

# 面向推荐系统的数据挖掘

计卓 1501 胡思勘 U201514898

2018 年 7 月 8 日

**摘要** 信息化世界存在着大量的数据，如何在大量的数据中为每一个用户找到其所需求的数据成为了一个问题。本文尝试以包括基于用户的协同过滤算法、基于物品的协同过滤算法以及其混合算法在内的多种不同的推荐算法对于数据进行挖掘分析，并在此过程中尝试对于推荐系统的实现提出一个行之有效的解决方法。

## 1 综述

### 1.1 数据集

本实验采用的数据集使用了 Jester 数据集<sup>1</sup>作为所有算法的训练和测试数据集。Jester<sup>2</sup>是一个为研究而开发的笑话推荐系统，使用的数据集中的数据则是次系统获取的真实用户数据。

数据集包含 3 个文件，共包含 73,421 个用户对于 100 个笑话的评分数据。数据为 xls 格式，每一行有 101 个数据，第一个为用户评价过的笑话个数，剩下的 100 个为用户对于 100 个笑话的评分，评分为-10 ~ 10 之间的实数，99 表示“null”，即未打分。数据集的 5, 7, 8, 13, 15, 16, 17, 18, 19, 20 是密集的，几乎所用的用户都给这些笑话打分过。

### 1.2 算法

本次实验采用多种不同的算法并将其结果进行对比。采用的算法包括基于用户的协同过滤算法、基于物品的协同过滤算法等方法。

如图1所示，在协同过滤算法中，用  $m \times n$  的矩阵表示用户对物品的喜好情况，一般用打分表示用户对物品的喜好程度，分数越高表示越喜欢这个物品。图中行表示用户，列表示物品， $U_{ij}$  表示用户  $i$  对物品  $j$  的打分情况。协同过滤分为两个过程：预测过程和推荐过程。预测过程是预测用户对没有购买过的物品的可能打分值，推荐是根据预测阶段的结果推荐用户最可能喜欢的一个或多个物品。

---

<sup>1</sup><http://www.ieor.berkeley.edu/~goldberg/jester-data/>

<sup>2</sup><http://eigentaste.berkeley.edu/>

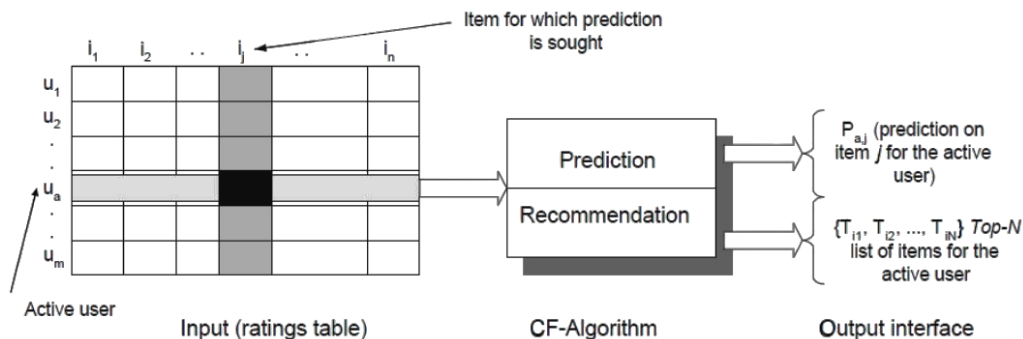


图 1: 协同过滤算法描述

### 基于用户的协同过滤算法

基于用户的协同过滤算法通过用户的历史行为数据发现用户对商品或内容的偏好，并对这些偏好进行度量和打分。根据不同用户对相同商品或内容的态度和偏好程度计算用户之间的关系。在有相同喜好的用户间进行商品推荐。如图2所示，如果用户 1、用户 3 两个用户都对于物品 2 和物品 3 显示出偏好，并且评价较高，那么用户 1 和用户 3 就属于同一类用户，可以将用户 1 偏好的物品 4 也推荐给用户 3。这里仅是一个简单的例子。在实际的应用过程中还要考虑偏好的程度，同一类用户的人数以及偏好物品的个数等多方面的因素。

