National University of Singapore
School of Computing
CS2109S: Introduction to AI and Machine Learning
Semester 2, 2024/2025

Tutorial 7
Unsupervised Learning

Summary of Key Concepts

In this tutorial, we will discuss and explore the following key learning points/lessons.

1. K-means algorithm
   (a) K-medoids algorithm
2. Kernel K-means clustering
   (a) Gaussian radial basis function (RBF) kernel
3. SVD

A K-means algorithm

The K-means algorithm is given as follows:

Algorithm 1: K-means clustering

for $k = 1$ ***to*** K **do**
  $\mu_k \leftarrow$ random location
while *not converged* **do**
  for $i = 1$ ***to*** m **do**
    $c^{(i)} \leftarrow argmin_k ||x^{(i)} - \mu_k||^2$
  for $k = 1$ ***to*** K **do**
    $\mu_k \leftarrow \frac{1}{|\{x^{(i)}|c^{(i)}=k\}|} \sum_{x \in \{x^{(i)}|c^{(i)}=k\}} x$

Note that a key fact of the K-means algorithm is that the algorithm never increases distortion.

1. From the key fact, it is clear that every iteration produces a partition with a lower or equal distortion. Prove from this fact that the K-means algorithm always converges. Convergence is when the centroids/medoids do not change after an iteration of the algorithm.

   Solution:

   The key idea is that we have a **finite number of possible partitions**. In particular, if we have N points and k clusters, there are at most k^N ways to cluster the points. Since we are also guaranteed that the distortion never increases, the

algorithm will eventually "run out" of partitions with a lower distortion (by the pigeonhole principle), and therefore will terminate.

***Bonus question**: How do we prove that the distortion never increases?*
Hint: Consider breaking down the K-means into two steps:

1. *Assign points to cluster*
2. *Recompute new centroids*

What happens to the distortion in both steps? Combining the two steps with a deterministic tie-breaking strategy, we can prove that the distortion decreases or remains the same at every iteration.

2. Although K-means always converges, it may get stuck at a bad local minimum. As mentioned in the lecture, one method of circumventing this is to run the algorithm multiple times and choose the clusters with the minimum distortion. Suggest some other ways to help the algorithm get closer to the global minimum.

Solution:

The problem lies in the random initialisation. We can come up with better ways of initialising the random initial state.

- We can choose the first centroid randomly. Then, choose the second centroid to be as far as possible from the first centroid. Then, choose the next centroid to be as far as possible from all of the previous centroids. Repeat until we have initialised all k centroids.
- We can also make use of a probabilistic method (K-means++). We start by choosing the first centroid randomly. To choose the rest of the centroids, we choose it randomly using a weighted probability distribution where the probability of choosing a point is proportional to $D(x)^2$, where $D(x)$ is the distance from x to the closest existing centroid.

3. You are given the following data.

i	1	2	3	4	5	6
x	1	1	2	2	3	3
y	0	1	1	2	1	2

Table 1: 6 data points on a 2D-plane

Cluster the 6 points in table 1 into **two** clusters using the K-means algorithm. The two initial centroids are (0, 1) and (2.5, 2).

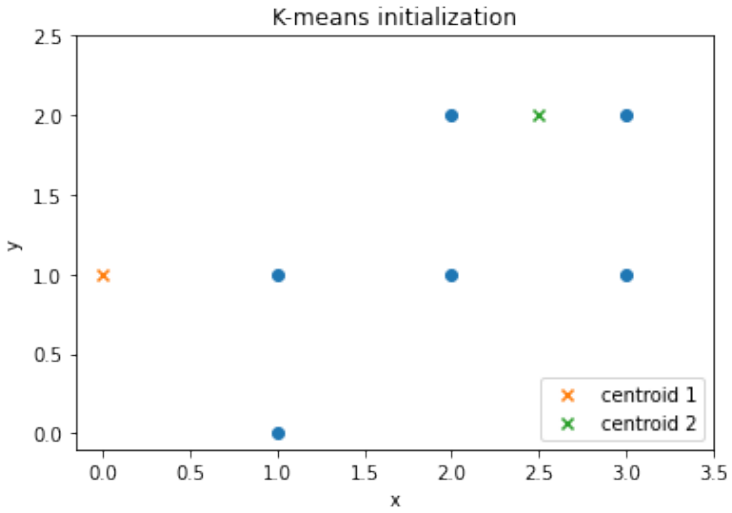

Figure 1: Initial configuration of K-means

Solution:

Iteration 1: Using first centroid = (0, 1) and second centroid = (2.5, 2), we get the table below.

Point	Squared distance to first centroid	Squared distance to second centroid	Assigned cluster
1	2	6.25	1
2	1	3.25	1
3	4	1.25	2
4	5	0.25	2
5	9	1.25	2
6	10	0.25	2

Computing the new centroids:

- Centroid 1 = ((1, 0) + (1, 1)) / 2 = (1, 0.5)
- Centroid 2 = ((2, 1) + (2, 2) + (3, 1) + (3, 2)) / 4 = (2.5, 1.5)

Iteration 2: Using first centroid = (1, 0.5) and second centroid = (2.5, 1.5), we get the table below.

Point	Squared distance to first centroid	Squared distance to second centroid	Assigned cluster
1	0.25	4.5	1
2	0.25	2.5	1
3	1.25	0.5	2
4	3.25	0.5	2
5	4.25	0.5	2
6	6.25	0.5	2

Computing the new centroids:

- Centroid 1 = ((1, 0) + (1, 1)) / 2 = (1, 0.5)
- Centroid 2 = ((2, 1) + (2, 2) + (3, 1) + (3, 2)) / 4 = (2.5, 1.5)

Since the centroids are the same as those from the previous iteration, the K-means algorithm has converged.

- The first cluster has centroid (1, 0.5) and contains points 1 and 2.
- The second cluster has centroid (2.5, 1.5) and contains points 3, 4, 5, and 6.

4. Cluster the 6 points in table 1 into **two** clusters using the K-medoids algorithm. The initial medoids are point 1 and point 3. The K-medoids algorithm is given in Algorithm 2.

Algorithm 2: K-medoids clustering

for $k = 1$ ***to*** K **do**
  $\mu_k \leftarrow$ random data point $x^{(i)}$
while *not converged* **do**
  for $i = 1$ ***to*** m **do**
    $c^{(i)} \leftarrow argmin_k ||x^{(i)} - \mu_k||^2$
  for $k = 1$ ***to*** K **do**
    $\mu_k \leftarrow \frac{1}{|\{x^{(i)}|c^{(i)}=k\}|} \sum_{x \in \{x^{(i)}|c^{(i)}=k\}} x$
  for $k = 1$ ***to*** K **do**
    $\mu_k \leftarrow argmin_{x^{(i)} \in \{x^{(i)}|c^{(i)}=k\}} ||x^{(i)} - \mu_k||^2$

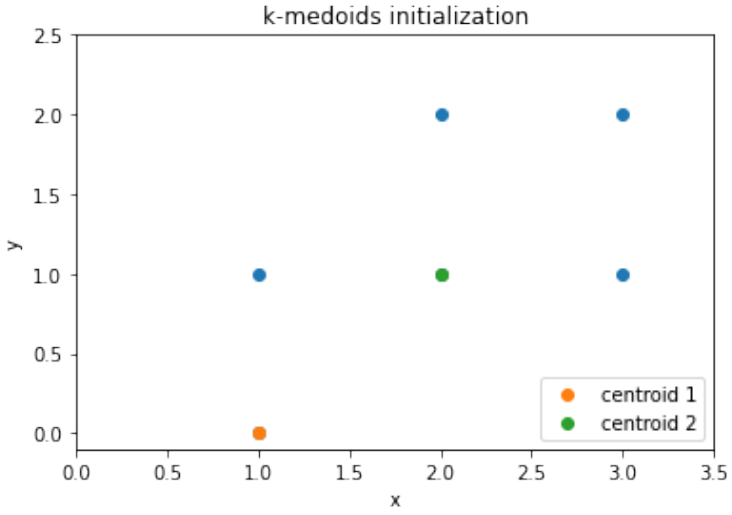

Figure 2: Initial configuration of K-medoids

Solution:

Iteration 1: Using first medoid = (1, 0) and second medoid = (2, 1), we get the table below.

