统计信号处理大作业

实验报告——应用最小二乘求解协同滤波问题

李　溢 2016011235

2019 年 5 月 23 日

目录

1　问题描述

关键词：协同滤波，填充稀疏矩阵

问题的经典背景为：假设有 1000 个用户和 200 部电影，用户 i 对电影 j 的打分为 M_{ij}，因此形成了一个 100x200 的评分矩阵 M，由于每个用户不一定对所有电影都打过分，因此矩阵 M 不会被完全填充。所要解决的问题就是填充矩阵。

2　问题建模

实验中已知稀疏矩阵 $M \in \mathbb{R}^{N\times S}$，假设满足 $M = UV^T$，其中 $M_{ij} = U(i,:)V(j,:)^T$，$U \in \mathbb{R}^{N\times rank}$，$V \in \mathbb{R}^{S\times rank}$，$rank << N, S$ 为一个常数，可近似看作 U、V 的秩。假设矩阵 M 低秩，由于矩阵的秩为矩阵奇异值向量的 0 范数，可将其放缩为 1 范数，即矩阵的核范数。矩阵的核范数满足

$$\|\mathbf{X}\|_* = \min_{\mathbf{U},\mathbf{V}|\mathbf{X}=\mathbf{U}\mathbf{V}^T} \frac{1}{2}\left(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2\right)$$

因此将上述问题转化为优化以下目标函数进行求解

$$\min_{\mathbf{U},\mathbf{V}} \left\|\mathbf{W} * \left(\mathbf{M} - \mathbf{U}\mathbf{V}^T\right)\right\|_F^2 + \frac{\lambda}{2}\left(\|\mathbf{U}\|_F^2 + \|\mathbf{V}\|_F^2\right)$$

其中 λ 是控制矩阵低秩程度的超参数，W 是标志矩阵，$W(i,j) = 1$ 表示 $M_{ij} \neq 0$，$*$ 表示矩阵对应元素相乘。

3　求解算法

3.1　ALS 交替最小二乘

3.1.1　优化目标

考虑优化如下目标函数

$$\mathbf{L}(\mathbf{U},\mathbf{V}) = \sum_{M_{xi}\neq 0}\left(\mathbf{M_{xi}} - \mathbf{U_{x:}}\mathbf{V_{i:}^T}\right)^2 + \lambda\left(|\mathbf{U_x}|^2 + |\mathbf{V_i}|^2\right)$$

其中 $U_{x:}$ 表示 U 的第 x 行，$V_{i:}$ 表示 V 的第 i 行，M_{xi} 表示用户 x 对用户 i 的评分。最后一项为正则化项。

3.1.2 最小二乘闭式解

先对 $U_{x:}$ 应用最小二乘

$$\frac{\partial \mathrm{L}}{\partial \mathrm{U_{x:}}} = 2\mathrm{V^T}\left(\mathrm{VU_x^T} - \mathrm{M_x^T}\right) + 2\lambda \mathrm{U_x^T} \tag{1}$$

令其等于 0，求出

$$\mathrm{U_x^T} = \left(\mathrm{V^TV} + \lambda \mathrm{I}\right)^{-1} \mathrm{V^TM_{x:}^T} \tag{2}$$

同理，对 $V_{i:}$ 应用最小二乘，得到

$$\frac{\partial \mathrm{L}}{\partial \mathrm{V_i}} = 2\mathrm{U^T}\left(\mathrm{UV_i^T} - \mathrm{M_i}\right) + 2\lambda \mathrm{V_{i:}^T} = 0 \tag{3}$$

$$\mathrm{V_i^T} = \left(\mathrm{U^TU} + \lambda \mathrm{I}\right)^{-1} \mathrm{U^TM_{:i}} \tag{4}$$

