# CSE/ISyE 6740-A Homework 1

Name: Archit Sengupta
GT ID: 904013900

September 2024

## 1 Question 1: Linear Algebra

### 1.1 A

**Proof 1**

$$E[x] = E_y[E_x[x|y]]$$

conditional expectation:

$$E[x|y] = \sum_x x\, P(x|y)$$

take the expectation with respect to $y$:

$$E_y[E_x[x|y]] = \sum_y E_x[x|y]\, P(y)$$

$$E_y[E_x[x|y]] = \sum_y \left( \sum_x x\, P(x|y) \right) P(y)$$

the expression becomes:

$$E_y[E_x[x|y]] = \sum_x x \left( \sum_y P(x|y)\, P(y) \right) = \sum_x x\, P(x)$$

Therefore:

$$E_y[E_x[x|y]] = E[x]$$

**Proof 2**

$$\mathrm{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$$

using the law of total expectation, we have:

$$\mathbb{E}[X^2] = \mathbb{E}[\mathbb{E}[X^2 \mid Y]] = \mathbb{E}[\mathrm{Var}(X \mid Y) + (\mathbb{E}[X \mid Y])^2]$$

Next, we rewrite the conditional second moment of $X$ in terms of its variance and first moment, and apply the law of total expectation:

$$\mathbb{E}[X^2] - (\mathbb{E}[X])^2 = \mathbb{E}[\mathrm{Var}(X \mid Y) + (\mathbb{E}[X \mid Y])^2] - (\mathbb{E}[\mathbb{E}[X \mid Y]])^2$$

$$= \mathbb{E}[\mathrm{Var}(X \mid Y)] + \mathbb{E}[(\mathbb{E}[X \mid Y])^2] - (\mathbb{E}[\mathbb{E}[X \mid Y]])^2$$

the terms in the second set of parentheses as the variance of the conditional expectation $\mathbb{E}[X \mid Y]$:

$$= \mathbb{E}[\mathrm{Var}(X \mid Y)] + \mathrm{Var}(\mathbb{E}[X \mid Y])$$

Thus, proven

## 1.2 B

Counterexample:

Consider three binary random variables $X_1, X_2, X_3$, each taking values in $\{0, 1\}$. Define their joint distribution as follows:

$$P(X_1 = 0, X_2 = 0, X_3 = 0) = P(X_1 = 1, X_2 = 1, X_3 = 0) = P(X_1 = 1, X_2 = 0, X_3 = 1)$$

$$= P(X_1 = 0, X_2 = 1, X_3 = 1) = \frac{1}{4}$$

This defines the following marginal distributions:

$$P(X_1 = X_2) = P(X_2 = X_3) = P(X_1 = X_3) = \frac{1}{2}$$

Thus, $X_1, X_2, X_3$ are pairwise independent. However, their joint probability:

$$P(X_1 = X_2 = X_3 = 1) = 0 \neq P(X_1 = 1)P(X_2 = 1)P(X_3 = 1) = \frac{1}{8}$$

This shows that they are not mutually independent.

## 1.3 C

**Part 1**
For $A^+A$:

$(A^+A)(A^+A) = A^+(AA^+A) = A^+A$ (using Moore-Penrose condition (a): $AA^+A = A$)

Thus, $A^+A$ is idempotent.

For $AA^+$:

$(AA^+)(AA^+) = A(A^+AA^+) = AA^+$ (using Moore-Penrose condition (b): $A^+AA^+ = A^+$)

Thus, $AA^+$ is idempotent.

**Part 2**
We claim that $A^+ = (A^TA)^{-1}A^T$ satisfies the Moore-Penrose conditions.
- **Condition (a)**

$$AA^+A = A((A^TA)^{-1}A^T)A = AI_n = A$$

Thus, $AA^+A = A$.
- **Condition (b)**

$$A^+AA^+ = (A^TA)^{-1}A^TA(A^TA)^{-1}A^T = (A^TA)^{-1}A^T = A^+$$

Thus, $A^+AA^+ = A^+$.
- **Condition (c)** For real matrices $A$, the conjugate transpose $(AA^+)^*$ is simply the regular transpose:

$$(AA^+)^* = \left(A\left(A^TA\right)^{-1}A^T\right)^T$$

Applying the transpose to this product:

$$(AA^+)^* = \left(A^T\right)^T\left(\left(A^TA\right)^{-1}\right)^TA^T$$

Since $\left(A^TA\right)^{-1}$ is symmetric (because $A^TA$ is symmetric), and the transpose of $A^T$ is $A$, we find:

$$(AA^+)^* = A\left(A^TA\right)^{-1}A^T = AA^+$$

Hence, $AA^+$ is symmetric.
- **Condition (d)** For real matrices $A$, the conjugate transpose $(A^+A)^*$ is simply the regular transpose:

