Clustering Algorithms Summary: K-means and Gaussian Mixture Models

Harry Fyjis-Walker

In unsupervised learning, targets Y are not provided. Two main types:

— Clustering: grouping X based on similarity

— Dimension Reduction: compressing X to lower a dimension space

1 Overview of Clustering Algorithms

Objective: In dataset S, we seek to partition examples X such that similar examples belong to the same group.

This is used to:

— Understanding hidden patterns of data

— Create additional features for modelling

— Anomaly detection

And in the following applications:

— Grouping Documents

— Customer Segmentation - group people by buying habits

— Image Segmentation - grouping pixels to identify objects (e.g. separating foreground from background)

There are a number of types of clustering (which ones map to which applications?):

1. Density-based Clustering (e.g. DBSCAN): finds areas where dots are packed tightly together; ignores sparse noise.
2. Distribution-based Clustering: assumes data is made of overlapping Gaussian clouds.
3. Centroid-based Clustering (e.g. K-means): groups data around central points
4. Hierarchical Clustering: creates a tree-like structure (dendrogram) showing how groups merge into larger groups.

1.1 Distance Functions

To measure the similarity integral to clustering, we need a mathematical way to measure closeness. We say that:

$$Distance = Similarity^{-1} \qquad \text{i.e. large distance = low similarity} \tag{1}$$

A valid distance function has three key properties:

1. *Symmetry, d(x,y) = d(y,x)* i.e. distance from London to Cambridge must be same as Cambridge to London
2. *Positive Separability, d(x,y) = 0 iff x=y* i.e. the distance is zero if and oly if the two points are exactly the same
3. Taking a detour through a third point (z) must always be $\geq$ the direct path (i.e. Triangular inequality, $d(x,y) \leq d(x,z) + d(z,y)$).
   - i.e. Either equal to: Watford is either directly en route from London to Cambridge i.e. $d(x,y) = d(x,z) + d(z,y)$
   - OR greater than: if not directly en route, it is a detour to get to Watford, so $d(x,y) < d(x,z) + d(z,y)$

Most distance functions used are specialised versions of the general *Minkowski Distance*. Given two data points in R^d, $x = (x_1, \ldots, x_d)$ and $y = (y_1, \ldots, y_d)$:

$$d(x,y) = \left(\sum_{i=1}^{d} |x_i - y_i|^p \right)^{1/p} \tag{2}$$

— *Euclidean Distance (p=2):* The "straight-line" distance between two points in a multi-dimensional space. Allowed to move diagonally - from (0,0) to (3,4), distance is $\sqrt{3^2 + 4^2} = 5$
   - e.g. calculating the physical distance between two GPS coordinates "as the crow flies"
   - Default for most clustering e.g. K-means
   - Works best when data is dense and features are measured in the same units (e.g. width and height in cm)
   - Very sensitive to outliers, as squares the difference

— Manhattan Distance (p=1): the distance between two points measured along axes at right angles. Can only move along the axes (left/right or up/down), not diagonally. From (0,0) to (3,4), distance is $|3-0| + |4-0| = 7$.
   - e.g. A taxi driving through a city grid like Manhattan.
   - More stable than euclidean as does not square difference (so outliers do not pull the distance as aggressively)
   - Used when features are high-dimensional or have different scales
   - For binary data, this is known as Hamming Distance (counts number of positions where the corresponding symbols are different e.g. comparing two binary strings of equal length - can be used in counting mutations in DNA sequences, matching users based on Yes/No preference tags, error detection in telecommunications

— Chebyshev/ Infinity Distance ($p = \infty$) i.e. $d(x, y) = \max_{i=1}^{d} |x_i - y_i|$

- e.g. Generally used when you want a bottleneck metric e.g. Quality control: if a part must fit into a box, only largest dimension matters for determining if it will fit; or in robotics - if a machine has two independent motors moving at same speed, the time to reach a destination is determined by whichever motor has the furthest to travel

Moving beyond fixed-coordinate vectors, there are many data types where "distance" is not about location in a grid, but about the effort to transform one thing into another: *Edit Distance: the minimum number of actions needed to transform one object to another.*

- Advantage: Minkowski distances require your data to be numerical vectors of the same length. If you are comparing two sentences, one might have five words and the other ten - the edit distance allows you to compare sequences of different lengths.

