Homework Set 2, CPSC 8420, Spring 2022

William Sherrer

Due 03/17/2022, Thursday, 11:59PM EST

Problem 1

For PCA, from the perspective of maximizing variance, please show that the solution of $\boldsymbol{\phi}$ to maximize $\|\mathbf{X}\boldsymbol{\phi}\|_2^2, s.t.\ \|\boldsymbol{\phi}\|_2 = 1$ is exactly the first column of $\mathbf{U}$, where $[\mathbf{U}, \mathbf{S}] = svd(\mathbf{X}^T\mathbf{X})$. (Note: you need prove why it is optimal than any other reasonable combinations of $\mathbf{U}_i$, say $\hat{\boldsymbol{\phi}} = 0.8 * \mathbf{U}(:,1) + 0.6 * \mathbf{U}(:,2)$ which also satisfies $\|\hat{\boldsymbol{\phi}}\|_2 = 1$.)

We say that this is equivalent to $\max tr(\phi^T X^T X \phi)$. If we compute $[\mathbf{U}, \mathbf{S}, \mathbf{V}] = svd(\mathbf{X}^T\mathbf{X})$, then we have $\phi_1 = U(:,1)$, and the variance $\|X\phi\|^2 = S^2(1,1)$. Using svd we can compute that $U\Sigma U^T = X^T X => \sum \lambda_i u_i u_i^T$. We start by letting $\phi_1 = U_1$. Therefore $\phi^T X^T X \phi$ can be rewritten as $U_1^T \Sigma \lambda_i u_i u_i^T U_1$ and since all of the columns of U are orthogonal to each other the equation will collapse to $= \lambda_1$.

Let's say that $X^T X v_1 = \lambda_1 v_1$. To find we multiply by X to get $X * X^T X v_1 = \lambda_1 X v_1$ and then further simplifying $XX^T = \lambda_1$. Therefore we can conclude that the corresponding eigenvalue is exactly equal to the variance of the data set.

The end result is that the first k principal components of X correspond exactly to the eigenvectors of the covariance matrix S ordered by their eigenvalues. Moreover, the eigenvalues are exactly equal to the variance of the dataset along the corresponding eigenvectors. Therefore we satisfy $\phi_1 = U(:,1)$, and the variance $\|X\phi\|^2 = S^2(1,1)$. This is optimal as $\hat{\phi} = \Sigma a_i u_i$ with a_i being a scalar. Since the magnitude $\|a\| = 1$ we can say that $\|\hat{\phi}\| = 1$

Problem 2

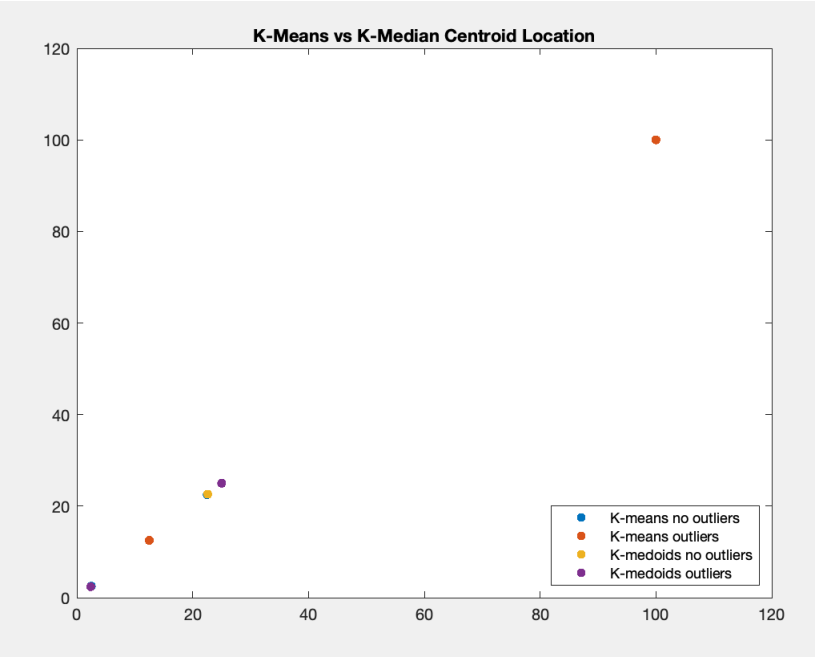
Why might we prefer to minimize the sum of absolute residuals instead of the residual sum of squares for some data sets? Recall clustering method K-means when calculating the centroid, it is to take the mean value of the data-points belonging to the same cluster, so what about K-medians? What is its advantage over of K-means? Please use a synthetic (toy) experiment to illustrate your conclusion.

