2017

基于内容和协同过滤混合模式的推荐系统评分预测实现

课程：大数据计算及应用

授课教师：刘杰 杨征路

小组成员：罗玲（1410623） 周玲军（1412665）

目录

基于内容和协同过滤混合模式的推荐系统评分预测实现

一、思路及算法

1、思路分析

我们可以通过所给文档集发现，根据 attribute.txt 可以得到电影所属属性，根据 train.txt 可以得到每个用户对于部分电影的评分。前者考虑了电影内容的相关性，所以可以采用基于内容的推荐方法；后者可以采用协同过滤的方法。

两种方法都有各自的优缺点，协同过滤算法擅长处理音乐视频等复杂的非结构化商品，自适应性好，能获得用户充分的隐式反馈等，但是它存在数据稀疏，新用户和新对象的引入问题；基于内容的方法优点是直观易解释，缺点是容易受属性限制**[1]**。所以若将两者结合起来，可以得到较好的准确度。

所以，我们采用综合基于内容和协同过滤的推荐方法，充分利用所给数据的信息训练模型，最后完成评分。

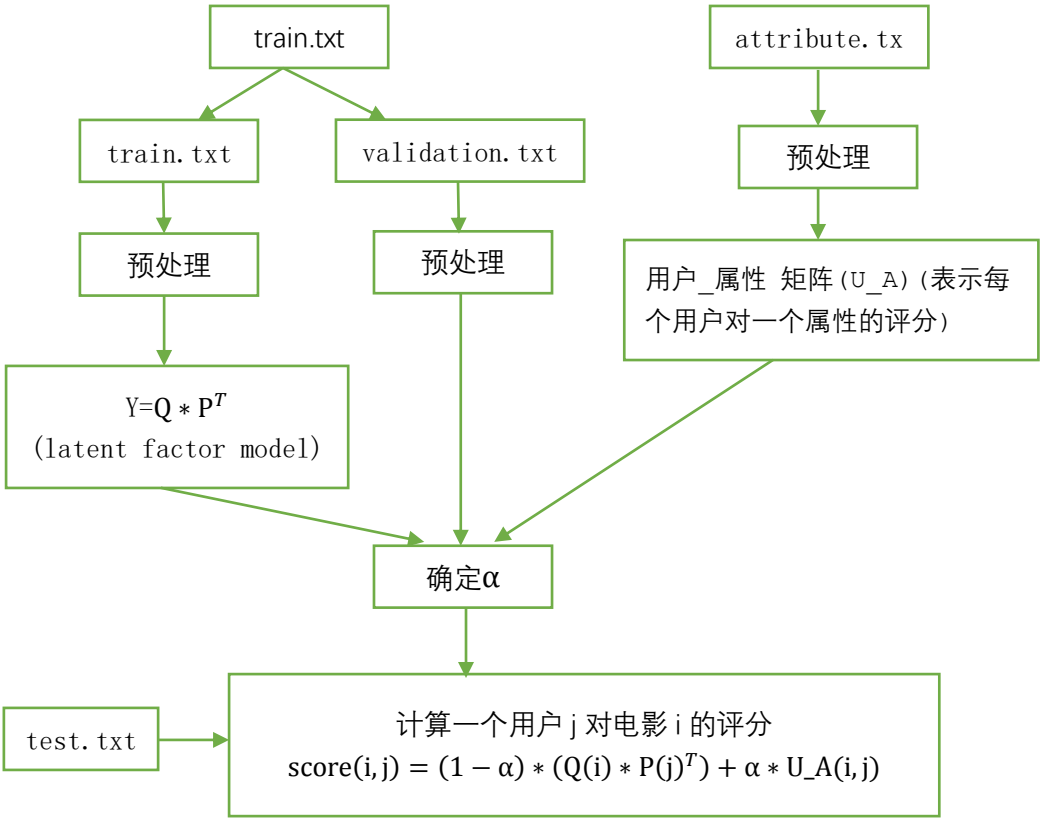
2、算法实现

使用语言： python matlab

算法核心公式：

$\mathrm{score(i,j)} = (1-\alpha) * (\mathrm{Q(i)} * \mathrm{P(j)}^T) + \alpha * \mathrm{U\_A(i,j)}$ (用户j对电影i的评分)**[2]**

