Homework Set 2, CPSC 8420, Spring 2022

Huang, Gangtong

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 $\|\hat{\boldsymbol{\phi}}\|_2 = 1$.)

Proof

Given $[\mathbf{U}, \mathbf{S}] = svd(\mathbf{X}^T\mathbf{X})$, we have $\mathbf{X}^T\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{U}^T$, where $\mathbf{S} = diag[s_1, s_2, ..., s_n]$ is a diagonal matrix, with $s_1 > s_2 >, ..., > s_n$ in descending order.
From the property of singular value decomposition, since in this case $\mathbf{U} = \mathbf{V}$, we know that the i-th column of $\mathbf{U}$, $\mathbf{u}_i$, is an eigenvector of $\mathbf{X}^T\mathbf{X}$ with eigenvalue s_i, hence we have:

$$\mathbf{X}^T\mathbf{X}\mathbf{u}_i = s_i\mathbf{u}_i \tag{1}$$

and:

$$\begin{aligned} \mathbf{u}_j^T\mathbf{X}^T\mathbf{X}\mathbf{u}_i &= \mathbf{u}_j^T s_i\mathbf{u}_i \\ &= s_i\mathbf{u}_j^T\mathbf{u}_i \end{aligned} \tag{2}$$

Since each eigenvectors $\mathbf{u}_i$ and $\mathbf{u}_j$ are orthogonal, we have $\mathbf{u}_j^T\mathbf{u}_i = \sigma_{ij}$, where

$$\sigma_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & i \neq j \end{cases}$$

Assume $\phi = \sum_{i=1}^n k_i\mathbf{u}_i$ is an arbitrary linear combination of u_1, u_2, ..., u_n, with $\|\phi\|_2 = 1$ or $\sum_{i=1}^n k_i^2 = 1$. We know that maximize $\|\mathbf{X}\boldsymbol{\phi}\|_2^2$ is equivalent to maximize$[tr(\phi^T\mathbf{X}^T\mathbf{X}\phi)]$. Then,

using Eqn. (2):

$$\begin{aligned}\phi^T\mathbf{X}^T\mathbf{X}\phi &= (\sum_{i=1}^{n} k_i\mathbf{u}_i^T)\mathbf{X}^T\mathbf{X}(\sum_{i=j}^{n} k_j\mathbf{u}_j) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}[k_i(\mathbf{u}_i^T\mathbf{X}^T\mathbf{X}\mathbf{u}_j)k_j] \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}[k_ik_j(\sigma_{ij})s_js_i] \\ &= \sum_{i=1}^{n} k_i^2s_i^2\end{aligned} \tag{3}$$

In this 1-dimensional case, $tr(\phi^T\mathbf{X}^T\mathbf{X}\phi) = \phi^T\mathbf{X}^T\mathbf{X}\phi = \sum_{i=1}^{n} k_i^2s_i^2$. Since $s_1 > s_2 >, ..., > s_n \geq 0$, we have: $tr(\phi^T\mathbf{X}^T\mathbf{X}\phi) = \sum_{i=1}^{n} k_i^2s_i^2 \leq k_1^2s_1^2$. The case $tr(\phi^T\mathbf{X}^T\mathbf{X}\phi) = k_1^2s_1^2$ with $k_1 = 1$ and $k_{i>1} = 0$, or the maximum $\|\mathbf{X}\boldsymbol{\phi}\|_2^2$ is only satisfied when $\phi = \mathbf{u}_1$, which is the first column of $\mathbf{U}$.

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 vs. squared residuals
The sum of residual squares can amplify the influence of data points with large distance from the centroid in the optimization objective. This can be problematic because the few outlier will produce larger penalty on minimization by squaring, and it makes the position of mean value shift towards the outlier, skewing the estimation.

K-medians vs. K-means
In the K-medians method, the medoid of a cluster must be selected among the data points, while in K-means method the centroid is the mean of a cluster. The advantage of K-median is that the restriction on the position of the medoid can prevent it from being shifted towards the outliers.

Synthetic experiment
In this 2D model, 3 clusters were generated by random sampling of normal distributions centering

at 0, 5, 10, respectively, while the standard deviation is 5 for all 3 clusters. An outlier is located in $x, y = 90 \sim 100$ region. Implemented with Mathematica.

