self.columns[1]).agg([list])[self.columns[0],
24            self.columns[2]]
25        self.globalMean = self.dataset[self.columns[2]].mean()
26
27        self.P, self.Q, self.bu, self.bi = self.sgd()
28
29    def _init_matrix(self):
30        """
31        初始化P和Q矩阵，同时为设置0，1之间的随机值作为初始值
32        """
33        # User-LF
34        P = dict(zip(
35            self.users_ratings.index,
36            np.random.rand(len(self.users_ratings),
37                            self.number_LatentFactors).astype(np.float32)
38        ))
39        # Item-LF
40        Q = dict(zip(
41            self.items_ratings.index,
42            np.random.rand(len(self.items_ratings),
43                            self.number_LatentFactors).astype(np.float32)
44        ))
45        return P, Q
46
47    def sgd(self):
48        """
49        使用随机梯度下降，优化结果
50        """
51        P, Q = self._init_matrix()
52
53        # 初始化bu、bi的值，全部设为0
54        bu = dict(zip(self.users_ratings.index,
55                      np.zeros(len(self.users_ratings))))

```

```

49         bi = dict(zip(self.items_ratings.index,
50                        np.zeros(len(self.items_ratings))))
51
52     for i in range(self.number_epochs):
53         print("iter %d"%i)
54         error_list = []
55         for uid, iid, r_ui in self.dataset.itertuples(index=False):
56             v_pu = P[uid]
57             v_qi = Q[iid]
58             err = np.float32(r_ui - self.globalMean - bu[uid] - bi[iid] -
59                               np.dot(v_pu, v_qi))
60
61             v_pu += self.alpha * (err * v_qi - self.reg_p * v_pu)
62             v_qi += self.alpha * (err * v_pu - self.reg_q * v_qi)
63
64             P[uid] = v_pu
65             Q[iid] = v_qi
66
67             bu[uid] += self.alpha * (err - self.reg_bu * bu[uid])
68             bi[iid] += self.alpha * (err - self.reg_bi * bi[iid])
69
70             error_list.append(err ** 2)
71
72         self.alpha = self.alpha * 0.93
73         print("mase = ", np.sqrt(np.mean(error_list)))
74
75     return P, Q, bu, bi
76
77 def test(self, testset):
78     """
79     测试函数
80     """
81     predict_rating = []
82     for uid, iid, real_rating in testset.itertuples(index=False):
83         try:
84             pred_rating = self.predict(uid, iid)
85             predict_rating.append(pred_rating)
86         except Exception as e:
87             print(e)
88     result = pd.DataFrame()
89     result['user_id'] = testset['user_id']
90     result['item_id'] = testset['item_id']
91     result['pred'] = predict_rating
92     result.to_csv('./data/result.csv', index=False)
93     print("Save success!")
94
95 def predict(self, uid, iid):
96     """
97     预测评分
98     """

```

```

96         if uid not in self.users_ratings.index or iid not in
97            self.items_ratings.index:
98             return self.globalMean
99
100         p_u = self.P[uid]
101         q_i = self.Q[iid]
102
103         return self.globalMean + self.bu[uid] + self.bi[iid] + np.dot(p_u, q_i)

```

以上实现过程简化为算法流程图如图4.7所示：

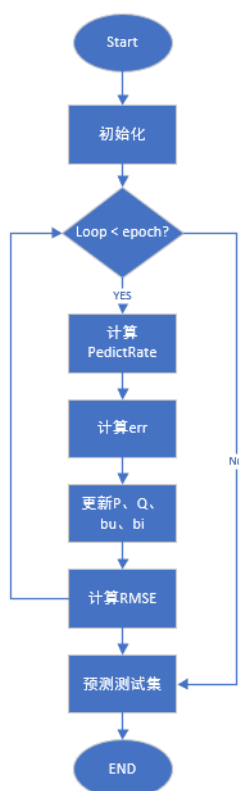


图 4.7: BiasSVD 算法流程图

5 基于内容的推荐算法实现

所谓基于内容的推荐算法 (Content-Based Recommendations) 是基于标的物相关信息、用户相关信息及用户对标的物的操作行为来构建推荐算法模型，为用户提供推荐服务。这里的标的物相关信息可以是对标的物文字描述的 metadata 信息、标签、用户评论、人工标注的信息等。用户相关信息是指人口统计学信息 (如年龄、性别、偏好、地域、收入等等)。用户对标的物的操作行为可以是评论、收藏、点赞、观看、浏览、点击、加购物车、购买等。基于内容的推荐算法一般只依赖于用户自身的行为为用户提供推荐，不涉及到其他用户的行为。

基于内容的推荐算法的基本原理是根据用户的历史行为，获得用户的兴趣偏好，为用户推荐跟他的兴趣偏好相似的标的物。具体的代码实现主要通过几个步骤。

第一步我们需要处理数据，获得包含所有物品的平均分：

