HOMEWORK 5

Venkata Saikiranpatnaik Balivada
Net ID: vbalivada
Wisc ID: 9084550244

Instructions: Use this latex file as a template to develop your homework. Submit your homework on time as a single pdf file. Please wrap your code and upload to a public GitHub repo, then attach the link below the instructions so that we can access it. Answers to the questions that are not within the pdf are not accepted. This includes external links or answers attached to the code implementation. Late submissions may not be accepted. You can choose any programming language (i.e. python, R, or MATLAB). Please check Piazza for updates about the homework. It is ok to share the experiments results and compare them with each other.

Please find the code and images for the homework inside the repo https://github.com/pao214/cs760_hw5.

1 Clustering

1.1 K-means Clustering (14 points)

1. **(6 Points)** Given n observations $X_1^n = \{X_1, \ldots, X_n\}$, $X_i \in \mathcal{X}$, the K-means objective is to find k ($< n$) centres $\mu_1^k = \{\mu_1, \ldots, \mu_k\}$, and a rule $f : \mathcal{X} \to \{1, \ldots, K\}$ so as to minimize the objective

$$J(\mu_1^K, f; X_1^n) = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}(f(X_i) = k) \|X_i - \mu_k\|^2 \tag{1}$$

Let $\mathcal{J}_K(X_1^n) = \min_{\mu_1^K, f} J(\mu_1^K, f; X_1^n)$. Prove that $\mathcal{J}_K(X_1^n)$ is a non-increasing function of K.

We prove that $\mathcal{J}_K(X^n) \geq \mathcal{J}_{K+1}(X^n)$ for any K.
First, we write down the equation for $\mathcal{J}$,

$$\mathcal{J}_K(X^n) = min_{\mu^K, f} \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}(f(X_i) = k) \|X_i - \mu_k\|^2$$

by substitution.
Let the optimal values be $\hat{\mu}^K, \hat{f}$.
Now, we define $\mu^{K+1} = \{..., \hat{\mu}_i^K, ..., 0\}$ and $f = \hat{f}$ for the new objective with K+1 centroids.

$$\begin{aligned}
J(\mu^{K+1}, f; X^n) &= \sum_{i=1}^{n} \sum_{k=1}^{K+1} \mathbb{1}(f(X_i) = k) \|X_i - \mu_k\|^2 \\
&= \sum_{i=1}^{n} \sum_{k=1}^{K+1} \mathbb{1}(\hat{f}(X_i) = k) \|X_i - \hat{\mu}_k\|^2 \\
&= \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}(\hat{f}(X_i) = k) \|X_i - \hat{\mu}_k\|^2 \\
&= \mathcal{J}_K(X^n)
\end{aligned}$$

The second line follows from substitution, the third line follows from the fact that $\hat{f}$ never maps to $K+1$, and the fourth line is a consequence of $\hat{\mu}, \hat{f}$ being the optimal arguments.

Finally, we observe that

$$\begin{aligned}\mathcal{J}_{K+1}(X^n) &= min_{\mu^{K+1},f} J(\mu^{K+1}, f; X^n) \\ &\leq J(\hat{\mu}^K, \hat{f}; X^n) \\ &= \mathcal{J}_K(X^n)\end{aligned}$$

Hence, $\mathcal{J}$ is a non-increasing function in K. ■

2. **(8 Points)** Consider the K-means (Lloyd's) clustering algorithm we studied in class. We terminate the algorithm when there are no changes to the objective. Show that the algorithm terminates in a finite number of steps.

Here's an outline of how we prove that the algorithm terminates.

(a) First, we show that the loss never increases, both E and M steps of the K-means algorithm.

(b) Next, we argue that there is a finite number of cluster assignments.

(c) Using the above fact, we deduce that some cluster assignment must be repeated during the algorithm.

(d) By using contradiction, we can prove that the algorithm must terminate unless there is a cycle.

(e) Finally, we prove that there are no such cycles.

Loss objective during the E and the M steps

First, let's look at the training objective

$$J(\mu^K, f; X^n) = \sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{1}(f(X_i) = k)\|X_i - \mu_k\|^2$$

We prove that this cannot increase in either the E-step or the M-step.

First, let's look at the E-step. This changes the cluster assignment function f to assign each point to the nearest centroid. The centroids themselves do not change in this step.

Say, the assignment for x_i changes from k to $\hat{k}$, then we have $\|x_i - \mu_{\hat{k}}\|^2 \leq \|x_i - \mu_k\|^2$ because we are choosing the closest centroid.

This implies that the new objective is

$$\begin{aligned}J(\mu^K, \hat{f}; X^n) &= \sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{1}(\hat{f}(X_i) = k)\|X_i - \mu_k\|^2 \\ &\leq \sum_{i=1}^{n}\sum_{k=1}^{K} \mathbb{1}(f(X_i) = k)\|X_i - \mu_k\|^2 \\ &= J(\mu^K, f; X^n)\end{aligned}$$

Here, we exploit the fact that the distance in 2-norm does not increase. Hence, the training objective/loss is non-increasing in the E-step.