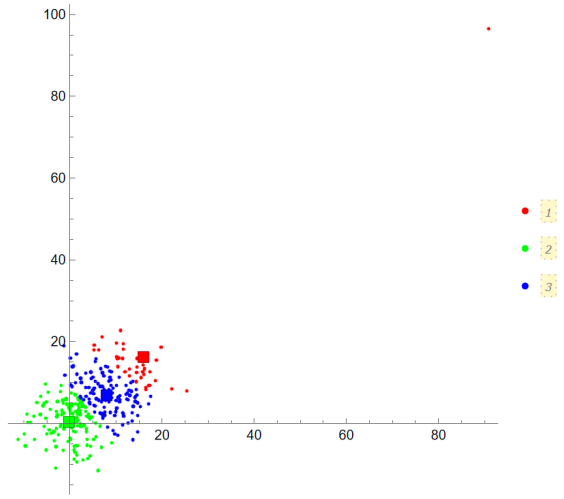

Figure 1: Clustering using K-means. Clusters differentiated by color. Data points are shown with dots and cluster means with squares.

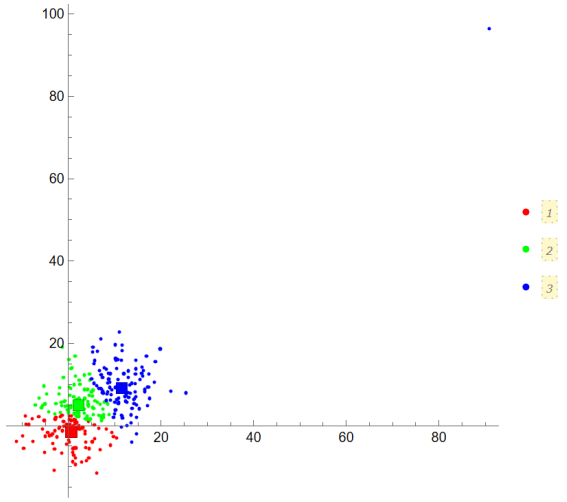

Figure 2: Clustering using K-medians. Clusters differentiated by color. Data points are shown with dots and cluster means with squares.

As shown in Fig. 1 and 2, K-means and K-medians got similar clustering results. The main difference is the centroid C of the upper-right cluster (red in Fig. 1, blue in Fig. 2): instead of being at the center of the cluster, C in Fig. 1 is shifted towards the outlier, while C in Fig. 2 is less affected by the outlier and still lies near the center of cluster by visual inspection.

Codes (Mathematica)

```
(*Dataset Preparation*)
(*Define number of dat points and clusters*)
nNormal = 100;
nOut = 1
nClust = 3;
nX = nNormal*nClust + nOut;
(*Initialize data points and random initial means*)
randomClust[a_, b_, n_] :=
RandomVariate[NormalDistribution[a, b], {n, 2}]
matX = Join[randomClust[0, 5, nNormal], randomClust[5, 5, nNormal],
randomClust[10, 5, nNormal], RandomReal[{90, 100}, {nOut, 2}]];

(*K-means*)
(*Random initial means*)
mean = Sort[RandomReal[{0, 10}, {nClust, 2}]];
(*L-2 Norm*)
dist[a_, b_] := Norm[a - b, 2]
(*Position of minimum in l_List*)
belong2[l_List] := Position[l, Min[l]]
(*Select all points in a cluster*)
sele[iCluster_] := Select[matX, clusters[#] == iCluster &]
meanTemp = {};
(*Number of iterations*)
iter = 0;
While[meanTemp !=
mean(*Stop when positions  of cluster means do not change*),
(*Distance b/w a data points and means*)
distances =
Table[dist[matX[[iX]], mean[[iM]]], {iX, 1, nX}, {iM, 1, nClust}];
(*Which cluster does each point belong to*)
clustering = Flatten[belong2 /@ distances];
clusters = AssociationThread[matX, clustering];
meanTemp = mean;
(*Update means with new cluster means*)
mean = Mean /@ sele /@ Range[1, nClust];
iter = iter + 1;]
iter
Show[ListPlot[sele /@ Range[1, nClust], PlotLegends -> Automatic,
PlotStyle -> {Red, Green, Blue}],
ListPlot[{#} & /@ mean, PlotMarkers -> {Red, Green, Blue}],
PlotRange -> All, AspectRatio -> 1]

(*K-medians*)
(*Random initial medians*)
median = Sort[RandomSample[matX, 3]];
```

```
(*L-1 Norm*)
dist[a_, b_] := Norm[a - b, 1]
(*Position of minimum in l_List*)
belong2[l_List] := Position[l, Min[l]]
(*Select all points in a cluster*)
sele[iCluster_] := Select[matX, clusters[#] == iCluster &]
medianTemp = {};
(*Number of iterations*)
iter = 0;
While[medianTemp !=
median(*Stop when positions  of cluster medians do not change*),
(*Distance b/w a data points and medians*)
distances =
Table[dist[matX[[iX]], median[[iM]]], {iX, 1, nX}, {iM, 1,
nClust}];
(*Which cluster does each point belong to*)

clustering = Flatten[belong2 /@ distances];
clusters = AssociationThread[matX, clustering];
medianTemp = median;
(*Update means with new cluster medians*)

median = Median /@ sele /@ Range[1, nClust];
iter = iter + 1;]
iter
Show[ListPlot[sele /@ Range[1, nClust], PlotLegends -> Automatic,
PlotStyle -> {Red, Green, Blue}],
ListPlot[{#} & /@ median, PlotMarkers -> {Red, Green, Blue}],
PlotRange -> All, AspectRatio -> 1]
```