$$(A^+A)^T = ((A^TA)^{-1}A^TA)^T$$

$$\left((A^TA)^{-1}A^TA\right)^T = A^TA\left(\left(A^TA\right)^{-1}\right)^T$$

Since $A^TA$ is a symmetric matrix ($\left(\left(A^TA\right)^{-1}\right)^T = \left(A^TA\right)^{-1}$):

$$\left((A^+A)\right)^T = A^TA\left(\left(A^TA\right)^{-1}\right)$$

This simplifies to:

$$\left((A^+A)\right)^T = \left(A^TA\right)\left(\left(A^TA\right)^{-1}\right)$$

Since $\left(A^TA\right)\left(\left(A^TA\right)^{-1}\right) = I$

Hence, $A^+A$ is symmetric.

Thus, $A^+ = (A^TA)^{-1}A^T$ satisfies all four Moore-Penrose conditions, proving that this is the pseudoinverse of $A$.

To verify this is a left inverse:

$$A^+A = (A^TA)^{-1}A^TA = I_n$$

Thus, $A^+$ is a left inverse of $A$.

For the right inverse, note that $A \in \mathbb{R}^{m\times n}$ with $m > n$ implies that $A$ has no full row rank, so $A^+$ is not a right inverse. Thus, no right inverse exists in general for $A$

**Part 3**

$$A = UDV^T$$
$$A^+ = (V^T)^{-1}D^+U^{-1}$$

Since V and U are both orthonormal,

$$V^T = V^{-1}$$

and

$$U^T = U^{-1}$$

The pseudo-inverse $A^+$ is:

$$A^+ = VD^+U^T$$

where $D^+$ is the pseudo-inverse of $D$. If $D$ is:

$$D = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r, 0, \ldots, 0)$$

## 1.4 D

**Part 1**

$$A(x)A^{-1}(x) = I$$

Differentiate both sides with respect to $x$:

$$\frac{d}{dx}\left[A(x)A^{-1}(x)\right] = \frac{dI}{dx}$$

Apply the product rule:

$$\frac{dA(x)}{dx}A^{-1}(x) + A(x)\frac{dA^{-1}(x)}{dx} = 0$$

$$A(x)\frac{dA^{-1}(x)}{dx} = -\frac{dA(x)}{dx}A^{-1}(x)$$
$$\frac{dA^{-1}(x)}{dx} = -A^{-1}(x)\frac{dA(x)}{dx}A^{-1}(x)$$

**Part 2**

$$df = \frac{\partial f}{\partial x_1}dx_1 + \frac{\partial f}{\partial x_2}dx_2 + \cdots + \frac{\partial f}{\partial x_m}dx_m$$

In matrix form, this can be written as:

$$df = \nabla f(x)^T dx$$

Here, $\nabla f(x)$ is the Jacobian matrix of $f$, which is an $n \times m$ matrix containing the partial derivatives of $f$ with respect to each component of $x$:

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_m} \end{bmatrix}$$

Therefore, the differential of $f$ with respect to $x$ is:

$$\frac{df}{dx} = \nabla f(x)$$

**Part 3**

To compute the gradient $dg(f(x))$, where $x \in \mathbb{R}^m$, $f : \mathbb{R}^m \to \mathbb{R}^n$, and $g : \mathbb{R}^n \to \mathbb{R}^l$, you need to use the chain rule for gradients.

- $f(x)$ maps $\mathbb{R}^m$ to $\mathbb{R}^n$. Thus, $f$ has $n$ outputs.
- $g(y)$ maps $\mathbb{R}^n$ to $\mathbb{R}^l$. Thus, $g$ has $l$ outputs.

- Gradient of $f$ with respect to x: Jacobian matrix $J_f \in \mathbb{R}^{n \times m}$.
- Gradient of $g$ with respect to y: Jacobian matrix $J_g \in \mathbb{R}^{l \times n}$.

Chain Rule
apply the chain rule. The chain rule states that:

$$\frac{\partial(g \circ f)}{\partial x} = \frac{\partial g}{\partial f} \cdot \frac{\partial f}{\partial x}$$

Here, $\frac{\partial g}{\partial f}$ is the Jacobian of $g$ with respect to $f$, which is $J_g$. $\frac{\partial f}{\partial x}$ is the Jacobian of $f$ with respect to $x$, which is $J_f$. Thus:

$$\frac{\partial(g \circ f)}{\partial x} = J_g \cdot J_f$$

**Dimensionality**

- The Jacobian matrix $J_f$ has dimensions $n \times m$.
- The Jacobian matrix $J_g$ has dimensions $l \times n$.

