机器学习与数据挖掘 ~ Homework4

19335084 黄梓浩

1 实验目的和实验要求

1.1 实验目的

实现协同滤波算法，分别实现User-based CF和Item-based CF两个版本。

1.2 实验要求

- 从GroupLens网站，找到自己认为合适的2个MovieLens数据集版本
- 推荐算法的评分预测效果用RMSE进行度量
- 考虑在不同的近邻个数k的情况下，User-based CF和Item-based CF的实验效果，并进行对比，分别找出User-based CF和Item-based CF的最佳的k。

2 算法原理

2.1 推荐系统

推荐系统是一种信息过滤系统，用于预测用户对物品的评分或偏好。推荐系统产生推荐列表的方式通常有两种：协同过滤以及基于内容推荐，或者基于个性化推荐。

本实验我们使用协同滤波的方法做根据用户电影评分的推荐系统。

2.2 collaborative filtering 协同滤波

协同滤波，指的是通过分析用户或者物品间的相似性，来预测用户可能感兴趣的内容，并将此内容推荐给用户。

相似性可以基于用户的个人信息（性别、年龄等），也可以是用户的历史行为记录（浏览过的商品、购物车的内容），在本实验中相似性是基于用户对于不同电影的评分。

协同滤波有三种类型，基于用户的(user-based)，基于物品的(item-based)，基于模型的(model-based)。

2.3 基于用户的和基于物品的方法步骤：

基于用户的协同滤波算法方法步骤如下：

- 收集用户对于物品的评分
- 找出要预测用户的最近邻集合（最近邻指的是，找出 k 个相似度与该用户最相近的用户），可以通过多种方式计算相似度，这里使用的是Pearson 相关性。
- 通过最近邻集合中的用户的评分来产生预测用户对一些物品的评分结果，整个算法的过程在后面会详细解释。

而基于物品的方法，除了在计算相似度方面不同以外，其余和基于用户的方法是一样的。区别在于，计算用户的相似度（比如用户和用户对电影评分的相似度），以及计算物品间的相似度（比如，电影和电影的相似度，通过对这个电影评了分的用户的评分集合来得出）

2.4 如何计算Pearson 相似度?

以 基于用户的 方式举例：

简单而言就是一个如下的公式，来计算用户间的相似度

- **Pearson correlation coefficient**
  - S_{xy} = items rated by both users **x** and **y**

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}}$$

用文字描述这个过程就是：

- 找出这两个用户都评过分数的物品（电影）
- 然后遍历这个共同物品清单，计算公式中小项的值，最后得结果。
- 计算 $(r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})$

  r_{xs}：用户x对当前遍历到的共同电影的评分

  $\overline{r_x}$：用户x对其评价所有电影的评分的平均数

  $\overline{r_y}$：用户y

如果是 基于物品的方式，那只要把共同物品清单换成共同用户清单就可以了。

3 代码 （使用 Python实现）

3.1 导入数据类型

导入数据格式为，[['用户ID','电影ID','评分'], [....], ...]

```
[[196,242,3],[186,302,3],[22,377,1]....]
```

3.2 my_collaborative_filtering 类

我希望实现一个这样的类，导入用户对电影评分数据后，执行类方法，就可以使用协同滤波方法来预测某用户对某物品的评分。

类变量如下，

```
self.user_dataset = {}# 以用户为键，记录每个用户对电影的评分
self.item_dataset = {}# 以电影为键，记录每个电影下的评分用户的评分
```

类方法如下：

```
    def get_avg_rating(self, data): # 求得该项的字典数据下的平均值
# 基于用户的方法
    def get_user_pearson(self, user1, user2):    #用pearson相关性求用户间的相似度
    def get_user_k_near_neigh(self, user, k):    #返回k个最近邻的相似用户
    def user_based_predict(self, user, item, k): #通过基于用户的方法预测某用户对某物品的评分

# 基于物品的方法
    def get_item_pearson(self, item1, item2):
    def get_item_k_near_neigh(self, item, k):
    def item_based_predict(self, user, item, k):#通过基于物品的方法预测某用户对某物品的评分
```