The most common form of edit distance is the *Levenshtein Distance*, which uses three basic operations:

- Insertion: adding a character (e.g. "cat" -> "cats")
- Deletion: removing a character (e.g. "cars" -> "car")
- Substitution: swapping one character for another (e.g. "cat" -> "bat")

In some versions of this algorithm, a substitution is treated as a deletion (cost=1) followed by an insertion (cost=1), so a substitution has cost=2.

NB kernels behave like "similarity functions" (as they measure how alike two points are).

1.2 K-means Clustering

We define:

- $x_i \in R^m$: Our data points (vectors).
- k: The number of clusters we want (we must decide this before we start).
- μ_j: The Centroid. This is the mathematical "center of gravity" for cluster j.
- $c(x_i)$: The assignment. This tells us which cluster ID (1, 2, ... k) point x_i belongs to.

K-means is a centroid-based clustering method that aims to find the best possible set of centroids, by minimising the Within-Cluster Sum of Squares (WCSS), also known as distortion:

$$J(c, \mu) = \sum_{i=1}^{n} \sum_{j=1}^{k} 1\{c_i = j\} \|x_i - \mu_j\|^2 \tag{3}$$

where $1\{c_i = j\}$ is an indicator function (1 if point i is assigned to cluster j, 0 otherwise), and $\|x_i - \mu_j\|^2$ is the squared L_2 norm (Euclidean distance). J therefore depends on the two sets of variables, the discrete assignments c and the continuous centroids μ. We cannot optimise both, so we alternative...

K-means is iterative, repeating the following two steps until it cannot improve any further:

1. Choose k (e.g. k=3)
2. Initialise randomly i.e. place our k (3) centroids in the dataspace at positions $(\mu_1 \dots \mu_k)$
3. Every data point x_i calculates its distance to each of the k centroids, and "joins" the cluster of the centroid that is closest (Euclidean distance) to minimise J with respect to c (fixed μ - centroid position fixed, c (which centroid each point is assigned to):

$$c(x_i) := \arg\min_j \|x_i - \mu_j\|^2 \tag{4}$$

4. Now the points are grouped, we minimise J with respect to μ (fixed c): i..e each centroid moves to the centre of gravity of the points that just joined it (take average of the coordinates of every point in cluster c_j:

$$\frac{\partial J}{\partial \mu_j} = \sum_{x_i \in c_j} -2(x_i - \mu_j) = 0 \qquad \text{Minimise partial deriv. of J wrt } \mu_j \text{ and set to zero}$$

$$\mu_j := \frac{1}{|c_j|} \sum_{x_i \in c_j} x_i \qquad \text{Solving for } \mu_j \tag{5}$$

5. The two last steps repeat, until we reach a state where no point changes its cluster assignment (convergence). (How to test for convergence?)

Points to note:

- Because each step is a minimization, the value of J is monotonically non-increasing. Since there are only a finite number of possible partitions, the algorithm must converge. (?)
- However, because the loss surface is non-convex, it is prone to local optima, meaning the final result depends heavily on the initial (random) positions of μ; can use K-means++, which chooses initial centroids that are far apart from each other, or run algorithm multiple times with different seeds and keep version with lowest final J.
- Limitations:
  - Because K-means uses L_2 norm, it implicitly assumes that clusters are spherical and have similar diameters - fails on elongated or "moon-shaped" data
  - Hard-clustering - a point belongs 100% to one cluster, even if it is exactly on the border between the two (compare to GMMs, which provide "soft" probabilities.

Choosing optimal k: As k increases, distortion $L(\mu, X)$ will always decrease, towards k = n (where n is number of data points) i.e. every point is its own centroid, distance from a point to its centroid is 0, so total distortion is therefore 0. Thus, algorithm will always say k=n is best solution, which is not useful for grouping. So we use the *Elbow Method*, which looks for the sweet spot where adding another cluster no longer provides a significant return on investment i.e. looking for the point of maximum curvature in the distortion-vs-k graph. So we seek k* such that:

— Significant Compression: $L(\mu, X)_{k^*}$ is significantly low, indicating that the clusters are sufficiently compact.

— Diminishing Returns: $L(\mu, X)_{k^*+1}$ does not offer a great improvement, meaning the marginal reduction in distortion is no longer worth the added complexity of an extra cluster.

