Homework 2, CPSC 8420, Fall 2020

Sadeghi Tabas, Sadegh

October 15, 2020

Solution to Problem 1

1.1.

for the first one we have:

$A_{k+1} = A_k - \lambda \nabla f(A)$

$h = \frac{1}{2}||X - A_k G_k||_F^2 \hookrightarrow A_{k+1} = A_k - \lambda \nabla_A h$

$\frac{\delta h}{\delta A} = (A_k G_k - X) G_k^T \hookrightarrow A_{k+1} = A_k - \lambda (A_k G_k - X) G_k^T = A_k - \lambda A_k G_k G_k^T + \lambda X G_k^T$

set $\lambda = \frac{A_k}{A_k G_k G_k^T} \hookrightarrow A_{k+1} = A_k - \frac{A_k}{A_k G_k G_k^T} A_k G_k G_k^T + \frac{A_k}{A_k G_k G_k^T} X G_k^T \hookrightarrow A_{k+1} = A_k \frac{X G_k^T}{A_k G_k G_k^T}$

for the second one we have:

$G_{k+1} = G_k - \lambda \nabla f(G)$

$h = \frac{1}{2}||X - A_k G_k||_F^2 \hookrightarrow G_{k+1} = G_k - \lambda \nabla_G h$

$\frac{\delta h}{\delta G} = (A_{k+1}^T A_{k+1} G_k - A_{k+1}^T X) \hookrightarrow G_{k+1} = G_k - \lambda (A_{k+1}^T A_{k+1} G_k - A_{k+1}^T X)$

$\hookrightarrow G_{k+1} = G_k - \lambda A_{k+1}^T A_{k+1} G_k - \lambda A_{k+1}^T X$

set $\lambda = \frac{G_k}{A_{k+1}^T A_{k+1} G_k} \hookrightarrow G_{k+1} = G_k - \frac{G_k}{A_{k+1}^T A_{k+1} G_k} A_{k+1}^T A_{k+1} G_k + \frac{G_k}{A_{k+1}^T A_{k+1} G_k} A_{k+1}^T X$

$\hookrightarrow G_{k+1} = G_k \frac{A_{k+1}^T X}{A_{k+1}^T A_{k+1} G_k}$

1.2.

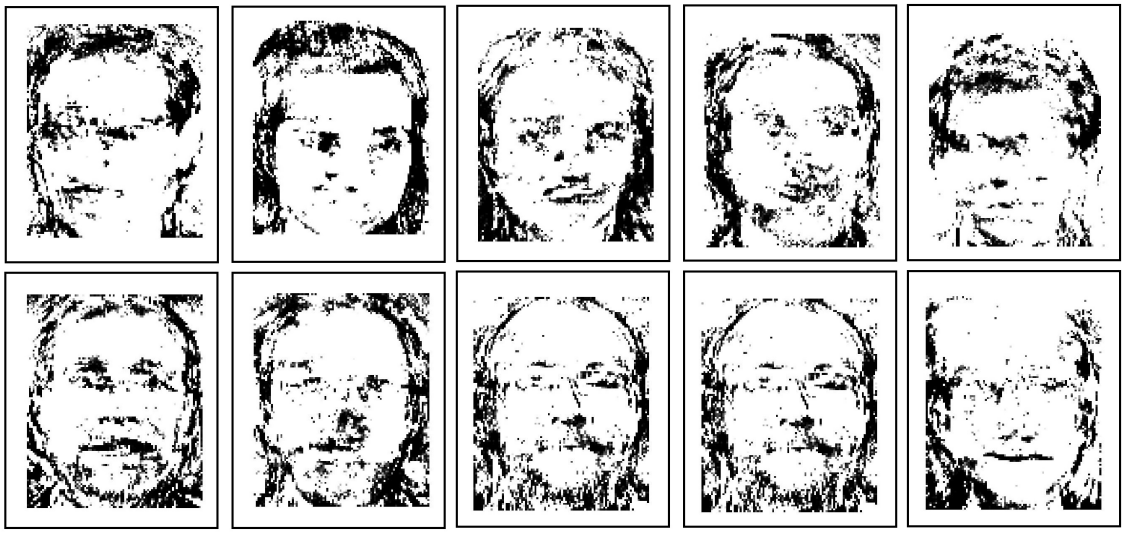

Figure 1: 10 feature faces obtained using MUA after 500 iterations

Solution to Problem 2

$f_A^{'} = \sum_{l=1}^{n} G_{kl} - \sum_{l=1}^{n} X_{il} \frac{G_{kl}}{(AG)_{il}}$

