Section 7: Gaussian Mixture Model

Section 7.1: Normal Distribution Probability Density Function

Why do we need Probability Density Functions

- Probability density functions are used in general in distribution-based clustering approaches
- Specifically for the Gaussian Mixture Model, need probability density function for the normal distribution

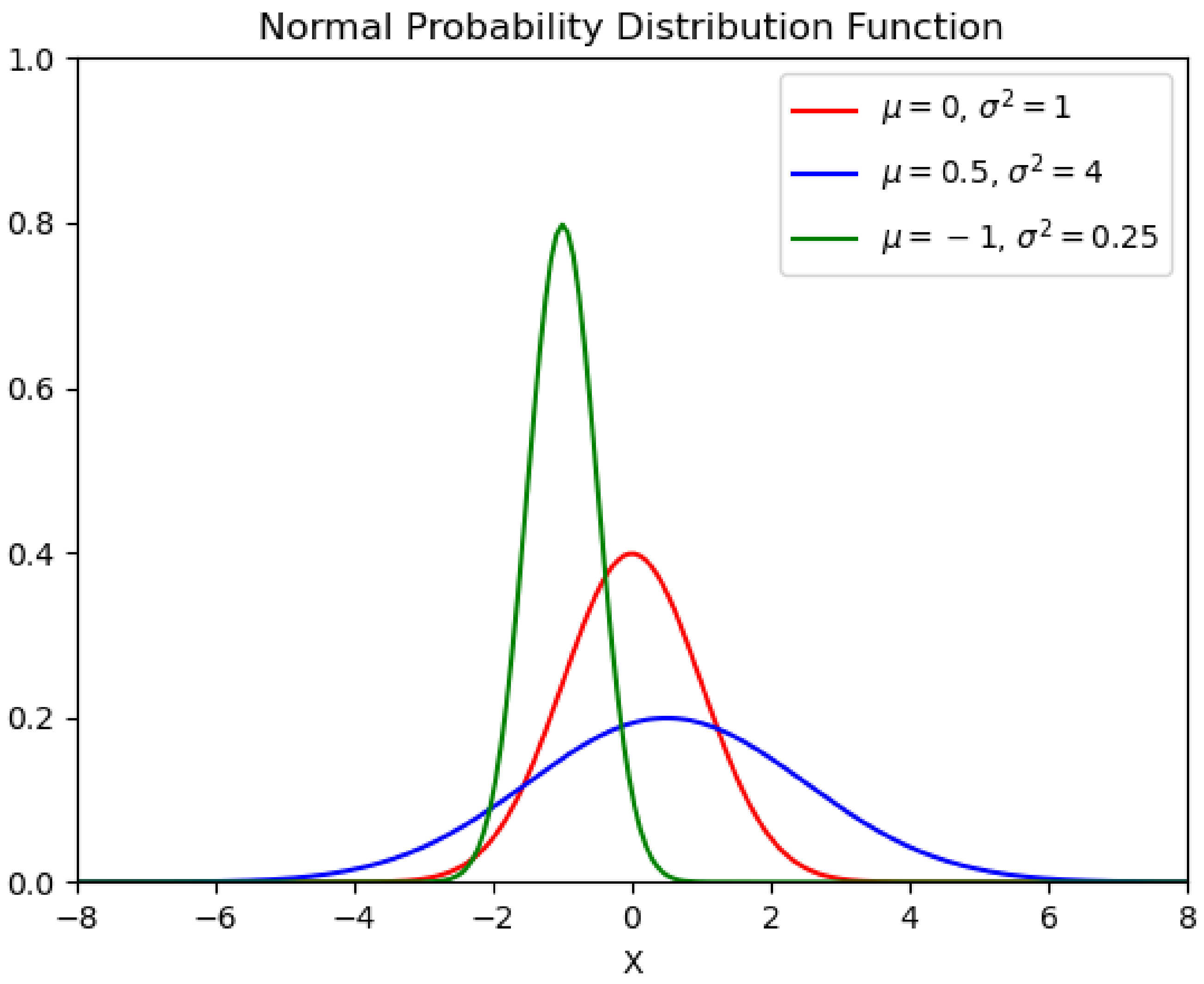
Normal Distribution: Probability Density Function