简而言之，基于用户的协同过滤算法主要分为 2 个步骤：

1. 寻找相似的用户集合；
2. 寻找集合中用户喜欢的且目标用户没有的进行推荐。

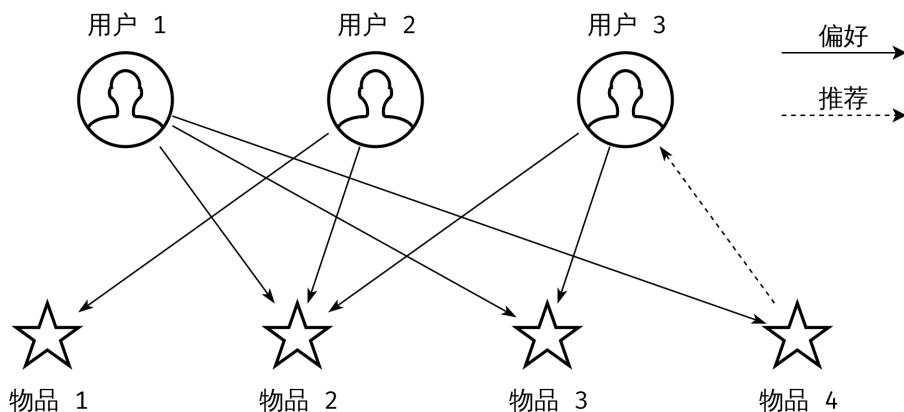


图 2: 基于用户的协同过滤算法原理

然而，基于用户的协同过滤算法存在两大问题：

1. 数据较为稀疏，往往在一个系统中有非常多的物品，而用户偏好的物品数量很少，购买相同物品的用户不够密集，导致无法找到一个用户的邻居进行推荐。
2. 算法的可扩展性较低，随着用户和物品数量的增加，为所有用户推荐所需时间呈几何模式增加，这往往是一个系统所不能接受的。

### 基于物品的协同过滤算法

与基于用户的协同过滤不同，基于物品的协同过滤根据所有的用户偏好计算物品之间的相似度，并把相似的物品推荐给用户。仍以图2为例，推荐物品 2 和物品 3 的人群相似，因此这两个商品相似，在用户 2 选择物品 2 时就可以推荐较为相似的物品 3。

由于物品间的相似性比较固定，因此可以预先计算不同物品之间的相似度，在性能上较优于基于用户的协同过滤算法。简而言之，基于物品的协同过滤算法分为 2 步：

1. 计算物品之间的相似度；
2. 根据物品之间的相似度以及用户历史行为给用户生成推荐列表。

上述为不同算法的概述，具体的算法实现见第3节。

## 2 数据预处理

由于原数据使用 99 表示用户未评分，而这样的数据在计算时会带来较大的不便，因此在预处理时首先将这些数据置为 0，相当于用户在进行-10~10 的连续打分中给这些数据打分为 0。可以证明，进行如此的预处理之后在进行各种距离计算时与忽略掉这些数据的结果是一致的，也就是说可以令其直接参与计算而不影响结果。

此外，在使用第三组数据作为训练数据集时，可以考虑预先剔除只给一个物品打分的用户，因为在计算推荐物品时这些用户不可能作出贡献（不可能选择此用户偏好的其他物品进行推荐）。

## 3 推荐算法实现

### 3.1 基于用户的协同过滤

#### 计算用户相似度

要实现基于用户的协同过滤算法，首先要进行计算的是用户间的相似度。计算用户的相似度主要有 4 种方法：Jaccard 距离、皮尔逊相关系数、闵可夫斯基距离以及余弦距离。其定义分别如下：

- Jaccard 系数主要用于计算布尔度量的个体间相似度，当个体的特征均使用布尔值进行度量时，无法衡量具体的差异大小，只能获得个体间的特征是否一致，其公式如下：

$$w_{uv} = \frac{N(u) \cap N(v)}{N(u) \cup N(v)}$$

其中， $w_{uv}$  为两个个体之间的相似度， $N(u)$  和  $N(v)$  分别为  $u$  和  $v$  所具有的属性。

- 皮尔逊相关系数是比欧几里德距离更加复杂的可以判断人们兴趣相似度的一种方法。它在数据不是很规范时，会倾向于给出更好的结果。其计算方法如下：

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_x \sigma_y} = \frac{E((X - \mu_x)(Y - \mu_y))}{\sigma_x \sigma_y} = \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - E^2(X)}\sqrt{E(Y^2) - E^2(Y)}}$$

其中  $E$  为数学期望， $\text{cov}$  为协方差。

- 两个点  $X$  和  $Y$  之间的闵可夫斯基距离定义为：

$$\left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

$p$  取 1 或 2 时的闵氏距离是最为常用的， $p = 2$  即为欧氏距离，而  $p = 1$  时则为曼哈顿距离。当  $p \rightarrow \infty$  时的极限情况下，可以得到切比雪夫距离：

$$\lim_{p \rightarrow \infty} \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}} = \max_{i=1}^n |x_i - y_i|$$