Now, we focus our attention on the M-step. We prove that the means of the current clusters form the optimal centroids. Let's take a look at the formula for the objective once again

$$J(\mu^K, \hat{f}; X^n) = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}(\hat{f}(X_i) = k) \|X_i - \mu_k\|^2$$

To get the optimal centroid, we differentiate w.r.t. μ and set it to zero.

$$\begin{aligned}
\nabla_{\mu_j} J(\mu^K, \hat{f}; X^n) &= \sum_{k=1}^{K} \sum_{i=1}^{n} \nabla \mathbb{1}(f(X_i) = k) \|X_i - \mu_k\|^2 \\
&= \sum_{k=1}^{K} \sum_{i=1}^{n} \nabla \mathbb{1}(f(X_i) = k)(X_i - \mu_k)^T (X_i - \mu_k) \\
&= \sum_{k=1}^{K} \sum_{i=1}^{n} \nabla \mathbb{1}(f(X_i) = k)(X_i^T - 2X_i^T \mu_k + \mu_k^T \mu_k) \\
&= \sum_{i=1}^{n} \mathbb{1}(f(X_i) = j)(2\mu_j - 2X_i) \\
&= 0
\end{aligned}$$

Solving for μ_j we get,

$$\mu_j = \frac{\sum_{i=1}^{n} \mathbb{1}(f(X_i) = j) X_i}{\sum_{i=1}^{n} \mathbb{1}(f(X_i)}$$

This is precisely the M-step of the K-means algorithm where we take the mean of the cluster to update the centroid. Hence, each step of the algorithm either decreases the loss objective or keeps it the same.

Importantly, we also observe that the Mean-squared error function is strictly convex because the double derivative is a positive constant. This in turn implies that the minimum is achieved only at the mean and the loss is strictly greater elsewhere. As a consequence, the loss is strictly decreasing unless the centers themselves do not change. Notice that this is precisely the criterion for algorithm termination.

Number of cluster assignments

Each point may be assigned to one of the k clusters. For n points, we may not have more than n^k distinct cluster assignments. Since the mean of cluster points is a deterministic function of the cluster points in each cluster, we only have a finite number of loss objectives during the run of the algorithm.

Termination

The loss is strictly decreasing as argued above. We also argued that the loss objective can only take a finite number of values. We prove that the algorithm terminates by contradiction. If the algorithm were to run infinitely and the losses are strictly decreasing. There are an infinite number of loss objectives that the algorithm must compute. This is in direct contradiction with the finite nature of cluster assignments. ■

1.2 Experiment (20 Points)

In this question, we will evaluate K-means clustering and GMM on a simple 2 dimensional problem. First, create a two-dimensional synthetic dataset of 300 points by sampling 100 points each from the three Gaussian distributions shown below:

$$P_a = \mathcal{N}\left(\begin{bmatrix}-1\\-1\end{bmatrix}, \sigma \begin{bmatrix}2, & 0.5\\0.5, & 1\end{bmatrix}\right), \quad P_b = \mathcal{N}\left(\begin{bmatrix}1\\-1\end{bmatrix}, \sigma \begin{bmatrix}1, & -0.5\\-0.5, & 2\end{bmatrix}\right), \quad P_c = \mathcal{N}\left(\begin{bmatrix}0\\1\end{bmatrix}, \sigma \begin{bmatrix}1 & 0\\0, & 2\end{bmatrix}\right)$$

Here, σ is a parameter we will change to produce different datasets.

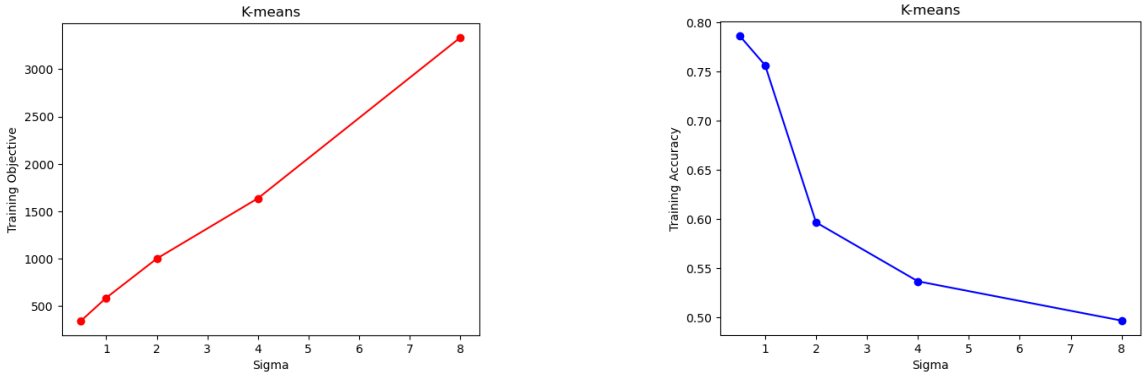
First implement K-means clustering and the expectation maximization algorithm for GMMs. Execute both methods on five synthetic datasets, generated as shown above with $\sigma \in \{0.5, 1, 2, 4, 8\}$. Finally, evaluate both methods on *(i)* the clustering objective (1) and *(ii)* the clustering accuracy. For each of the two criteria, plot the value achieved by each method against σ.

Guidelines:

- Both algorithms are only guaranteed to find only a local optimum so we recommend trying multiple restarts and picking the one with the lowest objective value (This is (1) for K-means and the negative log likelihood for GMMs). You may also experiment with a smart initialization strategy (such as kmeans++).
- To plot the clustering accuracy, you may treat the 'label' of points generated from distribution P_u as u, where $u \in \{a, b, c\}$. Assume that the cluster id i returned by a method is $i \in \{1, 2, 3\}$. Since clustering is an unsupervised learning problem, you should obtain the best possible mapping from $\{1, 2, 3\}$ to $\{a, b, c\}$ to compute the clustering objective. One way to do this is to compare the clustering centers returned by the method (centroids for K-means, means for GMMs) and map them to the distribution with the closest mean.