Probability density function for 1 dimensional case

- Let X be a real number
- Assume mean μ and variance σ^2 (both real numbers)
- Probability density function is

$$N(X, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(X-\mu)^2}{\sigma^2}}$$

Normal Probability Density Function in 1D



Normal Distribution: Probability Density Function

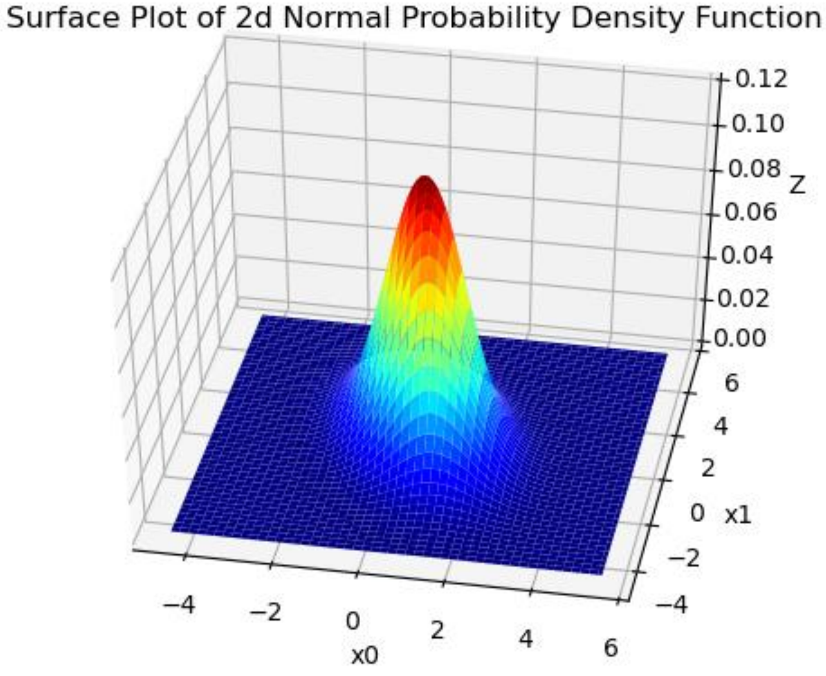
Probability density function for d dimensional case

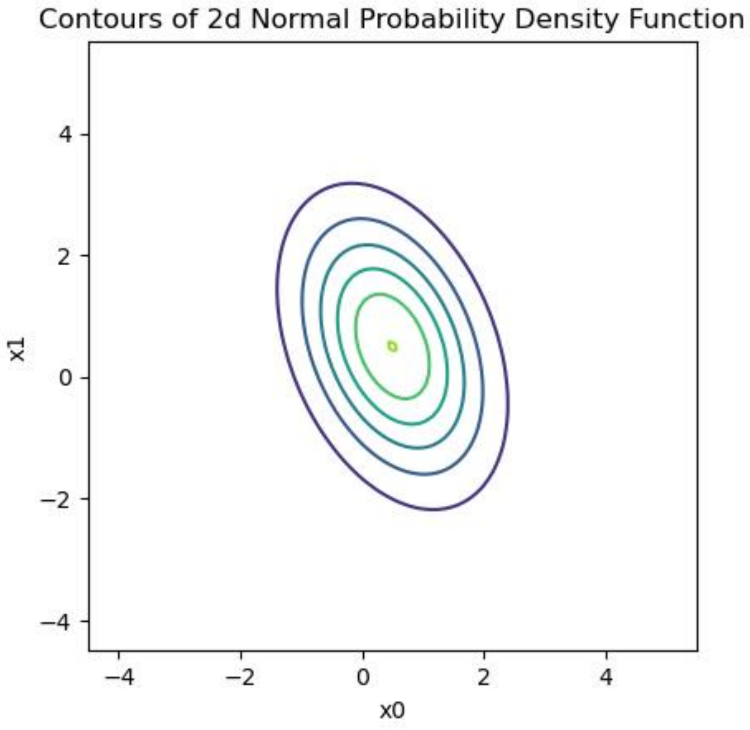
- Let X be a d-dimensional column vector
- Assume mean μ (d-dimensional column vector)
- Assume covariance matrix Σ (dxd matrix)
  - Covariance matrix is symmetric
  - Covariance matrix is assumed to be positive-definite (eigenvalues > 0)
- Let $|\Sigma|$ denote the determinant of Σ
- Probability density function is

$$N(X, \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(X-\mu)^T \Sigma^{-1} (X-\mu)}$$