$f_A^{''} = \sum_{l=1}^{n} X_{il} \frac{G_{kl}^2}{(AG)_{il}^2}$

let $s = max(A_{ik} - \frac{f^{'}}{f^{''}}, 0)$

$d = s - A_{ik}$ and $\lambda = cA_{ik}(f_A^{''})^{0.5}|d|$

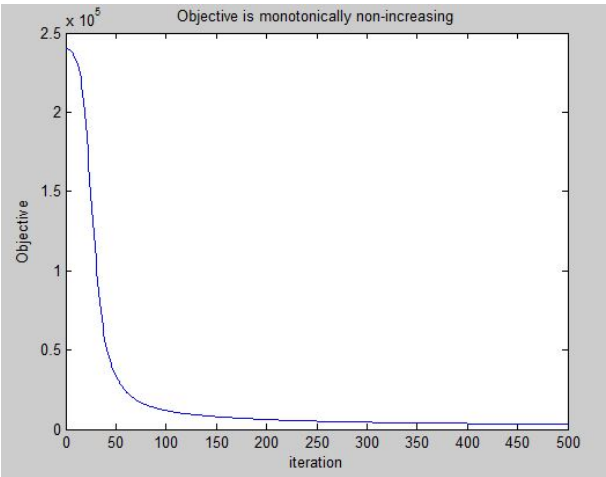

Figure 2: The above figure demonstrates that the objective is monotonically non-increasing with updates

Solution to Problem 3

3.1. if A be a matrix with more columns than rows ($p > n$), the columns of A must be linearly dependent, so the equation $Ax = 0$ must have a non-trivial solution x. Thus $(A^T A)x = A^T(Ax) = A^T 0 = 0$ and the equation $(A^T A)x = 0$ has a non-trivial solution. So since $(A^T A)$ is a square matrix, the Invertible Matrix Theorem says that the $(A^T A)$ is not invertible (and thus det $(A^T A) = 0$) so $(A^T A)^{-}1$ does not exists and the Vanilla solution is not applicable any more.

3.2.

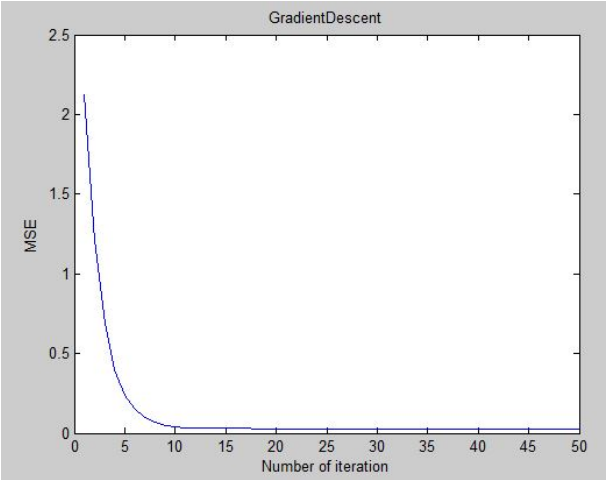

Figure 3: Gradient Descent method obtained the optimal solution with Linear Convergence rate if the fixed learning rate

3.3.

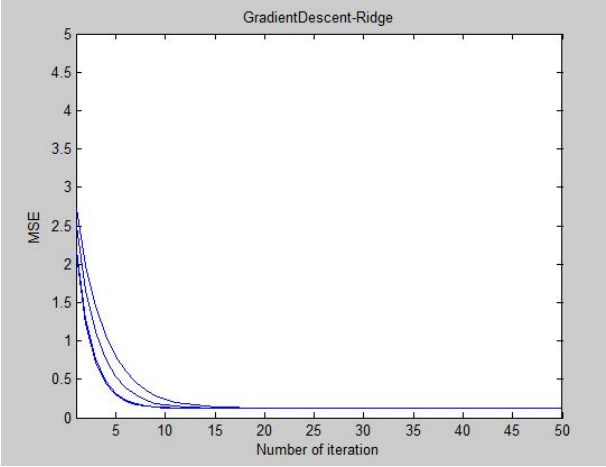

Figure 4: Gradient Descent method obtained the optimal solution for the ridge regression if the fixed learning rate and varies from 0.1, 1, 10, 100, 200

Solution to Problem 4

We can take an image which originally is rank 300+ (large size) and store it in a reasonably good representation matrix that has only i.e. rank 100. Sometimes, we can also cut off more of the original image without losing clarity. Based on the application we are going to use images we can compress images with different size and qulity, for example if we are going to put the image in a website, so the image should not have much size as user faces issues in loading the page. Thus, it is very important for us to be able to store all the image types in reasonably sized matrices. The Singular Value Decomposition makes this possible. If I want to select a good option between images provided in the figure 5, I will select the K=100, as its quality is acceptable (no significant difference with the original one based on what a human can see with their eyes) and the image size also reduced more than 300kbs (around 1/5). The option K=80 or 50 are not desired as their size is close to the one with K=100 (difference is less than 5.5kbs) but their difference in quality is sensible. Also the other 3, are not good options at all as their quality is very low.