3.1.3 迭代求解

每次先固定 V 求解 U，再固定 U 求解 V，这样交替进行优化，算法伪代码如下

Algorithm 1 ALS 协同滤波

Input: $M \in \mathbb{R}^{N\times S}$, λ,

Output: $X = UV^T$, $U \in \mathbb{R}^{N\times rank}$, $V \in \mathbb{R}^{S\times rank}$, $rank << N, S$

random initialization of U and V

for step < max step **do**

  $\mathrm{U^T} = \left(\mathrm{V^TV} + \lambda \mathrm{I}\right)^{-1} \mathrm{V^TM^T}$

  $\mathrm{V^T} = \left(\mathrm{U^TU} + \lambda \mathrm{I}\right)^{-1} \mathrm{U^TM}$

end for

3.1.4 实验结果

算法收敛时间大概为 1 min，超参数主要是特征的数量 rank 和 λ。

首先我固定 $\lambda = 10$，尝试了不同的 rank，计算在测试集上的 mse

rank(λ=10)	10	30	100
train mse	88.2968	71.5975	40.3695
test mse	98.635	105.2446	142.0029

表 1: rank 对 ALS 算法结果的影响

由此看来并不是特征的数量 rank 越大越好，rank 过大时很可能在训练集上产生了过拟合。

然后我固定 rank=10，取不同的 λ 值，计算在测试集上的 mse

λ(rank=10)	0.1	1	10	100	500	1000
mse	96.824	96.8548	97.3392	102.6232	131.8353	154.6974

表 2: λ 对 ALS 算法结果的影响

因此 $\lambda \leq 100$ 时的值对优化实际影响并不大，增大后优化效果下降。查看后我发现 $\lambda = 100$ 时 U^TU 与 V^TV 对角线元素的平均值大约在 350，此时式 2 和 4 中 λI 一项对于 U、V 的解影响不大。当 $\lambda = 1000$ 时，其对角元素平均值大约在 100。因此可以确认 λ 影响了 U、V 的低秩程度，当 λ 过大时，U、V 的秩偏少，不足以表征 M 对应的用户和电影的特点，因此优化效果变差。

注：实际实验中，最大迭代次数设为 1000。迭代终止条件除了最大的迭代次数，我还增加了一个 loss 的变化幅度，当 $abs(lossLast - lossNew)/lossNew < \epsilon$ 时也会停止迭代。其中 ϵ 也是引入的另一个超参数，不过由于实际实验中迭代时 U 和 V 的更新幅度越到后来越小，因此这一参数的影响并不很大，实验中我取 $\epsilon = 1e-7$

3.2 基于梯度下降的协同滤波

3.2.1 前述 ALS 算法的问题

根据上面基于 ALS 算法的结果，可以看到测试集上的 mse 数值在 100 左右，也就意味着平均预测值与真值的误差约为 10，而经过计算，实际测试数据的均值约为 13，这意味着实际上 ALS 算法得到的结果中，在测试数据对应的点上的预测值基本为 0。这并不意外，因为 ALS 的算法推导中存在一定的问题。

我们首先考虑式 1 和 3，真正的推导过程应为

$$\frac{\partial \mathrm{L}}{\partial \mathrm{U_x}} = 2\sum_{\mathrm{i,M_{xi}}\neq 0} V_{i:}^T \left(\mathrm{V_{i:}U_{x:}^T} - \mathrm{M_{xi}}\right) + 2\lambda \mathrm{U_x^T} \neq 2\mathrm{V^T}\left(\mathrm{VU_x^T} - \mathrm{M_x^T}\right) + 2\lambda \mathrm{U_x^T} \tag{5}$$

需要注意的是，这里只取了 $\{i|M_{xi} \neq 0\}$ 这一部分的 i 来计算关于 U_x 的梯度，严格来说不能化成式子 1 中的矩阵表达形式，因为式 1 中实际上相当于把原来没有观测数据的 M_{xi} 完全看作了 0，也即**认为这些点有观测数据，但是观测数据就等于 0**，由此优化出来的 U、V 必然是不对的，因为优化之后在这些点上的预测时会很接近于 0。

