中山大学数据科学与计算机学院
移动信息工程专业-数据仓库与数据挖掘
本科生实验报告

教学班级	1515	专业（方向）	软件工程
学号	15352334	姓名	吴佳卫

一、 实验题目

推荐系统

二、 实验内容

1. 算法原理

基于商品的协同过滤 IBCF

根据基于商品的协同过滤算法设计的推荐系统关注的是商品之间的相似度。一共有三种相似度的度量方法，分别是余弦相似度，皮尔逊相关系数以及余弦适应性相似度。

余弦相似度:

$$\mathrm{sim(i,j)} = \frac{\vec{i}\cdot\vec{j}}{\|i\|_2 * \|j\|_2}$$

皮尔逊相关系数:

$$\mathrm{sim(i,j)} = \frac{\sum_{u\in U}(R_{u,i}-\overline{R_i})(R_{u,j}-\overline{R_j})}{\sqrt{\sum_{u\in U}(R_{u,i}-\overline{R_i})^2}\sqrt{\sum_{u\in U}(R_{u,j}-\overline{R_j})^2}}$$

余弦适应性相似度:

$$\mathrm{sim(i,j)} = \frac{\sum_{u\in U}(R_{u,i}-R_u)(R_{u,j}-R_u)}{\sqrt{\sum_{u\in U}(R_{u,i}-R_u)^2}\sqrt{\sum_{u\in U}(R_{u,j}-R_u)^2}}$$

比较这三种相似度的度量方式，余弦相似度单纯的通过计算两个向量的夹角余弦值来评估它们的相似度。而皮尔逊相关系数刻画了变量间线性关系的强弱。余弦适应性相似度与余弦相似度本身比较像，

只不过是对两个变量指中心化后再求余弦相似度。

那么对于推荐系统来说，我们可以总结出以下三种度量方式的异同点。

算法	关注的用户	中心化的方式
余弦相似度	所有用户	无中心化
皮尔逊相关系数	只关注同时对 i 和 j 物品评价的用户	同时对 i 和 j 物品评价的用户对 i 评级的平均值
余弦适应性相似度	只关注同时对 i 和 j 物品评价的用户	用户已评级项目的平均值

在计算出物品之间的相似性后，下一步就是预测用户对新商品的评分，基本的思路就是使用用户打过的分数计算加权平均数，而权值就是两个物品之间的相似性度量。

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

公式中S_{ij}就是商品 i 与商品 j 之间的相似性，r_{xj}就是用户 x 对商品 j 的打分。

Slope One

Slope One 的本质是一种一元的线性模型，他的基本思路是根据用户已经评价过的商品评分的偏差来推算出新商品的评分。算法的第一步就是计算商品之间评分的偏差。

$$dev_{j,i} = \sum_{u \in U_{j,i}} \frac{R_{u,j} - R_{u,i}}{card(U_{j,i})}$$

计算偏差的方法就是计算同时购买过两件商品的用户的评分偏差的均值。计算出偏差后就可以预测用户对一件新的商品的评分了：

$$R_{u,j} = \frac{\sum_{i \in S(u)-\{j\}}(dev_{j,i} + R_{u,i})}{card(S(u) - \{j\})}$$

也就是说用户对于新商品的评分可以表达成已经评价过的商品的评分与两件商品之间评分的差值。

矩阵分解

矩阵分解算法的假设是通过分解评分矩阵（用户-商品）来找到用户因子矩阵和商品因子矩阵。为了判断矩阵分解效果的好坏首先算法设计了损失函数：

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

以上公式得到的矩阵$e_{i,j}^2$就代表了将矩阵 R 分解为矩阵 P 和 Q 之后的差值。最后将所有项求和就可以得到损失函数了：

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

当然根据损失函数我们就可以设计出矩阵 P 和 Q 的更新公式了（先通过偏导就算梯度，然后再根据梯度更新变量）

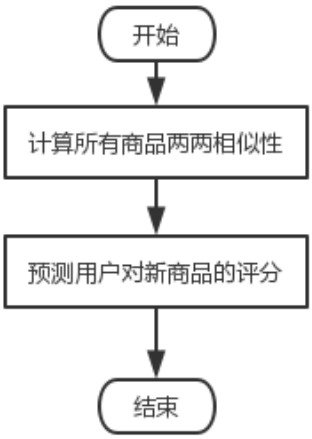
$$p_{i,k}{}' = p_{i,k} + 2 \propto e_{i,j} q_{k,j}$$