Point	Squared distance to first medoid	Squared distance to second medoid	Assigned cluster
1	0	2	1
2	1	1	2
3	2	0	2
4	5	1	2
5	5	1	2
6	8	2	2

Notice that for point 2, the distance between itself to the first medoid is the same as the distance between itself to the second medoid. For simplicity, we assign point 2 as a member of the second cluster.

Note: Typically for ties, we want to choose a deterministic tie-break strategy to avoid infinite loops or excessive iterations. For example, remaining in the same cluster in the case of a tie, or only updating the medoid if there is a decrease in distortion.

To compute the new medoids, we first calculate the centroids:

- Centroid 1 = (1, 0)
- Centroid 2 = ((1, 1) + (2, 1) + (2, 2) + (3, 1) + (3, 2)) / 5 = (2.2, 1.4)

We then find the closest point to each centroid to serve as the new medoid:

- For centroid 1, the closest point is point 1. Hence, we set point 1 as the new medoid.
- For centroid 2, the closest point is point 3. Hence, we set point 3 as the new medoid.

Since the medoids are the same as the initial ones, the K-medoids algorithm has converged.

B Kernel K-means clustering

1. Given the set of data points in the figure below, Clustering 1 should be considered better than Clustering 2. Can the K-means algorithm achieve the better clustering? Why or why not?

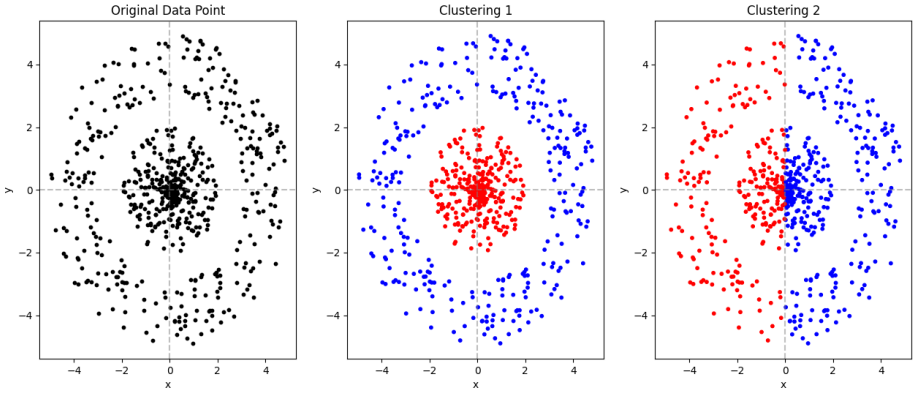

Figure 3: Different clustering approaches on the same dataset

Solution:

No, because K-means algorithm relies on Euclidean distance and assumes linearly separable clusters. K-means works best when clusters are convex and clearly separated by hyperplanes, but the desired clustering here is non-linear (circular). Since the algorithm assigns points based on proximity to centroids, it tends to create balanced, spherical clusters, making it unsuitable for this structure.

2. We would like to extend the K-means algorithm to non-linearly separable cases. To do that, write the squared Euclidean distance between two points $\mathbf{p}_i$ and $\mathbf{p}_j$ using vector norms (length) and the dot product. How can we apply the kernel trick?