算法流程：



算法说明：

(1) 首先将 train.txt 文件对于每个用户而言按照 7:3 的比例分为训练集 train.txt 和验证集 validation.txt。

(2) 然后按照 latent factor model 算法($Y = Q * P^T$)通过训练集得到Q, P矩阵；通过属性集数据 attribute.txt 以及训练集数据得到用户-属性矩阵 U_A。

(3) 接着利用公式$score(i,j) = (1 - \alpha) * (Q(i) * P(j)^T) + \alpha * U_A(i,j)$预测验证集数据，通过最小化 RMSE 来确定α的值。

(4) 利用公式$score(i,j) = (1 - \alpha) * (Q(i) * P(j)^T) + \alpha * U_A(i,j)$(此时α为确定值)来预测测试集数据。

二、实现过程

1、数据读入及处理

首先，我们先通过书写.py 小程序来统计一下 train.txt 的数据，得到一些信息比如电影的个数$num_movies = 624961$,用户的个数 $num\_users = 19835$,属性范围$num_attribute = 624951$。

然后，我们将 train.txt 文件的数据按照约 7:3 的数据比例分为两个文件 txt_train.txt 以及 txt_vali.txt（注意这个 7:3 的比例特别针对每个用户而言，例如 0 号用户已评分 100 部电影，则随机 70 部电影为训练集数据，其余 30 部电影为验证集数据）。

接着通过读入训练集的数据评分，创建稀疏矩阵 Y（$num\_movies * num\_users$），其中$Y(i,j) = socre$表示用户j 对电影的i的评分为socre。但实际上，真实评分时 100 分制的，而我们读入数据时将其除以 10，变为了 10 分制的，方便之后的系列操作。

关于稀疏矩阵的说明：

稀疏矩阵是一个用时间换空间的有效数据结构，普通的矩阵的索引很快，但是所占空间很大，所给的 train.txt 中的数据无法一次性加载到内存，而我们使用的稀疏矩阵只存储不为 0 的值以及其索引，所以空间相对很小，但是会耗费更多的时间。

数据集说明

在训练集中我们有很多用户评分为 0 的情况，其实际上是该用户购买了该电影，但是并没给予评分。所以不论是训练数据还是利用验证集计算 RMSE，我们

都是排除了这类情况的。

2、预处理数据

对于协同过滤算法而言，我们需要将数据进行了简单的归一化，结果为Y_{norm}(是一个num_movies $*$ num_users的稀疏矩阵)，公式如下：

$$Y_{norm}(i,j) = \frac{score(i,j) - row\_mean(j)}{norMaxMin(j)}$$

其中row_mean(j)表示用户j对于所有评分的平均值，$norMaxMin(j)$表示用户j评分最大值与最小值之差。做归一化的目的是让数据比较均匀地分布在[-1,1]之间；这不仅是为了后面数据处理的方便，而且保证程序运行时收敛加快。归一化的缺点在于预测任意一个新的评分时，输出的结果需要进行“反归一化”的运算，速度相对较慢。

说明：在实际操作中，我们对归一化数据和没有归一化的数据都进行了模型训练，发现效果差异不大（验证集的RMSE相差不大），于是在最终的结果代码中我们训练的是未归一化的数据，一方面计算速度更快，另一方面节省了内存开销。

3、模型的训练

3.1 协同过滤算法

3.1.1 算法说明

这里我们采用的是 latent factor model。

主要的思想是，我们将num_users $*$ num_movies)的矩阵(也就是 Ynorm 矩阵的转置近似拆分为如下形式：

$$R_{UI} = P_U Q_I = \sum_{k=1}^{K} P_{U,k} Q_{k,I}$$

$\hat{R}_{UI}$表示用户 I 对 U 的预测值，于是需要最小化以下式子来完成拆分：

$$C = \sum_{(U,I)\in K} (R_{UI} - \hat{R}_{UI})^2 = \sum_{(U,I)\in K} (R_{UI} - \sum_{k=1}^{K} P_{U,k} Q_{k,I})^2 + \lambda \|P_U\|^2 + \lambda \|Q_I\|^2$$

具体的算法伪代码如下：

1) 通过求参数 PUK 和 QKI 的偏导确定最快的下降方向（为了使得收敛速率更快，这里使用的是随机梯度下降法）

$$\frac{\partial C}{\partial P_{Uk}} = -2(R_{UI} - \sum_{k=1}^{K} P_{U,k} Q_{k,I}) Q_{kI} + 2\lambda P_{Uk}$$

$$\frac{\partial C}{\partial Q_{kI}} = -2(R_{UI} - \sum_{k=1}^{K} P_{U,k} Q_{k,I}) P_{Uk} + 2\lambda Q_{kI}$$

2) 迭代计算不断优化参数（迭代次数事先人为设置），直到参数收敛。

$$P_{Uk} = P_{Uk} + \alpha((R_{UI} - \sum_{k=1}^{K} P_{U,k} Q_{k,I}) Q_{kI} - \lambda P_{Uk})$$

$$Q_{kI} = Q_{kI} + \alpha((R_{UI} - \sum_{k=1}^{K} P_{U,k} Q_{k,I}) P_{Uk} - \lambda Q_{kI})$$

3.1.2 操作说明

这里我们将Ynorm 矩阵或者 Y 矩阵拆分为P, Q矩阵，P矩阵为num_users * num_features,Q 矩阵为num_features * num_movies。

关于num_features，λ，下降速率 α，迭代次数的设置，是在一次次实验中不断的尝试，根据验证集的 RMSE 来判断最终对这些参数的选取的。以下是部分

数据统计的结果：

特征数 num_features	λ	下降速率 α	迭代次数	是否正规化	是否过滤评分0项	训练时间(s)	验证集的RMSE
20	0.01	0.02	10	是	否	4880	40.42
20	0.01	0.02	55	是	否	26480	39.87
20	0.002	0.005	60	否	否	17568	23.36
20	0.002	0.005	100	否	否	29280	23.17
60	0.002	0.005	20	否	否	11712	24.32
60	0.002	0.005	30	否	否	23424	24.44
20	0.002	0.005	100	否	是	29280	18.5304
60	0.002	0.005	60	否	是	23520	15.7735
60	0.002	0.005	85	否	是	33660	15.9392

经过多次实验结果的最终统计，我们选择人工的设置num_features的值为 60；n(即迭代次数)为 60 次； α(即下降速率)的初始值为 0.005，λ的值为 0.002。α的值随着每一次迭代而逐渐变小，变为上一次迭代的 0.9。经过统计，通过尝试，我们发现 python 代码一次梯度下降的迭代需要大概 7.3 分钟，而 MATLAB 代码一次梯度下降的迭代>>3 小时。所以 LFM 算法的具体实现是 python 代码。模型的训练的总时间大概为：7.3*60≈438min。

具体的算法如下图所示：

```
def LearningLFM(P,Q,trainMatrix, numberOfFeatures, n, alpha, mylambda):
    # P:f*i  Q:f*u  R_:P'*Q  (i*u)
    P = P.T
    REMS_list=[]
    R_row, R_col, useless = find(trainMatrix)
    for step in range(0, n):
        eui_sum = 0
        for i in range(0, len(R_col)):
            user = R_row[i]
            item = R_col[i]
            rui = trainMatrix[user, item]
            # print("rui:"+str(rui))
            pui = Predict(user, item, P, Q, numberOfFeatures)
            eui = rui - pui
            eui_sum += eui*eui
            for f in range(0, numberOfFeatures):
                P[user, f] += alpha * (eui * Q[f, item] - mylambda * P[user, f])
                Q[f, item] += alpha * (eui * P[user, f] - mylambda * Q[f, item])
            if i % 100000 == 0:
                print(time.ctime())
                print("step:" + str(step) + " i:" + str(i))
        train_err = eui_sum / len(R_row)
        train_REMS = math.sqrt(train_err)
        print("train_REMS " + str(step) + " :" + str(train_REMS))
        REMS_list.append(train_REMS)
        alpha *= 0.9
        print(time.ctime())
        print("finish " + str(step) + " step(total:100)!")
    mdict = {"P": P, "Q": Q}
    scio.savemat("f60-10-PQ.mat", mdict)
    print(time.ctime())
    print("Save P,Q in PQ.mat successfully!")
    return  REMS_list
```

说明：P, Q矩阵初始化为随机值。在每一次迭代中，Predict 函数作用为通过现有的P, Q预测特定的值。其余的操作如具体算法所示。

3.2 content-based 算法

基于预处理阶段以及矩阵Attr，在 content-based 算法中还需要得到两个矩阵，一个是稀疏矩阵 user_attr(num_users * num_attribute)，user_attr(i, j) = score 表示用户 i 对所有属于属性 j 的电影的平均评分为 score；一个是向量 user_mean(num_users * 1)，user_mean(i) = score 表示每一个用户i对所有给过评分的电影的平均分为score。具体算法如下：

```
user_mean=zeros(num_users,1);
for i=1:num_users
    idx=find(R(:,i)==1);
    user_mean(i)=sum(Ynorm(idx,i))/size(idx,2);
end

User_attr=sparse(num_users,num_atrribute);
tic
for i=1:num_atrribute
    idx=find(Attr(:,i)==1);
    if(isempty(idx)~=1)
        fprintf('%d \r\n',i);
        u_count=zeros(num_users,1);
        for j=1:length(idx)
            u_t= find(R(idx(j),:)==1);
            User_attr(u_t,i)=User_attr(u_t,i)+Y(idx(j),u_t)';
            u_count(u_t)=u_count(u_t)+1;
        end
        tmp=find(u_count~=0);
        User_attr(tmp,i)=User_attr(tmp,i)./u_count(tmp);
    end
end
toc
```

说明一下user_attr矩阵的形成过程：对于任意一个属性i，找到该属性值下的所有电影，如果存在的话，所有看过该电影的用户需要加上对该电影的评分，最后对于每一个用户计算其平均值。假设属于属性i的电影有 x, y 两部电影，而用户 u1 对 x 评价为 9 分，对 y 评价为 6 分；用户 u2 只对电影 x 有评价，为 10 分，则$\text{user_attr}(\text{u1},\text{i}) = 7.5$，$\text{user\_attr}(\text{u2},\text{i}) = 10$。

3.3 混合算法

将 LFM 算法以及 content-based 算法综合起来的核心是寻找到预测公式$\text{score}(\text{i},\text{j}) = (1-\alpha) * (\text{Q}(\text{i}) * \text{P}(\text{j})^T) + \alpha * \text{U_A}(\text{i},\text{j})$中α值的最适合的大小，具体的算法步骤如下：

(1) 先通过 LFM 算法训练出的P,Q，通过公式$\text{score}(\text{i},\text{j}) = \text{Q}(\text{i}) * \text{P}(\text{j})^T$预测验证集的所有值，为$\text{vali_score1}$。

(2) 通过 content-based 算法预测验证集的所有值，为vali_score2。

(3) 将vali_score1与vali_score2的值规范化：由于vali_score1与vali_score2目前是十分制的为浮点型，利用 ceil 函数将其变为整型后再乘 10；然后检测是否存在不规范的值，比如大于 100 的值设为 100，小于 0 的值设置为 0。

(4) 最终的预测值为 $score = (1-\alpha) * vali\_score1 + \alpha * vali\_score2$ ，α（可取值[0,0.5]之间，阶梯为 0.002）取使得验证集的 RMSE 最小时的值。

核心算法如下：

```
vali_score1=zeros(num_users,6);
vali_score2=zeros(num_users,6);
tic
for i=1:num_users
    fprintf('%d \r\n',i);
    for j=1:6
        movie=movie_vali(i,j);
        idx=find(Attr(movie,:)==1);
        %[m,~]=find(Attr(:,idx)==1);
        if(isempty(idx)~=1)
            if(length(idx)==1&&User_attr(i,idx(1))~=0)
                vali_score2(i,j)=User_attr(i,idx(1));
            elseif(length(idx)==2&&User_attr(i,idx(1))~=0&&User_attr(i,idx(2))~=0)
                vali_score2(i,j)=(User_attr(i,idx(1))+User_attr(i,idx(2)))/2;
            elseif(length(idx)==2&&User_attr(i,idx(1))==0&&User_attr(i,idx(2))~=0)
                vali_score2(i,j)=User_attr(i,idx(2));
            elseif(length(idx)==2&&User_attr(i,idx(1))~=0&&User_attr(i,idx(2))==0)
                vali_score2(i,j)=User_attr(i,idx(1));
            else
                vali_score2(i,j)=user_mean(i);
            end
        else
            vali_score2(i,j)=user_mean(i);
        end
        vali_score1(i,j)=P(i,:)*Q(:,movie);
    end
end
vali_score1=ceil(vali_score1);
vali_score1=vali_score1*10;
vali_score2=ceil(vali_score2);
vali_score2=vali_score2*10;
toc
```

这段代码的目的是为了得到vali_score1与vali_score2，其中vali_score1就可根据 LFM 算法的公式得到，而vali_score2的计算步骤为，对于每一个用户i和电影 movie，在属性矩阵中找到该电影的所对应的属性（最多 2 个属性，最少无属性），

然后通过用户对属性的评分矩阵计算出vali_score2的值。

一共可以分为以下 5 种情况：

情况	计算得分公式
idx存在 1 个且**user_attr**(**i**, **idx**)存在	$\text{score} = \text{user\_attr}(\text{i}, \text{idx})$ (i的对属性idx的打分)
idx存在 2 个且**user_attr**(**i**, **idx**(**1**))存在	$\text{score} = \text{user\_attr}(\text{i}, \text{idx}(1))$ (i的对属性idx(1)的打分)
idx存在 2 个且**user_attr**(**i**, **idx**(**2**))存在	$\text{score} = \text{user\_attr}(\text{i}, \text{idx}(2))$ (i的对属性idx(2)的打分)
idx存在 2 个且**user_attr**(**i**, **idx**)都存在	$\text{score} = \text{sum}(\text{user\_attr}(\text{i}, \text{idx})))/2$ (i的对属性idx的平均打分)
其他	$\text{score} = \text{user\_mean(i)}$(i所有打分的平均值)

```
alpha=0;
scoreall=(1-alpha)*vali_score1+alpha*vali_score2;
scoreall=ceil(scoreall/10);
scoreall=scoreall*10;
count=0;
for i=1:6
    idx=find(Y_vali(:,i)==0);
    count=count+length(idx);
    scoreall(idx,i)=0;
end
scoreall=scoreall/10;
scoreall=scoreall*10;
l=num_users*6-count;
err=sqrt(sum(sum((scoreall-Y_vali).^2))/l);
for a=0:0.002:0.5
    fprintf('%d \r\n',a);
    scoreall=(1-a)*vali_score1+a*vali_score2;
    scoreall=ceil(scoreall/10);
    scoreall=scoreall*10;
    count=0;
    for i=1:6
        idx=find(Y_vali(:,i)==0);
        count=count+length(idx);
        scoreall(idx,i)=0;
    end
    l=num_users*6-count;
    tmperr=sqrt(sum(sum((scoreall-Y_vali).^2))/l);
    fprintf('%d \r\n',tmperr);
    if(tmperr<err)
        alpha=a;
        err=tmperr;
    end
end
```

这段代码的作用就是找出最好的α使得验证集的 RMSE 值最小。注意此时计算 RMSE 的公式为：

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(X_{obs,i} - X_{model,i})^2}{n}}$$

计算 RMSE 值时需要排除那些显示评分为 0 的项，所以$n = \text{totalnumber} - \text{count}$，其中 count为显示评分为 0 的项的个数。

此时我们计算出$\alpha = 0.178$，此刻$\text{RMSE} = 15.3577$。

4、预测评分

1. 数据的读入

对 test.txt 中数据进行考察，发现对于每个用户需要预测 6 个电影的评分，对数据进行读入，将电影编号存在矩阵$\mathrm{item}(\mathrm{num_users} * 6)$中，$\mathrm{item}(i, j) = t$表示用户$i$需要对电影$t$进行评分。

2. Content-based 评分

对于item矩阵，基于 content-based 算法得到$\mathrm{score1}$。

3. LFM 评分

根据模型训练时候得到的 Q，P 矩阵，对于任意用户i想要评分的电影t，得分为：

$$\mathrm{score}(i, t) = [Q(t, :) * P(i, :)']$$

对于$\mathrm{score1}$和$\mathrm{score2}$都需要修正不规范的评分。（具体类似 validation 处理过程）

4. 总体评分与输出

最终得分为$\mathrm{score} = (1 - \alpha) * \mathrm{score1} + \alpha * \mathrm{score2}$。通过 validation 的过程，我们将$\alpha$设置的为 0.178。

三、评价

1、评价说明

我们在评价推荐系统的打分情况时，可以根据它的运行时间，RMSE，内存消耗等指标来衡量这个推荐系统的好坏。

关于运行时间，我们将一个推荐系统的操作分为 offline 和 online，用户直观体验的时间为 online 时间。offline 的时间为 offline 的操作的总时间，offline 的操作包括得到训练数据，预处理数据，模型训练，调整参数等；online 的时间

为 online 的操作的总时间，online 的操作包括在模型训练完成的前提下运行测试数据。

2、内存消耗

对于整个训练过程中，我们需要的数据集以及其空间消耗情况如下：

数据集	性质	说明	内存消耗
Q	$\text{num\_movies} * 60$ 大小的矩阵	$Y_{norm} = \text{Q} * \text{P}^T$ 奇异分解后的结果，用于 latent factor model	295 MB
P	$\text{num\_users} * 60$ 大小的矩阵		
Y_{norm}	$\text{num\_movies} * \text{num\_users}$ 大小的稀疏矩阵	Y归一化后的值	0MB(不需要)
Y	$\text{num\_movies} * \text{num\_users}$ 大小的稀疏矩阵	训练集中用户对电影的评分	
R	$\text{num\_movies} * \text{num\_users}$ 大小的稀疏矩阵	统计训练集中是否存在用户对电影的评分	7.76 MB
user_mean	$\text{num\_users} * 1$ 大小的矩阵	每个用户评分平均值	6.99MB
user_attr	$\text{num\_users} * \text{num\_attribute}$ 大小的稀疏矩阵	每个用户对一种属性的评分	
Attr	$\text{num\_movies} * \text{num\_attribute}$ 大小的稀疏矩阵	每个电影拥有哪些属性	
合计			309.75MB

3、时间消耗

以下是 offline（主要）操作的时间消耗情况的统计结果：

操作	运行时间(min)
LFM 模型训练	438
Content-based 模型训练	206
利用验证集来确定α	70.97
总时间	714.97

以下是运行测试数据（给测试集打分的 online）的统计结果：

操作	运行时间(s)
加载测试集数据	6.787988
CFR 方法得分 score1	0.795998
Content-based 得分 score2	3833.948237
综合得分 $\text{score} = (1-\alpha) * \text{score1} + \alpha * \text{score2}$	0.000355
输出文件	48.910318
总时间	3890.2

4、验证集的 RMSE 以及参数α

RMSE	15.3577
α的值	0.178

5、结果分析与说明

可以发现，在 online 阶段，基于内容的推荐方法预测分数时相较于 LFM 方法预测分数时更加费时，一方面因为稀疏矩阵的索引很慢，另一方面 content-based 算法的计算量更大。

对于所以如果偏向加快运行时间，可以只使用 CFR 的 latent factor model 算法 ,如果偏向综合考虑所有已知数据集 ,就采用综合基于内容和 LFM 的办法。

包括加载预先训练的模型数据，读测试集数据，计算测试集评分，写回文件的所有过程，比较两者的时间消耗：

方法	空间消耗(MB)	时间消耗(s)	RMSE	考虑范围
综合两者	309.75	3890.2	15.3577	全面
仅采用 CFR	302.76	56.26	15.7735	片面

参考文献

[1] 李忠俊. 一种基于内容和协同过滤同构化整合的推荐系统模型. Computer science 36 卷 12 期

[2] 曹毅. 基于内容和协同过滤的混合模式推荐技术研究.