Normal Probability Density Function in 2D

Mean: $\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$ Covariance $\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 2 \end{bmatrix}$ X = $\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$

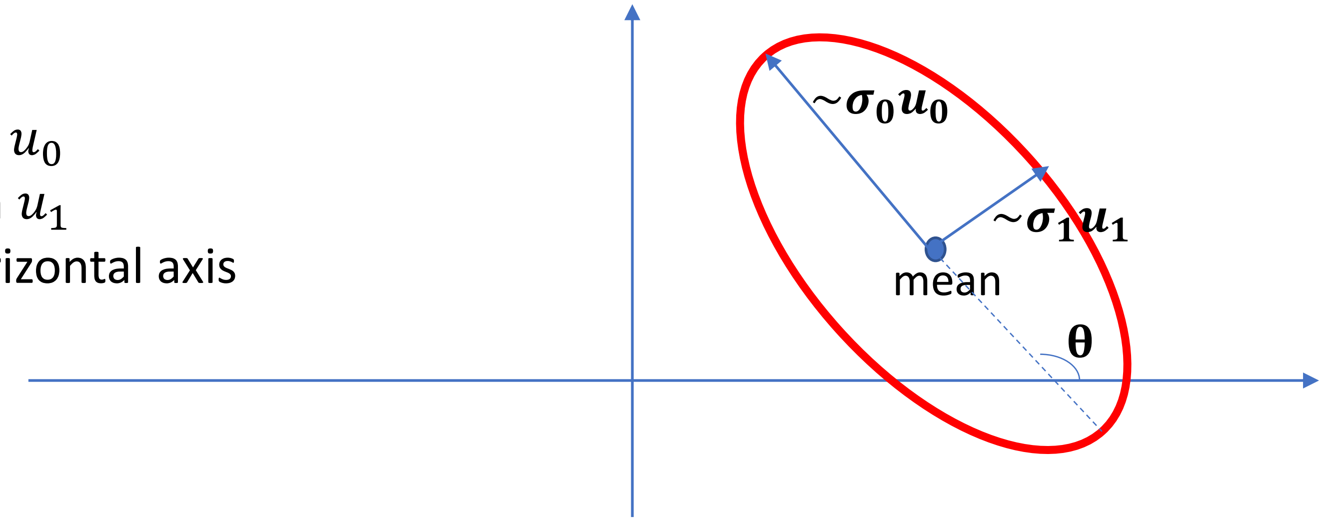




Normal Probability Density Function Contours

- Contours are curves in x0-x1 plane where $N(X, \mu, \Sigma)$ is constant
- Neat connection between contours and SVD
- Idea generalizes to higher dimensions (contours (level sets) are ellipsoids)
- In 2D: Mean: $\begin{bmatrix} \mu_0 \\ \mu_1 \end{bmatrix}$ Covariance: Σ SVD: $\Sigma = \begin{bmatrix} u_0 & u_1 \end{bmatrix} \begin{bmatrix} \sigma_0 & 0 \\ 0 & \sigma_1 \end{bmatrix} \begin{bmatrix} v_0^T \\ v_1^T \end{bmatrix}$

Mean is the centre of ellipse
Width proportional to σ_0 in direction u_0
Height proportional to σ_1 in direction u_1
Angle θ is angle made by u_0 with horizontal axis



Computational Complexity

- Determinant, inverse calculations require $O(d^3)$ operations as $d \to \infty$
- For each X, operations to compute $N(X, \mu, \Sigma)$ is $O(d^3)$ as $d \to \infty$
- Amount of memory for the calculation is $O(d^2)$ as $d \to \infty$

Normal Distribution PDF DEMO

Jupyter Notebook for demo:

- UnsupervisedML/Examples/Section08/Normal.ipynb

Course Resources at:

- https://github.com/satishchandrareddy/UnsupervisedML/

Section 7.2: Gaussian Mixture Model: Algorithm

Gaussian Mixture Model

- GMM is an approach for identifying clusters in a dataset
- Number of clusters is specified
- For each data point a probability of belonging to each cluster is computed and the point is assigned to cluster with highest probability
- Probabilities are based on normal distribution for each cluster
- Goal is to find mean, covariance, and weighting for normal distribution for each cluster
- Note this approach can be used with other distributions

GMM: Probability Density Function for Mixture

- Assume data points $X_0, X_1, X_2, \ldots, X_{M-1}$ in d dimensions
- Assume that there are K clusters:
  - Cluster k, denoted C_k, has mean μ_k, covariance Σ_k, and weight ϕ_k
  - Note that weights satisfy $\phi_0 + \cdots + \phi_{K-1} = 1$
- Probability density function for the mixture of Gaussians is:

$$P(X) = \sum_{k=0}^{K-1} \phi_k N(X, \mu_k, \Sigma_k)$$

- GMM: find most likely means, covariances, and weights for given set of data

GMM: Probability Density Function for Mixture

- Probability density of X and it is part of cluster k:

$$P(X \cap C_k) = \phi_k N(X, \mu_k, \Sigma_k)$$

- Conditional probability that data point is in cluster k given X is

$$P(C_k|X) = \frac{P(X \cap C_k)}{P(X)} = \frac{\phi_k N(X, \mu_k, \Sigma_k)}{\sum_{k=0}^{K-1} \phi_k N(X, \mu_k, \Sigma_k)}$$

GMM: Maximum Likelihood Estimation

- Joint probability density for $X_0, \ldots, X_{M-1}$ is given by likelihood function:

$$P(X_0,\ \ldots, X_{M-1}) = \prod_{i=0}^{M-1} P(X_i) = \prod_{i=0}^{M-1} \sum_{k=0}^{K-1} \phi_k N(X_i, \mu_k, \Sigma_k)$$

- Maximum Likelihood Estimation attempts to find the distributions (values of means $\{\mu_k\}$, covariances $\{\Sigma_k\}$, and weights $\{\phi_k\}$) that have the maximum likelihood for the given data points
- This is accomplished by maximizing the likelihood function subject to the constraint $\phi_0 + \cdots + \phi_{K-1} = 1$
- In practice, maximize the log likelihood function:

$$L = \log P(X_0,\ \ldots, X_{M-1}) = \sum_{i=0}^{M-1} \log \left[\sum_{k=0}^{K-1} \phi_k N(X_i, \mu_k, \Sigma_k) \right]$$

GMM: Solution for Expectation Maximization

- Use the method of Lagrange multipliers to determine solution of maximization problem. See Resources file ExpectationMaximization.pdf
- Define:

$$\gamma_{ki} = \frac{\phi_k N(X_i, \mu_k, \Sigma_k)}{\sum_{k=0}^{K-1} \phi_k N(X_i, \mu_k, \Sigma_k)}$$

- Can show

$$\mu_k = \frac{\sum_{i=0}^{M-1} \gamma_{ki} X_i}{\sum_{i=0}^{M-1} \gamma_{ki}} \quad \Sigma_k = \frac{\sum_{i=0}^{M-1} \gamma_{ki} (X_i - \mu_k)(X_i - \mu_k)^T}{\sum_{i=0}^{M-1} \gamma_{ki}} \quad \phi_k = \frac{\sum_{i=0}^{M-1} \gamma_{ki}}{M}$$

- Note that we can't solve the above the above equations as both the left and right sides contain $\{\mu_k\}$, $\{\Sigma_k\}$, $\{\phi_k\}$
- Need to use iterative approach: Expectation Maximization algorithm