Points break down: 7 points each for implementation of each method, 6 points for reporting of evaluation metrics.

The implementations are available in the GitHub link provided. See figure 1 for KMeans metrics and figure 2. We use the kmeans++ initialization method for both k-means and the GMM algorithm. The algorithm restarts a thousand times and retains the best-performing parameters in terms of the objective function. We terminate when the objective remains the same across iterations. The objective function for K-Means is given by $\sum_{i=1}^{n}\sum_{k=1}^{K}\mathbb{1}(f(X_i)=k)\|X_i-\mu_k\|^2$ and the objective function for GMM is given by $\sum_{i=1}^{n}\log(\sum_{j=1}^{k}p_\theta(X^{(i)}, z^{(i)}=j))$. The accuracy is computed as usual after finding the closest true center for each predicted centroid.



(a) KMeans: Mean Squared Distance Objective Function

(b) KMeans: Accuracy

Figure 1: KMeans

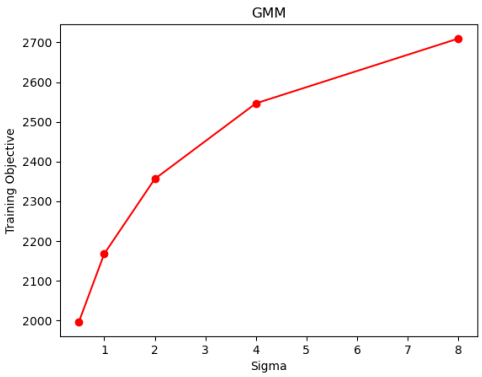
2 Linear Dimensionality Reduction

2.1 Principal Components Analysis (10 points)

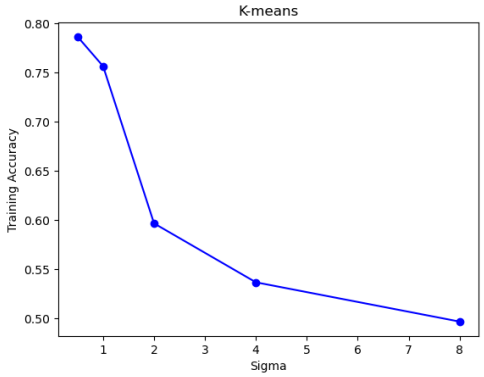
Principal Components Analysis (PCA) is a popular method for linear dimensionality reduction. PCA attempts to find a lower dimensional subspace such that when you project the data onto the subspace as much of the information is preserved. Say we have data $X = [x_1^\top; \ldots; x_n^\top] \in \mathbb{R}^{n\times D}$ where $x_i \in \mathbb{R}^D$. We wish to find a d ($< D$) dimensional subspace $A = [a_1, \ldots, a_d] \in \mathbb{R}^{D\times d}$, such that $a_i \in \mathbb{R}^D$ and $A^\top A = I_d$, so as to maximize $\frac{1}{n}\sum_{i=1}^{n}\|A^\top x_i\|^2$.

1. **(4 Points)** Suppose we wish to find the first direction a_1 (such that $a_1^\top a_1 = 1$) to maximize $\frac{1}{n}\sum_i (a_1^\top x_i)^2$. Show that a_1 is the first right singular vector of X.

   We maximize $\sum_i (a_1^\top x_i)^2$ since $\frac{1}{n}$ is independent of a_1 anyway. We can rewrite the expression as



(a) GMM: Objective Function



(b) GMM: Accuracy

Figure 2: GMM

$$\begin{aligned}\sum_i (a_1^\top x_i)^2 &= \sum_i \|x_i^T a_1\|^2 \\ &= \sum_i a_1^T x_i^T x_i a_1 \\ &= a_1^T (\sum_i x_i^T x_i) a_1 \\ &= a_1^T X^T X a_1\end{aligned}$$

The first line simply follows from the fact that $a_1^T x_i$ is a scalar and hence symmetric. The second line just expands the norm. We factor a_1 terms in the third line. Finally, we use the fact that $X^T X = \sum_i x_i^T x_i$ in the fourth line.

We now express a_1 in terms of the eigenvectors of $X^T X$. $a_1 = \sum_i c_i \psi_i$ where ψ_i are eigenvectors of $X^T X$ with eigenvalues λ_i. Substituting this, we have,

$$\begin{aligned}X^T X a_1 &= X^T X (\sum_i c_i \psi_i) \\ &= \sum_i c_i X^T X \psi_i \\ &= \sum_i c_i \lambda_i \psi_i\end{aligned}$$

The final line is simply the application of eigenvector scaling the vector, i.e. $X^T X \psi_i = \lambda_i \psi_i$. We now compute the complete expression

$$\begin{aligned}a_1^T X^T X a_1 &= a_1 (X^T X a_1) \\ &= (\sum_i c_i \psi_i)^T (\sum_i c_i \lambda_i \psi_i) \\ &= \sum_i \sum_j c_i c_j \lambda_j \psi_i^T \psi_j \\ &= \sum_i c_i^2 \lambda_i\end{aligned}$$

The second line is a simple substitution, the third line expands the terms, and the final line exploits the fact that the eigenvectors are orthonormal, i.e. $\psi_j^T \psi_i = \delta_{ij}$. Assuming that λ_1 is the largest eigenvalue, we have

$$\begin{aligned}\sum_i c_i^2 \lambda_i &\leq \sum_i c_i^2 \lambda_1 \\ &= \lambda_1 \sum_i c_i^2 \\ &= \lambda_1\end{aligned}$$

The first one follows from the fact that $\lambda_i \leq \lambda_1$. The second line simply takes λ_1 out of the loop. The final line is a consequence of a_1 being a unit vector. $a_1^T a_1 = (\sum_i c_i \psi)^T (\sum_i c_i \psi) = \sum_i c_i^2 = 1$.