Sum of absolute residuals is more resilient to outliers in a data set than sum of squares. In the sum of squares, the distances from the mean are squared, so large deviations are weighted more

heavily, and thus outliers can heavily influence it. In the sum of absolutes, the deviations of a small number of outliers are irrelevant. It is also more useful in data set where there isn't a clear mean or variance.

Similar to the same problem with mean for number sets, K-means can be prone to falling for outliers. The K-means algorithm shift the centroid to the mean vector of the cluster, whilst K-medians shifts to the median vector of the vector. If there are several outliers present in the dataset, this could vastly effect K-means in where it shifts it centroids.

Figure 1: Centroid position of data with and without outliers.



Some of the centroid points are not visible because they are in the same location as other centroid positions.

Problem 3

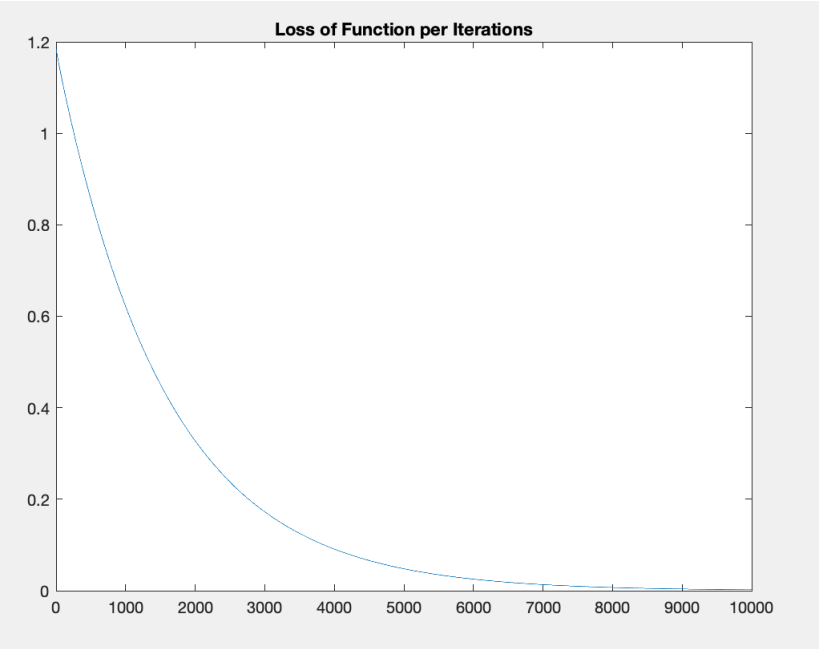
Let's revisit Least Squares Problem: $\underset{\boldsymbol{\beta}}{\text{minimize}}\ \frac{1}{2}\|\mathbf{y}-\mathbf{A}\boldsymbol{\beta}\|_2^2$, where $\mathbf{A}\in\mathbb{R}^{n\times p}$.

1. Please show that if $p > n$, then vanilla solution $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$ is not applicable any more. Least squares is applicable when a matrix has a set of linear equations. (more equations than unknowns). Since y is a linear combination of A, we assume that the columns of A are linearly independent, however, this is not the case when $p > n$ therefore the vanilla solution does not apply.

2. Let's assume $\mathbf{A} = [1, 2, 4; 1, 3, 5; 1, 7, 7; 1, 8, 9], \mathbf{y} = [1; 2; 3; 4]$. Please show via experiment re-