- 余弦距离，也称为余弦相似度，是用向量空间中两个向量余弦值作为衡量两个个体间差异大小的度量值。欧几里德距离相似，点  $X$ 、 $Y$  的余弦距离表示为：

$$\text{sim}(X, Y) = \cos \theta = \frac{X \cdot Y}{\|X\| \cdot \|Y\|}$$

在本实验中，由于物品的打分为连续的，为了尽可能的将打分考虑进取以及便于计算，采用余弦距离作为衡量用户之间距离的标准。

## 推荐算法

这里选用了余弦距离作为距离计算方法后，对于任意一个用户  $u$ ，选取与之  $k$  个用户，用集合  $S(u, k)$  表示，并将  $S$  中所有用户喜欢的物品提取出来，去除目标用户  $u$  已经喜欢的物品。然后对余下的物品进行评分与相似度加权，得到的结果进行排序。最后由排序结果对目标用户  $u$  进行推荐。其中，对于每个可能推荐的物品  $i$ ，预测的用户  $u$  对其感兴趣的程度可以用如下公式计算：

$$\hat{r}_{ui} = \frac{\sum_{v \in S(u, K) \cap N(i)} w_{uv} \times r_{vi}}{\sum_{v \in S(u, K) \cap N(i)} w_{uv}}$$

其中,  $N(i)$  为对于物品  $i$  打分的用户集合,  $r_{vi}$  为用户  $v$  对于  $i$  的打分, 而  $w_{uv}$  为用户  $u$  和用户  $v$  之间的相似程度。

具体的推荐算法如下, 传入的参数中,  $item$  为被推荐用户已选物品,  $table$  为训练数据,  $k$  为最近邻居数。

Algorithm 1: Recommend Algorithm

---

```

1: procedure RECOMMEND( $items, table, k$ )
2:   初始化空堆  $heap$ 
3:   for  $item$  in  $items$  do                                ▷ 计算所有用户与当前用户的相似度并入堆
4:     for  $user$  in 给  $item$  打过分的所有用户 do            ▷ 跳过没有交集的用户
5:       if 此用户未计算过 then
6:          $heap.push(sim(items, user), user)$ 
7:       end if
8:     end for
9:   end for
10:   $nearestUsers \leftarrow \{\}, recommend \leftarrow \{\}$ 
11:  for  $i = 1$  to  $k$  do                                       ▷ 找出前  $k$  个用户
12:     $nearestUsers.add(heap.pop())$ 
13:  end for
14:  for  $user$  in  $nearestUsers$  do                               ▷ 计算推荐物品以及预测分数
15:    for  $item$  in  $user$  打分过的物品 do
16:      if  $item$  in  $recommend$  then
17:         $recommend[mi] \leftarrow \sum_{v \in S(u,k) \cap N(i)} w_{uv} \times r_{vi} / \sum_{v \in S(u,k) \cap N(i)} w_{uv}$ 
18:      else
19:        更新  $recommend[mi]$ 
20:      end if
21:    end for
22:  end for
23:  return  $recommend$ 
24: end procedure

```

---

在这一算法下, 基于训练数据集 1 的一个可能的推荐列表如图3所示, 其中包含了这一用户所没有购买的商品推荐及其预测打分 (在最终输出中这些分数不会显示, 只显示正确率)。

```
~/H/DataMining | dev *... src python userBasedCF.py
7 : 0.06848093546846701
10 : 1.9200563413396696
17 : 0.1832468190552258
18 : 1.176132882090256
19 : 1.269379910548568
20 : 2.1212045715766146
22 : 2.77
24 : 4.13
25 : 1.9894596505098638
26 : 1.7324164507060105
27 : 3.9173584324420876
28 : 5.32555075002164
30 : 2.7041613683167554
31 : 4.442674708250144
34 : 4.268643030639361
35 : 4.621953072401164
37 : 1.4619199676996613
```

图 3: 一个推荐输出

## 3.2 基于物品的协同过滤

### 计算物品相似度

基于物品的协同过滤相似度计算方法与基于用户的类似，只不过其对象变为了物品，而评估相似度的指标变为了对两个物品进行打分的用户组的相似度。同样，在本实验的相似度计算中采用余弦相似度作为相似度的计算方法。

### 推荐算法

与基于用户的推荐算法不同，由于本数据集中的物品较少（100 个），因此可以先计算物品-物品相似度表，然后通过这一相似度表进行推荐。而对于物品的分数预测方法如下：

$$\hat{P}_{u,i} = \frac{\sum_{i,N} (s_{i,N} \times R_{u,N})}{\sum_{i,N} (|s_{i,N}|)}$$

其中  $s_{i,N}$  为物品  $i$  与物品  $N$  的相似度， $R_{u,N}$  为用户  $u$  对于物品  $N$  的打分。此时，基于用户的推荐算法如下，输入参数中 `items` 为用户已经打分的物品，而 `distance` 为预先计算好的物品相似度表：