Therefore, the resulting gradient of $g(f(x))$ with respect to $x$ will be a matrix of dimensions $l \times m$.
Summary
The chain rule applies here, giving us:

$$\frac{\partial(g \circ f)}{\partial x} = J_g \cdot J_f$$

with the dimensionality of the gradient being $l \times m$. This is why the chain rule works: it accounts for the transformation from $\mathbb{R}^m$ through $f$ to $\mathbb{R}^n$, and then through $g$ to $\mathbb{R}^l$.

# 2 Question 2: Maximum Likelihood

## 2.1 A

For a sample $x_1, x_2, \ldots, x_n$, the likelihood function is:

$$L(\lambda) = \prod_{i=1}^{n} f(x_i|\lambda) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i}$$

Simplify the product:

$$L(\lambda) = \lambda^n \prod_{i=1}^{n} e^{-\lambda x_i}$$

$$L(\lambda) = \lambda^n e^{-\lambda \sum_{i=1}^{n} x_i}$$

Take the natural logarithm to get the log-likelihood function:

$$\ell(\lambda) = \log L(\lambda) = \log\left(\lambda^n e^{-\lambda \sum_{i=1}^n x_i}\right)$$

Apply the properties of logarithms:

$$\ell(\lambda) = \log\left(\lambda^n\right) + \log\left(e^{-\lambda \sum_{i=1}^n x_i}\right)$$

$$\ell(\lambda) = n \log \lambda - \lambda \sum_{i=1}^n x_i$$

To find the maximum likelihood estimator, we take the derivative of the log-likelihood function with respect to $\lambda$ and set it to zero:

$$\frac{\partial \ell(\lambda)}{\partial \lambda} = \frac{n}{\lambda} - \sum_{i=1}^n x_i = 0$$

Solving for $\lambda$ gives:

$$\hat{\lambda} = \frac{n}{\sum_{i=1}^n x_i}$$

## 2.2 B

The likelihood function for a sample $x_1, x_2, \ldots, x_n$ is:

$$L(\theta) = \prod_{i=1}^n f(x_i|\theta) = \prod_{i=1}^n \theta x_0^\theta x_i^{-\theta-1}$$

Simplify :

$$L(\theta) = \theta^n x_0^{n\theta} \prod_{i=1}^n x_i^{-\theta-1}$$

$$L(\theta) = \theta^n x_0^{n\theta} \left(\prod_{i=1}^n x_i\right)^{-\theta-1}$$

Take the natural log to get the log-likelihood function:

$$\ell(\theta) = \log L(\theta) = \log\left(\theta^n x_0^{n\theta} \left(\prod_{i=1}^n x_i\right)^{-\theta-1}\right)$$

$$\ell(\theta) = n \log \theta + n\theta \log x_0 - (\theta+1) \log\left(\prod_{i=1}^n x_i\right)$$

$$\ell(\theta) = n \log \theta + n\theta \log x_0 - (\theta+1) \sum_{i=1}^n \log x_i$$

Differentiate with respect to $\theta$ and set to zero:

$$\frac{\partial \ell(\theta)}{\partial \theta} = \frac{n}{\theta} + n \log x_0 - \sum_{i=1}^{n} \log x_i = 0$$

$$\hat{\theta} = \frac{n}{\sum_{i=1}^{n} \log \frac{x_i}{x_0}}$$

### 2.3 C

The likelihood func for a sample $x_1, x_2, \ldots, x_n$ is:

$$L(\lambda) = \prod_{i=1}^{n} \frac{\lambda^{x_i} e^{-\lambda}}{x_i!}$$

Simplify :

$$L(\lambda) = \frac{\lambda^{\sum_{i=1}^{n} x_i} e^{-n\lambda}}{\prod_{i=1}^{n} x_i!}$$

Take the natural log:

$$\ell(\lambda) = \log L(\lambda) = \log \left( \frac{\lambda^{\sum_{i=1}^{n} x_i} e^{-n\lambda}}{\prod_{i=1}^{n} x_i!} \right)$$

$$\ell(\lambda) = \sum_{i=1}^{n} x_i \log \lambda - n\lambda - \log \left( \prod_{i=1}^{n} x_i! \right)$$

$$\ell(\lambda) = \left( \sum_{i=1}^{n} x_i \right) \log \lambda - n\lambda - \sum_{i=1}^{n} \log(x_i!)$$

Differentiate with respect to $\lambda$ and set to zero:

$$\frac{\partial \ell(\lambda)}{\partial \lambda} = \frac{\sum_{i=1}^{n} x_i}{\lambda} - n = 0$$

$$\lambda = \frac{1}{n} \sum_{i=1}^{n} x_i$$

## 3 Question 3: EigenVectors

### 3.1 Proof

**Objective:** We want to prove that for every sequence of vectors $x_1, \ldots, x_n$ such that

$$x_i \in \arg \max_{\|x\|=1,\, x^T x_j = 0 \text{ for } j<i} x^T M x,$$

each $x_i$ is an eigenvector of $M$.
We are given a matrix $M$ and vectors $x_1, x_2, \ldots, x_n$ such that: $\|x_i\| = 1$, $x_i^T x_j = 0$ for all $j < i$, $x_i$ maximizes $x^T M x$.