3.2.2 基于梯度下降的优化算法

仍旧考虑优化以下目标函数进行求解

$$L(U,V)=\min_{\mathbf{U},\mathbf{V}}\left\|\mathbf{W}*\left(\mathbf{M}-\mathbf{U}\mathbf{V}^{T}\right)\right\|_{F}^{2}+\frac{\lambda}{2}\left(\|\mathbf{U}\|_{F}^{2}+\|\mathbf{V}\|_{F}^{2}\right)\tag{6}$$

而前述 ALS 的问题实际上就在于计算梯度的过程中忽略了标志矩阵 W 及对应的点乘操作。如果保留矩阵 W，将式 6 分别对 U 和 V 求导，得到

$$\frac{\partial \mathrm{L}}{\partial \mathrm{U}}=2\left(\mathrm{W}*\left(UV^{T}-M\right)\right)\mathrm{V}+\lambda\mathrm{U}\tag{7}$$

$$\frac{\partial \mathrm{L}}{\partial \mathrm{V}}=2\mathrm{U}^{\mathrm{T}}\left(\mathrm{W}*\left(UV^{T}-M\right)\right)+\lambda\mathrm{V}\tag{8}$$

很容易验证，如果将上式中的 W 忽略后，就与前面推导的式 1、3 等价，但是实际上 W 并不能忽略，也不能直接分配到括号里边。上面两个式子并不能求出关于 U、V 的闭式解，因此不能直接应用最小二乘法，可以考虑应用梯度下降法优化目标函数。

3.2.3 梯度下降求解

每次基于 L 对 U 和 V 的梯度分别更新 U 和 V，交替进行优化，算法伪代码如下

Algorithm 2 基于梯度下降的协同滤波

Input: $M\in\mathbb{R}^{N\times S}$, λ, lr(learning rate)

Output: $X=UV^{T}$, $U\in\mathbb{R}^{N\times rank}$, $V\in\mathbb{R}^{S\times rank}$, $rank<<N,S$

random initialization of U and V

for step < max step **do**

$\frac{\partial \mathrm{L}}{\partial \mathrm{U}}=2\left(\mathrm{W}*\left(UV^{T}-M\right)\right)\mathrm{V}+\lambda\mathrm{U}$

$U=U-lr*\frac{\partial \mathrm{L}}{\partial \mathrm{U}}$

$\frac{\partial \mathrm{L}}{\partial \mathrm{V}}=2\mathrm{U}^{\mathrm{T}}\left(\mathrm{W}*\left(UV^{T}-M\right)\right)+\lambda\mathrm{V}$

$V=V-lr*\frac{\partial \mathrm{L}}{\partial \mathrm{V}}$

end for

3.2.4 实验结果

与 ALS 相似，特征数目 rank 过大时会出现严重的过拟合，因此最终取 rank=10，$\lambda=10$，梯度下降法中主要影响因素为学习率 lr 的调整。总的来说，实验中 lr>0.0002 时基本很快就会爆炸，即不收敛；lr 过小时优化速度太慢。最终我采用了阶梯衰减的学习率调整方案