---

Algorithm 2: Item-Based Recommend Algorithm

---

- 1: **procedure** RECOMMEND(*items*, *distance*)
- 2:     *recommend*  $\leftarrow$  {}

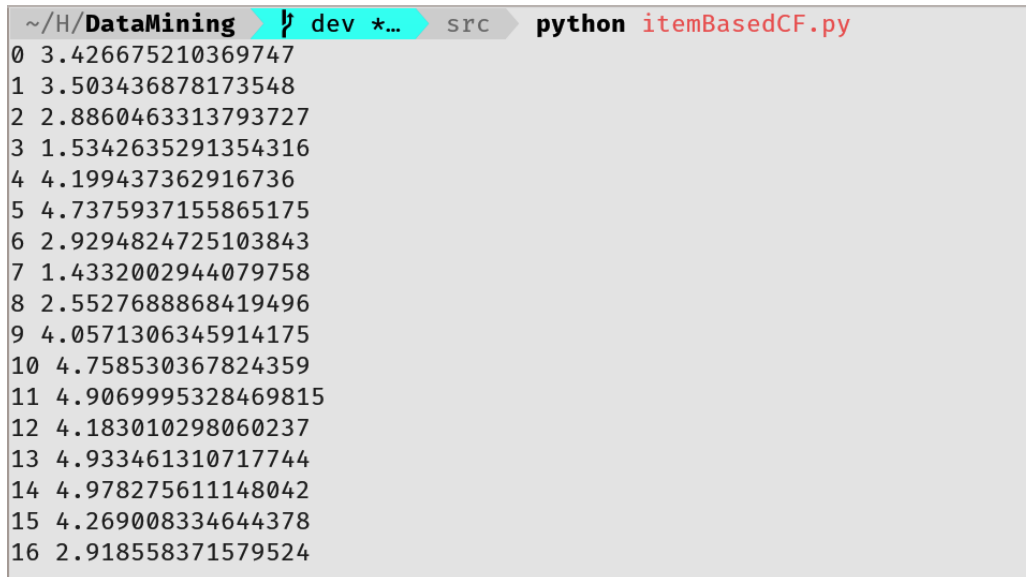
```

3:   for item in items do
4:       for otherItem in distance[item] do
5:           if item in recommend then
6:                $recommend[item] \leftarrow \frac{distance[item][otherItem] \times item.score}{distance[item][otherItem]}$ 
7:           else
8:               更新 recommend[item]
9:           end if
10:        end for
11:    end for
12:    return recommend
13: end procedure

```

---

完成算法后，对于随机的一个用户运行程序，得到的输出如图4所示，由于基于物品的协同过滤是对于未选择的物品按序计算的，因此也是按序输出的。



```

~/H/DataMining ▶ dev *... src ▶ python itemBasedCF.py
0 3.426675210369747
1 3.503436878173548
2 2.8860463313793727
3 1.5342635291354316
4 4.199437362916736
5 4.7375937155865175
6 2.9294824725103843
7 1.4332002944079758
8 2.5527688868419496
9 4.0571306345914175
10 4.758530367824359
11 4.9069995328469815
12 4.183010298060237
13 4.933461310717744
14 4.978275611148042
15 4.269008334644378
16 2.918558371579524

```

图 4: 基于物品的协同过滤的一个推荐

## 4 算法测试与结果评估

### 4.1 评价指标

要进行完整的测试，需要知道用于评价算法的各个指标。通常而言，用于评价的指标包括：

- 准确率：准确率就是最终的推荐列表中正确的推荐所占有的比例。
- 召回率：正确的推荐占全局的比例。
- 覆盖率：表示推荐的物品占了物品全集空间的比例。
- 新颖度：新颖度是为了推荐长尾区间的物品。用推荐列表中物品的平均流行度度量推荐结果的新颖度。如果推荐出的物品都很热门，说明推荐的新颖度较低，否则说明推荐结果比较新颖。

而对于本实验而言，由于打分是连续的，不能简单的判断推荐是否“正确”，因此需要使用其他的方法进行判断。为了能够尽可能准确地评价此推荐系统的准确程度，采用 NMAE (Normalized Mean Absolute Error) 对于结果进行评价。从 Jester 数据网站可以得知随机推荐的 NMAE 约为 33%，而 Jester 所使用的 Egintaste 协同过滤算法的准确度为 20% (2000 年)，以此为基准可以度量本实验所实现的算法的准确性。

(原文: *many papers, including ours, report Normalized Mean Absolute Error (NMAE) rates of approx 20%. How good is this compared with random guessing? In the Appendix to our paper, we show that if user ratings are uniformly distributed, random guessing yields NMAE = 33%.*)