1.3 Gaussian Mixture Models (Distribution-based Clustering)

The point of clustering is to try to answer the question: *Which points belong together?*

K-means assumes:

- Each cluster has a centre μ_k
- Each point belongs to exactly one cluster
- Distance to the centre designs assignment

However, real data is often messy and clusters overlap, some points are ambiguous, and some clusters are stretched, rotated, or have different sizes.

So GMMs, instead of asking *"Which cluster is closest?* ask *"Which cluster most likely generated this point?"*

The Gaussian Distribution and GMMs

A Gaussian distribution in D dimensions tells you *how likely is point x under this blob* and is written:

$$N(x\mu, \Sigma) \qquad \text{where } \mu \text{ is the centre and } \Sigma \text{ is the shape and orientation} \tag{6}$$

A GMM assumes that the data was generated by several Gaussian distributions. So, instead of fitting one Gaussian to all the data, we fit K Gaussians:

$$N(x|\mu_1, \Sigma_1), ..., N(x|\mu_k, \Sigma_k) \tag{7}$$

and combine them into p(x), the probability density of observing point x under the mixture model:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \quad \text{where} \quad \sum_{k=1}^{K} \pi_k = 1 \quad \text{and } \pi_k \text{ represents how common cluster k is} \tag{8}$$

The GMM approaches this by imagining a process of data point creation which follows the following:

- Pick a one Gaussian (one cluster) at random, based on the probabilities, π of each Gaussian
- Once the blob is chosen, generate a point x from that Gaussian

So its goal is to answers: "Which Gaussian probably generated this point?"

Obtaining the GMM

To frame the problem, we have data $X = x_1, x_2, ..., x_N$ but no labels.

— *Assumption:* The data are random samples drawn from some unknown probability distribution, $p(x|\theta)$ (where x are the random samples and θ are the parameters, that has the form of a mixture of Gaussians.

— *Goal*: We want to learn the parameters, θ, that make the observed data as probable as possible under the model (the "Maximum Likelihood Estimation (MLE)"):

$$\theta = (\pi_k, \mu_k, \Sigma_k)_{k=1}^{K} \tag{9}$$

So we want to obtain parameters of a GMM that make the observed data have the highest likelihood of occurring as possible. Therefore, we first need an expression for the likelihood for us to maximise:

- If a single point x is drawn from distribution $p(x|\theta)$, then the probability of observing it is

$$p(x|\theta) \tag{10}$$

- If we observe N independent data points, then

$$p(X|\theta) = \prod_{n=1}^{N} p(x_n|\theta) \qquad \text{the likelihood function} \tag{11}$$

  where the data X are fixed and we vary θ to maximise this probability $p(X|\theta)$.

- For a GMM:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{12}$$

  i.e. the weighted sum over the K Gaussians of the probabilities for contribution by each Gaussian. Therefore, as we assume i.i.d in X so $p(X|\theta) = p(x_1|\theta) \cdot p(x_2|\theta) \cdot ... \cdot p(x_N|\theta)$,

$$p(X|\theta) = \prod_{n=1}^{N} \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right) \tag{13}$$

  is the (marginal) likelihood function of GMM for the whole dataset (over all N data points).

- As products of many probabilities can become numerically unstable and messy, we take the log likelihood (as log is strictly increasing and maximising log-likelihood is equivalent to maximising likelihood):

$$\log p(X|\theta) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right) \tag{14}$$

  which is the function we want to maximise:

$$\max_{\pi,\mu,\Sigma} \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right) \tag{15}$$

So now we understand that in a GMM, the observed data are $X = (x_1, ..., x_N)$, and our goal is to find the parameters $\theta = (\pi_k, \mu_k, \Sigma_k)_{k=1}^{K}$ that maximise the marginal likelihood:

$$\log p(X|\theta) = \sum_{n=1}^{N} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right) \tag{16}$$

The Problem and the Latent variable solution

For this "incomplete" likelihood, for any given point x_n, we do not know which cluster it came from. Therefore, we have to account for every possibility so we sum over all possible clusters K (as shown above), weighted by their probabilities:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{17}$$

Then, accounting for the probabilities $x_1, ..., x_N$ of each data point falling inside the GMM (?), we get:

$$p(X|\theta) = \prod_{n=1}^{N} \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right) \tag{18}$$

— *Problem:* This is a log-of-sum, not sum-of-logs, so each Gaussian component interacts with every other component because of the sum inside the log (?). This means the parameters are interdependent in a complex way, and there is no closed-form solution for the parameters directly from this.

