Unsupervised Machine Learning with Python

Section 7.1: Normal Distribution Probability Density Function

Why do we need Probability Density Functions

- Probability density functions (PDFs) are used for distribution-based clustering approaches
- Specifically for the Gaussian Mixture Model clustering, need the pdf for the normal distribution

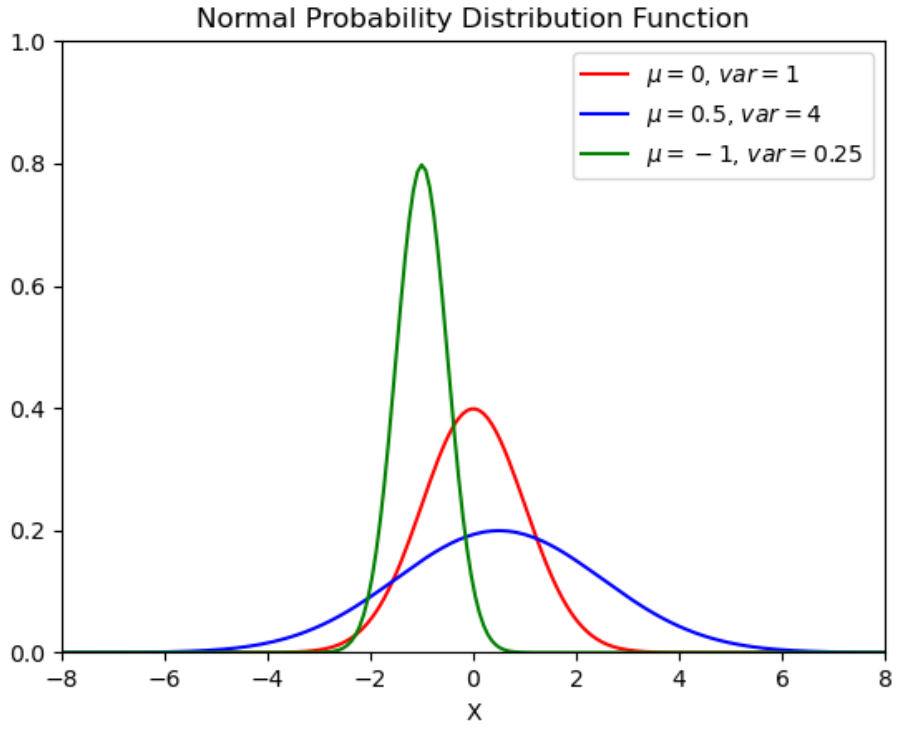
Normal Distribution PDF 1D

Probability density function for 1 dimensional case

- Let X be a real number
- Assume mean μ and variance v
- Probability density function:

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

- Area under pdf curve is 1



Normal Distribution PDF in Multiple Dimensions

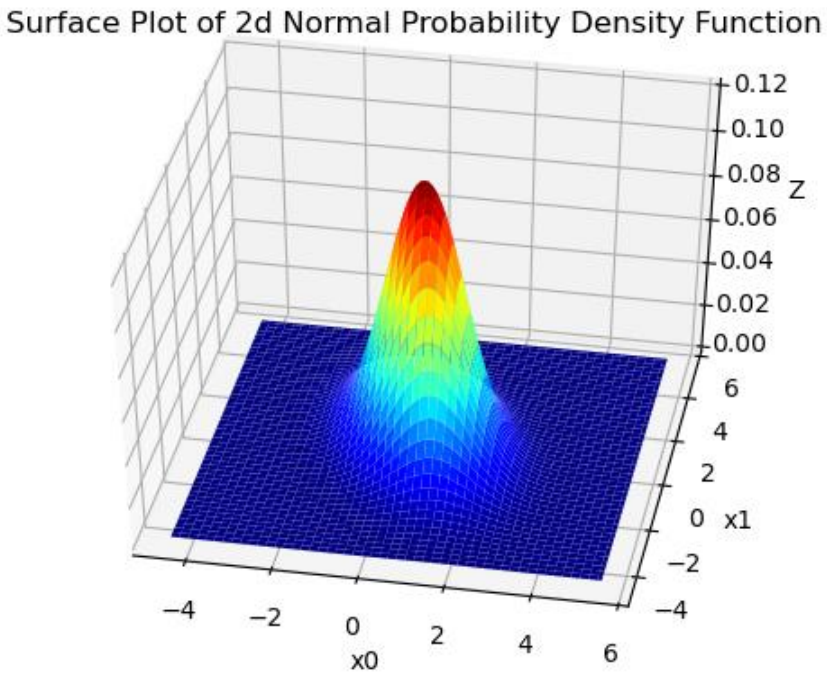
Probability density function for d dimensional case

