IITM-CS5691 : Pattern Recognition and Machine Learning　　Release Date: Apr 5, 2021
Assignment 2　　Due Date: Apr 18, 2021 (11.59pm)

Roll No: EE18B001　　Name: Abishek S
Collaborators (if any):
References (if any): Bishop, class slides

- Use LaTeX to write-up your solutions (in the solution blocks of the source LaTeX file of this assignment), and submit the resulting single pdf file at GradeScope by the due date. (Note: **No late submissions** will be allowed, other than one-day late submission with 10% penalty or four-day late submission with 30% penalty! Within GradeScope, indicate the page number where your solution to each question starts, else we won't be able to grade it! You can join GradeScope using course entry code **5VDNKV**).
- For the programming question, please submit your code (rollno.ipynb file and rollno.py file in rollno.zip) directly in moodle, but provide your results/answers in the pdf file you upload to GradeScope.
- Collaboration is encouraged, but all write-ups must be done individually and independently, and mention your collaborator(s) if any. Same rules apply for codes written for any programming assignments (i.e., write your own code; we will run plagiarism checks on codes).
- If you have referred a book or any other online material for obtaining a solution, please cite the source. Again don't copy the source *as is* - you may use the source to understand the solution, but write-up the solution in your own words.
- Points will be awarded based on how clear, concise and rigorous your solutions are, and how correct your code is. Overall points for this assignment would be **min**(your score including bonus points scored, 50).

1. (10 points) [SINGULARLY PCA!] Consider a dataset of N points with each datapoint being a D-dimensional vector in $\mathbb{R}^D$. Let's assume that:

   - we are in a high-dimensional setting where $D >> N$ (e.g., D in millions, N in hundreds).
   - the $N \times D$ matrix X corresponding to this dataset is already mean-centered (so that each column's mean is zero, and the covariance matrix seen in class becomes $S = \frac{1}{N}X^TX$).
   - the rows (datapoints) of X are linearly independent.

   Under the above assumptions, please attempt the following questions.

   (a) (3 points) Whereas X is rectangular in general, XX^T and X^TX are square. Show that these two square matrices have the same set of non-zero eigenvalues. Further, argue briefy why these equal eigenvalues are all positive and N in number, and derive the multiplicity of the zero eigenvalue for both these matrices.
   (Note: The square root of these equal positive eigenvalues $\{\lambda_i := \sigma_i^2\}_{i=1,\dots,N}$ are called the singular values $\{\sigma_i\}_{i=1,\dots,N}$ of X.)

Solution:
We will show that the set of non-zero eigenvalues of XX^T and X^TX are equal in two steps :

- Suppose λ_i is a non-zero eigenvalue of XX^T with eigenvector u_i.

$$\begin{aligned} &XX^Tu_i = \lambda_i u_i \\ &\implies X^TXX^Tu_i = \lambda_i X^Tu_i \qquad \text{(multiplying by } X^T \text{ on both sides)} \\ &\implies X^TX(X^Tu_i) = \lambda_i(X^Tu_i) \end{aligned}$$

  Hence, λ_i is an eigenvalue of X^TX with corresponding eigenvector X^Tu_i. Therefore, non-zero eigenvalues of $XX^T \subseteq$ non-zero eigenvalues of X^TX.
  (**NOTE :** λ_i being non-zero is important in the proof so that LHS is non-zero and hence, $X^Tv_i \neq \mathbf{0}$ and it is a valid eigenvector).

- Suppose λ_i is a non-zero eigenvalue of X^TX with eigenvector v_i.

$$\begin{aligned} &X^TXv_i = \lambda_i v_i \\ &\implies XX^TXv_i = \lambda_i Xv_i \qquad \text{(multiplying by } X \text{ on both sides)} \\ &\implies XX^T(Xv_i) = \lambda_i(Xv_i) \end{aligned}$$

  Hence, λ_i is an eigenvalue of XX^T with corresponding eigenvector Xv_i. Therefore, non-zero eigenvalues of $X^TX \subseteq$ non-zero eigenvalues of XX^T.

$\langle XX^Ty, y\rangle = \langle X^Ty, X^Ty\rangle = \|X^Ty\|^2 \geqslant 0$, hence XX^T is a positive semi-definite matrix. Hence all eigenvalues are all $\geqslant 0$, or all non-zero eigenvalues are positive.
$\mathrm{rank}(XX^T) = \mathrm{rank}(X) = \mathrm{rank}(X^T) = \mathrm{rank}(X^TX)$ and they are equal to N as the rows of X are linearly independent.
Multiplicity of eigenvalue zero of a matrix is the $\mathrm{nullity}$ of the matrix.

Using Fundamental theorem of linear algebra, $\mathrm{rank}(XX^T) + \mathrm{nullity}(XX^T) = N$ and $\mathrm{rank}(X^TX) + \mathrm{nullity}(X^TX) = D$. Therefore, $\mathrm{nullity}(XX^T) = 0$ and $\mathrm{nullity}(X^TX) = D - N$. Thus, the multiplicity of zero eigenvalue for XX^T is zero and for X^TX is $D - N$.
(The non-zero eigenvalues are hence N in both matrices and is consistent with the fact that the sets, hence their cardinalities are equal as we verified)

(b) (2 points) We can choose the set of eigenvectors $\{u_i\}_{i==1,\dots,N}$ of XX^T to be an orthonormal set and similarly we can choose an orthonormal set of eigenvectors $\{v_j\}_{j=1,\dots,D}$ for X^TX. Briefly argue why this orthonormal choice of eigenvectors is possible. Can you choose $\{v_i\}$ such that each v_i can be computed easily from u_i and X alone (i.e., without having to do an eigenvalue decomposition of the large matrix X^TX; assume $i = 1, \dots, N$ so that $\lambda_i > 0$ and $\sigma_i > 0$)?
(Note: $\{u_i\}, \{v_i\}$ are respectively called the left,right singular vectors of X, and computing them

along with the corresponding singular values is called the Singular Value Decomposition or SVD of X.)

Solution:
Real and symmetric matrices of dimension $N \times N$ have an orthonormal set of N eigenvectors from real spectrum theorem.
$(XX^T)^T = XX^T \implies XX^T$ is a real and symmetric matrix of dimension $N \times N$ hence there exists an orthonormal set of eigenvectors $\{u_i\}_{i-=1,\dots,N}$ for XX^T.
$(X^TX)^T = X^TX \implies X^TX$ is a real and symmetric matrix of dimension $D \times D$ hence there exists an orthonormal set of eigenvectors $\{v_j\}_{j=1,\dots,D}$ for X^TX.

Now, consider

$$\begin{aligned} &XX^Tu_i = \lambda_i u_i \\ &\implies X^TXX^Tu_i = \lambda_i X^Tu_i \qquad \text{(multiplying both sides by } X^T\text{)} \\ &\implies X^TX(X^Tu_i) = \lambda_i(X^Tu_i) \end{aligned}$$

The above is of the form $X^TXv_i = \lambda_i v_i$, where v_i is the eigenvector of X^TX.
Hence, we can obtain N such eigenvectors from the eigenvectors of XX^T. But let us verify that they form an orthonormal set.
For $i \neq j$, $\langle v_i, v_j\rangle = \langle X^Tu_i, X^Tu_j\rangle = \langle XX^Tu_i, u_j\rangle = \langle \lambda_i u_i, u_j\rangle = \lambda_i \langle u_i, u_j\rangle = 0$. Hence they form an orthogonal set (which implies they are linearly independent as well).
To make the eigenvectors normalized, we can choose $v_i = \frac{X^Tu_i}{\|X^Tu_i\|}$.
Thus we obtain orthonormal set of N eigenvectors of X^TX as $\{v_j\}_{j=1,\dots,N}$ easily from X and $\{u_i\}_{i=1,\dots,N}$.
Since we saw that X^TX and XX^T have the same set of non-zero eigenvalues, the remaining $D - N$ eigenvectors of X^TX correspond to 0 eigenvalue, and is not required for most applications (like PCA) as it is the minimum eigenvalue, though it can be found by finding the null space of X^TX.
Hence the choice of $v_i = \frac{X^Tu_i}{\|X^Tu_i\|}$ for $i = 1, \dots, N$

(c) (2 points) Applying PCA on the matrix X would be computationally difficult as it would involve finding the eigenvectors of $S = \frac{1}{N}X^TX$, which would take $O(D^3)$ time. Using answer to the last question above, can you reduce this time complexity to $O(N^3)$? Please provide the exact steps involved, including the exact formula for computing the normalized (unit-length) eigenvectors of S.

Solution:
We know that error when first M components in PCA is used to represent the data is given by $\sum_{i=M+1}^{D} u_i^T S u_i = \sum_{i=M+1}^{D} \lambda_i$. Since S is positive semidefinite, all eigenvalues are $\geqslant 0$

and hence the zero eigenvalues which come at last when eigenvalues are ordered in non-increasing order can be ignored because they dont cause any error. (They also don't cause any variance, as variance along u_i is given by $u_i^T S u_i = \lambda_i$ again)
Also, $X^T X v_i = \lambda_i v_i \implies \frac{1}{N} X^T X v_i = \frac{1}{N}\lambda_i v_i$, hence the eigenvectors remain the same, but eigenvalues are scaled by N.
So, from **part b**, we know that the N eigenvectors of S only is important (as the rest $D - N$ correspond to 0 eigenvalue), and are given by : $v_i = \frac{X^T u_i}{\|X^T u_i\|}$ for $i = 1, \dots, N$ corresponding to eigenvalue λ_i where u_i's are eigenvectors of $\frac{1}{N} X X^T$ corresponding to eigenvalues λ_i's (the top N eigenvalues of $\frac{1}{N} X X^T$ and $\frac{1}{N} X^T X$ will be the same).