To solve the maximization we can use the Lagrangian

The Lagrangian is:

$$\mathcal{L}(x) = x^T M x - \lambda(x^T x - 1) - \sum_{j<i} \mu_j x^T x_j.$$

where $\lambda$ is the Lagrangian multiplier associated with $\|x_i\|$
Taking the derivative and setting it to zero:

$$\nabla_x \mathcal{L}(x) = 2Mx - 2\lambda x - \sum_{j<i} \mu_j x_j = 0,$$

which simplifies to:

$$Mx = \lambda x + \frac{1}{2} \sum_{j<i} \mu_j x_j.$$

Since $x_i^T x_j = 0$ for $j < i$, multiplying both sides by $x_j^T$ gives:

$$Mx_i x_j^T = \lambda x_i x_j^T + \frac{1}{2} \sum_{j<i} \mu_j x_j x_j^T$$

Therefore,

$$Mx_i = \lambda x_i.$$

Thus
Each $x_i$ is an eigenvector of $M$, completing the proof.

## 3.2 Algorithm

The basic idea is to iteratively find the eigenvector corresponding to the largest eigenvalue, then project onto the orthogonal space to find the subsequent eigenvectors.

Given a matrix $M \in \mathbb{R}^{n \times n}$, the following algorithm computes a set of orthogonal eigenvectors $x_1, x_2, \ldots, x_n$ of $M$.

1. **Input:** A symmetric matrix $M \in \mathbb{R}^{n \times n}$.
2. **Initialize:** Choose a random starting vector $x_1 \in \mathbb{R}^n$ such that $||x_1|| = 1$.
3. **Step 1: Compute the first eigenvector**
   (a) Maximize $x^T M x$ to find the dominant eigenvector.

(b) Iterate using power iteration:

$$x_1^{(k+1)} = \frac{Mx_1^{(k)}}{||Mx_1^{(k)}||}$$

until convergence.

(c) Normalize $x_1$ to unit length: $x_1 = \frac{x_1}{||x_1||}$.

4. **Step 2: Compute subsequent eigenvectors** $x_2, x_3, \ldots, x_n$

(a) For $i = 2, 3, \ldots, n$:

i. Choose a random vector $x_i \in \mathbb{R}^n$.

ii. Orthogonalize $x_i$ with respect to the previously computed vectors $x_1, x_2, \ldots, x_{i-1}$ using Gram-Schmidt:

$$x_i' = x_i - \sum_{j=1}^{i-1} (x_i^T x_j) x_j$$

iii. Normalize $x_i'$ to unit length: $x_i = \frac{x_i'}{||x_i'||}$.

iv. Maximize $x^T M x$ in the subspace orthogonal to $x_1, x_2, \ldots, x_{i-1}$ using power iteration.

# 4 Question 4: Clustering

## 4.1 A

partial derivative of $J$ with respect to $\mu_k$:

$$\frac{\partial J}{\partial \mu_k} = \frac{\partial}{\partial \mu_k} \left( \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2 \right).$$

The squared Euclidean distance is given by

$$\|x_n - \mu_k\|^2 = (x_n - \mu_k)^T (x_n - \mu_k).$$

Thus,

$$\frac{\partial}{\partial \mu_k} \|x_n - \mu_k\|^2 = -2(x_n - \mu_k).$$

Therefore,

$$\frac{\partial J}{\partial \mu_k} = \sum_{n=1}^{N} \frac{\partial}{\partial \mu_k} \left( r_{nk} \|x_n - \mu_k\|^2 \right) = \sum_{n=1}^{N} r_{nk} \left( -2(x_n - \mu_k) \right).$$

$$\frac{\partial J}{\partial \mu_k} = -2 \sum_{n=1}^{N} r_{nk} (x_n - \mu_k).$$

Setting the derivative to zero to find the minimum,

$$-2\sum_{n=1}^{N} r_{nk}(x_n - \mu_k) = 0.$$

$$x_n \sum_{n=1}^{N} r_{nk} = \mu_k \sum_{n=1}^{N} r_{nk}.$$

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}.$$

This shows that $\mu_k$ is the mean of the data points assigned to the $k$-th cluster.