Solution:

Square norm of a vector $\mathbf{v}$,

$$\|\mathbf{v}\|^2 = \mathbf{v}^T\mathbf{v} = \mathbf{v} \cdot \mathbf{v}$$

The squared Euclidean distance between two points $\mathbf{p}_i$ and $\mathbf{p}_j$ is:

$$\begin{aligned}\|\mathbf{p}_i - \mathbf{p}_j\|^2 &= (\mathbf{p}_i - \mathbf{p}_j) \cdot (\mathbf{p}_i - \mathbf{p}_j) \\ &= \mathbf{p}_i \cdot \mathbf{p}_i - \mathbf{p}_i \cdot \mathbf{p}_j - \mathbf{p}_j \cdot \mathbf{p}_i + \mathbf{p}_j \cdot \mathbf{p}_j \\ &= \mathbf{p}_i \cdot \mathbf{p}_i - 2(\mathbf{p}_i \cdot \mathbf{p}_j) + \mathbf{p}_j \cdot \mathbf{p}_j \quad \text{(Symmetry of dot product)} \\ &= \|\mathbf{p}_i\|^2 - 2(\mathbf{p}_i \cdot \mathbf{p}_j) + \|\mathbf{p}_j\|^2.\end{aligned}$$

We apply the kernel trick, i.e., we replace the dot product $\mathbf{p}_i \cdot \mathbf{p}_j$ with a valid kernel function $k(\mathbf{p}_i, \mathbf{p}_j)$. The result is:

$$\mathbf{p}_i \cdot \mathbf{p}_i - 2(\mathbf{p}_i \cdot \mathbf{p}_j) + \mathbf{p}_j \cdot \mathbf{p}_j \to k(\mathbf{p}_i, \mathbf{p}_i) - 2k(\mathbf{p}_i, \mathbf{p}_j) + k(\mathbf{p}_j, \mathbf{p}_j).$$

We can use the kernel trick in the distance computation in K-means clustering.

3. For the remainder of the problem, you are given the following data.

   Given five points in Table 2 and two initial centroids, (1,1) and (-4,-4), we perform K-means clustering with two clusters. After convergence, we obtain:

- The first cluster has centroid (1, 1) and contains points 1, 2, 3, and 5.
- The second cluster has centroid (-4, -4) and contains point 4.

p	x	y
1	0	0
2	4	4
3	-4	4
4	-4	-4
5	4	-4

Table 2: 5 data points on a 2D-plane

The table below shows that the K-means algorithm has converged.

Point	Squared distance to first centroid	Squared distance to second centroid	Assigned cluster
1	2	32	1
2	18	128	1
3	34	64	1
4	50	0	2
5	34	64	1

- Centroid 1 $\mu_1 = ((-4, 4) + (0, 0) + (4, -4) + (4, 4))/4 = (1, 1)$.
- Centroid 2 $\mu_2 = (-4, -4)/1 = (-4, -4)$.

The Gaussian radial basis function (RBF) kernel maps data into a higher-dimensional space, where clusters potentially become linearly separable. For a point $\mathbf{p}_i = (x_i, y_i)$, the formula for the RBF kernel $k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_j)$ is given by:

$$k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_j) = \exp\left(-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\sigma^2}\right),$$

where σ is a parameter that controls the width of the kernel. Set $\sigma = 4$. Calculate $k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_j)$ for $i, j = 1, \ldots, 5$. Explain what the value $k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_j)$ represents.
Note: The matrix with the entries $k_{ij} := k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_j)$ is called a kernel matrix (for a given kernel function).

Solution:

The function $k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_j)$ represents the similarity between the points $\mathbf{p}_i$ and $\mathbf{p}_j$. The value of the kernel is close to 1 when the points are similar (i.e., close in Euclidean distance), and approaches 0 as the points become distant from each other. Based on

$$d^2(\mathbf{p}_1, \mathbf{p}_2) := \|\mathbf{p}_1 - \mathbf{p}_2\|^2 = (0-4)^2 + (0-4)^2 = 32$$

$$k_{\text{rbf}}(\mathbf{p}_1, \mathbf{p}_2) = \exp\left(-\frac{32}{2 \cdot 4^2}\right) = \exp(-1) \approx 0.37,$$

we evaluate the kernel function between all the data points:

$\mathbf{p}_i$	$k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_1)$	$k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_2)$	$k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_3)$	$k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_4)$	$k_{\text{rbf}}(\mathbf{p}_i, \mathbf{p}_5)$
(0, 0)	1	$\exp(-1)$	$\exp(-1)$	$\exp(-1)$	$\exp(-1)$
(4, 4)	$\exp(-1)$	1	$\exp(-2)$	$\exp(-4)$	$\exp(-2)$
(-4, 4)	$\exp(-1)$	$\exp(-2)$	1	$\exp(-2)$	$\exp(-4)$
(-4, -4)	$\exp(-1)$	$\exp(-4)$	$\exp(-2)$	1	$\exp(-2)$
(4, -4)	$\exp(-1)$	$\exp(-2)$	$\exp(-4)$	$\exp(-2)$	1

4. (Optional) Evaluate the distance to the previous cluster centers μ_1 and μ_2 using the kernel trick and the kernel matrix from the previous question.
Hint: The kernel trick is applied only to dot products between the data points and not *directly* to dot products between a data point and the cluster mean.

Solution:

The existing cluster centers are

$$\mu_1 = \frac{1}{4}\sum_{i\in\text{cluster1}} \mathbf{p}_i = \frac{1}{4}\left((-4,4)+(0,0)+(4,-4)+(4,4)\right), \tag{1}$$

$$\mu_2 = \sum_{i\in\text{cluster2}} \mathbf{p}_i = (-4,-4). \tag{2}$$

Performing the kernel trick in the distance to the cluster leads to

$$\begin{aligned}
d^2(\mathbf{p}_i,\mu_1) &= \|\mathbf{p}_i-\mu_1\|^2 && (3)\\
&= \mathbf{p}_i\cdot\mathbf{p}_i - 2\mathbf{p}_i\cdot\mu_1+\mu_1\cdot\mu_1 && (4)\\
&= \mathbf{p}_i\cdot\mathbf{p}_i - \frac{2}{4}\sum_{j\in\text{cluster1}}\mathbf{p}_i\cdot\mathbf{p}_j + \frac{1}{16}\sum_{k,j\in\text{cluster1}}\mathbf{p}_k\cdot\mathbf{p}_j && (5)\\
&\to\ k_{\text{rbf}}(\mathbf{p}_i,\mathbf{p}_i) - \frac{1}{2}\sum_{j\in\text{cluster1}} k_{\text{rbf}}(\mathbf{p}_i,\mathbf{p}_j) + \frac{1}{16}\sum_{k,j\in\text{cluster1}} k_{\text{rbf}}(\mathbf{p}_k,\mathbf{p}_j) =: d^2_{\text{rbf}}(\mathbf{p}_i,\mu_1). && (6)
\end{aligned}$$

In the last line, we have used the kernel trick, which defines a new distance $d^2_{\text{rbf}}(\mathbf{p}_i,\mu_1)$ based on the RBF kernel. Based on the numbers (kernel matrix) derived in the previous question, we can evaluate

$$\begin{aligned}
d^2_{\text{rbf}}(\mathbf{p}_1,\mu_1) &= k_{\text{rbf}}(\mathbf{p}_1,\mathbf{p}_1) - \frac{1}{2}\sum_{j\in\text{cluster1}} k_{\text{rbf}}(\mathbf{p}_1,\mathbf{p}_j) + \frac{1}{16}\sum_{k,j\in\text{cluster1}} k_{\text{rbf}}(\mathbf{p}_k,\mathbf{p}_j) && (7)\\
&= k_{\text{rbf}}(\mathbf{p}_1,\mathbf{p}_1) - \frac{1}{2}\left(k_{\text{rbf}}(\mathbf{p}_1,\mathbf{p}_1)+k_{\text{rbf}}(\mathbf{p}_1,\mathbf{p}_2)+k_{\text{rbf}}(\mathbf{p}_1,\mathbf{p}_3)+k_{\text{rbf}}(\mathbf{p}_1,\mathbf{p}_5)\right)\\
&\quad + \frac{1}{16}\,(\text{all corresponding 16 entries})\\
&= 1 - \frac{1}{2}\left(1+3\exp(-1)\right) + \frac{1}{16}\left(4+6\exp(-1)+4\exp(-2)+2\exp(-4)\right) && (8)\\
&\approx 0.37. && (9)
\end{aligned}$$