Naively finding the eigenvectors and eigenvalues of S would take $O(D^3)$.
On the other hand, finding the eigenvectors and eigenvalues of $\frac{1}{N} X X^T$ will take $O(N^3)$. Finding the eigenvectors of S from it requires $O(DN^2)$ for the matrix multiplication $X^T u_i$, and there are N such multiplications. Hence, the overall complexity reduces to $O(DN^3 + N^3) = O(DN^3)$, which will be much faster than $O(D^3)$ for $D >> N$. Also since the N eigenvalues of $\frac{1}{N} X X^T$ and S are same, finding it for S through the latter method needs only $O(N^3)$ time.

(d) (3 points) Exercise 12.2 from Bishop's book helps prove why minimum value of the PCA squared error J, subject to the orthonormality constraints of the set of principal axes/directions $\{u_i\}$ that we seek, is obtained when the $\{u_i\}$ are eigenvectors of the data covariance matrix S. That exercise introduces a modified squared error $\widetilde{J}$, which involves a matrix H of Langrange multipliers, one for each constraint, as follows:

$$\widetilde{J} = \mathrm{Tr}\left\{\widehat{U}^T S \widehat{U}\right\} + \mathrm{Tr}\left\{H(I - \widehat{U}^T \widehat{U})\right\}$$

where $\widehat{U}$ is a matrix of dimension $D \times (D - M)$ whose columns are given by u_i. Now, any solution to minimizing $\widetilde{J}$ should satisfy $S\widehat{U} = \widehat{U}H$, and one specific solution is that the columns of $\widehat{U}$ are the eigenvectors of S, in which case H is a diagonal matrix. Show that any general solution to $S\widehat{U} = \widehat{U}H$ also gives the same value for $\widetilde{J}$ as the above specific solution.
(Hint: Show that H can be assumed to be a symmetric matrix, and then use the eigenvector expansion i.e., diagonalization of H.)

Solution:
Observation 1 :
The second term of the equation for $\widetilde{J}$ is the lagrange multiplier term, which is present to ensure orthonormality of column vectors of $\widehat{U}$.
When we take the gradient of the $\widetilde{J}$ and set it to zero to obtain the solution $S\widehat{U} = \widehat{U}H$, notice that we would also take gradient wrt H, and since only the lagrangian term depends

on H, it would implicity make sure that the column vectors of $\widehat{U}$ (call them u_i's) form an orthonormal set of vectors.
Hence, $\widehat{U}^T\widehat{U} = I$ (Identity matrix of $(D-M) \times (D-M)$ dimensions) - **(1)**.

Observation 2 :
The general solution is $S\widehat{U} = \widehat{U}H$. That $\implies$ $\widehat{U}^T S\widehat{U} = H$ (pre-multiplying by $\widehat{U}^T$ and using 1). Since, $\left(\widehat{U}^T S\widehat{U}\right)^T = \widehat{U}^T S^T\widehat{U} = \widehat{U}^T S\widehat{U}$ (because S is covariance matrix and is real and symmetric), the RHS H is also a symmetric matrix.
We can also look at it intuitively as well. The lagrangian term penalizes $1 - \langle u_k, u_k\rangle$ and $\langle u_i, u_j\rangle$ for $i \neq j$ with some weight H_{kk} and H_{ij} respectively.
But $\langle u_i, u_j\rangle = \langle u_j, u_i\rangle$ and both H_{ij} and H_{ji} for $i \neq j$ penalize $\langle u_i, u_j\rangle = \langle u_j, u_i\rangle$ only. Hence, even if they are not equal (say $H_{ij} = h_1$ and $H_{ji} = h_2$), we can assign $H_{ij} = H_{ji} = \frac{h_1+h_2}{2}$ and end up with the same objective function $\widetilde{J}$.
Hence, H can be assume to be a symmetric matrix.

From observation 1, $I - \widehat{U}^T\widehat{U} = 0$, hence the second term in the expression for $\widetilde{J}$ is always zero for any general solution of $H, \widehat{U}$.
From observation 2, in any general solution H can be assumed to be symmetric and real ($\implies$ self-adjoint), and hence there exists an orthonormal basis of eigenvectors such that H can be diagonalized by real spectrum theorem (i.e) $H = V\Lambda V^T$, where H is of dimensions $(D-M) \times (D-M)$ and so is those of V, V^T and Λ. The column vectors of V, call them v_i's are the $(D-M)$ eigenvectors of H and the $(D-M)$ diagonal elements of the diagonal matrix Λ, call them λ_i's are the corresponding eigenvalues. From setting the gradients to zero we obtained $S\widehat{U} = \widehat{U}H$ as the solution.
Now,

$$\begin{aligned}
& Hv_i = \lambda_i v_i \\
\implies\ & \widehat{U}Hv_i = \widehat{U}\lambda_i v_i && \text{(pre-multiply by } \widehat{U}\text{)} \\
\implies\ & S\widehat{U}v_i = \lambda_i \widehat{U}v_i && \text{(from } S\widehat{U} = \widehat{U}H\text{)} \\
\implies\ & S\left(\widehat{U}v_i\right) = \lambda_i\left(\widehat{U}v_i\right)
\end{aligned}$$

Since the columns of $\widehat{U}$ are orthonormal ($\implies$ linearly independent), $\widehat{U}v_i \neq 0$ as long as $v_i \neq 0$, which is true as it is eigenvector of H.
Hence, all eigenvalues λ_i's of H (with eigenvector v_i's) are eigenvalues of S with corresponding eigenvectors $\widehat{U}v_i$'s.

The first term of $\widetilde{J}$,

$$\begin{aligned} \operatorname{Tr}\left\{\widehat{U}^T S \widehat{U}\right\} &= \operatorname{Tr}\{H\} \quad \text{(pre-multiplying } S\widehat{U} = \widehat{U}H \text{ by } \widehat{U}\text{)} \\ &= \sum_{i=1}^{D-M} \lambda_i \end{aligned}$$

where λ_i for $i = 1, 2, \cdots, D-M$ are the $(D-M)$ eigenvalues of H.
The last equality follows from the fact that the trace of a matrix is equal to the sum of it's eigenvalues (since the trace remains unchanged on similarity transformation of matrix). We can prove it like :

$$\begin{aligned} H &= V\Lambda V^T \\ \operatorname{Tr}\left\{V\Lambda V^T\right\} &= \sum_{i=1}^{D-M} \left(V\Lambda V^T\right)_{ii} \\ &= \sum_{i=1}^{D-M} \sum_{j=1}^{D-M} (V\Lambda)_{ij} V^T_{ji} \\ &= \sum_{i=1}^{D-M} \sum_{j=1}^{D-M} \sum_{k=1}^{D-M} V_{ik}\Lambda_{kj}V^T_{ji} \\ &= \sum_{j=1}^{D-M} \sum_{k=1}^{D-M} \Lambda_{kj} \left(\sum_{i=1}^{D-M} V_{ik}V^T_{ji}\right) \\ &= \sum_{j=1}^{D-M} \sum_{k=1}^{D-M} \Lambda_{kj} \left(\sum_{i=1}^{D-M} V^T_{ki}V_{ij}\right) \\ &= \sum_{j=1}^{D-M} \sum_{\substack{k=1 \\ k=j}}^{D-M} \Lambda_{kj} \\ &\qquad \text{(V - orthonormal matrix, hence } \sum_{i=1}^{D-M} V^T_{ki}V_{ij} = 0 \text{ if } k \neq j \text{ else 1)} \\ &= \sum_{j=1}^{D-M} \Lambda_{jj} = \sum_{j=1}^{D-M} \lambda_{jj} \end{aligned}$$

Hence using a general solution to $S\widehat{U} = \widehat{U}H$ we obtained the value of $\widetilde{J}$ as sum of $(D-M)$ eigenvalues of H, which is sum of (some arbitrary) $(D-M)$ eigenvalues of S (eigenvalues of $H \subseteq$ those of S), which is what we do in the specific solution where H is a diagonal matrix containing $(D-M)$ eigenvalues of S and $\mathrm{Tr}\{H\} = \sum_{i=1}^{D-M} \lambda_i$.

If we see in detail, in the specific solution we pick H to be a diagonal matrix with the minimum $(D-M)$ eigenvalues of S and $\widehat{U}$ as the corresponding eigenvectors of S as its diagonal entries, so that the modified squared error $\widetilde{J} = \sum_{i-1}^{D-M} \lambda_i$ is minimum. But choosing arbitrary $\widehat{U}$ and symmetric H for the general solution, we obtain $\widetilde{J}$ as sum of some $(D-M)$ eigenvalues of S which obviously will be $\geqslant$ the value of $\widetilde{J}$ obtained for the specific solution (since we pick the $(D-M)$ minimum ones in specific solution).

Therefore, the specific solution we obtain is not just a local maxima, but the global maxima as well (as it is as good as any other solution obtained by setting gradients of $\widetilde{J}$ wrt $\widehat{U}$ and H to zero).

2. (10 points) [TO SQUARE OR NOT SQUARE WITH K-MEANS]

   (a) (3 points) If instead of squared Euclidean distance, you use ℓ_1 norm in the objective function of (hard) K-means, then what are the new assignment and update equations? If your data contains outliers, would you prefer this over the regular K-means? Justify.

Solution:

ℓ_1 norm of vector x : $\|x\|_1$.

The distortion measure to minimize now would be :

$$J\left(r, \mu^{(1)}, \ldots, \mu^{(K)}\right) = \sum_{i=1}^{N} \sum_{j=1}^{K} r_j^{(i)} \left\|x^{(i)} - \mu^{(j)}\right\|_1$$

The assignment would remain the system, except the meaning of closest distance now changes.

Assignment step :

$$\begin{aligned} r_k^{(n)} &= 1 \qquad \text{if } \left\|x^{(n)} - \mu^{(k)}\right\|_1 \text{ is the smallest among all } \mu^{(i)}\text{'s } \forall i \in [1, K] \\ &= 0 \end{aligned}$$

For update step, the $r_k^{(n)}$'s are fixed, so we need to minimize $d(X, \mu^{(k)}) = \sum_{i=1}^{N_k} \left\|x^{(i)} - \mu^{(k)}\right\|_1$ wrt $\mu^{(k)}$ for the given set of $x^{(i)}$'s such that $r_k^{(i)} = 1$ (N_k is the number of such $x^{(i)}$'s allotted to $\mu^{(k)}$).