— *Sub-goal:* How can we "untangle" this sum to make the problem easier to solve?

— *Solution:* We introduce a latent (hidden) vairable z_n for each data point x_n, representing which Gaussian generated that point:

◇ z_n is a one-hot vector of length K (total number of Gaussians):

$$z_n = (0, ..., 0, 1, 0, ..., 0) \tag{19}$$

where the 1 indicates the Gaussian that generated x_n e.g. if x_n came from Gaussian 3, then $z_n = (0, 0, 1, 0, ..., 0)$. If we had z_n for every point, we would know the "true" cluster assignment for each point

◇ This allows us to change the function we are maximising. Before, we had to sum over all K clusters to find the likelihood; however, if we know Z, only one term in that sum can be true. We can write this mathematically using the exponent trick, which only keeps the Gaussian for which z_{nk} is 1 and puts all the others to the power of zero (i.e. = 1), leaving just the relevant Gaussian:

$$\begin{aligned} p(x_n, z_n|\theta) &= p(z_n|\theta) \cdot p(x_n|z_n, \theta) \quad \text{i.e. 1. Pick Gaussian; 2. Generate point from that Gaussian} \\ p(z_n|\theta) &= \prod_{k=1}^{K} \pi_k^{z_{nk}} \quad \text{Prior - since } z_n \text{ is one hot i.e. } z_{nk} \text{ is the 1 in } z_n = 0,1,0.. \\ p(x_n|z_n, \theta) &= \prod_{k=1}^{K} \mathcal{N}(x_n|\mu_k, \Sigma_k)^{z_{nk}} \end{aligned} \tag{20}$$

i.e. the conditional - if we know which cluster x_n came from (because z_n tells us), the probability is just that specific Gaussian. Again, if $z_{n2} = 1$, this product collapses to $\mathcal{N}(x_n|\mu_2, \Sigma_2)$. For a single point, we multiply the prior and conditional to get the posterior:

$$p(x_n, z_n|\theta) = \prod_{k=1}^{K} (\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k))^{z_{nk}} \tag{21}$$

And scaling this to a full dataset (again assuming each pair (x_n, z_n) is independent so can use p(a,b) = p(a) x p(b)):

$$p(X, Z|\theta) = \prod_{n=1}^{N} \left[\prod_{k=1}^{K} (\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k))^{z_{nk}} \right] \tag{22}$$

◇ Now, while before we had $\prod \Sigma$ and log(AB) = log(A) + log(B) but log(A+B) cannot be simplified so we get $\prod log \Sigma$ when we take the log, here we have a double product $\prod \prod$, so we get:

$$\log p(X, Z|\theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} \log \left([\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)]^{z_{nk}} \right) \tag{23}$$

and by $log(A^b) = blogA$:

$$\log p(X, Z|\theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \left[\log \pi_k + \log \mathcal{N}(x_n|\mu_k, \Sigma_k) \right] \tag{24}$$

so now each Gaussian's parameters affect only the points assigned to it i.e. latent variables have allowed us to pretend we know the hidden assignments, simplifying the problem mathematically.

Expectation-Maximisation: Estimating z

Step 1, E-Step (Expectation): Estimate the probability that each point belongs to each Gaussian ("responsibility" of each Gaussian)

Recall how a GMM imagines the data point creation process:

- Pick a one Gaussian (one cluster) at random, based on the probabilities, π of each Gaussian
- Once the blob is chosen, generate a point x from that Gaussian

We then reverse this using Bayes rule, to calculate the probability that $z_{nk} = 1$ i.e. that Gaussian k is responsible for data point x_n, by calculating its expected value. We say:

$$\begin{aligned} \gamma_{nk} &= E[z_{nk}] = p(z_{nk} = 1|x_n, \theta) \quad \text{probability it is cluster k gives data point and parameters} \\ \gamma_{nk} &= \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)} \quad \text{i.e. "how responsible is each Gaussian k for generating point } x_n\text{"} \end{aligned} \tag{25}$$

where the numerator is the probability that data point x_n came from Gaussian k (i.e. weighted by probability of Gaussian k), and denominator is total probability of x_n occurring under the entire mixed Gaussian shape (i.e. the sum over all possible clusters). So: probability point is under Gaussian k/ probability point is somewhere under whole GMM.