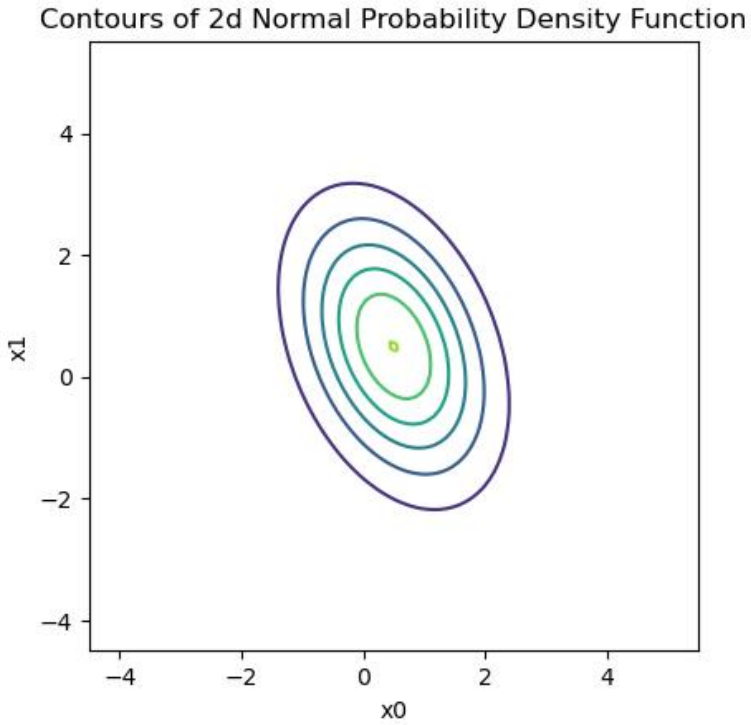
- Let X be a d-dimensional column vector
- Assume mean μ (d-dimensional column vector)
- Assume covariance matrix Σ (dxd matrix)
  - Covariance matrix is symmetric
  - Covariance matrix is assumed to be positive-definite (eigenvalues > 0)
- Let $|\Sigma|$ denote the determinant of Σ
- Probability density 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 Distribution PDF in 2D

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





Normal Distribution PDF Contours in 2D

Details in UnsupervisedML/Resources/UnsupervisedML_GMM.pdf

- Contours are curves (ellipses) in x0-x1 plane where $N(X, \mu, \Sigma) = \text{c}$ is constant
- Neat connection between contours and SVD of Σ
- In 2D: Mean: $\begin{bmatrix}\mu_0\\ \mu_1\end{bmatrix}$ Covariance: Σ SVD: $\Sigma = [u_0 \quad u_1]\begin{bmatrix}\sigma_0 & 0\\ 0 & \sigma_1\end{bmatrix}\begin{bmatrix}v_0^T\\ v_1^T\end{bmatrix}$