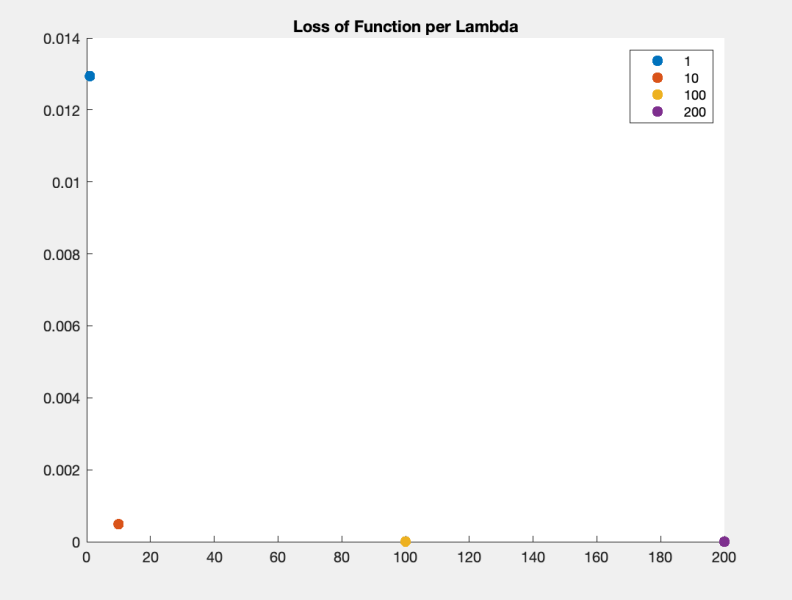
sults that Gradient Descent method will obtain the optimal solution with Linear Convergence rate if the learning rate is fixed to be $\frac{1}{\sigma_{max}(\mathbf{A}^T\mathbf{A})}$, and $\boldsymbol{\beta}_0 = [0; 0; 0]$.

Figure 2: The Loss will eventually converge as we increase # of iterations



3. Now let's consider ridge regression: $\underset{\boldsymbol{\beta}}{\text{minimize}}\ \frac{1}{2}\|\mathbf{y} - \mathbf{A}\boldsymbol{\beta}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{\beta}\|_2^2$, where $\mathbf{A}, \mathbf{y}, \boldsymbol{\beta}_0$ remains the same as above while learning rate is fixed to be $\frac{1}{\lambda+\sigma_{max}(\mathbf{A}^T\mathbf{A})}$ where λ varies from $0.1, 1, 10, 100, 200$, please show that Gradient Descent method with larger λ converges faster.

Figure 3: Loss of function per lambda



Problem 4

Please download the image from `https://en.wikipedia.org/wiki/Lenna#/media/File:Lenna_(test_image).png` with dimension $512 \times 512 \times 3$. Assume for each RGB channel data X, we have $[U, \Sigma, V] = svd(X)$. Please show each compression ratio and reconstruction image if we choose first $2, 5, 20, 50, 80, 100$ components respectively. Also please determine the best component number to obtain a good trade-off between data compression ratio and reconstruction image quality. (Open question, that is your solution will be accepted as long as it's reasonable.)

I personally believe that the 80 columns is a good tradeoff between compression ratio and image quality. The data showed is roughly 16% of the original image however when viewed at a reasonable distance it still looks very comparable to the original image.