To get the minimum, we set the gradient wrt μ to 0.

$\|x - \mu\|_1 = \sum_{i=1}^{D} |x_i - \mu_i|$, where D is the dimension of x. Hence, the gradient of $\|x - \mu\|_1$ wrt μ_i will be $+1$, -1 and 0 for $\mu_i > x_i$, $\mu_i < x_i$ and $\mu_i = x_i$ respectively.

So the gradient for $d(X, \mu^{(k)})$ will contain summation of 1's and -1's and 0's in every dimension, and for the gradient to be zero, $\mu^{(k)}$ has to be chosen such that the number of

points on either side of $\mu^{(k)}$ has to be equal in every dimension. That is,
Update step :

$$\mu^{(k)} = median(x^{(i)} \mid r_k^{(i)} = 1) \quad \forall k \in [1, K]$$

ℓ_1 norm might be preferred for noisy data because the geometric median is robust to outliers. This is because the distance to the outliers won't be considered and only the number of points on each side is considered to pick the middle point in the case of ℓ_1 norm, as opposed to ℓ_2 norm.

(b) (2 points) Consider a Gaussian mixture model with scalar covariance matrices: $\Sigma_r = \sigma^2 I$ where σ is a fixed parameter, r represents the mixture-component/cluster, and I the identity matrix. Show that for this model as σ tends to zero, the EM-based soft K-means algorithm (i.e., its assignment/update equations) become the same as the hard K-means algorithm.

Solution:
We had seen a Gaussian mixture model for Soft K-means with per-cluster prior and variances in class. Here, we have the variance fixed, hence apart from the update equation for variance everything would remain the same.
Assignment :

$$r_k^{(n)} = \frac{\pi_k \frac{1}{\left(\sqrt{2\pi\sigma}\right)^I} \exp\left(-\frac{1}{2\sigma^2} d\left(m^{(k)}, x^{(n)}\right)\right)}{\sum_{k'} \pi_{k'} \frac{1}{\left(\sqrt{2\pi\sigma}\right)^I} \exp\left(-\frac{1}{2\sigma^2} d\left(m^{(k')}, x^{(n)}\right)\right)} = \frac{\pi_k \exp\left(-\frac{1}{2\sigma^2} d\left(m^{(k)}, x^{(n)}\right)\right)}{\sum_{k'} \pi_{k'} \exp\left(-\frac{1}{2\sigma^2} d\left(m^{(k')}, x^{(n)}\right)\right)}$$

where I is the dimensionality of x, $m^{(k)}$ is the mean of cluster k, $d(m^{(k)}, x^{(n)}) = \left\| x^{(n)} - m^{(k)} \right\|$ and π_k is the prior of cluster k.
Update :

$$m^{(k)} = \frac{\sum_n r_k^{(n)} x^{(n)}}{R^{(k)}}$$

$$\pi_k = \frac{R^{(k)}}{\sum_k R^{(k)}}$$

where N is the number of total datapoints, $R^{(k)} = \sum_n r_n^{(k)}$ is the responsibility of cluster k. As $\sigma \to 0$ and for a given k, the exponential term $\exp\left(-\frac{1}{2\sigma^2} d\left(m^{(k)}, x^{(n)}\right)\right)$ with smallest $d\left(m^{(k)}, x^{(n)}\right)$ will decay slower, leading to :

$$\lim_{\sigma \to \infty} \frac{\pi_k \exp\left(-\frac{1}{2\sigma^2} d\left(m^{(k)}, x^{(n)}\right)\right)}{\sum_{k'} \pi_{k'} \exp\left(-\frac{1}{2\sigma^2} d\left(m^{(k')}, x^{(n)}\right)\right)} = 1$$

(as the denominator containing the sum decays as slow as the numerator only).
For the other $x^{(n)}$'s the limit will tend to 0, hence clearly indicating the soft K-means as-

signment step converges to the hard K-means assignment step.
Since the $r_n^{(k)}$ converges to that in hard K-means, the update steps become :

$$m^{(k)} = \frac{\sum_{n;r_n^{(k)}=1} x^{(n)}}{N_k}$$
$$\pi_k = \frac{N_k}{N}$$

where N is the number of datapoints and N_k is the number of datapoints belonging to cluster k as in the assignment step.
It is clear that the update and assignment steps become what we saw in hard K-means on sending $\sigma \to 0$.

(c) (5 points) We will see how K-means clustering can be done in polynomial time if the data points are along a line (1-dimensional or 1D).

i. (1 point) Consider four datapoints: $x_1 = 1, x_2 = 3, x_3 = 6$, and $x_4 = 7$; and desired number of clusters $k = 3$. What is the optimal K-means clustering in this case?

Solution:
Few observations regarding optimal solution in 1D :

- It is optimal for two points with $x_i \leqslant x_j$ to be assigned to clusters with centers μ_p and μ_q with $\mu_p \leqslant \mu_q$. Otherwise we can change the assignment to this, and the sum of squared distances to the respective cluster centers can be shown to become lesser.
  Put in another way, the optimal solution will involve partitioning the points into contiguous intervals. (will be proved in **part 3**)
- For a given set of points, the cluster center that minimizes the sum of squared distances will be the mean of the points assigned to that cluster, as we have already seen.

Making use of the above observations, we can come up with a Dynamic programming solution to obtain the global optimal solution (dynamic programming results in global optimal solution).
The dynamic programming solution is explained in **part 4**, and here we code it and run on the input given to obtain the global optimal assignment of points x_1, x_2, x_3, x_4 to 3 clusters as :
Cluster-1 : $x_1 = 1$ with cluster mean $\mu_1 = 1$
Cluster-2 : $x_2 = 3$ with cluster mean $\mu_2 = 3$
Cluster-3 : $x_3 = 6, x_4 = 7$ with cluster mean $\mu_3 = 6.5$
The distortion function in this case would be 0.5

ii. (1 point) You might think that the iterative K-means algorithm seen in class converges to global optima with 1D datapoints. Show that it can get stuck in a suboptimal cluster assignment for the problem in part (i).

Solution:
The K-means iterative algorithm involves initialising the cluster means to random values at the beginning. Consider the following initialisation for cluster means :
Cluster-1 : μ_1 initialised to 2
Cluster-2 : μ_2 initialised to 6
Cluster-3 : μ_3 initialised to 7
In the **Assignment** step,
$x_1 = 1$ will be assigned to $\mu_1 = 2$
$x_2 = 3$ will be assigned to $\mu_1 = 2$
$x_3 = 6$ will be assigned to $\mu_1 = 6$
$x_4 = 7$ will be assigned to $\mu_1 = 7$
In the **Update** step,
μ_1 will remain 2
μ_2 will remain 6
μ_3 will remain 7
In the next iteration the assignment and update both won't change, as the nearest cluster mean to $x_1 = 1$ and $x_2 = 3$ is $\mu_1 = 2$, and similarly for x_3 and x_4. Hence the iteration stops.
Now, the distortion function for the cluster assignment obtained is :

$$\sum_{i=1}^{n} \sum_{j=1}^{k} r_j^{(i)} (x_i - \mu_j)^2 = (1-2)^2 + (3-2)^2 = 2$$

Clearly, the distortion function is greater than for the solution we showed in **part 1**. Hence this is a suboptimal assignment, leading to a local optima.

iii. (3 points) Suppose we sort our data such that $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_n$. Show then that any optimal K-means clustering partitions the points into contiguous intervals, i.e. prove that each cluster in an optimal clustering consists of points $x_a, x_{a+1}, \ldots, x_b$ for some $1 \leqslant a \leqslant b \leqslant n$.

Solution:
Assume the contrary. Suppose the K-means algorithm terminated with an assignment that did not partition the points into contiguous intervals.
Then $\exists\ x_i, x_j$ such that $x_i <= x_j$ which are assigned to clusters q and p respectively with centers $\mu_q > \mu_p$. There are 5 possible cases :

- **CASE 1** - $\mu_p < \mu_q \leqslant x_i \leqslant x_j$

$$x_j - \mu_q < x_j - \mu_p \implies (x_j - \mu_q)^2 < (x_j - \mu_p)^2$$

  Hence, the assignment step would assign point x_j to cluster q, hence the K-means algorithm should not have terminated.

- **CASE 2** - $\mu_p < x_i \leqslant \mu_q \leqslant x_j$ or $\mu_p \leqslant x_i < \mu_q \leqslant x_j$

$$x_j - \mu_q < x_j - \mu_p \implies (x_j - \mu_q)^2 < (x_j - \mu_p)^2$$

  Hence, the assignment step would assign point x_j to cluster q, hence the K-means algorithm should not have terminated.

- **CASE 3** - $x_i \leqslant \mu_p < \mu_q \leqslant x_j$

$$x_j - \mu_q < x_j - \mu_p \implies (x_j - \mu_q)^2 < (x_j - \mu_p)^2$$

  Hence, the assignment step would assign point x_j to cluster q, hence the K-means algorithm should not have terminated.

- **CASE 4** - $x_i \leqslant \mu_p \leqslant x_j < \mu_q$ or $x_i \leqslant \mu_p < x_j \leqslant \mu_q$

$$x_i - \mu_p < x_i - \mu_q \implies (x_i - \mu_p)^2 < (x_i - \mu_q)^2$$

  Hence, the assignment step would assign point x_i to cluster p, hence the K-means algorithm should not have terminated.

- **CASE 5** - $x_i \leqslant x_j \leqslant \mu_p < \mu_q$

$$x_i - \mu_p < x_i - \mu_q \implies (x_i - \mu_p)^2 < (x_i - \mu_q)^2$$

  Hence, the assignment step would assign point x_i to cluster p, hence the K-means algorithm should not have terminated.