## 4.2 B

K-means aims to minimize the objective function:

$$J = \sum_{i=1}^{n} \|x_i - c_{\pi_i}\|^2$$

where $x_i$ is the $i$-th data point, $c_{\pi_i}$ is the center of the cluster to which $x_i$ is assigned, and $\pi_i$ is the cluster assignment of $x_i$.

**1. Cluster Assignment Step:**

In the cluster assignment step, each data point $x_i$ is assigned to the cluster $j$ such that:

$$\pi_i = \arg \min_{j=1,2,\ldots,k} \|x_i - c_j\|^2$$

This step minimizes the distance between data points and their assigned cluster centers, thus decreasing the objective function $J$.

**2. Center Assignment Step:**

In the center assignment step, each cluster center $c_j$ is updated to:

$$c_j = \frac{1}{|\{i : \pi_i = j\}|} \sum_{i:\pi_i=j} x_i$$

$$= \arg\min_{c} \sum_{i:\pi_i=j} \|x_i - c_j\|^2$$

This step minimizes the sum of squared distances between data points and their cluster centers within each cluster, further decreasing the objective function $J$.

Proof of Convergence

Since the K-means algorithm is iterative and alternates between cluster assignment and center update, and each step decreases the objective function $J$, it is guaranteed to decrease or remain the same with each iteration.

$$J^{(t+1)} \leq J^{(t)}$$

where $J^{(t)}$ is the value of the objective function at iteration $t$. Thus, it will converge to a finite value.

## 4.3 C

The average linkage method is more likely to produce clusters similar to those found by K-means for the following reasons

- Average linkage calculates the average distance between all pairs of points from two clusters.This metric tends to balance the distances across all points, which is conceptually similar to the way K-means optimizes cluster assignments.
- K-means clustering minimizes the within-cluster variance, which is the average squared distance between each point and the centroid of its cluster. Mathematically, K-means aims to minimize:

  $$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

  where $\mu_k$ is the centroid of cluster $C_k$.
- Similarly average linkage also computes average distance between all pairs of points in the 2 clusters. Both methods aim to minimize intra-cluster distances.

  Thus, average linkage is the most likely to yield results that align with those produced by K-means clustering.

## 4.4 D

The time complexity of the traditional hierarchical clustering algorithm using single linkage is $O(n^3)$ and its space complexity is $O(n^2)$

An improved algorithm for hierarchical clustering, using single linkage, can be using a priority queue (min-heap) to manage the merging process of clusters more efficiently.

**Algorithm Description:**

1. Initialize a min-heap to store pairs of clusters and their distances.

2. Compute initial distances between all pairs of clusters and insert them into the heap.

3. Repeatedly extract the minimum distance pair from the heap and merge the clusters.

4. Update distances of the newly formed cluster with all other clusters and adjust the heap accordingly.

5. Continue until all points are clustered into a single cluster.

**Time Complexity:** In this algorithm, each insertion and extraction operation from the min-heap takes $O(\log n)$ time. Since there are $O(n^2)$ possible cluster pairs, the overall time complexity is $O(n^2 \log n)$.

**Space Complexity Analysis:** The space complexity remains $O(n^2)$ due to the need to store the distance matrix and the min-heap.

**Applicability to Other Linkage Criteria:** This algorithm is optimized for single linkage clustering. For complete linkage, a similar approach using a priority queue can be applied, but the distance calculation will change to consider the maximum distance between clusters. For average linkage, the algorithm can be adapted to compute the average distance between clusters, but it may require additional computations to maintain the average distances efficiently.

**Summary:** While the linkage-based algorithm with a min-heap improves the time complexity for single linkage clustering to $O(n^2 \log n)$, it still maintains a space complexity of $O(n^2)$. Adapting this algorithm to complete or average linkage is feasible but requires modifications in distance computations.

# 5 Question 5: Programming

The Kmedoids and Kmeans programs were implemented for image compression using clustering. A variety of configurations were used to test out the system in each case.

## 5.1 K-medoids implementation

- After initial medoid selection, pixels were assigned to a specific medoid based on the distance from each pixel to the medoid.

- The medoids are then updated. Each cluster is iterated over, and a sample of pixels is chosen from each cluster. Within each sample, a distance metric is calculated for each pixel from the rest of the pixels. The pixel with the least distance metric is chosen as the new medoid for the cluster.

- The optimal sample size appeared to be 5% of each cluster. At 10%, computation became very slow. On a more powerful system, sampling might not be necessary, which would significantly improve the quality of the generated image.