So, λ_1 is the maximum value of the expression $\sum_i (a_1^T x_i)^2$ and is achieved when a_1 is the eigenvector of $X^T X$ with the largest eigenvalue. This is also the first right singular vector of X because

$$\begin{aligned}X &= USV^T \\ \implies X^T X &= VS^T U^T USV^T \\ &= VS^T SV^T\end{aligned}$$

Comparing this form with the eigendecomposition of $X^T X$, the eigenvector a_1 corresponds to the first vector in V. ■

2. **(6 Points)** Given $a_1, \ldots, a_k$, let $A_k = [a_1, \ldots, a_k]$ and $\tilde{x}_i = x_i - A_k A_k^\top x_i$. We wish to find a_{k+1}, to maximize $\frac{1}{n} \sum_i (a_{k+1}^\top \tilde{x}_i)^2$. Show that a_{k+1} is the $(k+1)^{th}$ right singular vector of X.

This follows naturally from the proof outlined in the previous solution. Here's a rough outline of the proof.

(a) First, we prove that none of the singular vectors in A_k maximize the term.
(b) Next, we prove that this is true for the complete subspace spanned by the first k singular vectors.
(c) Then, we prove that any of the vectors in the orthogonal subspace have the same effect on $\tilde{x}_i$ as x_i.
(d) Finally, we use the above claims to prove that $k+1$th right singular vector maximizes our objective.

First k right singular vectors

Consider a right singular vector a_i. We compute its dot product with $\tilde{x}_i$

$$\begin{aligned}a_i^T \tilde{x}_i &= a_i^T (x_i - A_k A_k^T x_i) \\ &= a_i^T x_i - a_i^T A_k A_k^T x_i \\ &= a_i^T x_i - [\delta_{ij}] A_k^T x_i \\ &= \mathbb{1}(i > k) a_i^T x_i\end{aligned}$$

The third line follows from the fact that the right singular vectors are orthonormal. The fourth line follows from the fact that the term on the right is zero when i is not one of the first k right singular vectors and is $a_i^T x_i$ otherwise.

An important consequence of this fact is that none of the first k singular vectors maximize the objective since $a_i^T \tilde{x}_i$ is always zero and thus the objective too. The objective is always non-negative since it is sum of squares. So, the first k singular vectors do not necessarily maximize the objective, they minimize it.

Also, any linear combination of the first k singular vectors is orthogonal to $\tilde{x}_i$ too. This is because dot product is a linear operation. This implies that none of the vectors in the subspace spanning the first k singular vectors can maximize the objective. We need to look for the vectors in the orthogonal space.

Optimal Vector

As observed earlier, the dot product with $\tilde{x}_i$ in the orthogonal space spanned by the remaining $D-k$ right singular vectors is the same as the dot product with the original vector x_i. This implies that the first right singular vector in this orthogonal space maximizes the objective (as seen in the proof of the previous question). This is precisely the $k+1$th right singular vector. ■

2.2 Dimensionality reduction via optimization (22 points)

We will now motivate the dimensionality reduction problem from a slightly different perspective. The resulting algorithm has many similarities to PCA. We will refer to method as DRO.
As before, you are given data $\{x_i\}_{i=1}^n$, where $x_i \in \mathbb{R}^D$. Let $X = [x_1^\top; \dots x_n^\top] \in \mathbb{R}^{n\times D}$. We suspect that the data actually lies approximately in a d dimensional affine subspace. Here $d < D$ and $d < n$. Our goal, as in PCA, is to use this dataset to find a d dimensional representation z for each $x \in \mathbb{R}^D$. (We will assume that the span of the data has dimension larger than d, but our method should work whether $n > D$ or $n < D$.)
Let $z_i \in \mathbb{R}^d$ be the lower dimensional representation for x_i and let $Z = [z_1^\top; \dots; z_n^\top] \in \mathbb{R}^{n\times d}$. We wish to find parameters $A \in \mathbb{R}^{D\times d}$, $b \in \mathbb{R}^D$ and the lower dimensional representation $Z \in \mathbb{R}^{n\times d}$ so as to minimize

$$J(A, b, Z) = \frac{1}{n}\sum_{i=1}^{n} \|x_i - Az_i - b\|^2 = \|X - ZA^\top - \mathbf{1}b^\top\|_F^2. \tag{2}$$

Here, $\|A\|_F^2 = \sum_{i,j} A_{ij}^2$ is the Frobenius norm of a matrix.

1. **(3 Points)** Let $M \in \mathbb{R}^{d\times d}$ be an arbitrary invertible matrix and $p \in \mathbb{R}^d$ be an arbitrary vector. Denote, $A_2 = A_1M^{-1}$, $b_2 = b_1 - A_1M^{-1}p$ and $Z_2 = Z_1M^\top + \mathbf{1}p^\top$. Show that both (A_1, b_1, Z_1) and (A_2, b_2, Z_2) achieve the same objective value J (2).