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.

Perform SVD on A:

$$SVD(A) = [U, S, V] \tag{4}$$

$$\mathbf{A} = \mathbf{USV}^T \tag{5}$$

where $\mathbf{S} \in \mathbb{R}^{n\times n}, \mathbf{S} = diag[s_1, s_2, ..., s_n]$ is diagonal and $\mathbf{U} \in \mathbb{R}^{n\times n}$ is unitary.

$$\begin{aligned} \mathbf{A}^T &= (\mathbf{USV}^T)^T \\ &= \mathbf{VS}^T\mathbf{U}^T \\ &= \mathbf{VSU}^{-1} \end{aligned} \tag{6}$$

Then we have:

$$\begin{aligned} \mathbf{A}^T\mathbf{A} &= \mathbf{VSU}^{-1}\mathbf{USV}^T \\ &= \mathbf{VS}^2\mathbf{V}^T \end{aligned} \tag{7}$$

Given $\mathbf{V} \in \mathbb{R}^{p\times n}$, $\mathbf{S}^2 \in \mathbb{R}^{n\times n}$, $\mathbf{V}^T \in \mathbb{R}^{n\times p}$, we have: $(\mathbf{A}^T\mathbf{A})^{-1} \in \mathbb{R}^{p\times p}$. However, $\mathbf{A}^T \in \mathbb{R}^{p\times n}$, so $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$ does not exist, so the vanilla solution $(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$ is not applicable.

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 results 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 $\boldsymbol{\beta}_0 = [0; 0; 0]$.

   Implemented with MatLab. According to definition of linear convergence rate of vanilla least squares: $\frac{f(x^+)-f(x^*)}{f(x)-f(x^*)} = 1 - \frac{\alpha}{\beta} = 1 - \frac{\sigma_{min}(\mathbf{A}^T\mathbf{A})}{\sigma_{max}(\mathbf{A}^T\mathbf{A})}$, we may calculate the linear convergence rate in two ways, i.e. $\frac{f(x^+)-f(x^*)}{f(x)-f(x^*)}$ and $1 - \frac{\sigma_{min}(\mathbf{A}^T\mathbf{A})}{\sigma_{max}(\mathbf{A}^T\mathbf{A})}$. With condition given and a tolerance $tol = 10^{-5}$ on $\|\|\beta^+ - \beta\|\|_2^2$, we can observe agreement between both methods: $\frac{f(x^+)-f(x^*)}{f(x)-f(x^*)} = 1 - \frac{\sigma_{min}(\mathbf{A}^T\mathbf{A})}{\sigma_{max}(\mathbf{A}^T\mathbf{A})} = 0.99968$.