```

1  # 包含所有物品平均分的字典grouped_data
2  def get_avg():
3      '''
4      获得平均分
5      '''
6      avg_data = train_dataset.groupby('item_id')['rank'].mean()
7      return avg_data

```

然后我们需要利用属性表得到所有物品的属性，这里因为有很多 None，因此我们将 None 的属性设置为 0，设置为随机的属性代表用户喜欢随机的东西，在本次实验中一个物品都有两个属性 (None 也算一个属性)，且设置每个属性分别占 0.5 比例。

```

1  # 得到存储所有物品种类的字典item_cast
2  def get_item_cast():
3      '''
4      得到所有种类
5      '''
6      attribute_1_values = attr_dataset['attribute_1'].unique().tolist()
7      attribute_2_values = attr_dataset['attribute_2'].unique().tolist()
8      # 合并两个列表并去重
9      item_cast = list(set(attribute_1_values + attribute_2_values))
10     return item_cast

```

然后我们需要获得一个种类下按照评分降序排列的物品列表，在完成上述两步后我们得到存储物品平均分的字典和存储物品种类的字典，然后我们根据物品种类的字典分别得到在该种类下的所有物品的平均分排序结果：

```

1  # 得到所有种类下排名前100的物品列表
2  def get_item_sort(avg_data, item_cast):
3      '''
4      得到种类前100
5      '''
6      item_sort = {}
7      merged_data = pd.merge(avg_data, attr_dataset, on='item_id')
8      for attribute in item_cast:
9          item_sort[attribute] = []
10         # 筛选出该种类的物品，并按照平均分进行降序排序)
11         item_sort[attribute].append(merged_data[merged_data['attribute_1'] ==
12         attribute].sort_values(by='rank', ascending=False)['item_id'])
12         item_sort[attribute].append(merged_data[merged_data['attribute_2'] ==
13         attribute].sort_values(by='rank', ascending=False)['item_id'])
13     return item_sort

```

数据处理完成后在对用户推荐之前，我们需要获得用户之前的偏好信息。偏好信息的评分计算如下：分值 = 用户评分 * 0.5。比如用户对物品 A 打了 80 分，该物品属于 B 和 C 类，那么对 B 和 C 的喜好程度上升 40。这里注意原来将 0 排除在外了，但是实际计算过程中发现属性为 0 去掉后严重影响了推荐的结果，因为对于绝大多数的用户评分后的物品的属性基本都是空的，因此这里还是将 0 当

成随机属性加入，当属于 0 属性的集的时候类似于热榜即推荐大多数用户评分过的物品。

```

1  # 得到用户的偏好
2  def get_user_like():
3      '''
4      得到用户最喜欢的几个种类
5      '''
6      user_like = {}
7      user_items = train_dataset.groupby('user_id')['item_id'].agg(list)
8      user_ratings = train_dataset.groupby('user_id')['rank'].agg(list)
9      user_count = len(user_items)
10     for uid in range(0, user_count):
11         user_like[uid] = {}
12         index = 0
13         for iid in user_items[uid]:
14             #找到物品属于的种类
15             attr = attr_dataset.iloc[iid][1:].tolist()
16             for a in attr:
17                 if a not in user_like[uid]:
18                     # print(uid, iid)
19                     user_like[uid][a] = 0
20                     user_like[uid][a] += user_ratings[uid][index] * 0.5
21                     index += 1
22     return user_like

```

最后就是得到用户最喜欢的前两个种类，然后根据用户 id，只需要查询用户偏好的种类，按照此种类查找所有物品平均分排序字典获得排名靠前的字典即可。

```

1  #对用户进行推荐
2  def recom(user_like, user_id, item_sort):
3      topk = 2
4      # 对用户喜欢的属性进行排序
5      for uid in user_id:
6          sorted_keys = sorted(user_like[uid], key=user_like[uid].get, reverse=True)
7          key = 0
8          if len(sorted_keys) != 0:
9              if sorted_keys[0] == 0:
10                 if len(sorted_keys) > 1:
11                     key = sorted_keys[1]
12                     print("该用户最喜欢种类", sorted_keys[1])
13                 else:
14                     key = sorted_keys[0]
15                     print("该用户最喜欢种类", sorted_keys[0])
16             else:
17                 key = sorted_keys[0]
18                 print("该用户最喜欢种类", sorted_keys[0])
19             #然后对其开始推荐
20             print("为用户", uid, "进行推荐: ")
21

```