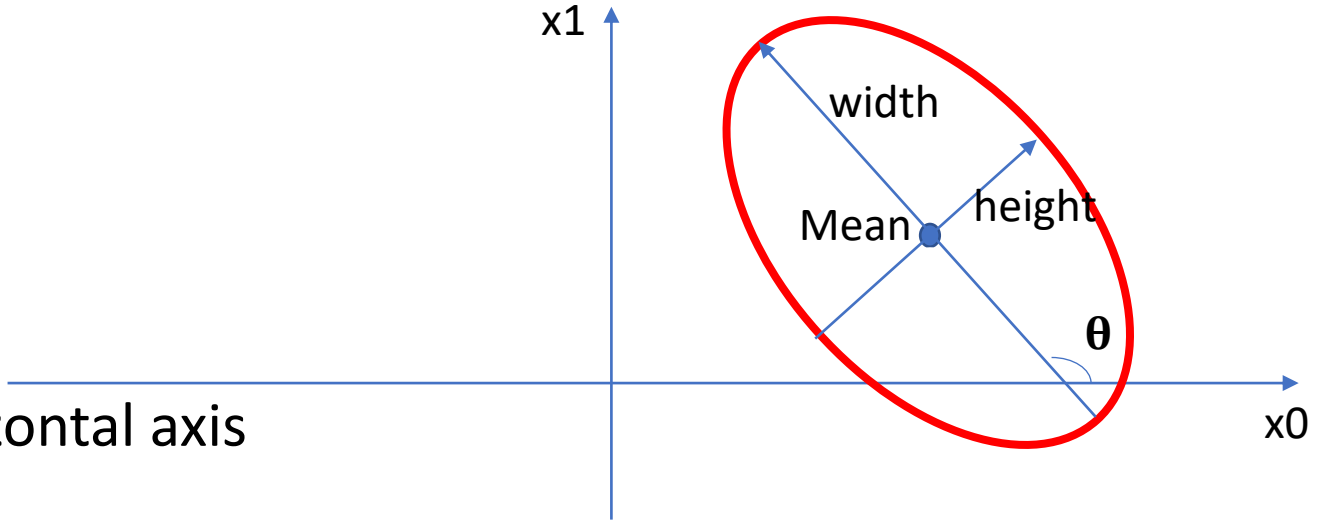
$$\alpha = \sqrt{-2\log[\text{c}2\pi\sqrt{|\Sigma|}]}$$

Mean = centre of ellipse

width = $2\alpha\sqrt{\sigma_0}$ in the u_0 direction

height = $2\alpha\sqrt{\sigma_1}$ in the u_1 direction

θ = angle made by u_0 with horizontal axis



Normal Probability Density Function Contours

- In higher dimensions contour is “multi-dimensional ellipsoid”
- Axis k is parallel to u_k (k’th column of U of SVD)
- Length along axis k of ellipsoid is proportional to $\sqrt{\sigma_k}$

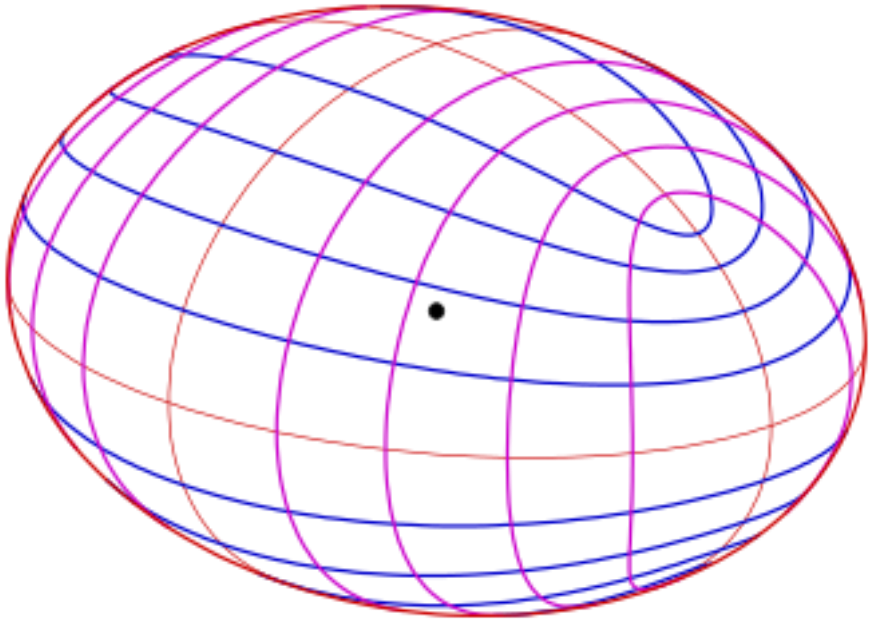


Image from Wikipedia Commons
https://en.m.wikipedia.org/wiki/File:Ellipsoid-kl.svg