Performing this computation for all points and both cluster centers, we evaluate the distances:

$\mathbf{p}_i$	$d^2_{\text{rbf}}(\mathbf{p}_i, \mu_1)$	$d^2_{\text{rbf}}(\mathbf{p}_i, \mu_2)$	Assignment
(0, 0)	0.37	1.26	1
(4, 4)	0.60	1.96	1
(-4, 4)	0.66	1.73	1
(-4, -4)	1.10	0.00	2
(4, -4)	0.66	1.73	1

Comparing the square distances to the centroid for the kernel and non-kernel versions, we see that:

- In the non-kernel version, the distances exhibits large values, reinforcing their separation in the original space.
- In the kernel version, distance values are more balanced, reducing the impact of large spatial differences.
- However, the assignments do not change in our example. This fact is explained by the lack of more data points and the existing centroids being a (local) minimum. Restarting kernel K-means and using more data points will achieve the desired clustering (see link below).

The kernel trick/feature transformations can help clustering methods, like kernel K-means, better separate data points into meaningful clusters based on their transformed distances.

This question introduces the concept of calculating distance in higher-dimensional space using the RBF kernel (and others). For more details on the kernel K-means clustering algorithm, refer to this optional reading: https://sites.google.com/site/dataclusteringalgorithms/kernel-k-means-clustering-algorithm.

C SVD

PCA can be thought of as a form of lossy compression. In this question, we will compress some data, provided in the form of an image for better visualisation. We will be making use of a classic test image `mandrill.png`, which can be downloaded from https://upload.wikimedia.org/wikipedia/commons/a/ab/Mandrill-k-means.png.

Figure 4: Image of Mandrill

You have been provided with a notebook `Tutorial7.ipynb` to aid you in answering the

questions below.

1. The current choice of $k = 9$ does not produce a very nice output. What is a good value for k? Justify your answer.

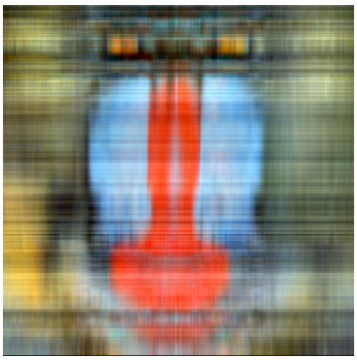

Figure 5: Image of Mandrill after k=9

Solution:

A good value would be $k = 286$ (This number can be determined through binary search, or simply iterating through the possible values of k).

When $k = 286$, the explained variance is $\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2} \geq 0.99$. Notice for smaller values of k with less explained variance, the image still looks reasonably good because the brain is good at recognising images so there is more room for compression for images. However, when using SVD for dimensionality reduction, a good rule of thumb is to use a value of k that retains 99% of the variance.

2. For the value of k you select in part 1, what is the space saved by doing this compression?

Solution:

When using $k = 286$, the (512×1536) 2D-array is now represented by U_{reduce} (512×286) and Z (286×1536). This demonstrates approximately 25.5% space saved.

$$\frac{(512 \times 286) + (286 \times 1536)}{512 \times 1536} = \frac{585728}{786432} = 0.745$$

If we wish to have more compression, we can choose a smaller k, but at the expense of image quality.

3. What are the drawbacks of this form of compression?

Solution:

This is a lossy compression as we do not regain 100% of the variance. For data with less redundancy, the value of k might be too large such that we would actually end up requiring more space as the decompression matrix U_{reduce} needs to be stored as well.

In practice, to compensate for this, PCA is done over a dataset of images so that U_{reduce} can be reused for every image for better compression.