```

22         for i in range(topk):
23             if i in item_sort[key][0]:
24                 print(item_sort[key][0][i])
25             elif i in item_sort[key][1]:
26                 print(item_sort[key][1][i])
27             else:
28                 print("无法推荐")
29     else:
30         print("无法推荐!")

```

6 实验结果

6.1 基于 BiasSVD 矩阵分解推荐算法结果

通过设置学习率为 0.0003、以及 p、q、bu、bi、k、epochs 分别设置为 0.00001、0.00001、0.01、0.01、10、30(经过测试在该条件下效果最好)，主要这里 p、q 的参数不能设置过大如果设置过大会导致浮点数计算溢出。

```

1  bsvd = BiasSvd(0.0003, 0.00001, 0.00001, 0.01, 0.01, 10, 30)
2  bsvd.train(train_dataset)

```

得到的训练结果如下，可以发现到最后迭代 30 次后 MASE 从原来的 32.10 降低到 21.01，如图6.8所示：

```

mase -----
iter 19
mase = 21.755358580649315
iter 20
mase = 21.644987700678854
iter 21
mase = 21.545385733629214
iter 22
mase = 21.455261948245944
iter 23
mase = 21.373517841453353
iter 24
mase = 21.29921270214835
iter 25
mase = 21.231536168979012
iter 26
mase = 21.1697865294105
iter 27
mase = 21.113353028532458
iter 28
mase = 21.06170175894698
iter 29
mase = 21.014363884290702

```

图 6.8: 训练结果图

然后得到 MASE 随时间变化的下降图如图6.9所示：

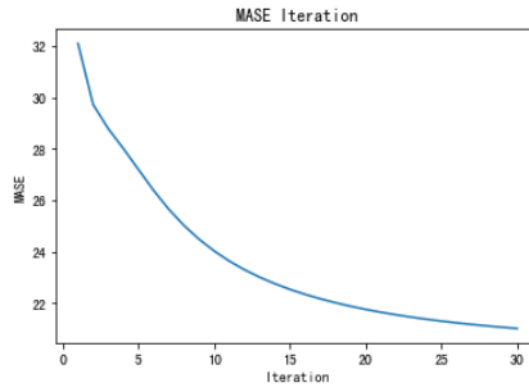


图 6.9: MASE 随时间变化图

然后调用 `test` 函数测试测试集并将结果写入 (这里是 csv 文件, 后面会将其转为 txt), 如图6.10所示:

	user_id	item_id	pred
1	0	208031	79.38511202160103
2	0	193714	55.64289905691143
3	0	393064	64.058644389291
4	0	207030	89.64142033861698
5	0	112040	59.10847516917589
6	0	464229	71.30240968727864
7	1	55971	88.90677002548841
8	1	583090	84.93282019238481
9	1	180171	91.88141196194567
10	1	617646	90.0144989570915
11	1	175835	90.50014976996688
12	1	553890	90.38875645743556
13	2	48916	79.72700171164117
14	2	238557	43.7892147286311
15	2	61148	9.853573456263845
16	2	378073	18.929706925256227
17	2	17863	36.7653227191807

图 6.10: test 数据集结果

同时将模型将 `pickle` 模块进行保存, 得到 `model.pkl`:

```

1 # 将对象保存到文件
2 with open('./results/model.pkl', 'wb') as f:
3     pickle.dump((self.P, self.Q, self.bu, self.bi), f)

```

然后下次我们就可以直接将模型拿出来使用而不用重新训练了, 下面是进行预测的代码:

```

1 attr_file_path = './data/itemAttribute.txt'
2 test_file_path = './data/test.txt'
3 train_file_path = './data/train.txt'
4 attr_csvfile_path = './data/attr.csv'
5 train_csvfile_path = './data/train.csv'
6 test_csvfile_path = './data/test.csv'
7 #将txt文档以csv的形式存储, 方便操作
8
9 #将itemAttribute转为csv格式
10 import csv

```

```

11 import pandas as pd
12 import numpy as np
13
14 train_dataset = pd.read_csv(filepath_or_buffer=train_csvfile_path,
15                             sep=',', header=None, names=['user_id', 'item_id',
16                                                         'rank'],
17                             skiprows=1)
18 test_dataset = pd.read_csv(filepath_or_buffer=test_csvfile_path,
19                             sep=',', header=None, names=['user_id', 'item_id',
20                                                         'rank'],
21                             skiprows=1)
22 attr_dataset = pd.read_csv(filepath_or_buffer=attr_csvfile_path,
23                             sep=',', header=None, names=['item_id',
24                                                         'attribute_1', 'attribute_2'],
25                             skiprows=1)
26
27 import matplotlib.pyplot as plt
28 import matplotlib.ticker as ticker
29
30 import numpy as np
31 import pandas as pd
32 from scipy.io import loadmat
33 from scipy.sparse import coo_matrix
34 import math
35 import pickle
36 import random
37
38 columns=["user_id", "item_id", "rank"]
39 dataset = pd.DataFrame(train_dataset)
40 users_ratings = dataset.groupby(columns[0]).agg([list])[columns[1], columns[2]]
41 items_ratings = dataset.groupby(columns[1]).agg([list])[columns[0], columns[2]]
42 globalMean = dataset[columns[2]].mean()
43
44 # 从文件加载对象
45 with open('./results/model.pkl', 'rb') as f:
46     P, Q, bu, bi = pickle.load(f)
47
48 def predict(uid, iid):
49
50     uid = int(uid)
51     iid = int(iid)
52
53     p_u = P[uid]
54     q_i = Q[iid]
55
56     return globalMean + bu[uid] + bi[iid] + np.dot(p_u, q_i)
57
58 while True:
59     uid = input("uid: ")

```