We prove that $X - Z_1A_1^T - \mathbf{1}b_1^T = X - Z_2A_2^T - \mathbf{1}b_2^T$ by proving that $Z_1A_1^T + \mathbf{1}b_1^T = Z_2A_2^T + \mathbf{1}b_2^T$.
Substituting

$$\begin{aligned} Z_2 &= Z_1M^T + \mathbf{1}p^T \\ A_2 &= A_1M^{-1} \\ b_2 &= b_1 - A_1M^{-1}p \end{aligned}$$

in $Z_2A_2^T + \mathbf{1}b_2^T$, we have,

$$\begin{aligned} Z_2A_2^T + \mathbf{1}b_2^T &= (Z_1M^T + \mathbf{1}p^T)(A_1M^{-1})^T + \mathbf{1}(b_1 - A_1M^{-1}p) \\ &= (Z_1M^T + \mathbf{1}p^T)((M^T)^{-1}A_1^T) + \mathbf{1}b_1 - \mathbf{1}A_1M^{-1}p \\ &= Z_1M^T(M^T)^{-1}A_1^T + \mathbf{1}p^T(M^T)^{-1}A_1^T + \mathbf{1}b_1 - \mathbf{1}A_1M^{-1}p \\ &= Z_1M^T(M^T)^{-1}A_1^T + \mathbf{1}b_1 + \mathbf{1}A_1M^{-1}p - \mathbf{1}A_1M^{-1}p \\ &= Z_1M^T(M^T)^{-1}A_1^T + \mathbf{1}b_1 \\ &= Z_1A_1^T + \mathbf{1}b \end{aligned}$$

The first line follows through substitution. We expand the terms in the second and the third lines. In the fourth line, we take the transpose of the scalar since scalars are symmetric and it cancels out with the trailing term in the fifth line. We simplify further by canceling the inverse to obtain the equivalence we desire. Since the matrices are the same, the forbenius norm and consequently the objectives are also equal. ■
Therefore, in order to make the problem determined, we need to impose some constraint on Z. We will assume that the z_i's have zero mean and identity covariance. That is,

$$\bar{Z} = \frac{1}{n}\sum_{i=1}^{n} z_i = \frac{1}{n}Z^\top \mathbf{1}_n = 0, \qquad S = \frac{1}{n}\sum_{i=1}^{n} z_i z_i^\top = \frac{1}{n}Z^\top Z = I_d$$

Here, $\mathbf{1}_d = [1, 1 \dots, 1]^\top \in \mathbb{R}^d$ and I_d is the $d \times d$ identity matrix.

2. **(16 Points)** Outline a procedure to solve the above problem. Specify how you would obtain A, Z, b which minimize the objective and satisfy the constraints.

   Hint: The rank k approximation of a matrix in Frobenius norm is obtained by taking its SVD and then zeroing out all but the first k singular values.

First, we figure out the value of b, the intercept in the affine transformation. For that, we differentiate the objective w.r.t. z_i and set to 0.

$$\begin{aligned}
\nabla_{z_j} J(A, b, Z) &= \nabla_{z_j} \frac{1}{n} \sum_{i=1}^{n} \|x_i - Az_i - b\|^2 \\
&= \frac{1}{n} \nabla_{z_j} (x_j - Az_j - b)^T (x_j - Az_j - b) \\
&= \frac{1}{n} \nabla_{z_j} ((x_j - b) - Az_j)^T ((x_j - b) - Az_j) \\
&= \frac{1}{n} \nabla_{z_j} z_j^T A^T A z_j - 2((x_j - b)A^T)^T z_j + (x_j - b)^T (x_j - b) \\
&= \frac{1}{n} 2A^T A z_j - 2(x_j - b)A^T
\end{aligned}$$

We now set this to zero. z_j now becomes

$$\begin{aligned}
\frac{1}{n} 2A^T A z_j - 2(x_j - b)A^T = 0 \\
\implies A^T A z_j = (x_j - b)A^T \\
\implies z_j = (A^T A)^{-1}(x_j - b)A^T
\end{aligned}$$

Applying the zero mean constraint, we have,

$$\begin{aligned}
\sum_j z_j = 0 \\
\implies \sum_j (A^T A)^{-1}(x_j - b)A^T = 0 \\
\implies (A^T A)^{-1}(\sum_j (x_j - b))A^T = 0 \\
\implies \sum_j (x_j - b) = 0 \\
\implies \sum_j x_j - nb = 0 \\
\implies \boxed{b = \frac{\sum_j x_j}{n}}
\end{aligned}$$

Here, we assume that A has no zero singular values.

Now that we determined the value of b, we now attempt to compute the values of Z, A. First, we define a zero mean matrix $Y = X - b$ to simplify our calculation.

Our objective is $J(A, b, Z) = \frac{1}{n}\|X - ZA^\top - \mathbf{1}b^\top\|_F^2 = \frac{1}{n}\|Y - ZA^\top|_F^2$. We use a key idea that we can obtain the rank-k approximation of a matrix in Forbenius norm by only retaining the top-k singular values of the original matrix. So, the rank-d approximation of $Y = U_{n\times n}\Sigma_{n\times D}V^T_{D\times D}$ is

$$\hat{Y} = \left(U_{n\times n}\begin{bmatrix} I_{d\times d} \\ 0_{n-d\times d}\end{bmatrix}\right)\left(\begin{bmatrix} I_{d\times d} & 0_{d\times n-d}\end{bmatrix}\Sigma_{n\times D}\begin{bmatrix} I_{d\times d} \\ 0_{D-d\times d}\end{bmatrix}\right)\left(\begin{bmatrix} I_{d\times d} & 0_{d\times D-d}\end{bmatrix}V^T_{D\times D}\right)$$

.