Computational Complexity

For computation of pdf in d dimensions:

- Determinant, inverse calculations require $O(d^3)$ operations as $d \to \infty$
- 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 $d \to \infty$
- If X is a dataset of M samples, then operations to compute $N(X, \mu, \Sigma)$ for all samples is $O(M)$ as $M \to \infty$

Normal Distribution PDF DEMO

Jupyter Notebook for demo:

- UnsupervisedML/Examples/Section07/Normal.ipynb

Course Resources at:

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

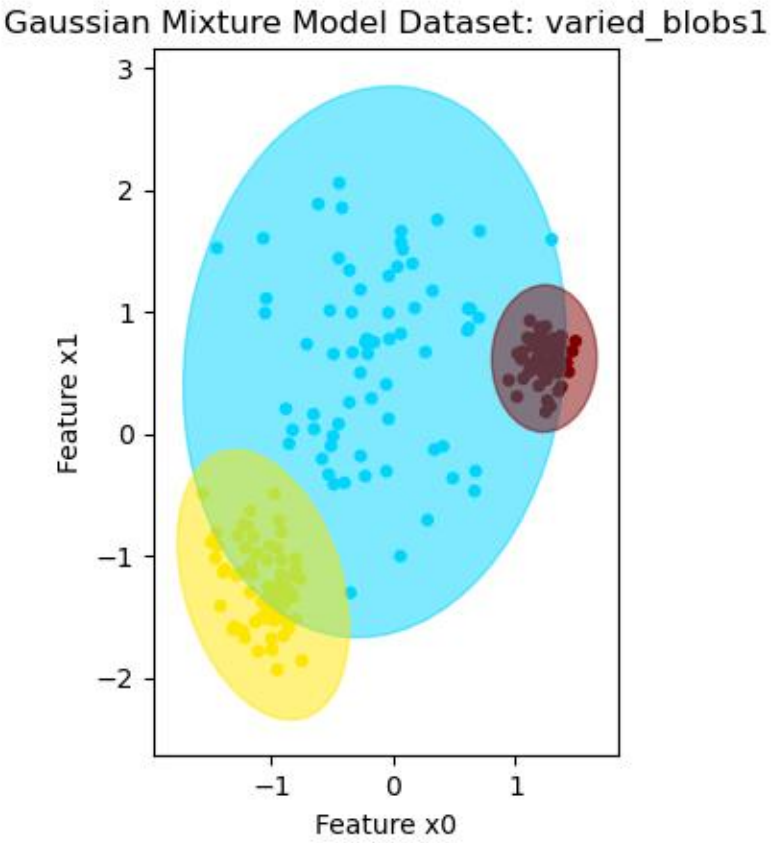
Unsupervised Machine Learning with Python

Section 7.2: Gaussian Mixture Model Algorithm

Gaussian Mixture Model

- GMM is a distribution based approach for identifying clusters in a dataset
- GMM is a “soft” clustering method – for each data point probability of belonging to each cluster is computed and point assigned to cluster with highest probability
- Probabilities are based on normal distribution probability density function (pdf) for each cluster
- This approach can be used with other distributions
- See UnsupervisedML_Resources.pdf for links to additional resources

GMM: Visualization of Results



Example in 2D with 3 clusters

- Filled elliptical regions represent 3 Gaussians in mixture
- Color of data point indicates cluster to which it most likely belongs