除了准确性指标，还需要考虑性能问题，由于基于用户的推荐算法所需要比较的用户数量较多，因此进行推荐的性能可能较差。在这个实验中，通过比较推荐所需的时间（不包含读取与预处理数据的时间）来对于性能进行评估。最终使用的评估算法如下：

Algorithm 3: Performance Estimation

---

```

1: procedure ESTIMATE
2:    $data \leftarrow \text{READDATA}(\text{trainData})$ 
3:    $data \leftarrow \text{PREPROCESS}(data)$ 
4:    $testCases \leftarrow \text{READDATA}(\text{testData})$ 
5:    $testCases \leftarrow \text{PREPROCESS}(testCases)$ 
6:    $avgNMAE \leftarrow 0$ 
7:   for  $i = 1$  to  $test - case - number$  do
8:     随机从  $testCases[i]$  中去除部分数据，存入  $testCase$  中
9:      $result \leftarrow \text{RECOMMEND}(testCase, data)$ 
10:     $intersect \leftarrow result \cap testCases[i]$ 
11:     $NMAE_i \leftarrow \frac{\sum_i |result.estimate[i] - result.real[i]|}{N \times (\hat{r}_{max} - \hat{r}_{min})}$ 
12:  end for
13:   $avgNMAE \leftarrow \frac{\sum_i NMAE_i}{test - case - number}$ 
14: end procedure

```

---



其中, *Recommend* 过程分别选用随机推荐、基于用户的协同过滤以及基于物品的协同过滤以进行比较。

## 4.2 评估过程

在  $n = 500$  的条件下进行测试

首先进行随机推荐测试, 运行 randomCF 进行推荐, 其结果如图5所示, 可以看出, NMAE 为 33.20%, 多次统计结果均在 32%~34% 之间, 由此可以验证上述的“随机推荐 NMAE 为 33%”这一基准。同时, 可以得知使用随机推荐处理 1000 个用户所需要的时间为 0.1 秒。

```
user 487 : 0.26359374999999996
user 488 : 0.28664583333333334
user 489 : 0.26576000000000005
user 490 : 0.305051724137931
user 491 : 0.27971999999999997
user 492 : 0.3309047619047619
user 493 : 0.4556111111111111
user 494 : 0.4132272727272729
user 495 : 0.41708823529411776
user 496 : 0.38625999999999994
user 497 : 0.37897058823529417
user 498 : 0.3689032258064516
user 499 : 0.33713888888888893
average NMAE: 0.33206254224789
elapsed time: 0.10366300000000006
~/H/DataMining > dev * src history | head -n 1
python randomCF.py
~/H/DataMining > dev * src Sun Jul 8 00:38:00 2018
```

图 5: RandomOutput

接下来, 对于基于用户的协同过滤进行测试, 测试在同样的条件下进行 (输入数据与测试数据相同), 在计算相邻用户时取  $k = 10$ , 结果如图6所示。从图中可以看出, 推荐的 NMAE 为 22.57% 明显优于随机推荐的 33%, 但相比于 Egintaste 算法的 20% 还有较大的改进空间。基于用户的协同过滤用时为 197.5 秒, 约为随机推荐算法的 1975 倍, 说明在用户基数较大时, 此算法较为低效。同时, 可以看出如对于第 499 个用户进行推荐时, 不能找到有效的推荐, 这说明此用户的偏好与其他用户重叠较低, 这也是基于用户的协同过滤算法的弊端之一。

最后, 对于基于物品的协同过滤进行测试, 测试结果如图7所示, 从图中可以看出, 对于 500 个用户推荐仅用时 1.65 秒, 这需要归功于预先建立的物品相似度表。由于此测试数据集的物品较少 (只有 100 件), 因而建立的物品相似度表的规模较小。在如电商网站等物品规模持续增大的数据集中, 需要采取其他的方法以减少计算的时间。

```

user 487 : 0.17069339062397046
user 488 : 0.20431759616516124
user 489 : 0.2753477052354561
user 490 : 0.11905287255568994
user 491 : 0.11615878269246645
user 492 : 0.25778780240092347
user 493 : 0.2808077558402735
user 494 : 0.12426913034663148
user 495 : 0.27697919614948546
user 496 : 0.2567904425280426
user 497 : 0.2824803153626327
user 498 : 0.3630768876119807
user 499 : no recommendation
average NMAE: 0.2257494286050096
elapsed time: 197.50902100000002
~/H/DataMining > dev * src history | head -n 1
python userBasedCF.py
~/H/DataMining > dev * src Sun Jul 8 00:41:58 2018