step	0-12	13-30	31-240	241-1000
lr	0.0002	0.00015	0.00013	0.00011

表 3: 学习率衰减

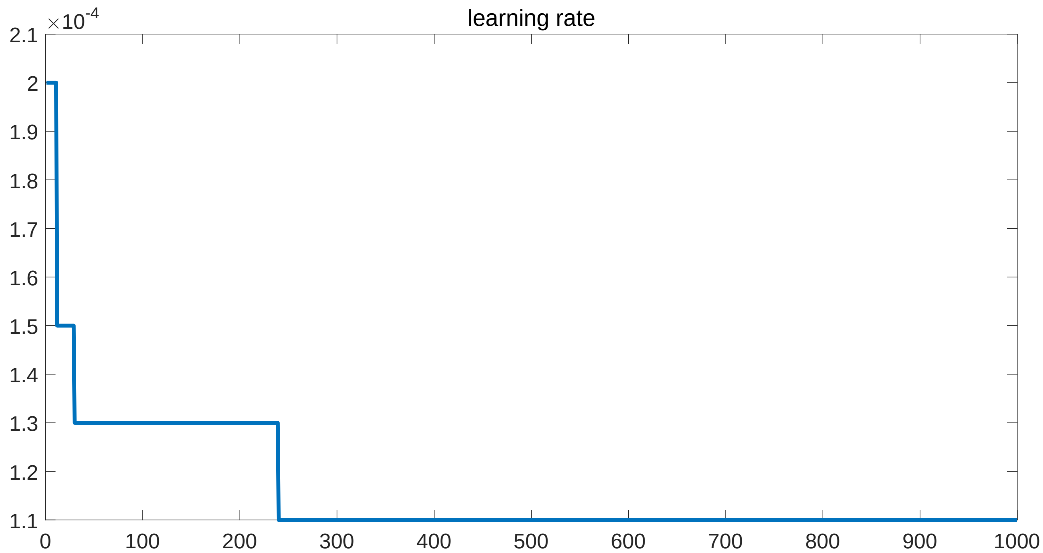

图 1: 学习率衰减方案

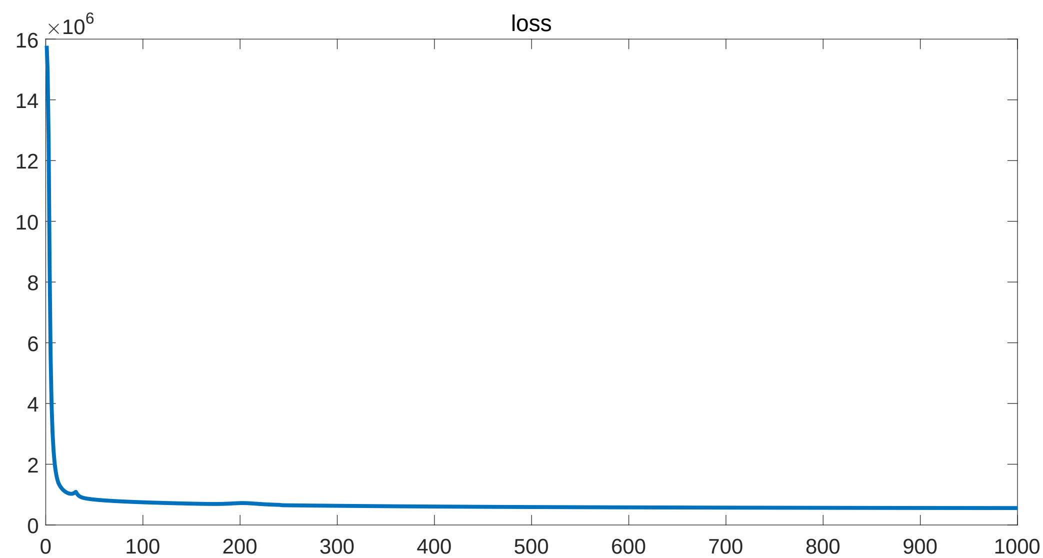

图 2: 梯度下降

实验中任何一个超参数的改变都需要对学习率衰减进行调整，当 λ 或 rank 较大时就需要调小 lr，因为此时梯度值较大，如果 lr 太大就会不收敛，衰减的区间也需要对应调整，最终的结果基本上是我针对 λ、rank、lr 等参数精细调整过的，也是实验中取得的最好的效果。最终效果如下，可以看到 mse 只有 8，平均每个观测数据的误差为 3 左右，而观测数据的真实平均值约为 13 左右，因此相对较准确，相比于之前的 ALS 算法则有很大的提升。