Figure 5: Compression ratio and reconstruction image if we choose first 2, 5, 20, 50, 80, 100 components respectively

Solution to Problem 5

As we know, the first principal component corresponds to the direction in which the projected data points have the largest variance and the second component is then taken orthogonal to the first and must again maximize the variance of the data points projected on it. But in PCA, outliers could change the direction of correct principal components and result in inaccuracy. As you can see in the figure 6; A) shows the correct direction of first two PCs while B) shows the change in the direction of PCs towards the outliers. A way to deal with contaminated data is to remove the outlying objects observed on the score plots and to repeat the PCA analysis again. Another, more efficient way is to apply a robust, i.e. not sensitive to outliers, variant of PCA. Robust PCA aims to obtain principal components that are not much influenced by outliers. For example, replacing the classical covariance matrix by a robust covariance estimator, such as the weighted MCD estimator or MM-estimators.

A second approach to robust PCA is use of Projection Pursuit techniques. These methods maximize a robust measure of spread to obtain consecutive directions on which the data points are projected.

and finally, as a scalable approach to robust PCA is to compute the average subspace spanned by the data; this can then be made robust by computing robust averages[1].

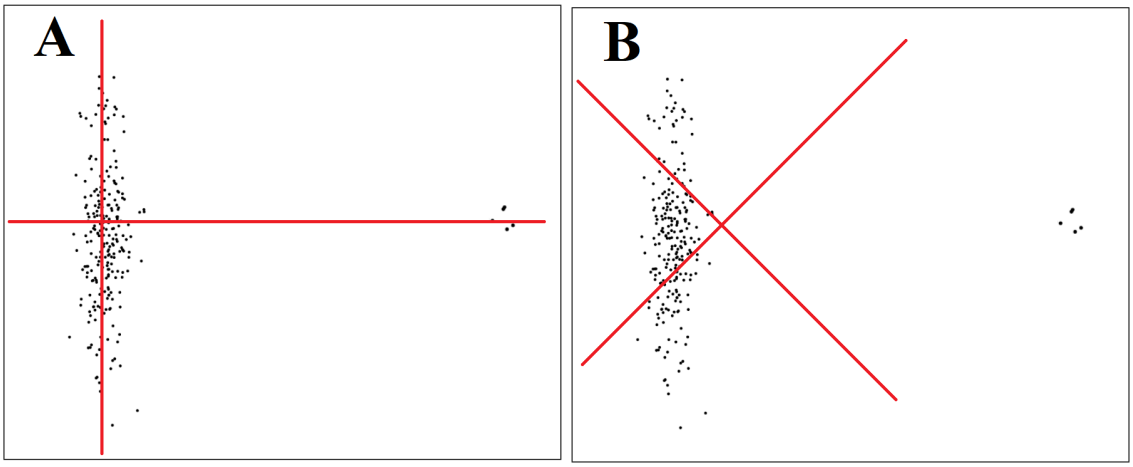

Figure 6: The first two principle components, A: without outlier; B: with outliers