```

图 6: 基于用户的协同过滤输出

从图中可以看出，平均 NAME 达到了 18.63，远优于基于用户的协同过滤。同时，由于是计算物品之间的相似度，因此不存在没有近邻而导致的不能推荐的情况，这也可以说明，在这一数据集下，基于物品的协同过滤的整体效果要好于基于用户的协同过滤。

```

user 487 : 0.1293098283116268
user 488 : 0.16730413471588484
user 489 : 0.2820695978572392
user 490 : 0.13339594866072893
user 491 : 0.27752571359756456
user 492 : 0.1829257030830215
user 493 : 0.12409696347846169
user 494 : 0.40649751215389235
user 495 : 0.1842327935917965
user 496 : 0.16334903632303915
user 497 : 0.10697782877886305
user 498 : 0.16452821718100655
user 499 : 0.19397972466608918
average NMAE: 0.18633525421227104
elapsed time: 1.5635589999999997
~/H/DataMining > dev * src history | head -n 1
python itemBasedCF.py
~/H/DataMining > dev * src Sun Jul 8 00:54:24 2018

```

图 7: 基于物品的协同过滤输出

### 4.3 评估结果

除了上述对于基于用户的协同过滤与基于物品的协同过滤的对比外，还需要对于不同的初始值进行计算以评估准确性随着用户已经打过的打分个数的变化。如图8所示，此图使用 itemCF 中的 trend 函数生成。其中，横轴为测试样例中已知部分的比例。如  $x = 10$  表示已知用户为前 10 个物品的打分，预测后 90 个物品的打分，而纵轴为平均 NMAE。使用同样的 500 个样例进行测试。可以发现，在已知打分的占比为 50%~60% 时预测的准确度最高。

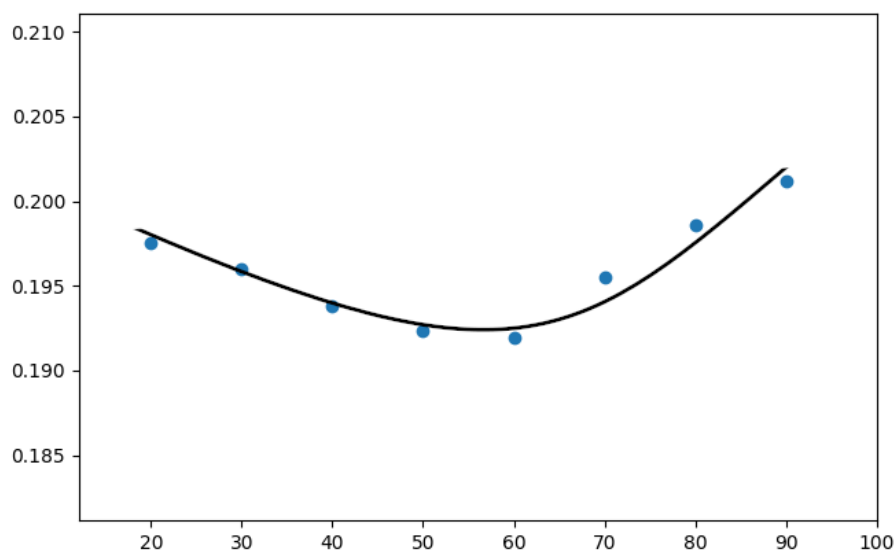


图 8: 基于物品的协同过滤—NMAE 随着已知分数比例的变化

而对于随机推荐而言，其准确率并不随已知的打分占比明显变化，如图9所示，平均 NMAE 一直稳定在约 33% 的位置。这也与预想的结果一致。将此结果与基于物品的协同过滤结果进行对比，说明已知用户偏好占全部物品的比例小于 50% 时预测的准确率呈增加趋势，而大于 50% 时呈减少趋势。在实际的应用中，用户观察并评价的物品数一般小于总物品数的 50%，那么在这个区间内，用户评价的越多，推荐就越准确，这也是符合直觉的。然而在物品数较少的情况下就需要考虑当用户评价的物品数占比过大时的“过拟合”状态。此外，从图中可以看到，即使用户的评价很少，NMAE 也可以维持在 20% 上下，这一结果依旧是十分不错的。

对于基于用户的随机推荐而言，由于计算所花费的时间过长，因此未能生成 NMAE 随已知打分比例的变化图，不过根据推测，其趋势应与基于物品的推荐一致。

综上，

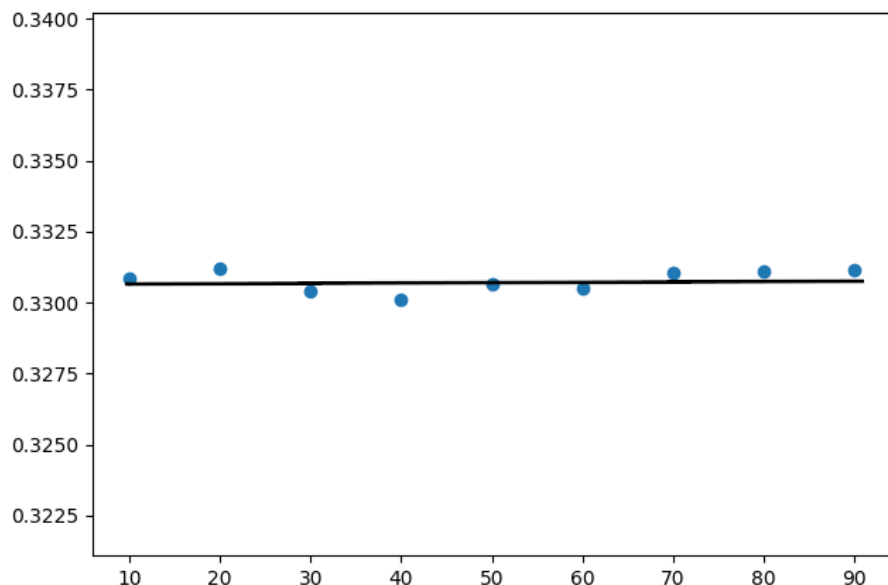


图 9: 随机推荐—NMAE 随着已知分数比例的变化

## 5 结语

本实验对于 Jester 数据集实现了基于用户的协同过滤算法以及基于物品的协同过滤算法，并将其与随机推荐进行比较，同时在基于物品的协同过滤算法上达到了较好的准确性以及性能。

从实验中可以总结，基于用户的协同过滤算法的特点是：

- 用户较少的场合，否则用户相似度矩阵计算代价很大；
- 适合时效性较强，用户个性化兴趣不太明显的领域；
- 对新用户不友好，对新物品友好，因为用户相似度矩阵不能实时计算；
- 很难提供令用户信服的推荐解释。

对应地，基于物品的协同过滤算法的特点有：

- 适用于物品数明显小于用户数的场合，否则物品相似度矩阵计算代价很大
- 适合物品丰富，用户个性化需求强的领域

- 对新用户友好，对新物品不友好，因为物品相似度矩阵不需要很强的实时性
- 利用用户历史行为做推荐解释，比较令用户信服

与其他优秀的推荐算法相比（如 eigintaste、混合算法、基于知识的推荐、基于模型的协同过滤等），本实验中实现的基于物品的协同过滤算法已经达到了较好的效果，但是仍然有较大的改进空间，这主要体现在计算所需要花费的时间上。由于计算各个用户以及各个物品之间的逻辑关联性很小，因此此算法有极大的可并行性，若充分利用多核服务器的并行性能，则能够很好的缩短计算所需要的时间。

## A 参考文献

- [1] Ken Goldberg, Theresa Roeder, Dhruv Gupta, and Chris Perkins. Eigentaste: A constant time collaborative filtering algorithm. *information retrieval*, 4(2):133–151, 2001.
- [2] Tavi Nathanson, Ephrat Bitton, and Ken Goldberg. Eigentaste 5.0: constant-time adaptability in a recommender system using item clustering. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 149–152. ACM, 2007.
- [3] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.
- [4] Zhi-Dan Zhao and Ming-Sheng Shang. User-based collaborative-filtering recommendation algorithms on hadoop. In *Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on*, pages 478–481. IEEE, 2010.

## B 源代码

此处只给出基于物品的协同过滤的源代码。基于用户的协同过滤以及随机推荐的代码整体框架与之类似，此处不再给出。

```
1 import logging
2 import pandas
3 import numpy
4 import sklearn
5 import math
6 import heapq
7 import matplotlib.pyplot as plot
8 import time
```

```

9
10 def readData(path):
11     """
12     read data into 2-dimensional array
13     :param path: path to the xml data file
14     :return: pandas data frame
15     """
16     logging.info("reading data file ", path)
17     dataframe = pandas.read_csv(path)
18     logging.info("data file ", path, " read")
19     return numpy.delete(numpy.array(dataFrame), numpy.s_[1:], axis=1)
20
21 def itemDistance(item1, item2, table):
22     """calculate similarity (cosine distance of two items)
23     :param user1: id of the first user / a list of item scores from the
24         first user
25     :param user2: id of the second user
26     :param table: original table
27     :return: distance of two user in floating point
28     """
29     itemUserTable = table.T
30     return numpy.dot(itemUserTable[item1], itemUserTable[item2]) / (
31         numpy.linalg.norm(itemUserTable[item1]) * numpy.linalg.norm(
32             itemUserTable[item2]))
33
34 def itemDistanceTable(table):
35     """get item-item distance table from original table
36     :param table: the original table
37     :return: item-item similarity matrix
38     """
39     result = [[0 for col in range(table.shape[1])] for row in range(
40         table.shape[1])]
41     for item1 in range(table.shape[1]):
42         for item2 in range(table.shape[1]):
43             if item1 == item2:
44                 result[item1][item2] = 0
45             else:
46                 distance = itemDistance(item1, item2, table)
47                 result[item1][item2] = distance
48                 result[item2][item1] = distance