rank	λ	train mse	test mse	收敛时间
10	10	4.4583	8.8111	$\sim$ 1 min

表 4: 梯度下降算法结果

除此之外，与上面相同，迭代终止条件除了最大迭代次数，我还增加了一个 loss 减小的幅度 $\epsilon = 1e-5$。

4 附录：源代码

注：由于编码问题，部分中文注释存在乱码

4.1 ALS 算法

```
% »      ALS µ      熸
clear all; close all; clc;
load data_train.mat;
%% hyperparameters
ranks = [10,30,100];
lambdas = [0.1,1,10,100,500,1000];      % ²» µ lambda² ¶100µ  ¹
epsilon = 1e-7;%0.00001;      % limit of iteration
%% randomly select train data and test data
%U = randn(943, rank);
%V = randn(1682, rank);
idx = randperm(90000,80000);
train = data_train(idx,:);
temp = 1:90000;
temp(idx) = 0;
temp(temp==0) = [];
test = data_train(temp,:);
```

```
% for train
M = zeros(943, 1682);
W = zeros(943, 1682);
M(sub2ind(size(M), train(:,1), train(:,2))) = train(:,3);
W(sub2ind(size(W), train(:,1), train(:,2))) = 1;
W0 = W;
W0(sub2ind(size(W0), test(:,1), test(:,2))) = 1;   % all data
% for test
Mtest = zeros(943, 1682);
Wtest = zeros(943, 1682);
Mtest(sub2ind(size(M), test(:,1), test(:,2))) = test(:,3);
Wtest(sub2ind(size(W), test(:,1), test(:,2))) = 1;
s = size(test,1);
bestmse=1e10;
%% train
for i=1:length(ranks)
    rank = ranks(i);
    lambda = 10;%lambdas(i);
    step = 0;
    U = randn(943, rank);
    V = randn(1682, rank);
    I = eye(rank);
    lossLast = 0;
    lossNew = norm(W.*(M-U*V'), 'fro').^2 + ...
        lambda/2*(norm(U,'fro').^2 + norm(V,'fro').^2);
    disp(lossNew);
    while step≤1000 && abs(lossLast-lossNew)/lossNew>epsilon
        lossLast=lossNew;
        lastU = U;
        U = M*V/(V'*V+lambda*I);
        lastV = V;
        V = M'*U/(U'*U+lambda*I);
        lossNew = norm(W.*(M-U*V'), 'fro').^2 + ...
            lambda/2*(norm(U,'fro').^2 + norm(V,'fro').^2);
        diffUmax = max(max(abs(U-lastU)));    % ...
            }´ ´ ¼ U£¬V±仯µ ¾ º
        diffUmean = mean(mean(abs(U-lastU)));
```

```
        diffVmax = max(max(abs(V-lastV)));
        diffVmean = mean(mean(abs(V-lastV)));
        diff = W0.*(M-U*V');
        diffmax = max(max(abs(diff)));
        diffmean = mean(mean(abs(diff(W0≠0))));
        if ¬mod(step, 500)
            disp([num2str(step), '  loss: ', num2str(lossNew),...
                '  △: ', num2str(abs(lossNew-lossLast)/lossNew),...
                '  diffmax: ',num2str(diffmax),...
                '  diffmean: ',num2str(diffmean),...
                '  diffUmax: ',num2str(diffUmax),...
                '  diffUmean: ',num2str(diffUmean),...
                '  diffVmean: ',num2str(diffVmean),...
                ]);

            testmean = mean(mean(Mtest(Wtest≠0)));     % mean value ...
                of test data
            difftest = Wtest.*(Mtest-U*V');
            difftestmax = max(max(abs(difftest)));     % ¼   õ½µ  ...
                UV        ¾          º  ¾
            difftestmean = mean(mean(abs(difftest(Wtest≠0))));
            mse = 1/s * norm(Wtest.*(Mtest-U*V'), 'fro').^2;
            disp(['      test     ',num2str(mse),...
                '  mean: ',num2str(testmean),...
                '  difftestmax: ',num2str(difftestmax),...
                '  difftestmean: ',num2str(difftestmean)]);
            disp(' ');
        end
        step = step+1;
    end
    % train mse
    s = size(train,1);
    mse = 1/s * norm(W.*(M-U*V'), 'fro').^2;
    disp(['train    ',num2str(mse)]);
    % test
    Mtest = zeros(943, 1682);
    Wtest = zeros(943, 1682);
```

```
    Mtest(sub2ind(size(M), test(:,1), test(:,2))) = test(:,3);
    Wtest(sub2ind(size(W), test(:,1), test(:,2))) = 1;
    s = size(test,1);
    mse = 1/s * norm(Wtest.*(Mtest-U*V'), 'fro').^2;
    if mse < bestmse
        bestmse = mse;
        X = U*V';
        save X_als.mat X;
    end
    disp(['test     ',num2str(mse),...
        '     step',num2str(step)]);
end
```