3.3 计算相似度（Pearson相关性）

一开始求两个用户的评分的平均值，

然后取两个用户共同评价过的电影，遍历这个共同电影集，

通过公式求值。（完全是这个公式的过程：

- **Pearson correlation coefficient**
  - S_{xy} = items rated by both users x and y

$$sim(x,y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})(r_{ys} - \overline{r_y})}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \overline{r_x})^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \overline{r_y})^2}}$$

以下展示的代码都是基于用户的方法，基于物品的方法和基于用户的方法在计算过程方式上没有区别，只是变量使用上不一样（user_dataset 换成了 item_dataset）

```
def get_user_pearson(self, user1, user2): #用pearson相关性求用户间的相似度
    user1_data = self.user_dataset[user1]
    user2_data = self.user_dataset[user2]
    avg_x = self. get_avg_rating(user1_data)
    avg_y = self. get_avg_rating(user2_data)
    # get common movie
    common_list = []
    tmp = {}
    for (m, r) in user1_data.items():
        tmp.setdefault(m, 0)
        tmp[m] += 1
    for (m, r) in user2_data.items():
        tmp.setdefault(m, 0)
        tmp[m] += 1
    for (m, count) in tmp.items():
        if(count==2):
            common_list.append(m)
    cov = sig_x = sig_y = 0.0
    for item in common_list:
        x = user1_data[item]
        y = user2_data[item]
        cov += (x - avg_x) * (y - avg_y)
```

```
        sig_x += (x - avg_x) ** 2
        sig_y += (y - avg_y) ** 2
    sig_x = sqrt(sig_x)
    sig_y = sqrt(sig_y)
    if (sig_x*sig_y == 0):
        return 0.0
    return cov / (sig_x * sig_y)
```

3.4 预测某用户对某电影的评分：

求该用户对其余用户的相似度，通过相似度大小排序，取最相似的k个返回集合

```
def get_user_k_near_neigh(self, user, k):  # 返回k个最近邻的相似用户
    neigh = []
    for (u, data) in self.user_dataset.items():
        if (u != user):
            p = self.get_user_pearson(user, u)
            neigh.append([u, p])
    return (sorted(neigh, key=itemgetter(1), reverse=True))[:k]
```

先返回 k 个与该用户最相似的用户集（包括相似度数据）；

排除掉其中并没有对该电影评分的用户集；

然后，通过这个方法求出预测评分：

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

其实就是把计算出了相似度当作权重，来计算评分的加权平均数罢了

```
def user_based_predict(self, user, item, k):  # 通过基于用户的方法预测某用户对某物品的评分
    # 先取k个最相似的
    k_near_neigh = self.get_user_k_near_neigh(user, k)
    valid_neigh = []
    # 在k个中找对该电影评了分的项
    for nei in k_near_neigh:
        if (item in self.user_dataset[nei[0]].keys()):
            valid_neigh.append(nei)
    if (len((valid_neigh)) == 0):
        return 0.0
    # Σ(Wi,1)*(rating i,item) / Σ(Wi,1),
    up = down = 0.0
    for nei in valid_neigh:
        rating = self.user_dataset[nei[0]][item]
        up += nei[1] * rating
        down += nei[1]
    if (down == 0.0):
        return 0.0
    return up / down
```