Thus in all the possible cases, we end up with a contradiction. Hence, the statement that optimal K-means partition the points into contiguous intervals is **true**.

iv. (1 point) [BONUS] Show a $O(kn^2)$ dynamic programming algorithm of k-means when the data is 1-dimensional.

Solution:
The observations described in **part 1** leads to the following Dynamic programming

solution to find the global optimal assignment of k clusters to points $x_1, \dots, x_n$ where $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_n$ WLOG.
Let $DP[i][m]$ be the cost of optimally clustering $x_1, \dots, x_m$ into i clusters.
Let $\mathtt{cost}(j, m)$ be the cost of clustering points $x_j, \cdots, x_m$ into a single cluster (i.e) the sum of squared distance of points $x_j, \cdots x_m$ to their cluster center (which will be the mean of $x_j, \cdots, x_m$ for the optimal case).
Base cases : $DP[0][0] = 0$ (there is no cost incurred on assigning 0 points to 0 clusters)
Rest all elements of DP matrix initialised to ∞
For $i > 0$,

$$DP[i][m] = \min_{j=1}^{m} DP[i-1][j-1] + \mathtt{cost}(j, m)$$

Using another matrix

$$PREV[i][m] := \arg\min_{j=1}^{m} DP[i-1][j-1] + \mathtt{cost}(j, m)$$

it is possible to get the construct the cluster assignment. $PREV[i][m]$ stores the index such that $x_{PREV[i][m]}$ will act as the starting point for i^{th} cluster's interval that is ending at x_m and results in minimum overall cost of clustering $x_1, \cdots, x_m$ into i clusters.
The dynamic programming solution results in the global optimal solution for the clustering of 1D datapoints.
Each update equation for $DP[i][m]$ takes $O(n)$ steps (if cost(j,m) takes $O(1)$ time) to find the optimal choice of j, and there are $O(kn)$ elements in the matrix to fill. Hence the solution has $O(kn^2)$ time and $O(kn)$ space complexity.
For calculating $\mathtt{cost}(i, j)$ we can use another DP. Notice that,

$$\sum_{l=i}^{j} (x_l - m_{i,j})^2 = \sum_{l=i}^{j-1} (x_l - m_{i,j-1})^2 + \frac{j-i}{j-i+1}(x_j - m_{i,j-1})^2$$

where $m_{i,j}$ is the mean of points $x_i, \cdots, x_j$.
The $m_{i,j}$ can also be calculated along with $\mathtt{cost}(i, j)$ using a recurrence equation.
Hence, the recurrence equations will be :

$$\begin{aligned} m_{i,j} &= \frac{(j-i)m_{i,j-1} + x_j}{j-i+1} \\ \mathtt{cost}(i,j) &= \mathtt{cost}(i, j-1) + \frac{j-i}{j-i+1}(x_j - m_{i,j-1})^2 \\ \forall \quad & j > i \end{aligned}$$

Base case : $\mathtt{cost}(i, i) = 0$ and $m_{i,i} = x_i$
The above DP takes $O(n^2)$ time and space and can be pre-computed, so that they can be used in the calculating $DP[i][m]$ and $PREV[i][m]$ in $O(1)$.
So overall time complexity of the solution would be $O(n^2 + kn^2) = O(kn^2)$.

3. (10 points) [THINKING HIERARCHICALLY...] Consider some of the most common metrics of distance between two clusters $A = \{a_1, a_2, \ldots, a_m\}$ and $B = \{b_1, b_2, \ldots, b_n\}$.

- Minimum distance between any pair of points from the two clusters

$$\min_{a \in A, b \in B} ||a - b||$$

- Maximum distance between any pair of points from the two clusters,

$$\max_{a \in A, b \in B} ||a - b||$$

- Average distance between *all* pairs of points from the two clusters,

$$\frac{1}{m \cdot n} \sum_{i=1}^{m} \sum_{j=1}^{n} ||a_i - b_j||$$

As discussed in class, we can obtain clusters by *cutting* the hierarchical tree with a line that crosses at required number of points (K).

(a) (2 points) Which of the three distance/dissimilarity metrics described above would most likely result in clusters most similar to those given by K-means? (Consider the hierarchical clustering method as described in class and further *cut* the tree to obtain K clusters. Assume K is power of 2.) Explain briefly.

Solution:
Average distance/dissimilarity metric would most likely result in clusters most similar to those given by K-means. This is because K-means tries to make sure the distance of point to the centroid they are assigned to (can be approximately thought of as the average distance between the point and other points in the same cluster in average linkage clustering) is lesser than the distance of the point to any other cluster centroid (can be approximately thought of as the average distance between the point to points in another cluster). Hence the objective in both the methods is nearly the same. Also, both the methods tend to create clusters that are compact and contains close-knit points (we saw that K-means can be considered as fitting Gaussian mixture models with $\sigma \to 0$).
On the other hand, single linkage clustering tend to create chain clusters and complete linkage clustering tend to form compact circles, hence is different in the type/shape of clusters they form. Also they look at the extreme distances between two clusters hence have an objective quite different from K-means (that's the reason for the different shape of clusters they favour as well).

(b) (3 points) Which among the three metrics discussed above (if any) would lead hierarchical clustering to correctly separate the two moons in Figure 1a? How would your answer change (if at all) in case of Figure 1b? Explain briefly.

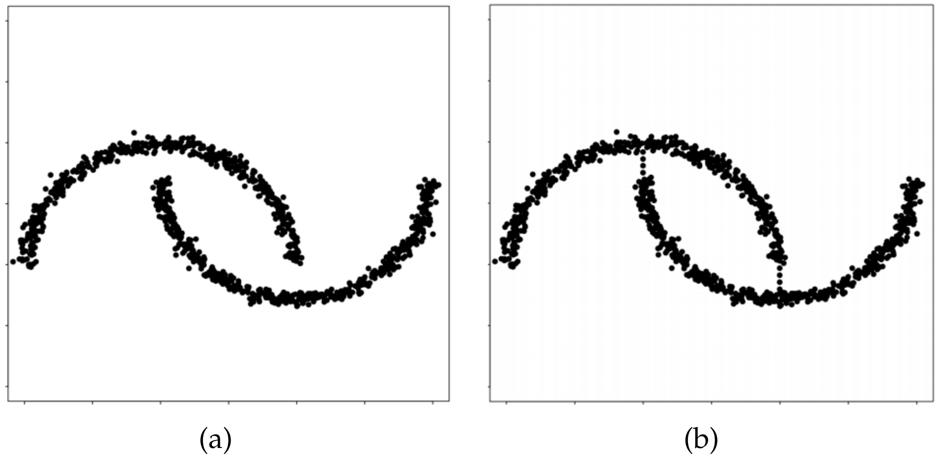


Figure 1: (a) Standard moon crescent distribution. (b) Moon crescent distribution with data points in adjoining area.

Solution:
Single linkage clustering (using minimum distance metric) can separate the two moons in Figure 1a as it tends to form spectrum or chain (straight or curved) clusters. If the points of a cluster are chained, the minimum separation between two points within the cluster would be smaller than that between two separate chains.
The complete linkage clustering (using maximum distance metric) or clustering using average distance metric cannot separate the moons in Figure 1a which tend to form compact circles and compact close-knit collectives respectively. The maximum separation (or the average separation) between two points within a cluster is not necessarily less than between two clusters in this case since the moons are formed by long chains (and not compact ones), hence they won't work well.

For Figure 1b, none of the linkage metrics given will work. The complete and average linkage do not work for the same reasons as described above. Now, for the single linkage case, we see that here the moons are no longer separated from each other. Single linkage doesn't care about the number of points separating the two moons as is visually evident, but only the minimum distance between two points, hence the two chains are now joined from the point of view of single linkage clustering, hence not guaranteed to be separated by the single linkage clustering as intended.

(c) (3 points) Consider the distance matrix in Table 1. Show the hierarchy of clusters created by the minimum distance hierarchical clustering algorithm, along with the intermediate steps. Finally, draw the dendrogram with edge lengths indicated.
(Note: You can draw the dendrogram on paper and upload the screenshot.)

Table 1: Distance between nodes

	A	B	C	D	E
A	0	0.73	6.65	4.61	5.24
B	0.73	0	4.95	2.90	3.45
C	6.65	4.95	0	2.24	1.41
D	4.61	2.90	2.24	0	1
E	5.24	3.45	1.41	1	0

Solution:

STEP - 1

Clusters A and B have the minimum distance between them.

	{A,B}	C	D	E
{A,B}	0	4.95	2.90	3.45
C	4.95	0	2.24	1.41
D	2.90	2.24	0	1
E	3.45	1.41	1	0

Let $d(X, Y)$ be the minimum distance between clusters X and Y. $d(X, X) = 0$ trivially. Also the matrix is symmetric, $sod(X, Y) = d(Y, X)$.

Cluster distances update : $d(\{A, B\}, C) = min(6.65, 4.95)$, $d(\{A, B\}, D) = min(4.61, 2.90)$ and $d(\{A, B\}, E) = min(3.45, 1.41)$.

Height in dendrogram from the base to the node formed (X) : $h_1 = \frac{0.73}{2} = 0.365$

STEP - 2

Clusters D and E have the minimum distance between them.

	{A,B}	C	{D,E}
{A,B}	0	4.95	2.90
C	4.95	0	1.41
{D,E}	2.90	1.41	0

Cluster distances update : $d(\{D, E\}, \{A, B\}) = min(2.90, 3.45)$ and $d(\{D, E\}, C) = min(2.24, 1.41)$.