Figure 1: Compression ratio $= \frac{2}{512} = 0.004$

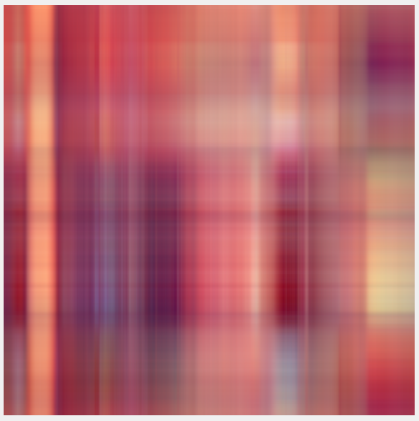


Figure 2: Compression ratio $= \frac{5}{512} = 0.01$

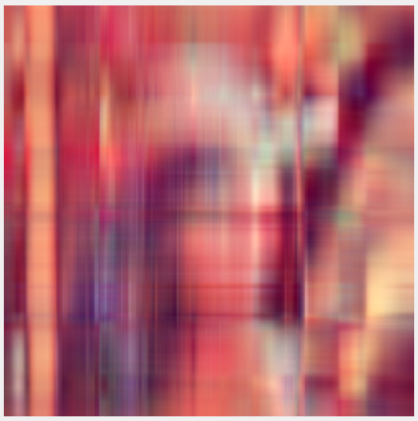


Figure 3: Compression ratio $= \frac{20}{512} = 0.04$

Figure 4: Compression ratio $= \frac{50}{512} = 0.097$

Figure 5: Compression ratio $= \frac{80}{512} = 0.156$

Figure 6: Compression ratio = $\frac{100}{512} = 0.195$

Code for homework

Kmeans vs Kmedians toy experiment

```
clc;
rng default;
points_amount = 100;
x = reshape(linspace(0, 5, 26), 26, 1);
y = reshape(linspace(20, 25, 26), 26, 1);
X_1 = [x x;y y];
outliers = [100 100];
X_2 = cat(1, X_1, outliers);

[idx1, C1] = kmeans(X_1, 2);
[idx2, C2] = kmeans(X_2, 2);
[idx3, C3] = kmedoids(X_1, 2);
[idx4, C4] = kmedoids(X_2, 2);

h = figure;
plot(C1(:,1), C1(:,2), '.', 'MarkerSize',15);
hold on
plot(C2(:,1), C2(:,2), '.', 'MarkerSize',15);
plot(C3(:,1), C3(:,2), '.', 'MarkerSize',15);
plot(C4(:,1), C4(:,2), '.', 'MarkerSize',15);
l = legend('K-means no outliers', 'K-means outliers', 'K-medoids no
    outliers', 'K-medoids outliers');
l.Location = 'southeast';
xlim([0, 120])
ylim([0, 120])
title('K-Means vs K-Median Centroid Location')
hold off
```

```
waitfor(h)
```

Gradient Descent using Vanilla Least Squares

```
clc;
A = [1, 2, 4;1, 3, 5; 1, 7, 7; 1, 8, 9];
y = [1;2;3;4];
b_0 = [0;0;0];
y_hat = A * b_0;
[U, S, V] = svd(A'*A);
sigma_max = max(max(S));
learning_rate = 1 /(sigma_max);
num_iterations = 10000;
%compute least squares
beta_truth = (A'*A)\(A'*y);

[beta, costhistory] = gradient_least(A, y, b_0, learning_rate,
    num_iterations, beta_truth);

h = figure;
plot(costhistory);
title("Loss of Function per Iterations")
waitfor(h);
```

Gradient Calculation for Least Squares

```
function [ beta, costHistory ] = gradient_least( A, y, beta,
    learningRate, repetition, beta_truth)

    costHistory = zeros(repetition, 1);
    for i = 1:repetition

        derivative = A' * (A * beta - y);

        beta = beta - learningRate * derivative;

        %MSE loss
        costHistory(i) = sum((beta - beta_truth).^2);

    end
end
```

Gradient Descent using Ridge Regression

```
clc;
A = [1, 2, 4;1, 3, 5; 1, 7, 7; 1, 8, 9];
y = [1;2;3;4];
b_0 = [0;0;0];
y_hat = A * b_0;
```

```
[U, S, V] = svd(A'*A);
sigma_max = max(max(S));
num_iterations = 1;
lambdas = [0.1, 1, 10, 100, 200];
costhistory_array = zeros(num_iterations, length(lambdas));

for i=1:length(lambdas)
    lambda = lambdas(i);
    %compute ground truth
    beta_truth = (A'*A + lambda * eye(3))\ A'*y;

    learning_rate = 1 /(lambda + sigma_max);

    [beta, costhistory] = gradient_ridge(A, y, b_0, lambda,
       learning_rate, num_iterations, beta_truth);
    costhistory_array(:, i) = costhistory;
end
h = figure;
%plot(costhistory_array(:, 1));
hold on
plot(1, costhistory_array(:, 2), '.', 'MarkerSize', 20);
plot(10, costhistory_array(:, 3), '.', 'MarkerSize', 20);
plot(100, costhistory_array(:, 4), '.', 'MarkerSize', 20);
plot(200, costhistory_array(:, 5), '.', 'MarkerSize', 20);
hold off
legend(["1", "10", "100", "200"])
title("Loss of Function per Lambda")
waitfor(h);
```

Gradient Calculation for Ridge Regression

```
function [ beta, costHistory ] = gradient_ridge( A, y, beta, lambda,
   learningRate, repetition, beta_truth)

    costHistory = zeros(repetition, 1);
    for i = 1:repetition

        derivative = (A' * (A * beta - y)) + (2 * lambda * beta);

        beta = beta - learningRate * derivative;

        %MSE loss
        costHistory(i) = sum((beta - beta_truth).^2);

    end
end
```