基于物品的 类方法和上面的代码除了变量使用上不一样，其余都是一样的。

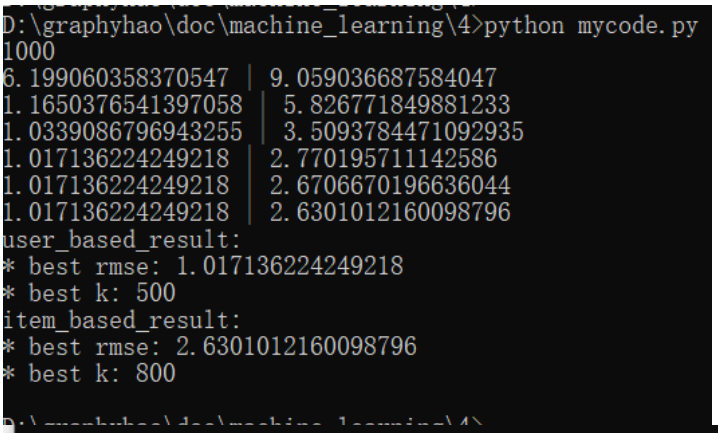
4 结果分析

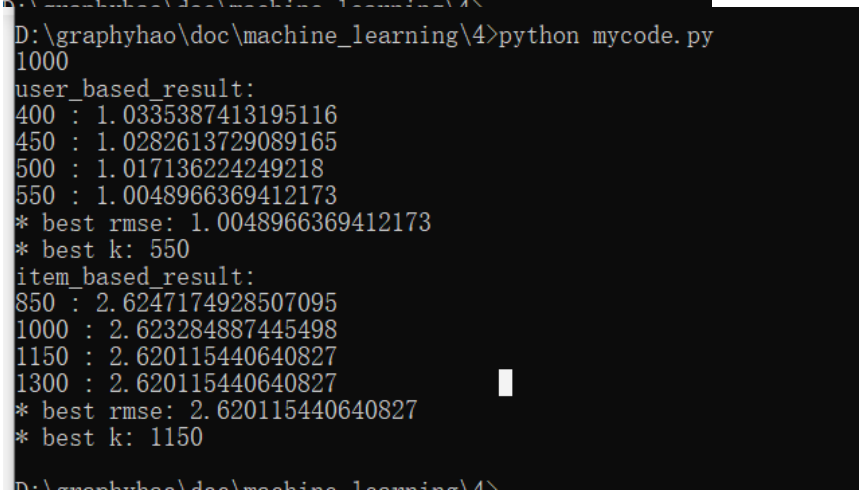
数据集使用（在data文件夹下）：

```
MovieLens 100K Dataset
MovieLens Latest Datasets
```

数据集1：**MovieLens 100K Dataset** 的对比结果（上图为GenLouvain，下图为我的）：

```
评价数 100000
用户数 1000
电影数 1700
```





```
D:\graphyhao\doc\machine_learning\4>python mycode.py
1000
user_based_result:
525 : 1.0044596652603441
575 : 1.0044596652603441
600 : 1.0044596652603441
* best rmse: 1.0044596652603441
* best k: 525
item_based_result:
1350 : 2.6520029122403512
1400 : 2.6520029122403512
1450 : 2.6520029122403512
* best rmse: 2.6520029122403512
* best k: 1350
```

因为计算一次k的情况的时间过长，所以我只是选用了几个k的采样点，来分析最好的 k可能落在的范围。

最后的结果是，

基于用户的方法，最好的k在 550附近；基于物品的方法，最好的k在1150附近。

数据集2：**MovieLens Latest Datasets** 的对比结果 （上图为GenLouvain，下图为我的）：

```
评价数 100000
用户数 600
电影数 9000
```

```
1008
user_based_result:
50 : 6.473662910489902
150 : 2.1882294778883957
200 : 1.507194766965479
400 : 1.2117724365936626
500 : 1.2117724365936626
* best rmse: 1.2117724365936626
* best k: 400
```

```
item_based_result:
50 : 6.8371289630877525
500 : 3.9300625775151024
1500 : 1.5778526784724118
3000 : 1.356123542335212
5000 : 1.3555427677724334
7000 : 1.3554767339010287
* best rmse: 1.3554767339010287
* best k: 7000
```

```
1008
user_based_result:
250 : 1.3078832001210898
300 : 1.1466120476518251
350 : 1.0613850300393646
* best rmse: 1.0613850300393646
* best k: 350
item_based_result:
3500 : 1.2483381127139548
4000 : 1.2483381127139548
4500 : 1.2483381127139548
* best rmse: 1.2483381127139548
* best k: 3500
```

对第二数据集取k值，最后的结果是，

基于用户的方法，最好的k在 350附近；基于物品的方法，最好的k在3500附近。

此算法还可以继续优化，对于pearson计算和求k近邻可以使用矩阵的方式优化，可以大量降低时间复杂度。