Height in dendrogram from the base to the node formed (Y) : $h_3 = \frac{1}{2} = 0.5$

STEP - 3

	{A,B}	{C,{D,E}}
{A,B}	0	2.90
{C,{D,E}}	2.90	0

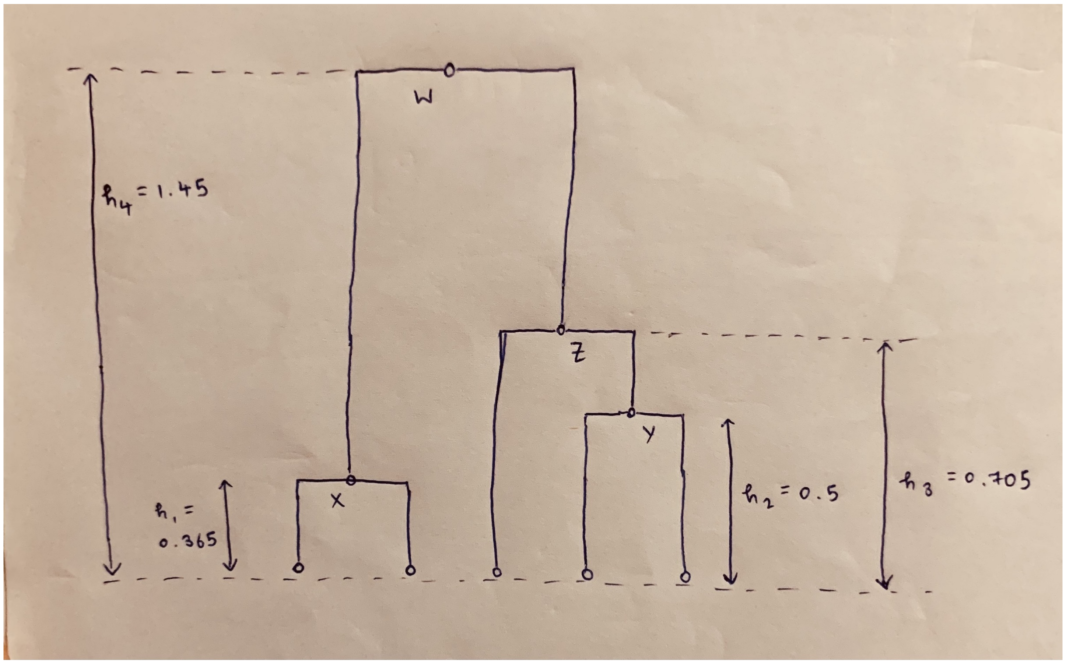
Cluster distances update : $d(\{A,B\},\{C,\{D,E\}\}) = min(4.95, 2.90)$.
Height in dendrogram from the base to the node formed (Z) : $h_3 = \frac{1.41}{2} = 0.705$

STEP - 4

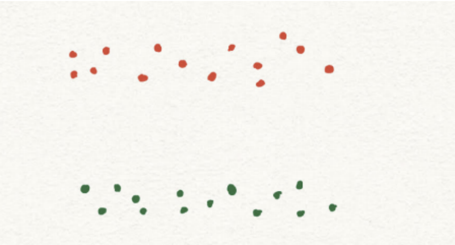
	{{A,B},{C,{D,E}}}
{{A,B},{C,{D,E}}}	0

Height in dendrogram from the base to the final node formed (*W*) : $h_4 = \frac{2.90}{2} = 1.45$
Final clustering: $\Big\{\{A, B\}, \{C, \{D, E\}\}\Big\}$

The result drawn as a dendrogram looks like :



(d) (2 points) Which distance metric (minimum, maximum and average) is more likely to generate following results given in Figure 2a for $K = 2$? Why?



(a)

Figure 2: Result produced for $K = 2$ clusters. Red points belong to one cluster and green to the other.

Solution:
The two clusters comprise of a long (almost straight) chain. Hence, single linkage clustering (using minimum distance metric) is more likely to result in the separation given in Figure 2a, as it tends to form clusters that are straight or curvilinear chains, which can be long.
Complete linkage clustering (using maximum distance metric) and UPGMA (using average distance metric) on the other hand, are less likely to result in this configuration shown in Figure 2a because they tend to form clusters that are compact (circular or close-knit ones respectively) and do not favour long or chained clusters.

4. (10 points) [CUTTING SPECTRAL APART] One of the several ways to express a given dataset is by using a *graph*. Each of the N datapoints in the dataset can be thought of as a vertex/node in a graph and any two datapoints can be connected in the graph with an edge whose non-negative weight W_{ij} indicates the similarity between the ith and jth datapoints. We will look at methods to partition this graph into two clusters, especially one that gained early prominence in computer vision. These methods can be recursively applied to partition the graph into any required number of clusters.

   (a) (1 point) A graph cut is a technique that separates a given graph into two disjoint sets of vertices and the degree of similarity (formally called the *cut cost*) between the two sets is given by the sum of weights of the edges between the sets (i.e., edges whose two endpoint nodes lie in different sides of the partition). The obvious method to separate the data into two is by choosing a partition that has the minimum cut cost. What do you think is/are the drawback(s) of this method? (Hint: Think about the sizes of the two sets in the partition.)

Solution:
Say, we partition n points into two disjoint sets, with x and $n - x$ nodes respectively.
The number of edges between the two sets is $x(n - x)$, which is maximum when the two sets are equal in size (i.e) $x = \frac{n}{2}$ and the number of edges is $\frac{n^2}{4}$ in this case. Hence, if we separate the data (or the corresponding indices of the data points) into two sets U and V,

by choosing a partition which has minimum cut cost, where cut cost is given by :

$$\sum_{\substack{u\in U\\ v\in V}} W_{uv}$$

the method will most likely not suggest separating the data into equal sets, even when it needs to be. Let us consider an example to verify this.
Suppose say we use $\exp(-d(x_i, x_j))$ (where $d(x_i, x_j)$ is the euclidean distance between points x_i and x_j) as the measure of similarity. Since $d(x_i, x_j) \geqslant 0$ always, $0 < W_{x_i x_j} \leqslant 1$. Now if there are two sets of 2D points with equal elements (say, $= \frac{n}{2}$) lying in different regions, hence forming 2 clusters is quite obvious by looking at it itself.
If we split the points into two sets of 1 and $n-1$ points, the cut cost will be atmost $n-1$, since $W_{x_i x_j} \leqslant 1$.
If we split the points into two sets U and V of $\frac{n}{2}$ and $\frac{n}{2}$ as it should be for the problem, the cut cost will $\geqslant \frac{n^2}{4} W_{min}$, where W_{min} is $\min_{\substack{u\in U\\ v\in V}} W_{uv}$ and will correspond to the maximum separation between nodes in the two sets.
For obtaining the intended solution using the min cut cost method, $n-1$ should be $> \frac{n^2}{4} W_{min} \implies W_{min} < 4\left(1 - \frac{1}{n}\right)\frac{1}{n}$. Thus there is a threshold for the maximum distance between two points from the two sets, above which only the min cut cost method will lead to the true solution of splitting the points into two equal clusters. And, this threshold for maximum distance also increases with the number of points, because RHS for W_{min} decreases with increase in n.
Hence, clearly the min cut cost proposed in this question favours splitting the points into two skewed sets, with number of elements in one much greater than in the other, even in the case where it needs to form two nearly equal clusters.

(b) (2 points) Due to the above drawback(s), we use a variation of the min cut method called normalized cut to partition the graph into two. The problem of finding the minimum-cost normalized cut can be reduced to this problem:

$$min_y \frac{y^T(D-W)y}{y^T D y} \text{ subject to } y \in \{1, -c\}^N \text{ and } y^T D\mathbf{1} = 0$$

where y_i takes one of the two discrete values $\{1, -c\}$ to indicate which side of the cut/partition the ith datapoint belongs to, W is the symmetric $N \times N$ similarity (non-negative edge-weights) matrix, D is a di agonal matrix called the degree matrix with $d_{ii} = \Sigma_j W_{ij}$, and $\mathbf{1}$ is a vector whose entries are all ones. This expression (not including the constraints) is called the *Generalized Rayleigh's Quotient* (GRQ). The matrix in the numerator, $D - W$ is the called the Laplacian Matrix, denoted by L. Prove that the Laplacian matrix is a singular matrix.

Solution:

Laplacian $L = D - W$. W is given to be a symmetric matrix with non negative real elements. D is a diagonal matrix with real elements.

Hence $L = D - W$ is symmetric and real $\implies$ self-adjoint, and by real spectrum theorem is diagonalizable wrt an eigenvector basis.

If an operator (square matrix) A has 0 as an eigenvalue, then $\mathrm{nullity}(A) > 0$ which implies matrix A is non-invertible (i.e) singular.

Clearly, L is a square matrix (operator). Consider $L\mathbf{1}$:

$$L\mathbf{1} = (D - W)\mathbf{1} = \begin{bmatrix} \sum_j (D_{1j} - W_{1j}) \\ \vdots \\ \sum_j (D_{ij} - W_{ij}) \\ \vdots \end{bmatrix} = \begin{bmatrix} D_{11} - \sum_j W_{1j} \\ \vdots \\ D_{ii} - \sum_j W_{ij} \\ \vdots \end{bmatrix} = \begin{bmatrix} \sum_j W_{1j} - \sum_j W_{1j} \\ \vdots \\ \sum_j W_{ij} - \sum_j W_{ij} \\ \vdots \end{bmatrix} = \mathbf{0}$$

Hence, $\mathbf{1}$ is an eigenvector of L with eigenvalue 0. Thus, L is a singular matrix.

(c) (3 points) As the above minimization problem is NP-hard with the two constraints, we first let go of both constraints. Then, the above GRQ can be minimized over $y \in \mathbb{R}^N$ by solving the generalized eigenvalue system $(D - W)y = \lambda D y$. Show that this equation can be expressed in the form $(ALA)z = \lambda z$, by expressing A, z in terms of D, W, y. Compute the eigenvector corresponding to the smallest eigenvalue of the matrix $M = ALA$.

Solution:

Let us assume D is positive definite (as it can be guaranteed by assuming that each node is connected to atleast one other node with a positive weight, that is true for similarities like $W_{ij} = \exp(-d(x_i, x_j)) \implies W_{ij} > 0$). Hence, it has a unique self-adjoint and positive-definite square root $\sqrt{(D)}$. Since, D is diagonal matrix, say is :

$$D = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \lambda_N \end{bmatrix}$$

the unique self-adjoint positive-definite square root is given by :

$$\sqrt{D} = \begin{bmatrix} \sqrt{\lambda_1} & 0 & \cdots & 0 \\ 0 & \sqrt{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \sqrt{\lambda_N} \end{bmatrix}$$

where $\lambda_1, \cdots, \lambda_N$ (all > 0) are the eigenvalues of D.