The intermediate matrices are introduced to select only the top d singular values/vectors. Now, we wish to express this d-rank approximation as ZA^T where Z has unit covariance and is shape $n \times d$. The shape of A^T is $d \times D$. There is a clear choice here where we can use the term in left brackets for Z.

$$\begin{aligned} Z^TZ &= \begin{bmatrix} I_{d\times d} & 0_{n-d\times d}\end{bmatrix}U^TU\begin{bmatrix} I_{d\times d} \\ 0_{n-d\times d}\end{bmatrix} \\ &= \begin{bmatrix} I_{d\times d} & 0_{n-d\times d}\end{bmatrix}\begin{bmatrix} I_{d\times d} \\ 0_{n-d\times d}\end{bmatrix} \\ &= I_{d\times d}I_{d\times d} + 0 \\ &= I_{d\times d} \end{aligned}$$

Instead, we want Z^TZ to be $nI_{d\times d}$, so we define

$$\boxed{Z = \sqrt{n} \times U\begin{bmatrix} I_{d\times d} \\ 0_{n-d\times d}\end{bmatrix}}$$

This is proportional to the first d columns of U. This has zero mean because $z_i = (A^TA)^{-1}A^T(x_i - b)$ and $\sum_i x_i - b = 0$.

Now factorizing A^T, we have

$$\boxed{A = \frac{1}{\sqrt{n}}\left(V_{D\times D}\begin{bmatrix} I_{d\times d} \\ 0_{D-d\times d}\end{bmatrix}\right)Diag(\sigma_d)_{d\times d}}$$

We have the $\frac{1}{\sqrt{n}}$ factor to compensate for the proportionality constant on Z. The expression in the bracket selects the first d right singular vectors. Finally, we right multiply this with the $d \times d$ diagonal matrix containing the top-d singular values in its diagonal.

Ideally, we would like to express Z in terms of Y to be able to apply the affine transformations on new unseen examples. We do this in the next question.

3. **(3 Points)** You are given a point x_* in the original D dimensional space. State the rule to obtain the d dimensional representation z_* for this new point. (If x_* is some original point x_i from the D–dimensional space, it should be the d–dimensional representation z_i.)

We look at the formula for optimal z_i that we obtained by differentiating the objective, i.e. $z_i = (A^TA)^{-1}A^T(x_i - b)$. Generalizing this to any x, we have $z_* = \boxed{(A^TA)^{-1}A^T(x_i - b)}$

Alternatively, we can also express Z in the previous question in terms of Y. We know that $Y = U_{n\times n}\Sigma_{n\times D}V^T_{D\times D}$.

We right multiply Y by $V_{D\times D}\begin{bmatrix} I_{d\times d} \\ 0_{D-d\times d}\end{bmatrix}Diag(\frac{1}{\sigma_d})_{d\times d}$.

We claim this returns the top-d left singular vectors. First, notice that right multiplying Y with $V_{D\times D}$ gets $U_{n\times n}\Sigma_{n\times D}V^T_{D\times D}V_{D\times D} = U_{n\times n}\Sigma_{n\times D}$. Next, we multiply by the column selector that selects the first d columns. This results in $U_{n\times n}\begin{bmatrix} Diag(\sigma_d) \\ 0\end{bmatrix}$. Finally, multiplying by $Diag(\frac{1}{\sigma_d})$, results in $U_{n\times n}\begin{bmatrix} I_{d\times d} \\ 0\end{bmatrix}$. This is simply the first d columns of U as desired.

Hence, we represent Z as $\sqrt{n} \times YV_{D\times D} \begin{bmatrix} I_{d\times d} \\ 0_{D-d\times d} \end{bmatrix} Diag(\frac{1}{\sigma_d})_{d\times d}$.

Hence, for any new point x_*, we compute z_* as $\boxed{\sqrt{n} \times Diag(\frac{1}{\sigma_d}) \left(\begin{bmatrix} I_d & 0 \end{bmatrix} V^T \right) (x_* - b)}$. Here, $Diag(\frac{1}{\sigma_d})$ represents the reciprocals of top-d singular along the diagonal. The matrix in the brackets represents the top-d right singular vectors. This is equivalent to the formula at the top, just expressed differently.

2.3 Experiment (34 points)

Here we will compare the above three methods on two data sets.

- We will implement three variants of PCA:
  1. "buggy PCA": PCA applied directly on the matrix X.
  2. "demeaned PCA": We subtract the mean along each dimension before applying PCA.
  3. "normalized PCA": Before applying PCA, we subtract the mean and scale each dimension so that the sample mean and standard deviation along each dimension is 0 and 1 respectively.
- One way to study how well the low dimensional representation Z captures the linear structure in our data is to project Z back to D dimensions and look at the reconstruction error. For PCA, if we mapped it to d dimensions via $z = Vx$ then the reconstruction is $V^\top z$. For the preprocessed versions, we first do this and then reverse the preprocessing steps as well. For DRO we just compute $Az + b$. We will compare all methods by the reconstruction error on the datasets.
- Please implement code for the methods: Buggy PCA (just take the SVD of X) , Demeaned PCA, Normalized PCA, DRO. In all cases your function should take in an $n \times d$ data matrix and d as an argument. It should return the the d dimensional representations, the estimated parameters, and the reconstructions of these representations in D dimensions.