GMM: Expectation Step

- Input: data points data points $X_0, X_1, X_2, \ldots, X_{M-1}$
- Input: most recently computed means $\{\mu_k\}$, covariances $\{\Sigma_k\}$, weights $\{\phi_k\}$
- Update conditional probabilities:

$$\gamma_{ki} = \frac{\phi_k N(X_i, \mu_k, \Sigma_k)}{\sum_{k=0}^{K-1} \phi_k N(X_i, \mu_k, \Sigma_k)}$$

- Recall that γ_{ki} is the probability that point X_i is in cluster C_k. Hence the most likely cluster for X_i is

$$Cluster(X_i) = argmax_k(\gamma_{ki})$$

GMM: Maximization Step

- Input: data points data points $X_0, X_1, X_2, \ldots, X_{M-1}$
- Input: most recently computed conditional probabilities $\{\gamma_{ki}\}$
- Update estimated number of points in cluster k:

$$M_k = \sum_{i=0}^{M-1} \gamma_{ki}$$

- Update Weights:

$$\phi_k = \frac{M_k}{M}$$

- Update Means:

$$\mu_k = \frac{1}{M_k} \sum_{i=0}^{M-1} \gamma_{ki} X_i$$

- Update Covariances:

$$\Sigma_k = \frac{1}{M_k} \sum_{i=0}^{M-1} \gamma_{ki} X_i X_i^T$$

GMM: Initialization

- Input: data points $X_0, X_1, X_2, \ldots, X_{M-1}$
- Initial weights (pick to be all the same):

$$\phi_k = \frac{1}{K} \quad \mathrm{k} = 0, \ldots, \mathrm{K} - 1$$

- Initial means:
  - Random approach: pick K points randomly from among data points
  - K Means ++ approach: use K means ++ approach

- Initial covariances: compute covariance of all data points and use same value for all clusters

$$\Sigma_k = \frac{1}{M} \sum_{i=0}^{M-1} (X_i - \mu)(X_i - \mu)^T \quad \mathrm{k}=0, \ldots, K-1$$

GMM: Expectation Maximization Algorithm

- Input: data points data points $X_0, X_1, X_2, \ldots, X_{M-1}$
- Specify: number of clusters K
- Specify: tolerance for stopping iteration

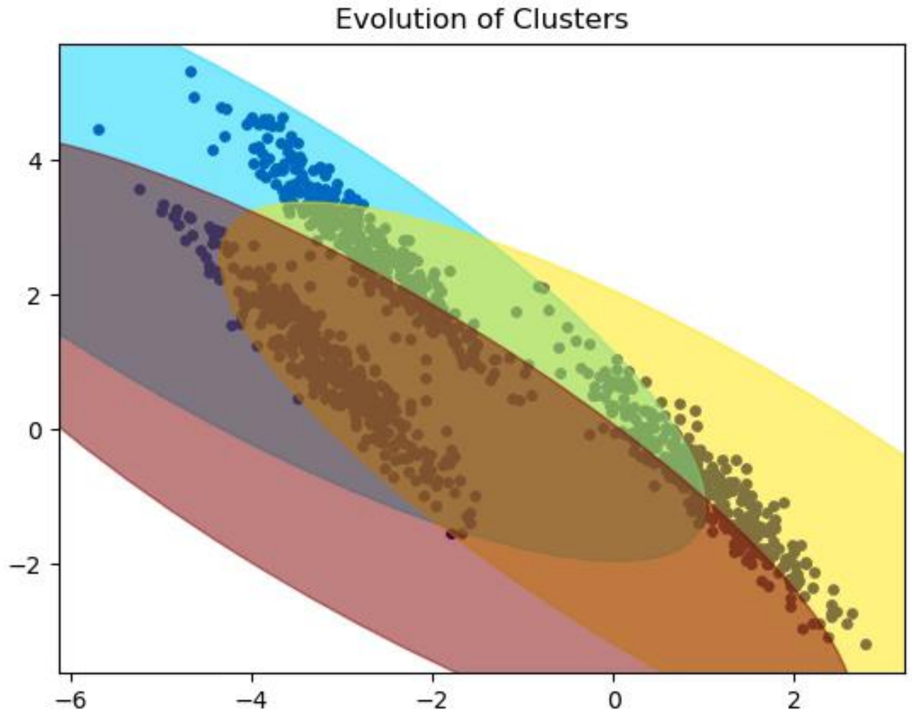
(1) Initialization: compute initial means $\{\mu_j\}$, covariances $\{\Sigma_j\}$, weights $\{\phi_j\}$