- The distance metric chosen was the average of the Manhattan distances between the pixels. Euclidean distances did not offer much benefit over Manhattan distances, so Manhattan was selected for performance reasons.
- After each update, the clusters are regenerated according to the same process as in step 1.
- There are multiple stopping criteria: an iteration limit and a convergence check based on the change in the total distance metric.
- I also tried other implementations such as PAM ( partition around medoids) without sampling, with granular cost functions, however these turned out to be too computationally intensive for my system.

## 5.2 Time Comparison for K-Means and K-Medoids

| k | K-medoids (s) | K-means (s) |
|---|---|---|
| 3 | 23.07 | 0.69 |
| 5 | 13.26 | 0.83 |
| 8 | 12.36 | 1.40 |
| 15 | 6.86 | 3.50 |
| 25 | 5.54 | 5.37 |

Table 1: Timing comparison between K-medoids and K-means clustering algorithms for different values of $k$.

- **Scalability with $k$:** K-medoids execution time decreases with increasing $k$, likely due to reduced iterations or fewer data points per cluster. Despite this, the K-medoids code remains significantly slower than K-means even after sampling
- **Efficiency:** K-means consistently outperforms K-medoids in terms of execution time. This efficiency is because to K-means' simpler iterative updates and direct cluster assignments.
- **Convergence:** The reduced execution time for K-medoids at higher $k$ shows faster convergence with more clusters.
- **Conclusion:** K-means is favored for its speed, especially with smaller $k$. As $k$ increases, while K-medoids becomes more efficient.

## 5.3 Output Quality Comparison

In all cases, images output by K-means were overwhelmingly better than K-Medoids. This is probably due to loss of data due to sampling.

## 5.4 Centroid/Medoid Initialization

2 Different systems were used for center initializaiton

- **Random** : This is the most common way to initialize centers. A random collection of $k$ pixels are chosen to be centroids
- **Unique centers**: For certain images with large sections of solid colors, we sometimes get multiple Centers with the same RGB values, this is because these colors are dominant. This leads to one of the centers remaining empty. It might be beneficial to select only unique pixels as the initial centers. In my tests I got mixed results . Some runs gave better quality images with unique selection, while some were better with random. This remains to be explored with larger datasets.