4.2 梯度下降算法

```
% »     GD μ      燦
clear all; close all; clc;
load data_train.mat;
%% hyperparameters
rank = 10;
bound = [12, 30, 240, 600];
lrs = [0.00015, 0.00013, 0.00011, 0.00011];
lr = 0.0002;
lambda = 10;
epsilon = 0.00001;          % limit of iteration
%% learn
U = randn(943, rank);
V = randn(1682, rank);
epoch = 5;
%% randomly select train data and test data
idx = randperm(90000,80000);
train = data_train(idx,:);
temp = 1:90000;
temp(idx) = 0;
temp(temp==0) = [];
```

```
test = data_train(temp,:);
%% train
M = zeros(943, 1682);
W = zeros(943, 1682);
M(sub2ind(size(M), train(:,1), train(:,2))) = train(:,3);
W(sub2ind(size(W), train(:,1), train(:,2))) = 1;
step = 0;
train_loss = zeros(1001);
% ¸ ª -¼¶• Ĭ 1e07, 1e04, 1e05
lossLast = norm(W.*(M-U*V'), 'fro').^2 + ...
    lambda/2*(norm(U,'fro').^2 + norm(V,'fro').^2);
gradU = 2*W.*(U*V'-M)*V + lambda*U;  % ¸ ª  ¬100 -¼¶
lastGradU = gradU;
U = U - lr * gradU;
gradV = 2*(U'*(W.*(U*V'-M)))' + lambda*V;
lastGradV = gradV;
V = V - lr * gradV;
lossNew = norm(W.*(M-U*V'), 'fro').^2 + lambda/2*(norm(U,'fro').^2 ...
    + norm(V,'fro').^2);
while step<1000 && abs(lossLast-lossNew)/lossNew>epsilon
    lossLast = lossNew;
    gradU = 2*W.*(U*V'-M)*V + lambda*U;
    lastGradU = mean(abs(gradU(:)));
    U = U - lr * gradU;
    gradV = 2*(U'*(W.*(U*V'-M)))' + lambda*V;
    lastGradV = mean(abs(gradV(:)));
    V = V - lr * gradV;
    lossNew = norm(W.*(M-U*V'), 'fro').^2 + ...
        lambda/2*(norm(U,'fro').^2 + norm(V,'fro').^2);
    if ¬mod(step, 50)
        disp([num2str(step), '  loss: ', num2str(lossNew),...
                '  △: ', num2str(abs(lossNew-lossLast)/lossNew)]);
    end
    temp = find(bound==step);
    if any(temp)
        lr = lrs(temp);
    end
```

```
    step = step+1;
    train_loss(step) = lossNew;
end
s = size(train,1);
mse = 1/s * norm(W.*(M-U*V'), 'fro').^2;
disp(['train   ',num2str(mse)]);
%% test
M = zeros(943, 1682);
W = zeros(943, 1682);
M(sub2ind(size(M), test(:,1), test(:,2))) = test(:,3);
W(sub2ind(size(W), test(:,1), test(:,2))) = 1;
s = size(test,1);
mse = 1/s * norm(W.*(M-X), 'fro').^2;
disp(['test    ',num2str(mse)]);
%% save
X = U*V';
save X.mat X;
plot(train_loss(1:step),'linewidth',3);
title('loss');
set(gca,'fontsize',16);
```