GMM: Probability Density Function for Mixture

- For a single Gaussian (d dimensions), density function is normal distribution pdf:

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

- Assume that there are K clusters:
  - Cluster k, denoted S_k, has mean μ_k, covariance matrix Σ_k, and weight ϕ_k
  - 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: Maximum Likelihood Estimation

- Assume data points: X_0, …, X_{M-1}
- Joint distribution pdf 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 $\{\mu_k\}$, covariance matrices $\{\Sigma_k\}$, 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: Expectation Maximization

- Cannot solve maximization problem exactly so employ Expectation Maximization algorithm
  - Make an initial guesses for means $\{\mu_k\}$, covariance matrices $\{\Sigma_k\}$, weights $\{\phi_k\}$
  - Iteratively improve the guesses until convergence
    - Perform Expectation Step
    - Perform Maximization Step

- Details in UnsupervisedML/Resources/UnsupervisedML_GMM.pdf
  - Use Lagrange multipliers approach from multi-variable calculus

GMM: Expectation Step

- Input: data points $X_0, X_1, X_2, \ldots, X_{M-1}$
- Input: current means $\{\mu_k\}$, covariance matrices $\{\Sigma_k\}$, weights $\{\phi_k\}$
- Update conditional probabilities for cluster k=0,…,K-1 and points i=0,…,M-1

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

(γ matrix is K rows and M columns)

- Update Log Likelihood function value:

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

- Update Cluster assignment: Most likely cluster to which X_i belongs is k that gives the largest value of γ_{ki}

GMM: Maximization Step

- Input: 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 Matrices:

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

GMM: Initialization

- Initial weights (all the same):

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

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

- Initial covariances: compute covariance matrix of all data points and use same value for all clusters: let μ denote the mean of the entire data set

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

GMM: Expectation Maximization Algorithm

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

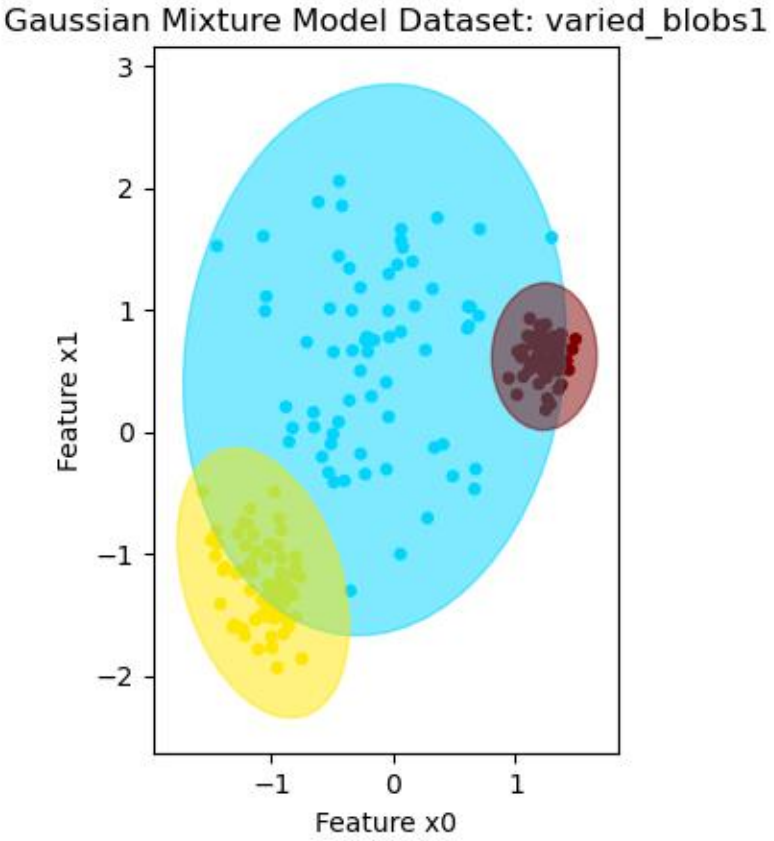
(1) Initialization: compute initial means $\{\mu_k\}$, covariances $\{\Sigma_k\}$, weights $\{\phi_k\}$