   Codes (Matlab):

```
clear all;

A = [1, 2, 4; 1, 3, 5; 1, 7, 7; 1, 8, 9];
y = [1; 2; 3; 4];

S = svd(A'*A);

beta = [0;0;0];

s_min = min(S); s_max = max(S); t = 1/s_max;

beta_tmp = 1;

beta_optimal = pinv(A'*A)*A'*y;
```

```
iter = 0;
tol = 10^-5;
while (beta_tmp ~= beta)
grad_f = A'*(A*beta - y);
beta_1 = beta - t*grad_f;
beta_tmp = beta;
conv_rate = norm(beta_1 - beta_optimal,1)/norm(beta - beta_optimal,1);
beta = beta_1;
iter = iter + 1;
if norm(beta - beta_tmp,2) <= tol; break; end
end

fprintf("n_iter = %d\n",iter)
fprintf("Linear convergence rate by original definition: %f\n", ...
conv_rate)

fprintf("Linear convergence by 1 - sigma_min(A'A)/sigma_max(A'A): %f\n", ...
1 - s_min/s_max)
```

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.

   Linear convergence rate for ridge regression: $\frac{f(x^+) - f(x^*)}{f(x) - f(x^*)} = 1 - \frac{\sigma_{min}(\mathbf{A}^T\mathbf{A}) + \lambda}{\sigma_{max}(\mathbf{A}^T\mathbf{A}) + \lambda}$. With varied λ, the number of iterations needed for $\|\|\beta^+ - \beta\|\|_2^2$ to drop below the tolerance $tol = 10^{-6}$ is shown in Fig. 3.

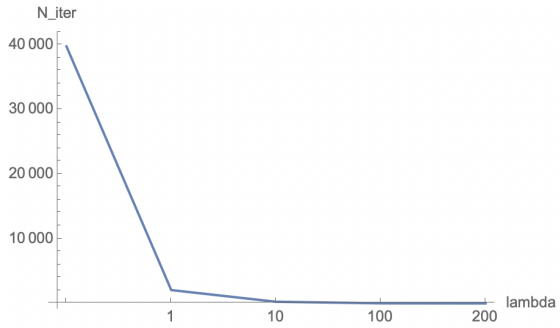

Figure 3: Number of iterations with varied λ.

We can see that as λ increases, the number of iterations dreases, i.e. the convergence becomes faster.

Codes (MatLab):

```
clear all;
```

```
A = [1, 2, 4; 1, 3, 5; 1, 7, 7; 1, 8, 9];
y = [1; 2; 3; 4];

S = svd(A'*A);

beta = [0;0;0];

s_min = min(S); s_max = max(S);

beta_tmp = 1;

% Linear convergence rate by original definition

tol = 10^-6;
for lambda = [ 0.1, 1, 10, 100, 200 ]
%step size
t = 1/(s_max+lambda);

beta_optimal = pinv(A'*A+lambda)*A'*y;

iter = 0;
while (beta_tmp ~= beta) & (iter <= 100000)
grad_f = 2*A'*(A*beta - y) + lambda*beta;
beta_1 = beta - t*grad_f;
beta_tmp = beta;
%         conv_rate = norm(beta_1 - beta_optimal,1)/norm(beta - beta_optimal,1);
beta = beta_1;
iter = iter + 1;
if norm(beta - beta_tmp,2) <= tol; break; end
end
fprintf("iter = %d\n", iter)
fprintf("lambda = %d\n", lambda)
%     fprintf("iter = %d\n", iter)
%     fprintf("Linear convergence rate by original definition: %d\n", ...
%         conv_rate)

fprintf("Linear convergence by 1 - (lambda+sigma_min(A'A))/(lambda+sigma_max(A'A)): %f\n",
1 - (s_min+lambda)/(s_max+lambda))

end
```

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.)

PCA implemented with Mathematica. The reconstructed images using the first $2, 5, 20, 50, 80, 100$ columns of $\mathbf{U}$:

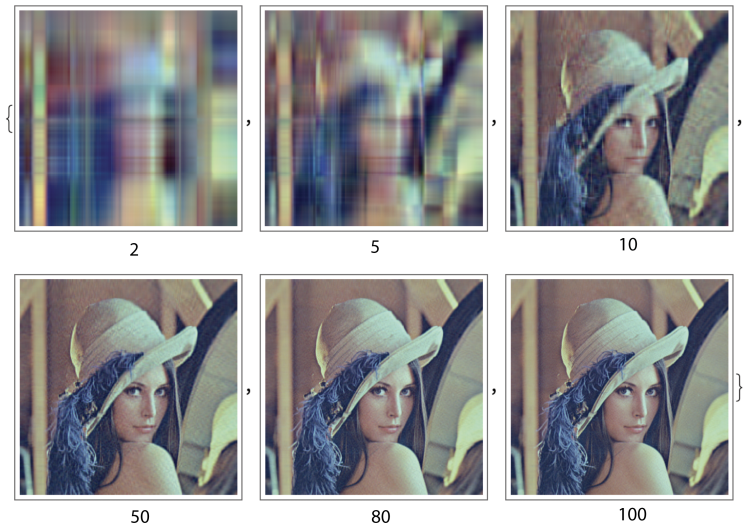

Figure 4: Reconstructed images using the first $2, 5, 20, 50, 80, 100$ columns of $\mathbf{U}$.