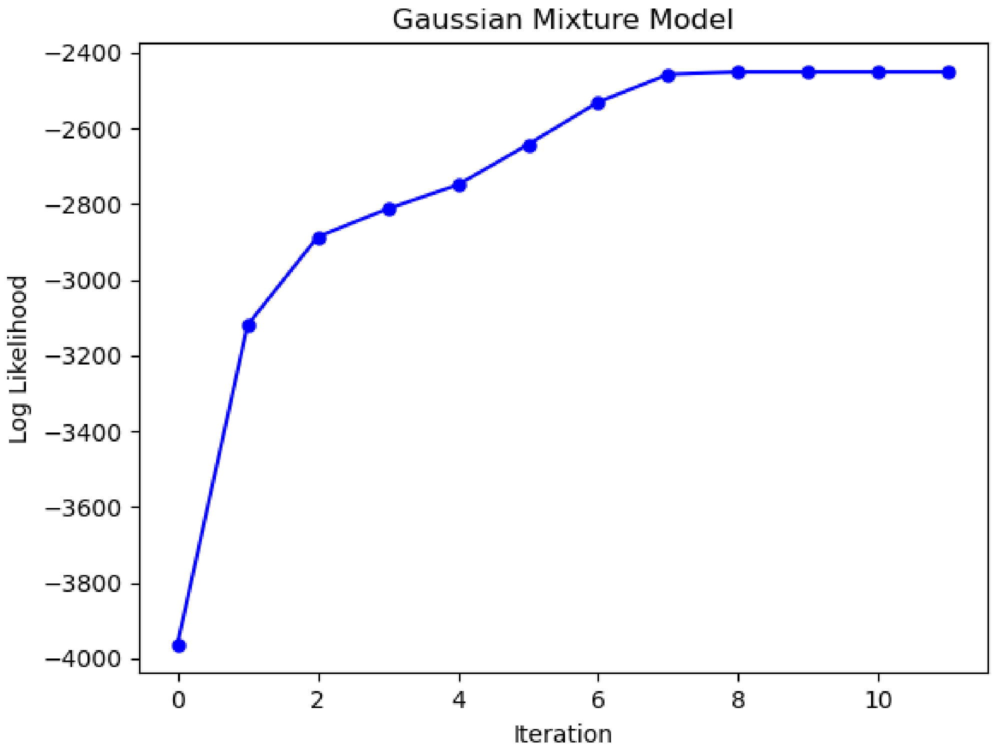
(2) While maxdist>tolerance

- Expectation step: update conditional probabilities $\{\gamma_{ki}\}$,
- Maximization step: update means $\{\mu_k\}$, covariances $\{\Sigma_k\}$, weights $\{\phi_k\}$
- Compute maximum distance between current and previous means

GMM: Example

- Dataset: sklearn "aniso" dataset with 1000 points
- Specify 3 means pick initial means from random from data points
- Specify stopping tolerance of 10^{-5}





GMM: Complexity

Expectation Step: compute MK coefficients

- Each coefficient requires $O(d^3)$ operations to compute normal

Maximization Step: compute K means, K covariances, and K weights

- Each mean computation is $O(Md)$
- Each covariance computation is $O(Md^2)$
- Each weight computation is $O(M)$

Assume K, d, and number of iterations are fixed, then GMM takes $O(M)$ operations

GMM: Notes

- User must specify number of clusters
- Can use an elbow type approach based on log likelihood function to determine appropriate number of clusters
- No guarantee that global maximum of log likelihood function is found – may get to local maximum
- May be memory or speed issues if number of dimensions d is large

Section 7.3: Gaussian Mixture Model: Code Design

Gaussian Mixture Model Code Design

- This section contains information about design of the GMM Code
- Design is based on algorithm described in Section 8.2
- Stop video here, if you would like to do code design yourself