So now we have a variable that represents how likely an individual Gaussian is to contain a given point. We then define a new function, Q, which is this Expected Complete-Data Log-Likelihood, involving replacement of the unobservable z_{nk} with the fixed number (expectation) γ_{nk}:

$$Q(\theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} [\log \pi_k + \log \mathcal{N}(x_n|\mu_k, \Sigma_k)] \tag{26}$$

which we can then differentiate to find the value of each parameter that maximise this log-likelihood Q.

Step 2, M-Step (Maximisation): Update the Gaussian parameters using those probabilities to maximise likelihood (Q)

To maximise the log-likelihood Q, we need to optimise parameters π_k, μ_k, and Σ_k. We do this by taking the derivatives of Q with respect to each parameter, setting to zero, and solving. This gives us the following:

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \tag{27}$$

which represents the effective number of points assigned to cluster k. e.g. if we have for cluster 1: $\gamma(x_1) = 0.8, \gamma(x_2) = 0.3, \gamma(x_3)0.6$, then $N_1 = 0.8 + 0.3 + 0.6$ i.e. cluster 1 effectively "owns" 1.7 points.

So, for the mean, we define:

$$\mu_k^{new} = \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} x_n \tag{28}$$

where if γ_{nk} is large i.e. there is a high probability that point x_n belongs to cluster k, then this point strongly pulls the mean for cluster k, μ_k. If γ_{nk} is small, x_n barely affects the mean. As such, the Gaussian moves towards the weighted centre of the points it is "responsible" for.

For the covariance,

$$\Sigma_k^{new} = \frac{\sum_{n=1}^{N} \gamma_{nk} (x_n - \mu_k^{new})(x_n - \mu_k^{new})^T}{\sum_{n=1}^{N} \gamma_{nk}} \tag{29}$$

i.e. points that strongly belong to cluster k influence shape strongly, while those that only weakly belong have little affect on covariance.

For the mixing coefficient,

$$\pi_k^{new} = \frac{1}{N} \sum_{n=1}^{N} \gamma_{nk} = \frac{N_k}{N} \tag{30}$$

where N_k is the effective number of points in cluster k and N is the total number of points i.e. π_k is the fraction of data mass belonging to cluster k.

Overall Workflow

1. Initialisation: start with a best guess for the parameters θ: π_k, μ_k, and Σ_k
2. E-Step: Calculate γ_{nk}
3. M-Step: Update π_k, μ_k, and Σ_k using γ_{nk}
4. Calculate Marginal Log-Likelihood using these new parameters to see if there is an improvement

Derivation of M-Step Formulae

1. Deriving the mean, μ_k:
   - ◇ Starting with expanded definition of $Q(\theta)$:

$$Q(\theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma_{nk} \left[\log \pi_k - \frac{1}{2} \log |2\pi\Sigma_k| - \frac{1}{2} (x_n - \mu_k)^T \Sigma_k^{-1} (x_n - \mu_k) \right] \tag{31}$$

   - ◇ Using identity $\frac{\partial}{\partial s}(a - s)^T W (a - s) = -2W(a - s)$:

$$\begin{aligned} \frac{\partial Q}{\partial \mu_k} &= \sum_{n=1}^{N} \gamma_{nk} \left[-\frac{1}{2} \cdot (-2\Sigma_k^{-1}(x_n - \mu_k)) \right] \\ \frac{\partial Q}{\partial \mu_k} &= \sum_{n=1}^{N} \gamma_{nk} \Sigma_k^{-1} (x_n - \mu_k) \end{aligned} \tag{32}$$

◇ Setting derivative to zero and multiplying both sides by Σ_k^{-1} (as it is a constant matrix across the summation):

$$\begin{aligned} \sum_{n=1}^{N} \gamma_{nk} \Sigma_k^{-1} (x_n - \mu_k) &= 0 \\ \sum_{n=1}^{N} \gamma_{nk} (x_n - \mu_k) &= 0 \end{aligned} \tag{33}$$

◇ Distribute the sum, pull out μ_k as it is not dependent on n, and solve for μ_k:

$$\begin{aligned} \sum_{n=1}^{N} \gamma_{nk} x_n - \sum_{n=1}^{N} \gamma_{nk} \mu_k &= 0 \\ \sum_{n=1}^{N} \gamma_{nk} x_n &= \mu_k \sum_{n=1}^{N} \gamma_{nk} \\ \mu_k^{new} &= \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{\sum_{n=1}^{N} \gamma_{nk}} \end{aligned} \tag{34}$$