```
57     iid = input("iid: ")
58     print(predict(uid, iid))
```

然后就可以输入用户 id 和物品 id 进行预测了，如图6.11所示：

```
uid: 2
iid: 4
61.011183366768115
uid: 123
iid: 546
62.340893202277634
uid: 123
iid: 47542
74.33198457523278
uid: 12
iid: 3415
59.96468158888821
uid: 1
iid: 1
64.61407421775654
```

图 6.11: 基于 BiasSVD 推荐算法的推荐算法

6.2 基于内容的推荐算法结果

通过以下方式得到用户列表的推荐内容：

```
1 avg_data = get_avg()
2 item_cast = get_item_cast()
3 item_sort = get_item_sort(avg_data, item_cast)
4 user_like = get_user_like()
5 user_id = [0, 56, 89, 1234, 12345]
6 recom(user_like, user_id, item_sort)
```

得到的推荐结果如图6.12所示：

```
user_id = [0, 56, 89, 1234, 12345]
recom(user_like, user_id, item_sort)
```

```
该用户最喜欢种类 113323
为用户 0 进行推荐:
[ 41221 512332]
[]
该用户最喜欢种类 490104
为用户 56 进行推荐:
[326428 318586]
[]
该用户最喜欢种类 0
为用户 89 进行推荐:
[510219 602266]
[153594 362117]
该用户最喜欢种类 332729
为用户 1234 进行推荐:
[620010 476263]
[]
该用户最喜欢种类 141519
为用户 12345 进行推荐:
[185184 571267]
[]
```

图 6.12: 基于内容推荐算法的推荐结果

6.3 实验总结

本次实验旨在研究并实现推荐算法：基于物品的协同过滤推荐算法、基于 BiasSVD 矩阵分解的推荐算法以及基于内容的推荐算法。以下是对每种算法的实现和实验结果的总结。

1. 基于物品的协同过滤推荐算法：

- 实现方法：该算法通过计算物品之间的相似度，为用户推荐与他们过去喜欢的物品相似的其他物品。
- 实验过程：我们使用了用户-物品评分矩阵来计算物品之间的相似度，并根据相似度得分为用户生成推荐列表。

2. 基于 BiasSVD 矩阵分解的推荐算法：

- 实现方法：该算法通过将用户-物品评分矩阵分解为用户矩阵和物品矩阵，并引入偏差项来建模用户和物品的隐含特征，从而生成推荐结果。
- 实验过程：我们使用 SVD（奇异值分解）算法对评分矩阵进行分解，并通过优化目标函数来学习用户和物品的隐含特征。
- 实验结果：与基于物品的协同过滤相比，基于 BiasSVD 的推荐算法在准确性和推荐效果方面有所提高。通过引入偏差项，该算法可以更好地捕捉用户和物品的个性化特征，从而提供更准确的推荐结果。

3. 基于内容的推荐算法：

- 实现方法：该算法通过分析物品的内容特征，如关键词、标签等，为用户推荐与他们过去喜欢的物品具有相似内容的其他物品。
- 实验过程：我们使用了物品的文本描述、关键词等内容信息，并通过文本相似度计算来确定物品之间的相似度。

- 实验结果：基于内容的推荐算法在能够捕捉物品之间的语义相似性方面表现出色。它不依赖于用户的历史行为，因此可以应对冷启动问题。然而，该算法可能受限于内容特征的质量和完整性。

总结来说，本次实验展示了基于物品的协同过滤推荐算法、基于 BiasSVD 矩阵分解的推荐算法以及基于内容的推荐算法的实现和效果。每种算法都有其优势和限制，选择适当的算法取决于具体的应用场景和数据特征。未来的研究可以进一步探索不同推荐算法的组合和优化策略，以提升推荐系统的性能和用户体验。

7 附录

7.1 小组分工

- 杨鑫：完成数据预处理，ItemCF、BiasSVD、基于内容的推荐算法代码和实验结果，完成实验报告的撰写。
- 吴晨宇：查找资料。

7.2 仓库

Github [RecommendSystem](#)