  Please see the GitHub link for the implementations in the Jupyter notebook.
- You are given two datasets: A two Dimensional dataset with 50 points `data2D.csv` and a thousand dimensional dataset with 500 points `data1000D.csv`.
- For the $2D$ dataset use $d = 1$. For the $1000D$ dataset, you need to choose d. For this, observe the singular values in DRO and see if there is a clear "knee point" in the spectrum. Attach any figures/ Statistics you computed to justify your choice.

  We observe a sharp dip after the first singular value and then a noticeable knee point after $d = \boxed{31}$ total singular values. See figure 3.
- For the $2D$ dataset you need to attach the a plot comparing the orignal points with the reconstructed points for all 4 methods. For both datasets you should also report the reconstruction errors, that is the squared sum of differences $\sum_{i=1}^{n} \|x_i - r(z_i)\|^2$, where x_i's are the original points and $r(z_i)$ are the D dimensional points reconstructed from the d dimensional representation z_i.

  See figure 4 for reconstruction plots of the four Dimension reduction methods employed. The blue points are the original points and the red ones are reconstructed. See table 1 for reconstruction errors of each method and dataset pair. We report the average error in this case.
- **Questions:** After you have completed the experiments, please answer the following questions.
  1. Look at the results for Buggy PCA. The reconstruction error is bad and the reconstructed points don't seem to well represent the original points. Why is this?
     Hint: Which subspace is Buggy PCA trying to project the points onto?

     In PCA, we are trying to find one direction that is most prominent. When we observe the figure 4a, it is apparent that the eigenvector onto which the PCA is projecting is pointing towards the data but is

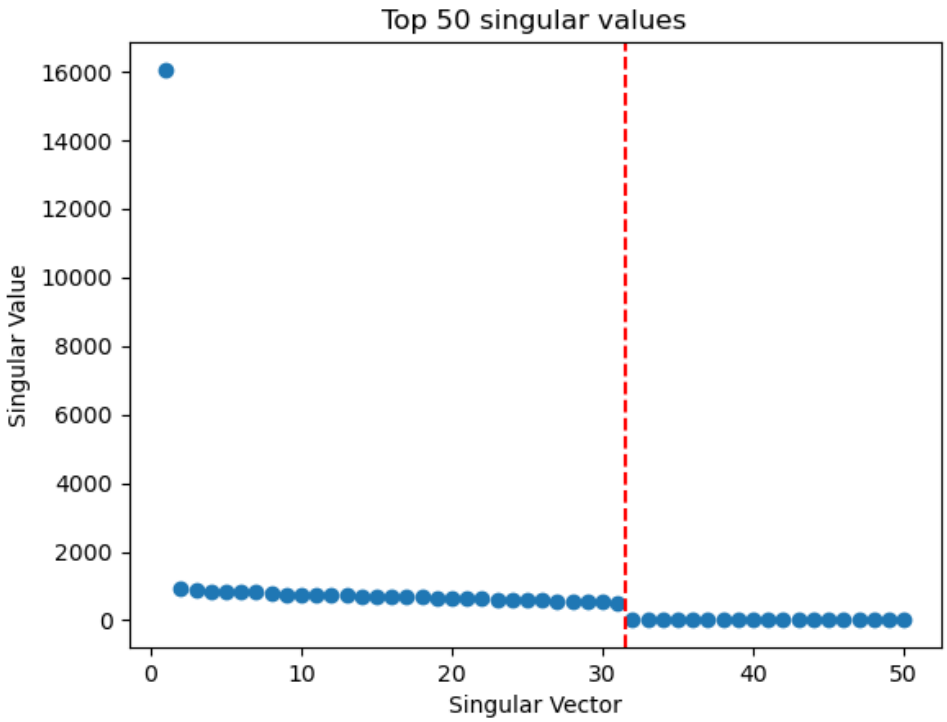


Figure 3: Singular values with 1-based indexing.