Gaussian Mixture Model Code Design: To Do

Component	Description
class gaussianmm	class gaussianmm derived from clustering_base class and relevant variables and methods
driver_gaussianmm	driver for Gaussian Mixture Model
normal	function for computing multi-dimensional normal distribution probability density function (pdf)
create_ellipse_patch_details	function for generating ellipse that shows “footprint” of normal pdf

class gaussianmm: Principal Variables

Variable	Type	Description
self.meansave	list of list of means	Contains cluster means for each iteration Example with 2 iterations and 3 clusters $[[\begin{bmatrix}1\\2\end{bmatrix}, \begin{bmatrix}2\\1\end{bmatrix}, \begin{bmatrix}3\\2\end{bmatrix}],[\begin{bmatrix}3\\1\end{bmatrix}, \begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}3\\3\end{bmatrix}]]$ self.meansave[i][j] is the mean for iteration i and cluster j
self.Sigmasave	list of list of covariance matrices	Contains the covariance matrices for each iteration Example with 2 iterations and 3 clusters $[[\begin{bmatrix}1 & 2\\2 & 2\end{bmatrix}, \begin{bmatrix}3 & 1\\1 & 2\end{bmatrix}, \begin{bmatrix}3 & 2\\2 & 2\end{bmatrix}],[\begin{bmatrix}1.5 & 1\\1 & 1.5\end{bmatrix}, \begin{bmatrix}0.5 & 1\\1 & 1.5\end{bmatrix}, \begin{bmatrix}1.2 & 1.1\\1.1 & 1.2\end{bmatrix}]]$ self.Sigmasave[i][j] is the covariance matrix for iteration i and cluster j
self.weightsave	list of list of weights	Contains the weights for each cluster for each iteration Example 2 iterations and 3 clusters [[0.2, 0.3, 0.5], [0.3, 0.3, 0.4]]
self.gamma	2d numpy array	Contains the conditional probabilities γ_{ki} computed in the Expectation Step

gaussianmm class: Key Methods

Method	Input	Description
initialize_algorithm	initialization (string)	Initialize self.clustersave, self.loglikelihoodsave, self.Sigmasave, and self weightsave. Initialize self.meansave using “random” or “kmeans++” initialization
fit	X (2d numpy array) max_iter (integer) tolerance(float)	Performs Gaussian Mixture Model approach until distance between current and previous means is less than tolerance. Take at most max_iter iterations Return: self.loglikelihoodsave
expecation		Performs expectation step for Gaussian Mixture Model – specifically, updates self.gamma
maximization		Performs maximization step for Gaussian Mixture Model – specifically, updates self.meansave, self.Sigmasave, self.weightsave
update_cluster_assignment		Updates cluster assignments based on current self.gamma

gaussianmm class: Key Methods

Method	Input	Description
compute_distance	X (2d numpy array) list_cluster (list of cluster means)	Compute distance between each data point and point in list_cluster Return: 2d array containing distance between each data point and each point in list_cluster
compute_diff		Determine maximum distance between current and previous estimate for means Return: maximum difference in means
plot_cluster	level (integer) title,xlabel,ylabel (strings)	Plot the data points with cluster assignments and the footprints of each of the normal distributions for given iteration (level) Return: nothing
plot_cluster_animation	level (integer) title,xlabel,ylabel(string)	Creates animation showing data points and evolution cluster assignments and the footprints of each of the normal distributions Return: nothing

Additional Functions

Method	Input	Description
normal	X (2d numpy array) mu (numpy column array) Cov (2d numpy array)	Given the mean, and covariance matrix, this function computes the normal pdf for each of the data points in X Return: array of normal pdf values
create_ellipse_patch_details	mu (numpy column array) Cov (2d numpy array) weight (float) contour (float)	This function finds the ellipse in x0-x1 plane for which weighted normal pdf is equal to contour. Return: centre, width, height, and angle for ellipse

Section 7.4: Gaussian Mixture Model: Code Walkthrough

GMM Clustering: Code Walkthrough

- Code is located in:

  Folder: UnsupervisedML/Code/Programs

  Files: normal.py, gaussianmm.py, driver_gaussianmm.py, normal.py
- Stop video here, if you would like to do coding yourself before seeing my implementation