(2) While change in cluster means is greater than ε

- Expectation step: update conditional probabilities $\{\gamma_{ki}\}$ and log likelihood function
- Compute the cluster assignments based on $\{\gamma_{ki}\}$
- Maximization step: update means $\{\mu_k\}$, covariance matrices $\{\Sigma_k\}$, weights $\{\phi_k\}$
- Compute change (distance) between current and previous cluster means

Typically, also specify a maximum number of iterations in while loop

GMM: Visualization of Results

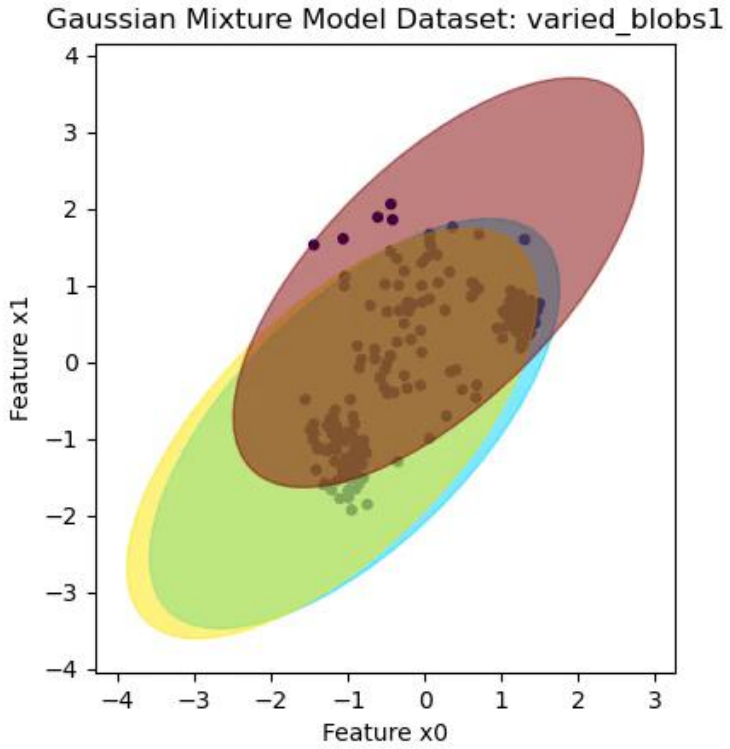


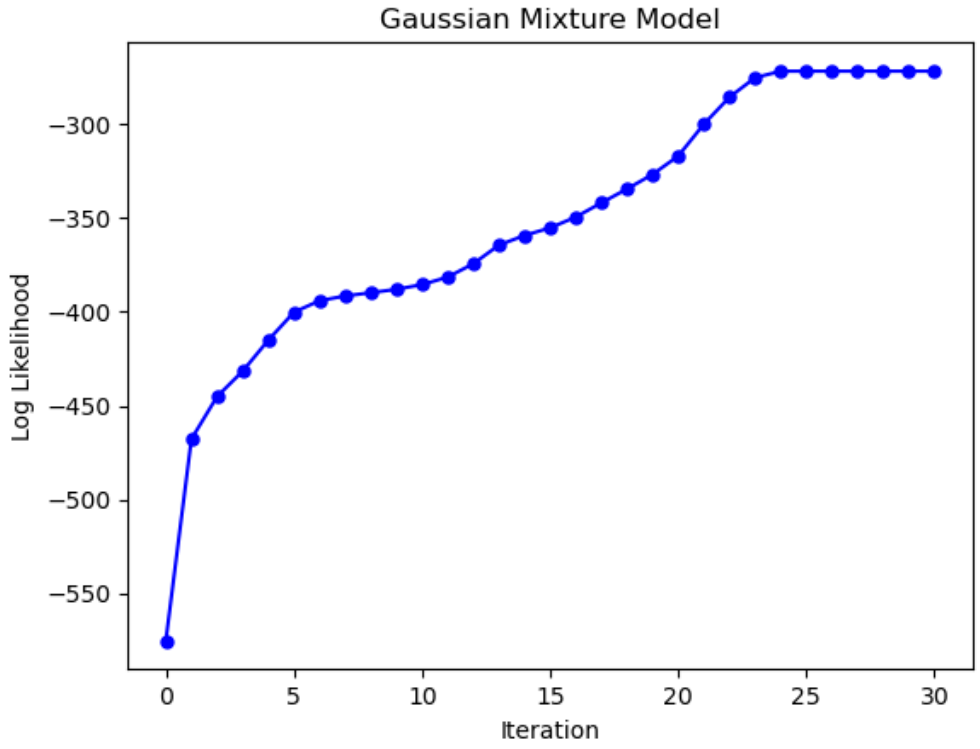
Example in 2D with 3 Clusters

- Show “filled” elliptical region for each Gaussian in mixture