References

[1] Gharibnezhad, Fahit, Luis E. Mujica, and José Rodellar. "Applying robust variant of Principal Component Analysis as a damage detector in the presence of outliers." Mechanical Systems and Signal Processing 50 (2015): 467-479.

```
% Code for the Question 1
 clc
 clear
 close all

load atnt40x10x112x92ML5FoldNaturalCrossValidationSplit.mat
X=M;
maxiter=500;
A0=rand(400,400);
G0=rand(400,10304);
A = A0; G = G0;
[m,n] = size(X);
for iter=1:maxiter
    iter
    AtA = A'*A;
    Gradient_G = AtA*G - A'*X;
    Gb =G;
    G = G - Gb./(AtA*Gb).*Gradient_G;
    GGt = G*G';
    Gradient_A = A*(GGt) - X*G';
    Ab = A;
    A = A - Ab./(Ab*GGt).*Gradient_A;
    S = sum(A,1);
    A = A./repmat(S,m,1);
    G = G.*repmat(S',1,n);
end
for i=1:10
    Vec=G(i,:);
    Mat=vec2mat(Vec,112);
    figure(i)
    imshow(Mat');
    print (figure(i),'-djpeg', num2str(i))
end
```

```
% Code for the Question 2


 clc
 clear
 close all

load atnt40x10x112x92ML5FoldNaturalCrossValidationSplit.mat
X=M;
maxiter=500;
A0=zeros(400,400);
G0=zeros(400,10304);
A = A0; G = G0;
[m,n] = size(X);
for iter=1:maxiter
    iter
    AtA = A'*A;
    Gradient_G = AtA*G - A'*X;
    Gb =G;
    G = G - Gb./(AtA*Gb).*Gradient_G;
```

```
    GGt = G*G';
    Gradient_A = A*(GGt) - X*G';
    Ab = A;
    A = A - Ab./(Ab*GGt).*Gradient_A;
    S = sum(A,1);
    A = A./repmat(S,m,1);
    G = G.*repmat(S',1,n);

    Obj(iter,:)=1/2*(sum((X-A*G).^2));
end
plot([1:maxiter],Obj(:,1))
xlabel('iteration')
ylabel('Objective')
title('Objective is monotonically non-increasing')
% for i=1:10
%     Vec=G(i,:);
%     Mat=vec2mat(Vec,112);
%     figure(i)
%     imshow(Mat');
%     print (figure(i),'-djpeg', num2str(i))
% end
```

```
% Code for the Question 3.2
 clc
clear

A = [1, 2, 4; 1, 3, 5; 1, 7, 7; 1, 8, 9];
Y = [1; 2; 3; 4];
maxiter=50;
tol=0.000001;

n = size(A,1);
beta = zeros(1,size(A, 2));
vt = rand(1, size(A, 2));
t = beta;
for i =1 : maxiter
        r = (1/n)*(A'*(A * beta' - Y));
        LR = (r' * r)/ (r' * (A' * A) * r);
        beta = beta - LR * r';
        mse = (1/(2*n)) * sum((A * beta'-Y).^2);
        err_beta = norm(beta - t)/norm(beta);
        disp(['Iteration ' num2str(i) ' : Mean-squared error = ' num2str(mse)]);
        if err_beta ≤ tol
            fprintf('\n Converged \n')
        break
        end
    t = beta;
    MSE(i)=mse;
end

plot([1:maxiter],MSE)
title('GradientDescent')
xlabel('Number of iteration')
ylabel('MSE')
```

```
% Code for the Question 3.3



clc
clear
close all
A = [1, 2, 4; 1, 3, 5; 1, 7, 7; 1, 8, 9];
Y = [1; 2; 3; 4];
maxiter=50;
tol=0.000001;

lambda=[0.1, 1 , 10, 100, 200];
for j=1:length(lambda)
    n = size(A,1);
    beta = zeros(1,size(A, 2));
    vt = rand(1, size(A, 2));
    t = beta;

for i =1 : maxiter
        r = (1/n)*(A'*(A * beta' - Y));
        LR=1/(lambda(j)+svds((A' * A),1));
        beta = beta - LR * r';
        mse = (1/(2*n)) * sum((A * beta'-Y).^2)+ sum((beta).^2);
        err_beta = norm(beta - t)/norm(beta);
        disp(['Iteration ' num2str(i) ' : Mean-squared error = ' num2str(mse)]);
        if err_beta ≤ tol
            fprintf('\n Converged \n')
        break
        end
    t = beta;
    MSE(j,i)=mse;
end

plot([1:maxiter],MSE(j,:))
hold on
axis([1 50 0 5])
end
title('GradientDescent-Ridge')
xlabel('Number of iteration')
ylabel('MSE')
```

```


% Code for the Question 4
clc
clear
close all

%% Reading Image
I = imread('Lenna.png');
imshow(I)

%% dividing RGB channel data
```

```
R = I(:,:,1);
G = I(:,:,2);
B = I(:,:,3);

R1=im2double(R);
G1=im2double(G);
B1=im2double(B);

%% Singular Value Decomposition
[ur,sr,vr]=svd(R1);
[ug,sg,vg]=svd(G1);
[ub,sb,vb]=svd(B1);

Cr=zeros(size(R1));
Cg=zeros(size(G1));
Cb=zeros(size(B1));

%% Apply different compression ratios and save coressponding reconstruction images
k=[2,5,20,50,80,100];

for i=1:length(k)
    Cr=zeros(size(R1));
    Cg=zeros(size(G1));
    Cb=zeros(size(B1));

    for j=1:k(i)
        Cr=Cr+sr(j,j)*ur(:,j)*vr(:,j).';
        Cg=Cg+sg(j,j)*ug(:,j)*vg(:,j).';
        Cb=Cb+sb(j,j)*ub(:,j)*vb(:,j).';
    end

    Cr(k(i))=1;
    Cg(k(i))=1;
    Cb(k(i))=1;
    R2=im2uint8(Cr);
    G2=im2uint8(Cg);
    B2=im2uint8(Cb);

    NewI(:,:,1)=R2;
    NewI(:,:,2)=G2;
    NewI(:,:,3)=B2;

    figure(i+1)
    imshow(NewI,[]);
    name=['Lenna(',num2str(k(i)),')']
    print (figure(i+1),'-djpeg', name)

end
```