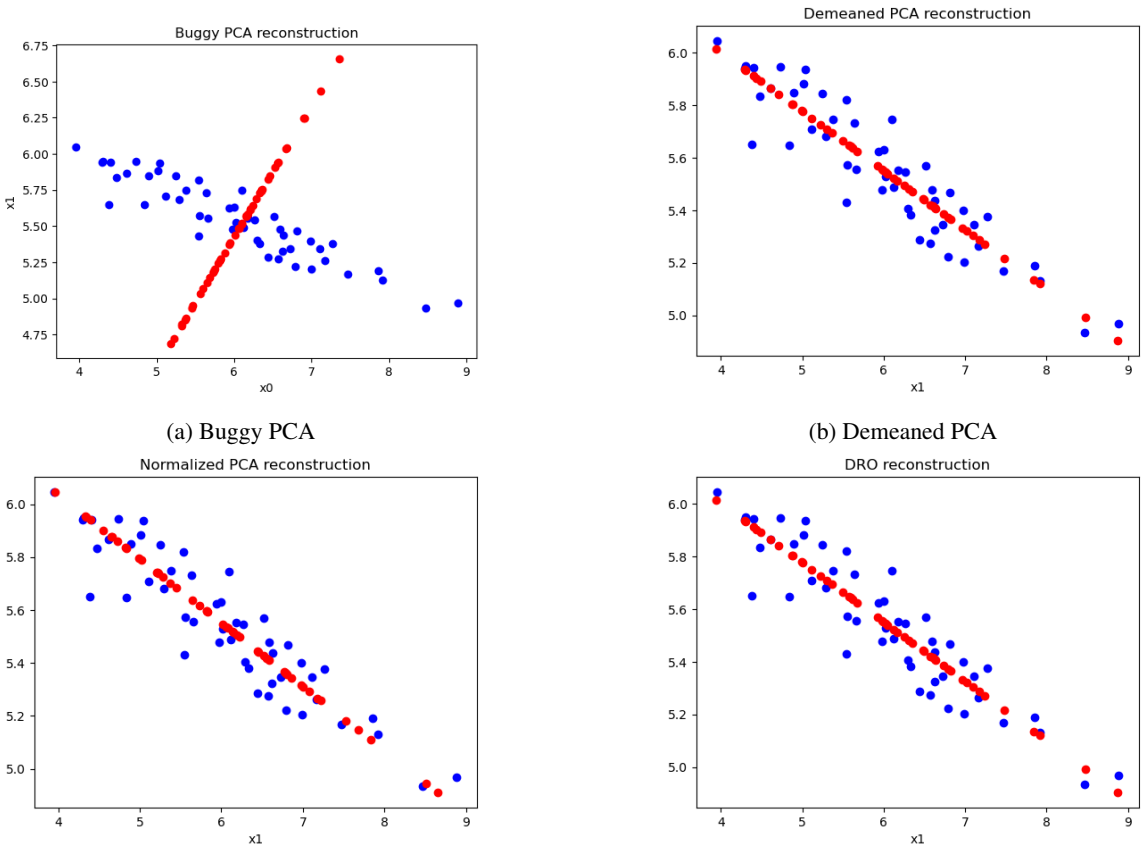
Method/Dimension	2D	1000D
Buggy PCA	0.89	272.77
Demeaned PCA	**0.01**	**271.40**
Norm PCA	0.05	272.06
DROP	**0.01**	**271.40**

Table 1: Reconstruction errors

also constrained to pass through the origin. We want to capture the highest variance direction which is fairly orthogonal to the eigenvector because there is no line through the origin that fits the data very well. Centering/demeaning solves this issue because we do not have to worry about the intercept in that case and lines through origin tend to fit the data better after transformation. A similar argument generalizes to higher dimensions.

2. The error criterion we are using is the average squared error between the original points and the reconstructed points. In both examples DRO and demeaned PCA achieves the lowest error among all methods. Is this surprising? Why?

DRO algorithm is designed explicitly to minimize the average squared error between the original and reconstructed points. Hence, DRO is provably optimal for all affine transformations. First, we argue that each of the mentioned methods are linear/affine transformations and hence cannot have a better mean squared error. The buggy PCA version uses a matrix multiplication to embed in lower dimension as well as to reconstruct back in the original dimension. So, this is clearly linear. The normalized version in addition to matrix multiplications has additional multiplications/divisions with stddev and additions/subtractions with the mean. These operations are clearly linear too. Intuitively, norm PCA fails to perform optimally since we scale independently on each feature without accounting for any correlation between features. Finally, the demeaned PCA is the same operation as DRO. This is because in demeaned PCA, we subtract the mean before embedding and add the mean during reconstruction similar to DRO. The operation $(A^TA)^{-1}A^T(X-b)$ in DRO is equivalent to embedding in demeaned PCA sans some proportionality constants, one for each feature dimension. These proportionality constants are accounted for when reconstructing, so demeaned PCA is operationally



(a) Buggy PCA

(b) Demeaned PCA

(c) Norm PCA

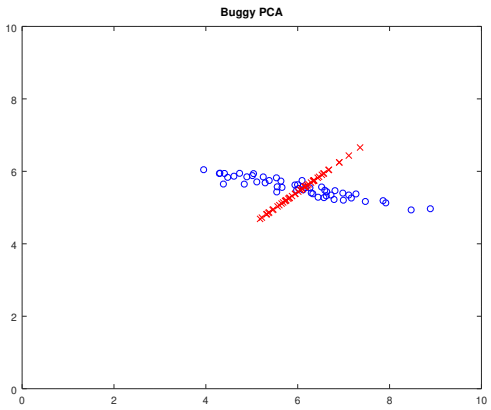
(d) DRO

Figure 4: Reconstruction for the 2D dataset

the same as DRO giving optimal reconstruction errors. While the reconstruction error is similar, the numerical stability of the algorithms may be different because of the proportionality constants (not explored here).

- Point allocation:
  - Implementation of the three PCA methods: **(6 Points)**
  - Implementation of DRO: **(6 points)**
  - Plots showing original points and reconstructed points for 2D dataset for each one of the 4 methods: **(10 points)**
  - Implementing reconstructions and reporting results for each one of the 4 methods for the 2 datasets: **(5 points)**
  - Choice of d for $1000D$ dataset and appropriate justification: **(3 Points)**
  - Questions **(4 Points)**

Answer format:
The graph bellow is in example of how a plot of one of the algorithms for the 2D dataset may look like:



The blue circles are from the original dataset and the red crosses are the reconstructed points.

And this is how the reconstruction error may look like for Buggy PCA for the 2D dataset: 0.886903