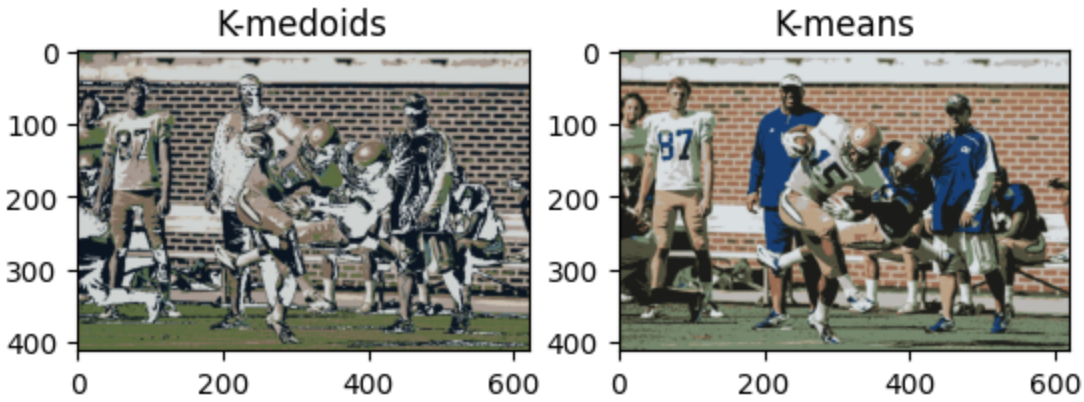
In [ ]:
```
import os
import imageio
import numpy as np
from matplotlib import pyplot as plt
import time

def distance_metric(cluster, pixel):
    x= np.linalg.norm(cluster - pixel,ord =1)/len(cluster)
    return x
def generate_clusters(pixels, medoids,order=2):
    distances = np.linalg.norm(pixels[:, np.newaxis] - medoids, axis=2,ord=o
    clusters = np.argmin(distances, axis=1)
    return clusters
def select_initial_medoids_unique(pixels, K):
    unique_pixels = np.unique(pixels, axis=0)  # Remove duplicate rows (pixe
    medoids = unique_pixels[np.random.choice(unique_pixels.shape[0], K, repl
    return medoids
def select_inital_medoids_random(pixels, K):
    return pixels[np.random.choice(pixels.shape[0], K, replace=False)]

def kmeans(pixels, K):

    """ Function to perform K-means clustering on the given pixels. """
    # Flatten the image to a 2D array of pixels
    pixels = pixels.reshape(-1, 3)

    # Randomly initialize the cluster centroids
    centroids = select_inital_medoids_random(pixels, K)
    iterations = 0

    while iterations<10:
        # Assign each pixel to the nearest cluster center
        clusters = generate_clusters(pixels, centroids)
        # Calculate new cluster centroids
        new_centroids= np.zeros(centroids.shape)
        for k in range(K):
            cluster = pixels[clusters == k]
            if len(cluster) == 0:
                continue
            new_centroids[k] = cluster.mean(axis=0)
        #new_centroids = np.array([pixels[clusters == k].mean(axis=0) for k
        # Check for convergence (if centroids do not change)
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids
        iterations+=1

    return clusters, centroids

def kmedoids(pixels, K):
    """Function to perform K-medoids clustering on the given pixels."""
    pixels = pixels.reshape(-1, 3)  # Flatten the image to a 2D array of pix
    #medoids = pixels[np.random.choice(pixels.shape[0], K, replace=False)]
    medoids = select_inital_medoids_random(pixels, K)
```

```
    clusters = generate_clusters(pixels,medoids,order=1)  # Initialize clust
    new_medoids = np.zeros(medoids.shape)
    iterations = 0  # Count iterations
    distance_metrics = []
    while iterations < 10 and not np.array_equal(medoids, new_medoids):  # L
        total_metric = 0
        medoids =new_medoids
        for i in range(K):
            cluster = pixels[clusters == i]
            if len(cluster) == 0:
                continue
            current_min = distance_metric(cluster, medoids[i])
            sample_size = len(cluster)//20 if len(cluster) > 20 else 1
            sample_idx = np.random.choice(len(cluster), sample_size, replace

            sample = cluster[sample_idx]
            for pixel in sample:
                metric = distance_metric(cluster, pixel)
                if metric < current_min:
                    current_min = metric
                    new_medoids[i] = pixel
            total_metric+= current_min
        distance_metrics.append(total_metric)
        new_clusters = generate_clusters(pixels, medoids,order=1)
        iterations+=1

        if min(abs(np.array(distance_metrics[-2:]) - distance_metrics[-1])) 
            break
        else:
            clusters = new_clusters

    for idx,medoid in enumerate(medoids):
        if idx not in clusters:
            print(f'Medoid {idx} is empty')
    return clusters, medoids


def main():
    # Load the image files directly
    directory = os.getcwd()  # Get the directory of the script
    K = 2  # Number of clusters

    # Loop through all image files in the directory
    time_taken = []
    for filename in os.listdir(directory):

        if filename.endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff', '.gi
            image_file_name = os.path.join(directory, filename)
            print(image_file_name)
            K=8

            t = time.time()
            im = np.asarray(imageio.imread(image_file_name))
            fig, axs = plt.subplots(1, 2)

            #Apply K-medoids
```

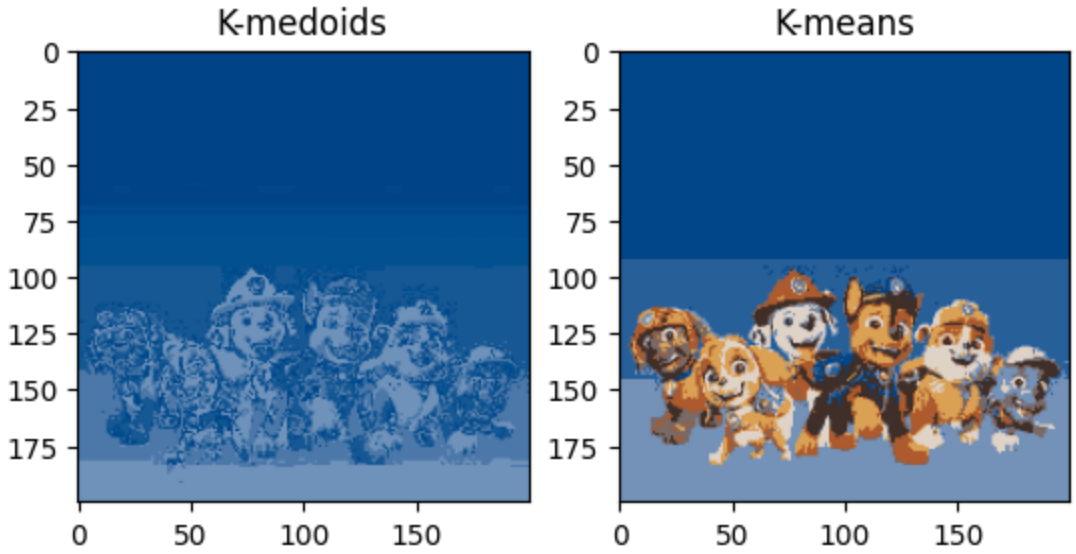
```
            classes, centers = kmedoids(im, K)
            new_im = np.asarray(centers[classes].reshape(im.shape), im.dtype
            imageio.imwrite(os.path.basename(os.path.splitext(image_file_nam
            axs[0].imshow(new_im)
            axs[0].set_title('K-medoids')
            # time_taken.append(time.time() - t)
            # print(f'Time taken: {time.time() - t:.2f} seconds')

            #Apply K-means
            classes, centers = kmeans(im, K)
            new_im = np.asarray(centers[classes].reshape(im.shape), im.dtype
            imageio.imwrite(os.path.basename(os.path.splitext(image_file_nam
            axs[1].imshow(new_im)
            axs[1].set_title('K-means')
            # time_taken.append(time.time() - t)
            # print(f'Time taken: {time.time() - t:.2f} seconds')
            plt.show()

if __name__ == '__main__':
    main()
```

```
/home/archit/Desktop/CDA-Assignments/HW1/programming/football.bmp
```