Since PCA uses the principal componants with the largest variances, we can use the variances of each channel in the reconstructed image relative to the original as a measurement of the image quality, i.e. how much of the information (variance) of the original data is retained.

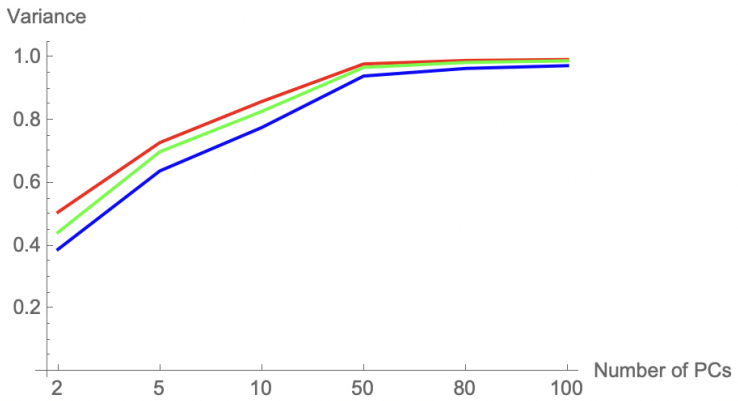

Figure 5: Relative variance of the R, G, B channels of reconstructed images with different number of PCs. Each channel is colored by its corresponding color.

We can see that with 50 PCs, all the 3 channels effectively reach over 90% of the variance in the original image, and increasing the number of PCs thereafter does not significantly improve image quality. Therefore we can conclude that 50 is the best component number for image reconstruction.

Codes (Mathematica)

```
(*Import picture*)

cwd = NotebookDirectory[];
original = Import[cwd <> "Lenna_(test_image).png", "Data"];
pxl = N[Flatten[original, 1]];
mean = Mean[pxl];
pxlCtr = # - mean & /@ pxl;
(*RGB channels*)
R = ArrayReshape[pxlCtr[[All, 1]], {512, 512}]; G =
ArrayReshape[pxlCtr[[All, 2]], {512, 512}]; B =
ArrayReshape[pxlCtr[[All, 3]], {512, 512}];

(*PCA and reconstruction*)
Clear[recon];
(*Reconstruction single channel*)

recon[channel_, nPC_] := Module[{u, s, v, picrecon},
(*SVD*){u, s, v} = SingularValueDecomposition[channel];
(*Reconstruction using nPC columns*)
picrecon =
u[[All, ;; nPC]].s[[;; nPC, ;; nPC]].Transpose[v[[All, ;; nPC]]];
picrecon]
(*Reconstruct 3 channels*)

reconRGB[nPC_] :=
```

```
ArrayReshape[
Transpose[
Flatten /@ {recon[R, nPC], recon[G, nPC], recon[B, nPC]}], {512,
512, 3}]
rgbPlot[plot_] :=
ArrayPlot[ArrayReshape[plot, {512, 512, 3}],
ColorFunction -> Function[p, RGBColor[p[[1]], p[[2]], p[[3]]]],
ColorFunctionScaling -> True]

(*Plotting*)
rgbPlot[reconRGB[#]] & /@ {2, 5, 20, 50, 80, 100}

(*Evaluation*)
varPic[picArr_] := N[Variance[ArrayReshape[picArr, {512*512, 3}]]]
VarOrig = varPic[original];
percentVar =
varPic[reconRGB[#]]/VarOrig & /@ {2, 5, 10, 50, 80, 100};
ListLinePlot[Transpose[percentVar],
Ticks -> {Transpose[{Range[1, 6], {2, 5, 10, 50, 80, 100}}],
Automatic}, AxesLabel -> {"Number of PCs", "Variance"},
PlotStyle -> {Red, Green, Blue}, PlotRange -> Full]
```