$$q_{k,j}{}' = q_{k,j} + 2 \propto e_{i,j} q_{i,k}$$

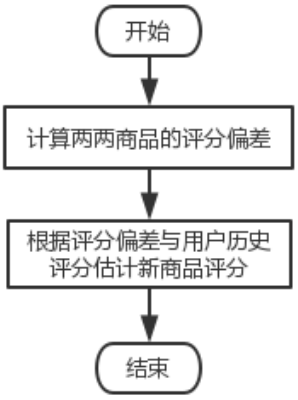
更新至目标函数收敛后就可以将矩阵 P 和 Q 相乘生成预测的用户评分矩阵$\hat{R}$了。

2. 算法流程

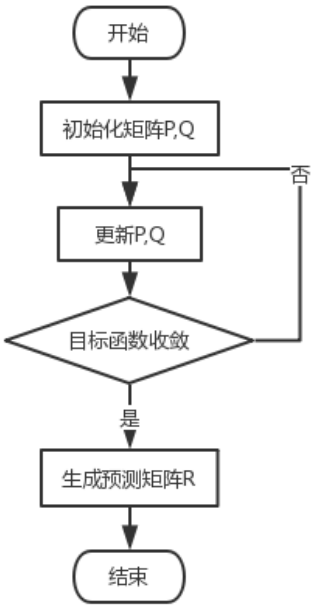
基于商品的协同过滤 IBCF



Slope One



矩阵分解



3. 关键代码截图

基于商品的协同过滤 IBCF

计算商品之间的相似性（余弦适应性相似度）；

```
[user_num,item_num] = size(train);
R_aver = mean(train,2);
S = zeros(item_num,item_num);
for i=1:item_num
    for j=i+1:item_num
        temp1=0;
        temp2=0;
        temp3=0;
        for k=1:user_num
            temp1 = temp1 + ((train(k,i)-R_aver(k))*(train(k,j)-R_aver(k)));
            temp2 = temp2 + ((train(k,i)-R_aver(k))^2);
            temp3 = temp3 + ((train(k,j)-R_aver(k))^2);
        end
        S(i,j) = temp1/(sqrt(temp2)*sqrt(temp3));
        S(j,i) = S(i,j);
    end
end
```

预测用户对于新的商品的评分;

```
[test_size,~] = size(test);
score = zeros(test_size,1);
for i=i:test_size
    temp1=0;
    temp2=0;
    for j=1:item_num
        temp1 = temp1 + (S(test(i,2),j) * train(test(i,1),j));
        temp2 = temp2 + S(test(i,2),j);
    end
    score(i,1) = temp1 / temp2;
end
```

Slope One

计算商品两两之间的评分差值;

```
[user_num,item_num] = size(train);
dev = zeros(item_num,item_num);
for i=1:item_num
    for j=i+1:item_num
        user_cnt=0;
        for k=1:user_num
           if(train(k,i)~=0 && train(k,j)~=0)
               dev(i,j) = dev(i,j)+ (train(k,i) - train(k,j));
               user_cnt = user_cnt + 1;
           end
        end
        if(user_cnt~=0)
            dev(i,j) = dev(i,j) / user_cnt;
            dev(j,i) = dev(i,j);
        else
            dev(i,j) = 0;
            dev(j,i) = 0;
        end
    end
end
```

预测用户对新商品的评分;

```
[test_num,~] = size(test);
score = zeros(test_num,1);
for i=1:test_num
    item_cnt = 0;
    for j=1:item_num
        if(train(test(i,1),j)~=0)
            item_cnt = item_cnt + 1;
            score(i,1) = score(i,1) + (dev(test(i,2),j) + train(test(i,1),j));
        end
    end
    if(item_cnt~=0)
        score(i,1) = score(i,1) / item_cnt;
    else
        score(i,1) = 0;
    end
end
```

矩阵分解

目标函数;

```
function [e,loss] = calc_loss(R,P,Q)
[m,n] = size(R);
[~,k] = size(P);
e = R;
e2 = zeros(m,n);
for i=1:m
    for j=1:n
        for l=1:k
            e(i,j) = e(i,j) - (P(i,l)*Q(l,j));
        end
        e2(i,j) = e(i,j)^2;
    end
end
loss = sum(sum(e2));
end
```

更新公式;

```
while(abs(loss1-loss0)>0.001)
    for i=1:m
        for j=1:n
            for l=1:k
            P(i,k) = P(i,k) + 2 * alpha * e(i,j) * Q(k,j);
            Q(k,j) = Q(k,j) + 2 * alpha * e(i,j) * P(i,k);
            end
        end
    end
    loss0 = loss1;
    [e,loss1] = calc_loss(train,P,Q);
end
```

三、 实验结果及分析

1. 实验结果展示示例

首先我们先来了解一下实验的评测方法与评测指标，由于推荐系统本身实现的功能是预测用户对新商品的评分并且推荐出评分最高的前 k 个商品，因此很难直接从预测出的分数上来判断模型的好坏，因此我们可以做这样处理：对于测试集预测出的一组评分和真实的评分分别取平均值，并认为高于平均分的商品是用户喜欢的商品，低于平均分为不喜欢的商品。最后判断预测结果和真实结果是否一致。

```
answer = test(:,3);
score = MF(train,test);
[test_size,~] = size(test);
aver_answer = sum(sum(answer)) / test_size;
aver_score = sum(sum(score)) / test_size;
```

最后评价的指标是准确率，召回率与 f1 值，这是一组分类器模型中常用的指标。计算方式如下：

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

$$\text{f1} = \frac{2 * TP}{2 * TP + FP + FN}$$

其中 TP 代表本身为正且预测为正的结果个数，FN 代表本身为正且预测为负的结果个数，FP 代表 FN 代表本身为负且预测为正的结果个数。准确率就可以理解为“预测为正的结果内有多少是正确的”，召回率理解为“本身为正的结果有多少被预测正确了”，f1 值是以上两者的调和平均数，是一个更加客观的指标。

```
for i=1:test_size
    if(answer(i,1)>=aver_answer && score(i,1)>=aver_score)
        TP = TP + 1;
    elseif(answer(i,1)>=aver_answer && score(i,1)<aver_score)
        FN = FN + 1;
    elseif (answer(i,1)<aver_answer && score(i,1)>=aver_score)
        FP = FP + 1;
    elseif (answer(i,1)<aver_answer && score(i,1)<aver_score)
        TN = TN + 1;
    end
end
precision = TP / (TP + FP);
recall = TP / (TP + FN);
f1 = (2 * TP) / (2 * TP + FP + FN);
result = [precision recall f1];
```

值得一提的是还有一种衡量预测结果的指标均方根误差 RMESE:

$$\mathrm{RMSE}(\mathrm{X},\mathrm{h}) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2}$$

RMSE 是可以作为回归模型的评测指标的，通过计算平方根的均值衡量观测值同真值之间的偏差。但是本次实验推荐系统的目的应该是判断用户是否喜欢某一件商品，而不是用户对于商品的真实评分，因此使用准确率，召回率与 f1 值就足以衡量模型的好坏了。

首先对数据进行预处理并转换为 mat 格式。预处理的方式如下：随机划分出训练集与测试集，然后将训练集转化为用户-商品评分矩阵。

```
function pre_treated(A)
[max_A,~]=max(A,[],1);
num_user = max_A(1);
num_item = max_A(2);
train = zeros(num_user,num_item);
test0 = zeros(num_user,3);
[n,~] = size(A);
test_cnt = 1;
for i=1:n
    flag = Rand();
    if(flag == 0)
        train(A(i,1),A(i,2)) = A(i,3);
    elseif(flag==1)
        test0(test_cnt,1) = A(i,1);
        test0(test_cnt,2) = A(i,2);
        test0(test_cnt,3) = A(i,3);
        test_cnt = test_cnt + 1;
    end
end
test = test0(1:test_cnt-1,:);
save U2 train test;
end
```

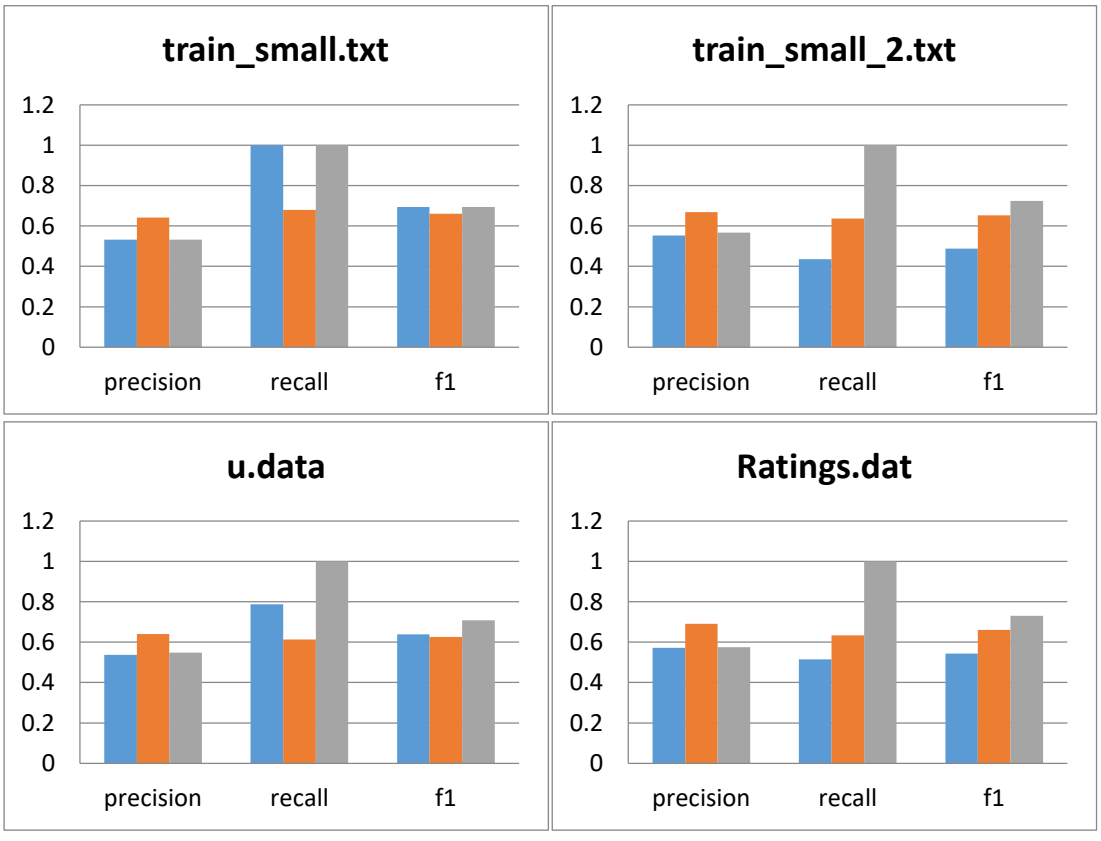
训练集与测试集的比例被设定为 4:1;

```
function R = Rand()
a = rand(1,1);
if(a>0.2)
    R = 0;
elseif(a<=0.2)
    R = 1;
end
end
```

2. 评测指标展示即分析

	IBCF			Slope One			矩阵分解		
	precision	recall	f1	precision	recall	f1	precision	recall	f1
train_small.txt	0.5319	1.0000	0.6944	0.6415	0.6800	0.6602	0.5319	1.0000	0.6944
train_small_2.txt	0.5527	0.4356	0.4872	0.6683	0.6370	0.6523	0.5674	1.0000	0.7240
u.data	0.5366	0.7874	0.6382	0.6398	0.6125	0.6258	0.5485	1.0000	0.7084
Ratings.dat	0.5726	0.5156	0.5426	0.6905	0.6333	0.6607	0.5750	1.0000	0.7301

整理成图标如下：（三种颜色分别是三个模型）



从图表及数据中可以看出，IBCF 的表现整体比较稳定，但是也有出现全部预测为“喜欢”即召回率为 1 的情况。Slope One 的表现更加出色一些，并没有出现全部为 1 的情况。而矩阵分解几乎把所有的商品都记为了用户喜欢，结果并不理想。

Slope One 会出现 recall 为 1 是可以解释的，因为在计算评分偏差的时候需要找到同时买过两件商品的用户，因此可能有的商品无法找到相应的用户，会导致全部预测为 1 的情况。

至于矩阵分解都为 1 的情况，因为算法本身是有局限性的，P、Q 矩阵更新的次数不够或者分解的方式不好都会导致预测失败。