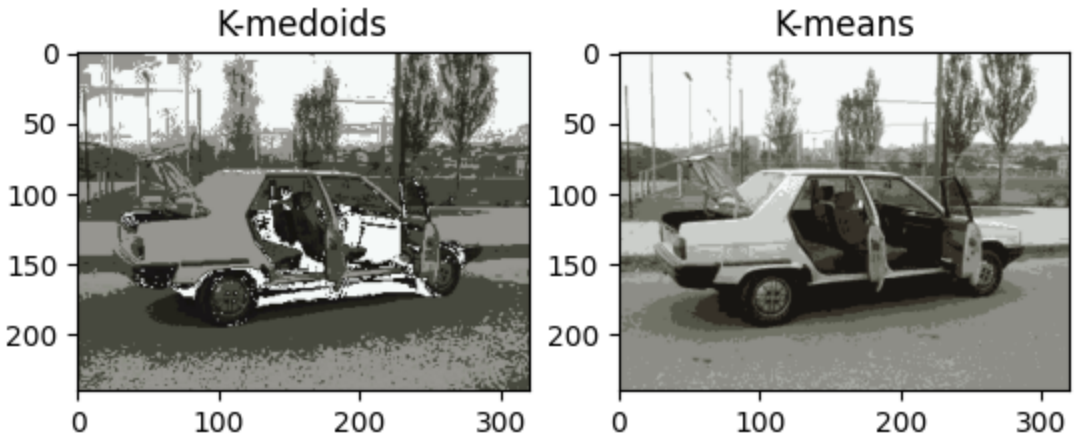
```
/tmp/ipykernel_88012/3341785279.py:107: DeprecationWarning: Starting with Im
ageIO v3 the behavior of this function will switch to that of iio.v3.imread.
To keep the current behavior (and make this warning disappear) use `import i
mageio.v2 as imageio` or call `imageio.v2.imread` directly.
  im = np.asarray(imageio.imread(image_file_name))
```



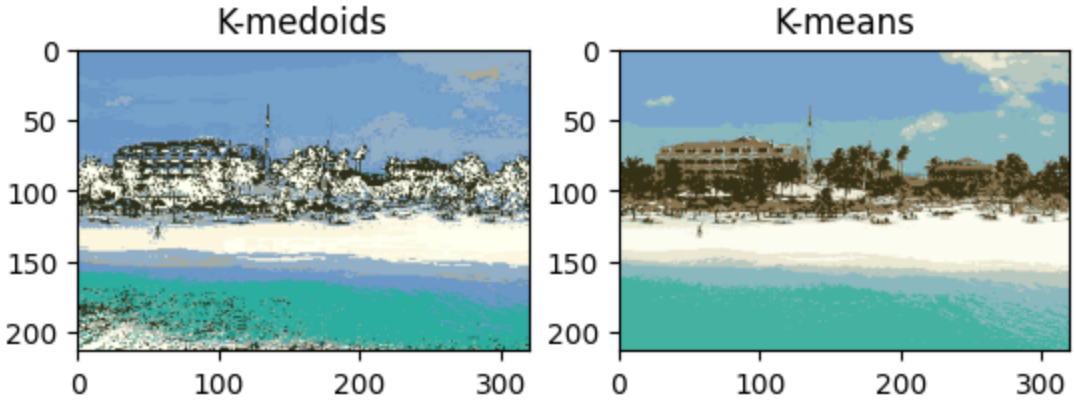

```
/home/archit/Desktop/CDA-Assignments/HW1/programming/dog.jpg
```




```
/home/archit/Desktop/CDA-Assignments/HW1/programming/car.jpg
```




```
/home/archit/Desktop/CDA-Assignments/HW1/programming/beach.bmp
```




In [ ]: