一. 数据整理

数据整理实验使用了全部数据来构建矩阵。

直接对所提供的数据导入，先记录用户 ID 与实际的数目的对应关系，然后生成 USER*ITEM 的训练和测试矩阵，并随之生成对应的数据描述矩阵，movie.txt 中各行不等长无法直接导入，需要逐行读取，并且使用逗号分隔，由于电影名称当中有逗号会带来干扰，生成数据的时候需要使用前两个逗号进行分割才能正常读入数据。由于本次实验数据较少，最后一组数据 movie.txt 直接使用 excel 进行处理，并读入。

实际代码如下，代码文件为 datapro.m

```
%构建用户ID数据表
username = load('users.txt');
[m,n]=size(username);
for i =1:m
    users(username(i)) = i;
end
save users.mat users;
clear username;
%构建训练集行为矩阵
tic
train = load('netflix_train.txt');
train = train(:,1:3);
[m,n]=size(train);
for i =1:m
    trainM(users(train(i,1)),train(i,2))=train(i,3);
end
trainA=double(trainM~=0);
toc
save trainMitrx.mat trainM trainA;
clear trainM trainA
t1=clock;
%构建测试集行为矩阵
test = load('netflix_test.txt');
test = test(:,1:3);
[m,n]=size(test);
for i= 1:m
    testM(users(test(i,1)),test(i,2))=test(i,3);
end
testA=double(testM~=0);
t2=clock;
etime(t2,t1);
save testMitrx.mat testM testA;
clear all;

[movieN,~, movieM] = xlsread('movie.xlsx');
save movieM.mat movieN movieM
clear all;
```

二. 协同过滤

通常来说，协同过滤有基于用户的协同过滤和基于内容的协同过滤。本次实验实现的协同过滤基于用户的相似性，当然如果在本次实验当中知道内容也就是说电影之间的相关性，比如类别，也可以做基于内容的协同过滤。

本次实验基于用户的协同过滤算法步骤如下：

1. 根据原始训练集构建用户相似矩阵。
2. 在测试集上，根据用户相似矩阵，预测每一项的值。实际上只需要考虑存在值得点。但是根据公式，需要使用所有用户的相似度之后来进行归一化。RMSE 计算公式当中的 n 应该为测试集项的数目。

本次实验，在 win 8,16G RAM,Matlab R2017a 平台上实现。由于基本上都使用矩阵实现，相对运行较快，协同过滤实验运行时间约为 34s,RMSE 值为 1.0184

上面的计算在预测分数的时候对分母还是采用相似矩阵乘以描述矩阵来计算，也就是只加权点评过的用户的数据，而对未点评过的用户不进行处理，因为未点评的数据分数按照 0 来处理，如果都进行加权的话，会导致预测结果偏小，RMSE 变大，实验表明，如果对所有用户进行加权平均，RMSE 约为 2.56，和预期相似。

代码实现见文件 UserCF.m 内容如下：

```
%% 实现基于用户的协同过滤
load testMitrx.mat;
load trainMitrx.mat

t1=clock;
%计算相似度矩阵
norm=sqrt(sum(trainM.^2,2)); % 求训练集范数
normM=norm*norm'; %  范数矩阵
sim=(trainM*trainM')./normM;% x*y/|x||y|
% 预测分数，计算RMSE
% normS=repmat(sum(sim,2),1,10000);

score=((sim*trainM)./(sim*trainA)).*testA; % 预测并做归一化,选择有分数的测试点
% normS=repmat(sum(sim,2),1,10000);
% score=((sim*trainM)./normS).*testA; % 预测并做归一化,选择有分数的测试点
rmse=RMSE(score,testM);
t2=clock;
disp(['运行时间：',num2str(etime(t2,t1)),'s'])

%% 计算RMSE
function [rmse]= RMSE(A,B)
diff=A-B;
tem=double(diff~=0);
n=sum(sum(tem));
rmse=sqrt(sum(sum(diff.*diff))/n);
end
```

三. 基于梯度下降的矩阵分解算法

这是一个非常常见的优化问题，使用最基本的 SGD 算法来进行系数的优化，在用 UV 乘

积逼近原始矩阵的情况下，增加对矩阵范数控制，减小过拟合。

算法的基本步骤如下：

（1） 随机初始化矩阵

（2） 计算梯度

（3） 更新权值参数

（4） 判断停止条件是不是成立

实际代码见 Mdec.m

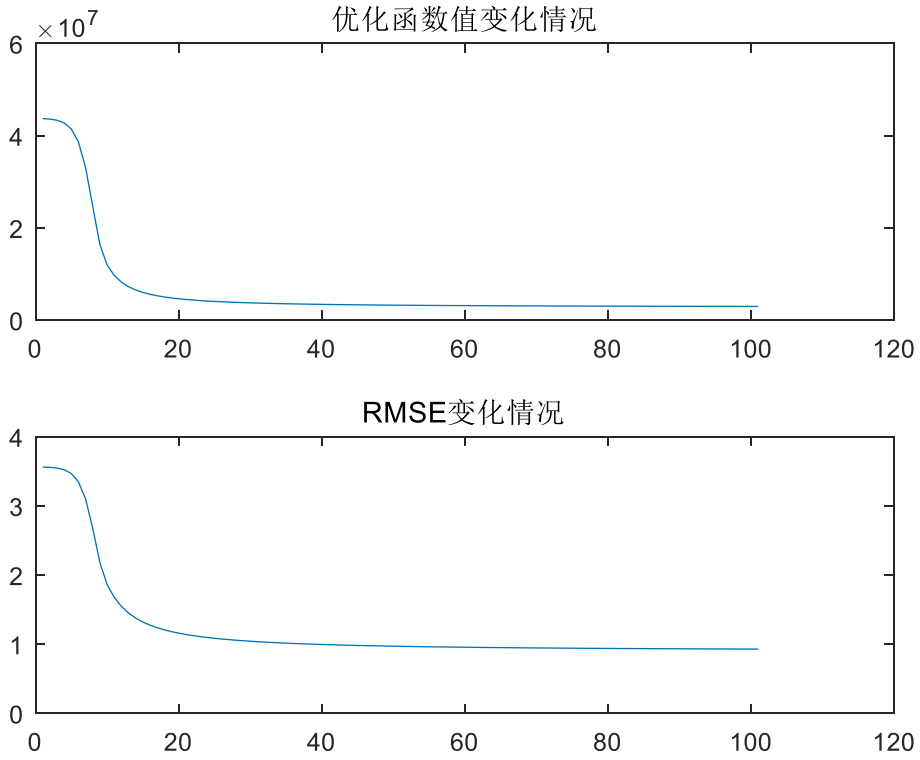
```
load testMitrx.mat;
load trainMitrx.mat


lr=0.0001;%设定α的值
weight=0.01;%设定λ的值
k=100;%设定k的值
 U=0.01*rand(10000,k);%初始化U矩阵
 V=0.01*rand(10000,k);%初始化V矩阵

 thresld=10E-6;
 pos=1;
 loss=0;
 RM=0;
 t1=clock;
while( pos>0 )
    diff=trainA.*(U*V'-trainM);
    dU=diff*V+2*weight*U;%J对U的偏导
    dV=diff'*U+2*weight*V;%J对V的偏导
    U=U-lr*dU;%更新U
    V=V-lr*dV;%更新V
    [J,R]= floss(U,V,weight,trainA,trainM,testA,testM)%计算loss以及RMSE
    loss=[loss,J]; % 存储loss
    RM=[RM,R];
    if(pos>100) %迭代停止条件
        break;
    end
%     if( abs(J-loss(pos))<thresld)%循环结束的判定条件，R的值低于阈值
%         break;
%     end
    pos=pos+1;
    t3=clock;
end
t2=clock;
disp(['运行时间: ',num2str(etime(t2,t1)),'s'])
%%  作图
loss=loss(2:end);
RM=RM(2:end);
subplot(2,1,1)
plot(loss)
title('优化函数值变化情况')
subplot(2,1,2)
plot(RM)
title('RMSE变化情况')


%% 计算loss

function [J,R]= floss(U,V,weight,trainA,trainM,testA,testM)
    XX=U*V';%产生估计矩阵
    diff=trainA.*(XX-trainM);% 估计矩阵结果和已有值之间的误差。
    J=sum(sum(diff.*diff))*0.5+sum(sum(U.*U))*weight+sum(sum(V.*V))*weight;%计算目标函数J的值
    testX=testA.*XX;
    R=RMSE(testX,testM);
end



%% 计算RMSE
function [rmse]= RMSE(A,B)
diff=A-B;
tem=double(diff~=0);
n=sum(sum(tem));
rmse=sqrt(sum(sum(diff.*diff))/n);
end
```

1） 给定 k=50，lamda=0.01 的情形

设定算法终止条件是两次 loss 之间的差值小于 10e-6，但是由于运行时间过长，所以将算法的终止条件变成迭代 100 次之后停止。到此除学习率之外的所有超参数都已经确定。

手动设定学习率 alpha 为 0.01，会导致 loss 曲线先下降，然后迅速上升，这是典型的由于数据太少导致的过拟合。典型的解决办法是使用 early stop，然后调小学习率。本实验由于相对简单，不再赘述。

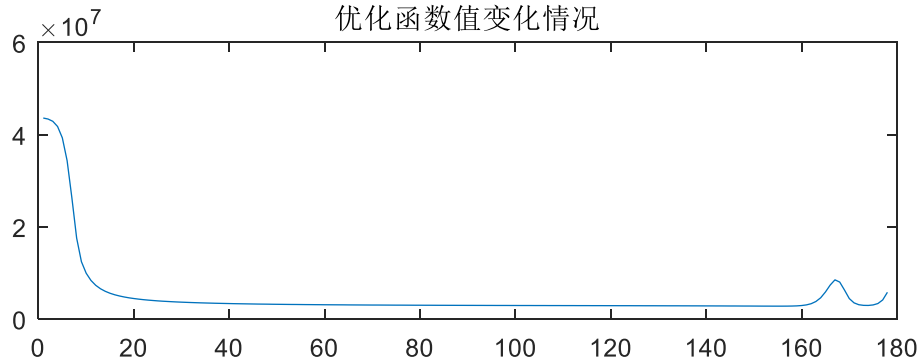
手动设定学习率 alpha 为 0.0001，算法的收敛曲线如下：

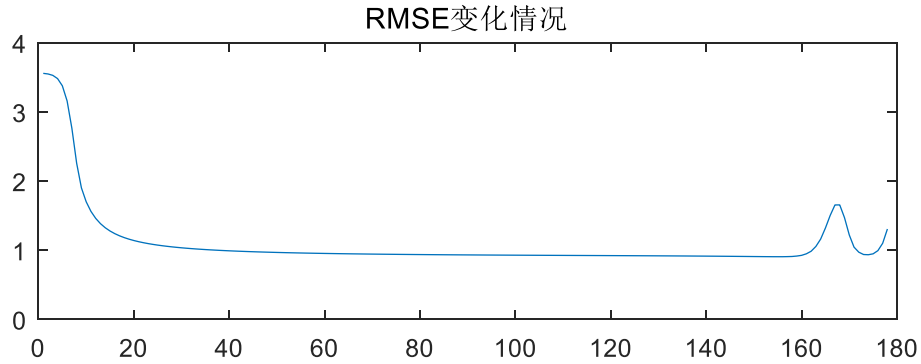


结果分析：

显然，在将学习率改小之后，算法能够相对比较稳定的收敛，这和预期一致。本次实验在结束的时候 RMSE 为 0.9272，运行时间约 365s，基本上在迭代 100 次的时候函数已经基本收敛。本次实验使用的是最简单的梯度下降算法，如果需要进一步调优，可以进一步调小学习率，然后得到更好的结果。

简单分析给定的优化函数，如果最小化的只是 $X-U*V^T$ 的范数，则会使未知的部分的估计值基本为 0，因为 X 在未知部分的值为 0，增加指示矩阵以后，$U*V^T$ 对未知部分的估计被指示矩阵消除，因为指示矩阵在未知部分的值为 0，做阿达玛积的目的也在于此。通常来说，给定一个小的随机初始矩阵以及一个小的学习率，结果通常能够收敛，实验结果和预期吻合。但是如果设置较大的初始值或者较大的学习率，loss 曲线也会出现跑飞的情况。实验当中，如果学习率设置的相对较大，loss 曲线可能会在后面收敛的时候震荡。比如出现下图情况：

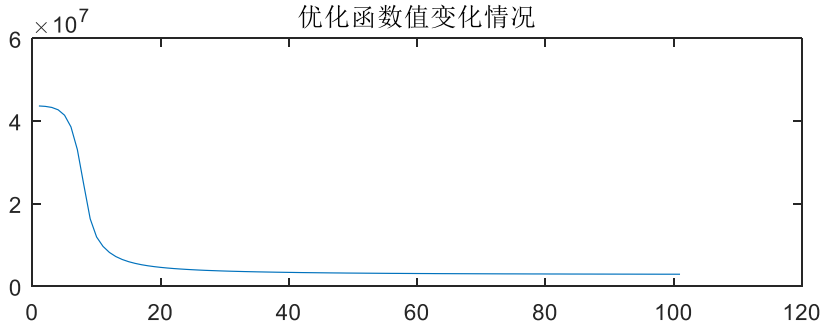


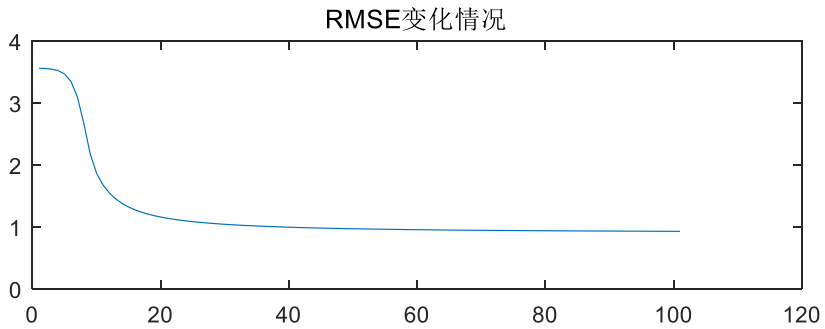


2） 调整参数，求解最优解。

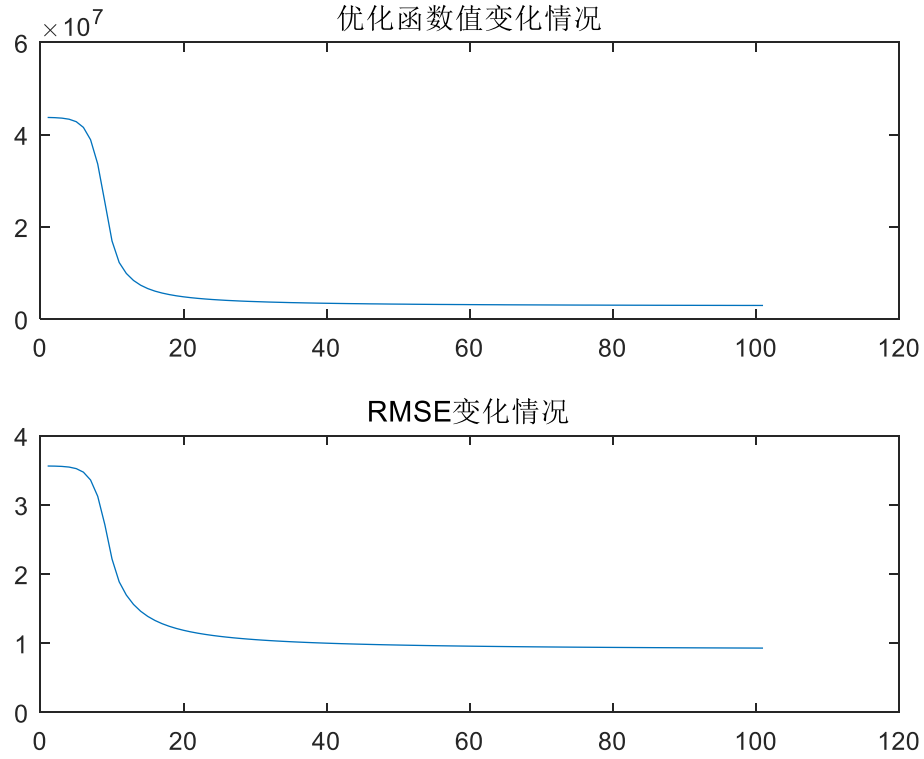
以下实验学习率都为 0.0001，迭代停止条件为迭代次数不超过 100 次。

给定 lamda=0.1,k=50 的情况下，RMSE 为 0.9273，收敛曲线如下：

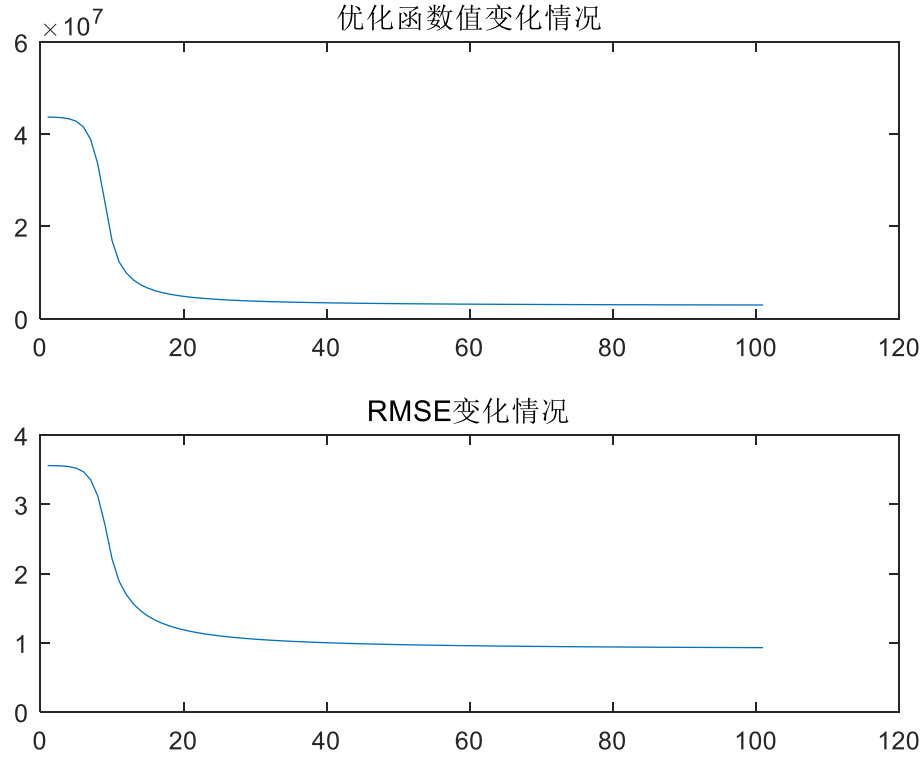




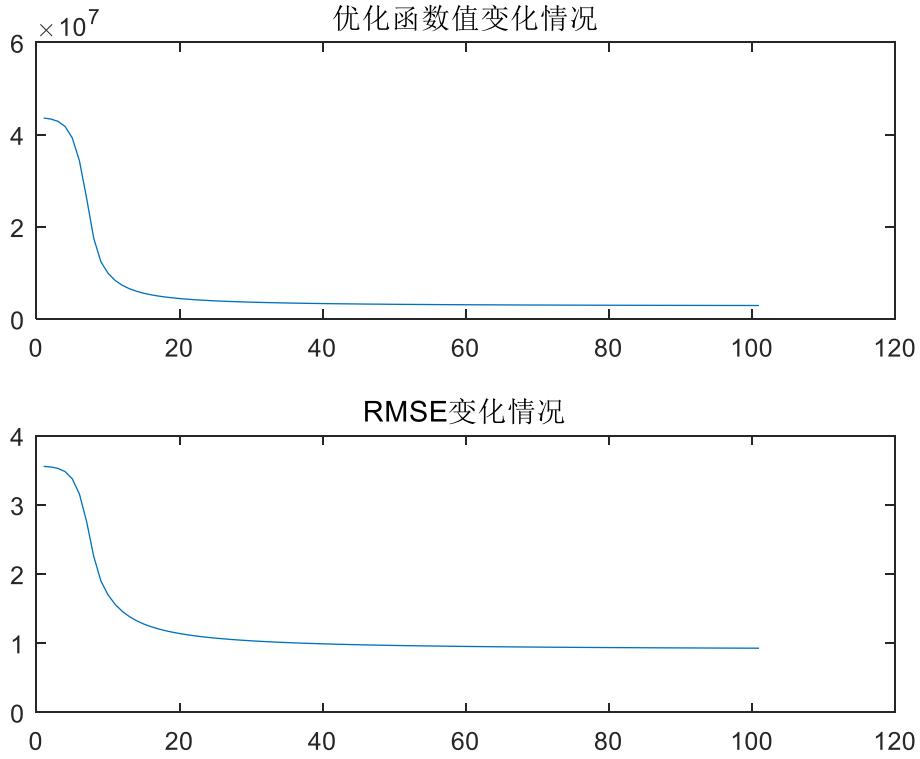
给定 lamda=0.01,k=20 的情况下，RMSE 为 0.9280，收敛曲线如下：



给定 lamda=0.1,k=20 的情况，RMSE 为 0.9281，收敛曲线如下：



给定 lamda=0.01，k=100，RMSE 为 0.9265



结果简单分析：不同的 k 值对 RMSE 影响相对较大，lamda 的取值对结果的影响不是很大，所以就目前的实验结果而言，单从 RMSE 来看，k=100，lamda=0.01，效果更好一点。但是存储的矩阵 U,V 的数据增多。可以预期，增大 k 值，RMSE 还会进一步提高，最优取决于我们在复杂度和 RMSE 之间做什么样的取舍。实验当中如果将收敛条件改成两次 RMSE 之差小于一定值，RMSE 的结果还会进一步提高。

四. 结果对比与分析

对比两种算法，协同过滤算法可理解性强，计算原理直观简单，运行时间也相对比较短，通常来说能够取得很好的预测结果，如果再增加一些额外的信息，对用户的相似性进行修正，可能会得到更好的效果。如果能够得到更多的数据，结果也会更好。但是我们也应该看到，协同过滤的算法在用户增加项目之后，需要对原始的矩阵进行重新计算，这样计算量相对较大，成本较高。

矩阵分解算法，理论上讲如果参数调整合适，从量化的角度，会比协同过滤得到更好的近似效果，但可解释性相对较差。对矩阵的分解可以理解为抽取原始矩阵当中重要信息的过程，低秩的近似可以看成去噪声的过程。矩阵分解算法如果增加数据，可以在原始结果的基础上进行简单的迭代，就可以得到很好的结果。矩阵分解算法的问题在于优化起来相对比较困难，选择最优超参数的过程非常耗时。

五. 选作部分

考虑给出的打分时间信息，实际上时间信息可以给出用户在某一时间段对某一

类电影的喜好，如果从时间上来看，打分时间越靠近的两个用户，通常来讲这一阶段的兴趣也比较相似。考虑对用户相似度再增加一项，也就是时间维度，待预测用户和用户的打分时间差作为参考，则其相似度可以定义为：

$$Sim(x,y)=x*y/|x||y|+a(1-|t_x-t_y|/b)$$

可以看出，当两个用户打分时间相等的时候，我们增加一个值 a，如果值不相等，则对差值除以 b，（本次实验当中用户打分集中在 1999-2005，可以取值为 10）。我们也可以看到这样增加一项之后，两个用户的相似度可以大于 1，简单分析之后其实这并没有什么影响，因为最终的求预测分数的时候会除以所有相似度求和的值，类似于 softmax 当中做归一化，所以单个 sim 本身的值大小并没有什么影响。

本次实验当中，设置 a 为 1/1000，b 为 10，最终的 RMSE 值为 1.019

设置 a 为 1/5000,b 为 10 的时候，最终的 RMSE 值为 1.0187

设置 a 为 1/6000,b 为 10 的时候，最终的 RMSE 值为 1.0187

从 RMSE 的角度，最终的结果相比原来的协同过滤的结果差别并不是很大。虽然调整参数有可能使得 RMSE 小于协同过滤的结果，但是这样的性能提升显然并不能令人满意。分析其中的原因，显然在相似矩阵增加了修正项之后，如果完全没有作用，由于 bias 给出的值很大，对原来的结果会产生很大影响，导致性能下降，但是本次实验当中，性能基本相当，原因可能在于这两个度量方式可能存在很强的相关性，增加的时间信息并没有给出太多有用的信息，才产生了现在的结果。

数据处理和代码实现文件为 选作/datapro.m 选作/userCF.m

具体实现代码如下

```
%% 实现基于用户的协同过滤
load testMitrx.mat;
load trainMitrx.mat
load testT.mat;
load trainT.mat

t1=clock;
%计算相似度矩阵
norm=sqrt(sum(trainM.^2,2)); % 求训练集范数
normM=norm*norm'; %  范数矩阵
sim=(trainM*trainM')./normM;% x*y/|x||y|
%增加修正项
bias=zeros(10000,10000);
for i=1:10000
    buffer=repmat(trainT(i,:),10000,1);
    dif=abs(trainT-buffer);%求解时间差
    dif(dif>10)=0;
    A=double(dif~=0);
    dif=A.*(1-dif/10)/1000;% 1-时间差/10,由于矩阵稀疏，考虑a=1/1000，使总和不超过1
    cor=sum(dif,2);
    cor(i)=0;
    bias(i,:)=cor';
    bias(:,i)=cor;
end
bias=bias+eye(10000,10000);
%进行相似性修正
sim=sim+bias;

% 预测分数，计算RMSErmse
score=((sim*trainM)./(sim*trainA)).*testA; % 预测并做归一化,选择有分数的测试点
rmse=RMSE(score,testM);
t2=clock;
disp(['运行时间：',num2str(etime(t2,t1)),'s'])

%% 计算RMSE
function [rmse]= RMSE(A,B)
diff=A-B;
tem=double(diff~=0);
n=sum(sum(tem))
rmse=sqrt(sum(sum(diff.*diff))/n);
end
```