```

```

46     for index in range(len(result)):
47         maxnum = numpy.max(result[index])
48         minnum = numpy.min(result[index])
49         for column in range(len(result)):
50             if result[index][column] > 0 and maxnum:
51                 result[index][column] /= maxnum
52             elif minnum:
53                 result[index][column] /= minnum
54     return numpy.array(result)
55
56 def recommend(items, itemDisTable):
57     """recommend items for user
58     :param items: the items this user already scored
59     :param table: original data table (training set)
60     :param k: k nearest neighbor
61     :return: estimated items and score, in a dictionary of item:
62             estimate-score
63     """
64     result = {}
65     for itemIndex in range(len(itemDisTable)):
66         if items[itemIndex] == 0:
67             result[itemIndex] = 0
68             # calculate via scored items
69             base = 0
70             for scoredItemIndex in range(len(items)):
71                 if items[scoredItemIndex] != 0:
72                     result[itemIndex] += itemDisTable[itemIndex][
73                         scoredItemIndex] * items[scoredItemIndex]
74                     base += itemDisTable[itemIndex][scoredItemIndex]
75             if base:
76                 result[itemIndex] /= base
77     return result
78
79 def preProcess(table):
80     """
81     do pre-processing
82     :param table: original table
83     :return: preprocessed table
84     """
85     table[table == 99] = 0
86     return table

```

```

85
86 def trend():
87     # read from csv file
88     data = readData("data/jester-data-1.csv")
89     data = preProcess(data)
90     itemDisTable = itemDistanceTable(data)
91
92     testData = readData("data/jester-data-2.csv")
93     testData = preProcess(testData)
94
95     totalError = 0
96     totalCount = 0
97
98     startTime = time.clock()
99
100    jList = []
101    avgNMAEList = []
102    for j in range(10,100,10):
103        for i in range(500):
104            # test the nth line of data
105            test = testData[i]
106
107            # random remove something from test
108            testRemoved = test.copy()
109            testRemoved[:j] = [0] * j
110
111            # get recommend list
112            result = recommend(testRemoved, itemDisTable)
113
114            # calculate NMAE
115            err = 0
116            count = 0
117            rMax = -10
118            rMin = 10
119            if len(test):
120                for index in result:
121                    if test[index] != 0:
122                        # # update rmax and rmin
123                        # if(result[index] > rMax):
124                        #     rMax = result[index]
125                        # if(result[index] < rMin):

```



```

126             # rMin = result[index]
127             # error
128             err += math.fabs(test[index] - result[index])
129             count += 1
130         if count:
131             totalError += err / count / 20
132             totalCount += 1
133             print("user ", i , ": ", err / count / 20)
134         else:
135             print("user ", i , ": no recommendation")
136
137     endTime = time.clock()
138     print("average NMAE: ", totalError / totalCount)
139     print("elapsed time: ", endTime - startTime)
140
141     avgNMAEList.append(totalError / totalCount)
142     jList.append(j)
143     plot.scatter(jList, avgNMAEList)
144     plot.show()
145
146 def main():
147     # read from csv file
148     data = readData("data/jester-data-1.csv")
149     data = preProcess(data)
150     itemDisTable = itemDistanceTable(data)
151
152     testData = readData("data/jester-data-2.csv")
153     testData = preProcess(testData)
154
155     totalError = 0
156     totalCount = 0
157
158     startTime = time.clock()
159     for i in range(500):
160         # test the nth line of data
161         test = testData[i]
162
163         # random remove something from test
164         testRemoved = test.copy()
165         testRemoved[:50] = [0] * 50
166

```

```

167         # get recommend list
168         result = recommend(testRemoved, itemDisTable)
169
170         # calculate NMAE
171         err = 0
172         count = 0
173         rMax = -10
174         rMin = 10
175         if len(test):
176             for index in result:
177                 if test[index] != 0:
178                     # error
179                     err += math.fabs(test[index] - result[index])
180                     count += 1
181             if count:
182                 totalError += err / count / 20
183                 totalCount += 1
184                 print("user ", i , ": ", err / count / 20)
185             else:
186                 print("user ", i , ": no recommendation")
187
188         endTime = time.clock()
189         print("average NMAE: ", totalError / totalCount)
190         print("elapsed time: ", endTime - startTime)
191
192 if __name__ == "__main__":
193     main()

```