So, $D = \sqrt{D}\sqrt{D}$. Since, eigenvalues of D are all positive, so are the eigenvalues of $\sqrt{D}$ and so $\sqrt{D}$ is invertible.
Now,

$$\begin{aligned}
&(D - W)y = \lambda Dy \\
&\implies (D - W)Iy = \lambda Dy \qquad \text{(I is identity matrix)} \\
&\implies (D - W)\left((\sqrt{D})^{-1}\sqrt{D}\right)y = \lambda Dy \\
&\implies (D - W)(\sqrt{D})^{-1}\left(\sqrt{D}y\right) = \lambda\sqrt{D}\sqrt{D}y \\
&\implies \left((\sqrt{D})^{-1}(D - W)(\sqrt{D})^{-1}\right)\left(\sqrt{D}y\right) = \lambda\left(\sqrt{D}y\right) \qquad \text{(premultiply by } (\sqrt{D})^{-1}\text{)}
\end{aligned}$$

The above brings the equation to the form $(ALA)z = \lambda z$, where $A = (\sqrt{D})^{-1}$, $L = D - W$, $M = (\sqrt{D})^{-1}(D - W)(\sqrt{D})^{-1}$ and $z = \sqrt{D}y$.
$M = ALA$ is a self-adjoint operator, as it symmetric (because $L = D - W$ is symmetric and A is diagonal matrix, so $(ALA)^T = A^T L^T A^T = ALA$), so it's eigenvalues are real. Also, we saw in spectral clustering that $L = D - W$ is positive semi definite hence $\langle Lx, x\rangle \geqslant 0 \quad \forall x$. Put $x = Ay$, $\langle LAy, y\rangle \geqslant 0 \quad \forall y$. Since A is diagonal matrix, $A^T = A$, hence $\langle ALAy, y\rangle \geqslant 0 \quad \forall y$.
Hence $M = ALA$ is also positive semi definite, and the eigenvalues are $\geqslant 0$.
So, the smallest eigenvalue is **zero**. We saw in **part b** that the eigenvector corresponding to eigenvalue 0 for $D - W$ is $\mathbf{1}$.
Hence, $(D - W)y = \lambda y$ reduces to $(D - W)\mathbf{1} = 0$, when $y = \mathbf{1}$ and λ becomes 0. Now the equivalent equation $(ALA)z = \lambda z$ should also satisfy it, hence $(ALA)z = 0$ when we put $\lambda = 0$ and $y = \mathbf{1} \implies z = \sqrt{D}y = \sqrt{D}\mathbf{1} = \sqrt{D}$.
Therefore, $\sqrt{D}$ is the eigenvector of $M = ALA$ corresponding to the smallest eigenvalue 0.

(d) (4 points) Now, let's bring back the constraint that $y^T D\mathbf{1} = 0$ (the constraint that y takes only two discrete values can remain relaxed as a final real-valued solution $\{y_i\}$ can be clustered using 2-means for instance to identify the desired partition). Prove that the GRQ above (subject to $y^T D\mathbf{1} = 0$) is minimized when y is the eigenvector corresponding to the second smallest eigenvalue of the generalized eigenvalue system $(D - W)y = \lambda Dy$.
(Hint: You can use the following fact. Let A be a real symmetric matrix. Under the constraint that x is orthogonal to the $(j-1)$ eigenvectors corresponding to the $(j-1)$ smallest eigenvalues of A, the Rayleigh's quotient $\frac{x^T Ax}{x^T x}$ is minimized when x is the eigenvector corresponding to the j^{th} smallest eigenvalue.)

Solution:

$\mathrm{D} = \sqrt{\mathrm{D}}\sqrt{\mathrm{D}}$ from **part c**. Now,

$$\begin{aligned} y^{\mathsf{T}}(\mathrm{D}-W)y &= y^{\mathsf{T}}\sqrt{\mathrm{D}}(\sqrt{\mathrm{D}})^{-1}(\mathrm{D}-W)(\sqrt{\mathrm{D}})^{-1}\sqrt{\mathrm{D}}y \\ &= \left(\sqrt{\mathrm{D}}y\right)^{\mathsf{T}}(\sqrt{\mathrm{D}})^{-1}(\mathrm{D}-W)(\sqrt{\mathrm{D}})^{-1}\left(\sqrt{\mathrm{D}}y\right) \\ &= z^{\mathsf{T}}Mz \end{aligned}$$

and

$$y^{\mathsf{T}}\mathrm{D}y = y^{\mathsf{T}}\sqrt{\mathrm{D}}\sqrt{\mathrm{D}}y = \left(\sqrt{\mathrm{D}}y\right)^{\mathsf{T}}\left(\sqrt{\mathrm{D}}y\right) = z^{\mathsf{T}}z$$

where M and z are as defined in **part c**.
Hence, the GRQ :

$$min_y \frac{y^{\mathsf{T}}(\mathrm{D}-W)y}{y^{\mathsf{T}}\mathrm{D}y} \equiv min_z \frac{z^{\mathsf{T}}Mz}{z^{\mathsf{T}}z}$$

$$\begin{gathered} \text{The constraint } y^{\mathsf{T}}\mathrm{D}\mathbf{1} = 0 \\ \equiv \\ y^{\mathsf{T}}\sqrt{\mathrm{D}}\sqrt{\mathrm{D}}\,\mathbf{1} = 0 \\ \equiv \\ \left(\sqrt{\mathrm{D}}y\right)^{\mathsf{T}}\sqrt{\mathrm{D}}\mathbf{1} = 0 \\ \equiv \\ z^{\mathsf{T}}u_1 = 0 \end{gathered}$$

where $u_1 = \sqrt{\mathrm{D}}\,\mathbf{1}$ is the eigenvector of $M = (\sqrt{\mathrm{D}})^{-1}(\mathrm{D}-W)(\sqrt{\mathrm{D}})^{-1}$ corresponding to the eigenvalue 0, as we saw in **part c**.
Thus the objective translates to :

$$min_z \frac{z^{\mathsf{T}}Mz}{z^{\mathsf{T}}z} \quad \text{subject to } z^{\mathsf{T}}u_1 = 0$$

Here, M is real and symmetric because :
$\left((\sqrt{\mathrm{D}})^{-1}(\mathrm{D}-W)(\sqrt{\mathrm{D}})^{-1}\right)^{\mathsf{T}} = (\sqrt{\mathrm{D}})^{-1}(\mathrm{D}-W)^{\mathsf{T}}(\sqrt{\mathrm{D}})^{-1} = (\sqrt{\mathrm{D}})^{-1}(\mathrm{D}-W)(\sqrt{\mathrm{D}})^{-1}$.
So, we have to minimize $\frac{z^{\mathsf{T}}Mz}{z^{\mathsf{T}}z}$ wrt z, under the constraint that z is orthogonal the (1st) eigenvector of M corresponding to the smallest eigenvalue 0. And, the solution to this problem is z being the eigenvector corresponding to the 2nd smallest eigenvalue of M (from the **fact** given).
Hence, the solution is the eigenvector corresponding to the second smallest eigenvalue of the eigenvalue system $My = \lambda y$, which is equivalent to the eigenvalue system $(\mathrm{D}-W)y = \lambda \mathrm{D}y$ as proved in **part c**.

5. (10 points) [LIFE IN LOWER DIMENSIONS...] You are provided with a dataset of 1797 images in a folder here - each image is 8x8 pixels and provided as a feature vector of length 64. You will try your hands at transforming this dataset to a lower-dimensional space, and clustering the images in this reduced space.

   Please use the template .ipynb file in the same folder to prepare your solution. Provide your results/answers in the pdf file you upload to GradeScope, and submit your code separately in this moodle link. The code submitted should be a rollno.zip file containing two files: rollno.ipynb file (including your code as well as the exact same results/plots uploaded to Gradescope) and the associated rollno.py file.

   Write the code from scratch for both PCA and clustering. The only exception is the computation of eigenvalues and eigenvectors for which you could use the numpy in-bulit function.

   (a) (3 points) Run PCA algorithm on the given dataset. Plot the cumulative percentage variance explained by the principal components. Report the number of principal components that contribute to 90% of the variance in the dataset.

Solution:

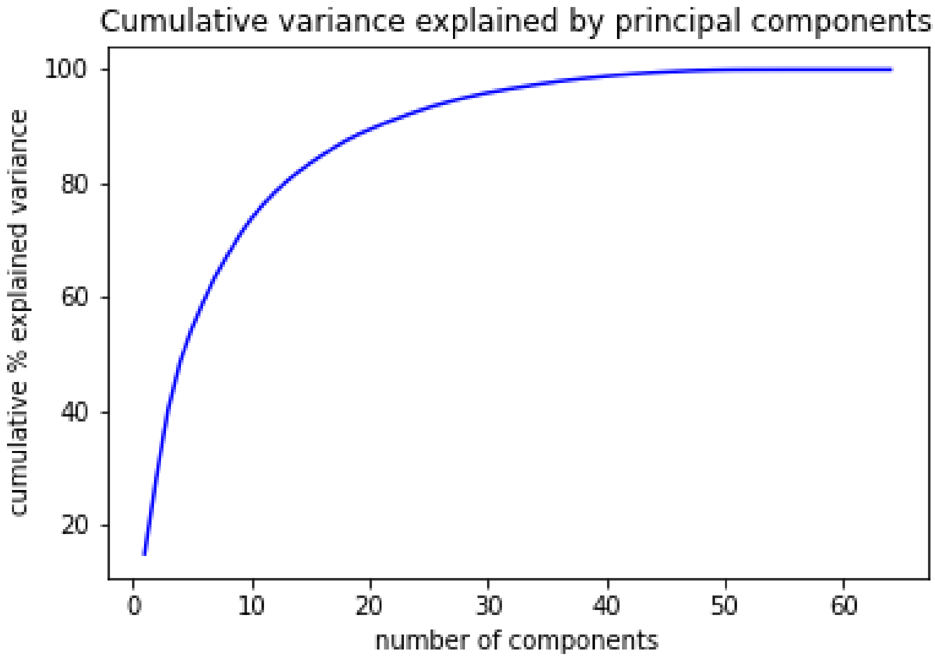


Figure 3: plotting cumulative variance explained for $1 - 64$ components

The first **21** components contribute 90.3199% (> 90%) of the variance

(b) (3 points) Perform reconstruction of data using the dimensionality-reduced data considering the number of dimensions [2,4,8,16]. Report the Mean Square Error (MSE) between the original data and reconstructed data, and interpret the optimal dimension $\widehat{\mathrm{d}}$ based on the MSE values.

Solution:

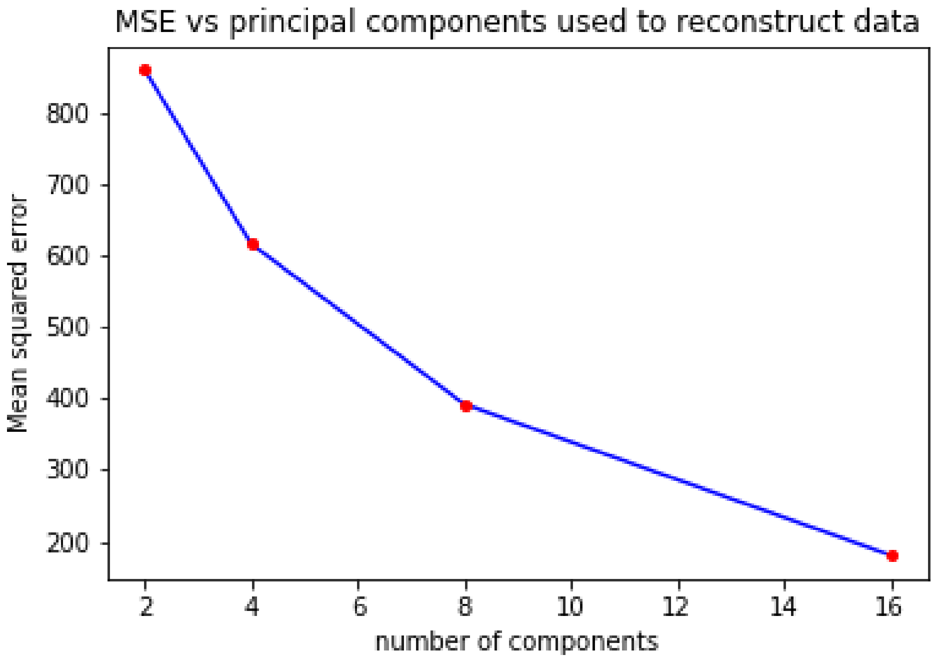


Figure 4: plotting MSE when reconstructed using $[2, 4, 8, 10]$ components

The MSE between original data and data reconstructed :

- using **2** components : 858.9448
- using **4** components : 616.1911
- using **8** components : 391.7947
- using **16** components : 180.9397

Optimal choice of $\hat{\mathrm{d}}$:
Out of the list $[2, 4, 8, 16]$ given, we choose $\hat{\mathrm{d}} = 16$, since it has the least MSE of 180.9397 and a variance explained of 84.9402% (close to 85%), hence retaining the individuality of the data to a good extent and reconstructing original data without much loss.

(c) (3 points) Apply K-means clustering on the reduced dataset from last subpart (b) (i.e., the $\mathbb{R}^{64}$ to $\mathbb{R}^{\widehat{d}}$ reduced dataset; pick the initial k points as cluster centers during initialization). Report the optimal choice of K you have made from the set [1...15]. Which method did you choose to find the optimum number of clusters? And explain briefly why you chose that method.

Also, show the 2D scatter plot (consider only the first two dimensions of optimal $\widehat{d}$) of the datapoints based on the cluster predicted by K-means (use different color for each cluster).

Solution:
We choose the optimal value of K (number of clusters) as **9**.
For choosing optimal value of K, we calculate the **WSS** (within-cluster sum of square error), which is the sum of squared error of each point from it's predicted cluster center for different values of K and plot it. It is basically the objective function (distortion measure) that we maximize through the iterative algorithm in K-Means clustering. We choose the the elbow point (where the plot bends from high slope to low slope region) as the optimal value for K. The intuition behind this is as follows:

- The WSS will clearly reduce with increase in K as the model with more number of clusters can perform as good as the one with lesser number of clusters by simply not assigning certain clusters to any points.
- But having more number of clusters makes it harder to analyse or observe patterns in the data.
- Just after the elbow point, after which the WSS starts reducing very slowly, we get diminished returns which is not worth the additional cost, almost similar to overfitting.

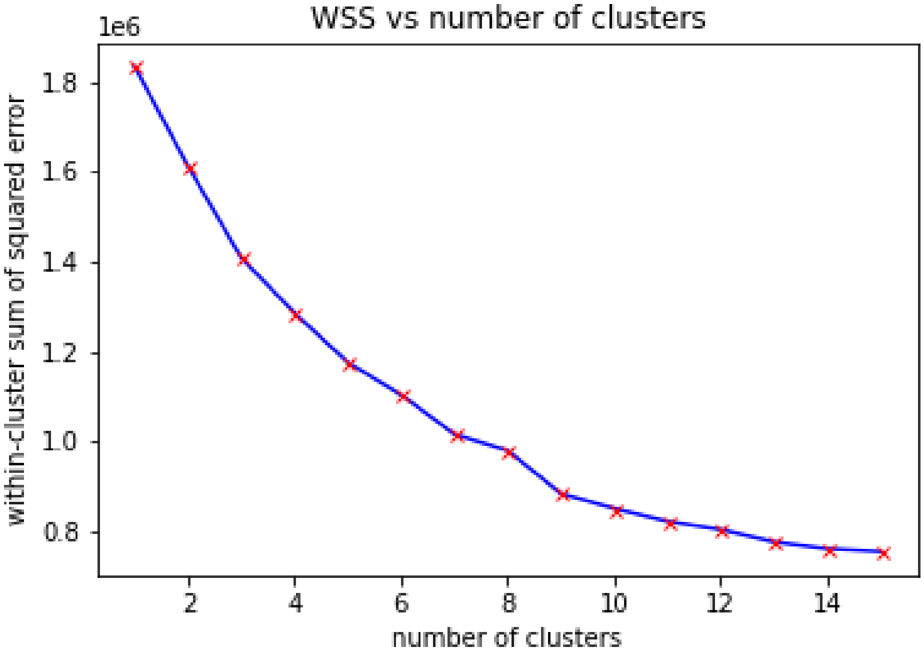


Figure 5: plotting WSS score for different values of K (number of clusters)

Though we can see that the elbow point occurs at $K = 9$ from the WSS plot, it is not very obvious, as there seems to be quite a few number of choices. To re-emphasize and verify the most optimal choice of K we then plot the **average silhouette value** for all the datapoints. The silhouette value is a measure of how similar a point is to its own cluster (cohesion) compared to other clusters (separation) and it ranges from -1 to $+1$. A high silhouette value indicates that the point is well matched to its own cluster and poorly matched to neighboring clusters.

From the silhouette value plot, it is clear that the maximum value (peak) occurs at $K = 9$, which helps us to zero in on the elbow point and also serves as a sanity check to the choice.

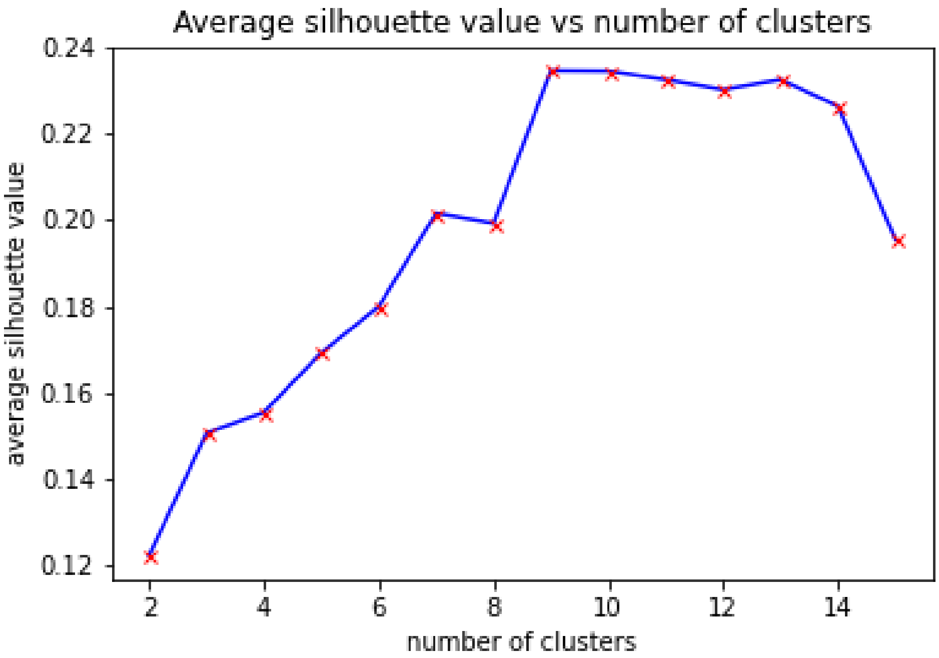


Figure 6: plotting average silhouette value for different values of K (number of clusters)

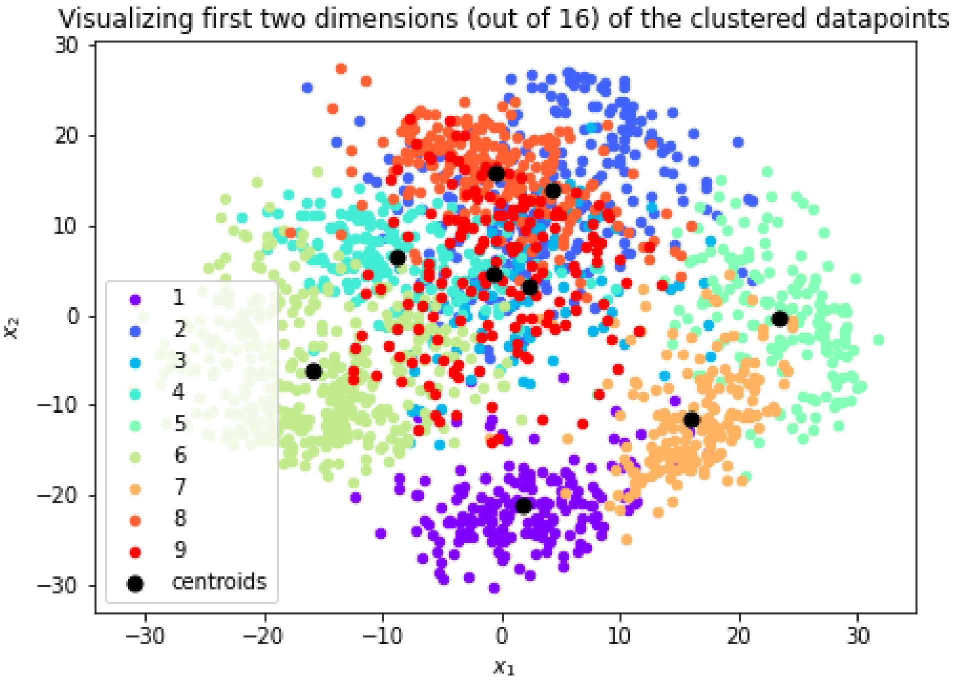


Figure 7: reducing data to $\hat{d} = 16$ dimensions using PCA, clustering it using K-Means and visualizing the first 2 dimensions of the datapoints

(d) (1 point) Summarise and explain your observations from the above experiments. Is the PCA+K-means clustering consistent with how your brain would cluster the images?

Solution:
Observations regarding PCA :

- We expect the MSE to decrease (and the explained variance to increase) monotonically as number of components used to reconstruct the data is increased, and that is exactly what the MSE plot and the explained variance plot also display. Also, the rate of decrease in MSE (or increase in explained variance) slows down as number of components is increased as in evident from the plots. All this can be explained by noting that the variance explained by representing data using M components (out of D components) is :

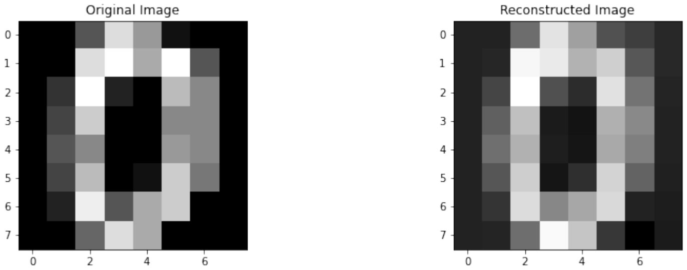
$$\sum_{i=1}^{M} \lambda_i$$

and the MSE for the case is :

$$\sum_{i=M+1}^{D} \lambda_i$$

where λ_i for $i = 1, 2, \cdots, D$ are the eigenvalues of $S = \frac{1}{N}X^TX$ sorted in non-increasing order. The MSE is monotonically decreasing (and explained variance increasing) with number of principal components becasue all eigenvalues of S are positive, since it is a positive semi-definite matrix. Alse, the rate of decrease in MSE (or increase in explained variance) becomes slower because the eigenvalues that occur later are smaller in magnitude.

- Out of the list $[2, 4, 8, 16]$ given, we choose $\hat{d} = 16$, since it has the least MSE of 180.9397 and a variance explained of 84.9402% (close to 85%), hence retaining the individuality of the data to a good extent and reconstructing original data without much loss. We can observe this by comparing one sample from the original and corresponding reconstructed image and notice how closely the reconstructed one resembles the original in Figure 8.



(a)

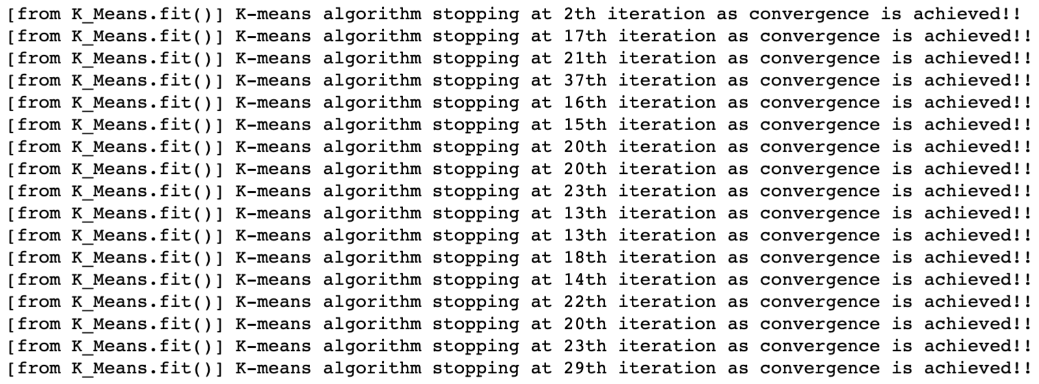
(b)

Figure 8: (a) Original Image (b) Image reconstructed using $\hat{d} = 16$ principal components

Observations regarding K-Means:

- We have set a default maximum number of iterations the K-Means will run to 100 and defined convergence as achieved when the $r_i^{(n)}$ matrix (storing 1 when $x^{(i)}$ is assigned to cluster with centroid $m^{(n)}$), doesn't change after the E-step and M-step. We however observe (from Figure 9) that the K-Means algorithm took much lesser iterations than 100 to converge (maximum being 37, on an average about 20), assuring the quickness of the algorithm.

- The only possibility where randomness can creep in during PCA+K-Means is during sorting of eigenvalues when the eigenvectors corresponding to equal eigenvalues can be placed in any order (the centroid initialisations are fixed). But np.linalg.eigh() which returns the eigenvalues (and corresponding eigenvectors) in non-decreasing order seems to do it in a deterministic way, as there was no change in the metric values or scores or even the clustering scatter plot, how many ever times I ran the program. Hence, as a whole there is no randomness involved, and therefore exists slight possibility that the clustering is unable to come out of a local minima solution if stuck there.



```
[from K_Means.fit()] K-means algorithm stopping at 2th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 17th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 21th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 37th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 16th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 15th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 20th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 20th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 23th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 13th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 13th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 18th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 14th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 22th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 20th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 23th iteration as convergence is achieved!!
[from K_Means.fit()] K-means algorithm stopping at 29th iteration as convergence is achieved!!
```

Figure 9: screenshot showing K-Means converging within 100 iterations when running for $K = 1, 2, \cdots, 15$ on the 16-dimensional points from PCA

Comments on PCA+K-Means :

We can see in the Figure 7 the K-Means clustering for the points reduced to $\hat{d} = 16$ dimensions using PCA. The plot however does not seem to match what we would have done/expected manually, as K-Means is known to form convex clusters. This is because we only visualize the first 2 dimensions out of the 16 dimensions, hence the clusters which seem to overlap in 2-dimensions may actually be separated out when considered the other dimensions.

However, to confirm that the K-Means algorithm is indeed working correctly, we reduce the datapoints to 2 dimensions using PCA, and plot the result of K-Means clustering again as shown in Figure 10. This time we see that the clustering exactly matches what we have in mind and gives us convex sets as one would expect.

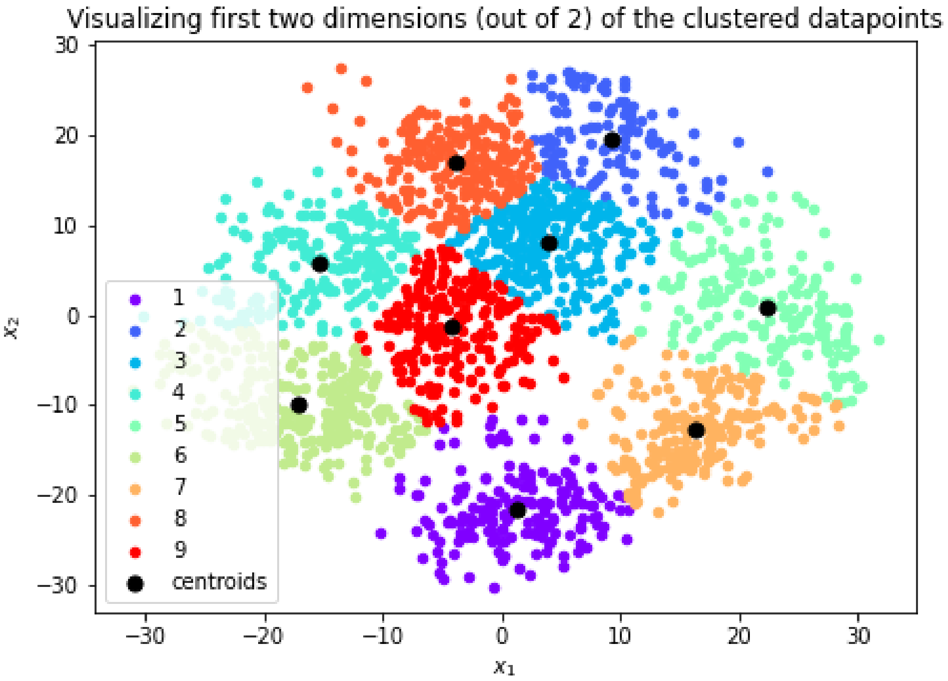


Figure 10: reducing data to 2 dimensions using PCA, clustering it using K-Means and visualizing the datapoints