- Each elliptical region shows where
$$\phi_k N(X, \mu_k, \Sigma_k) \geq 0.002$$
  - Size: depends on weight
  - Centre: depends on mean
  - Ellipse: width/height ratio and angle depend on covariance matrix
- Color of data point indicates cluster to which it most likely belongs
- In higher dimensions regions are multi-dimensional ellipsoids

GMM: Example

- Dataset: sklearn varied_blobs1 data set with 200 points
- Specify 3 clusters and pick initial means at random from data points
- Set stopping tolerance $\varepsilon = 10^{-5}$





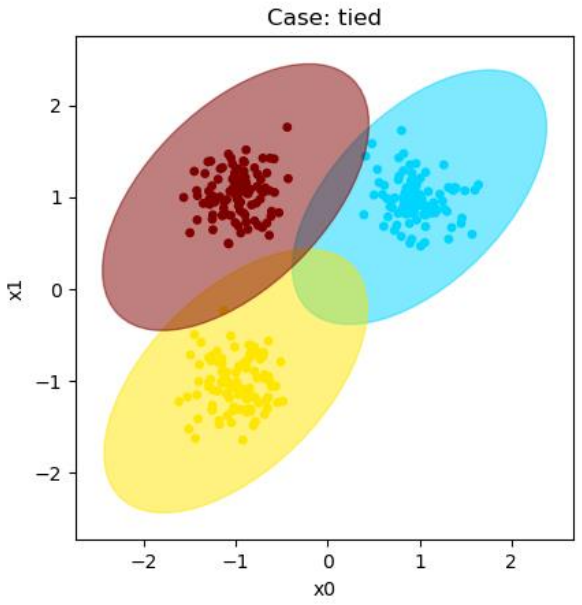
GMM: Complexity

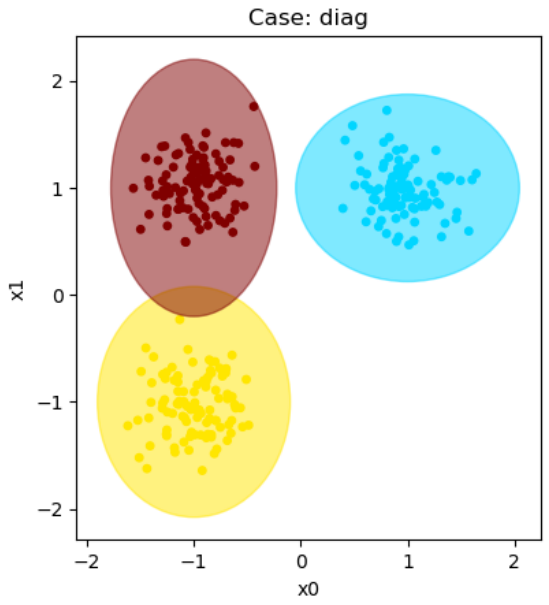
Assume dataset with M data points in d dimensions

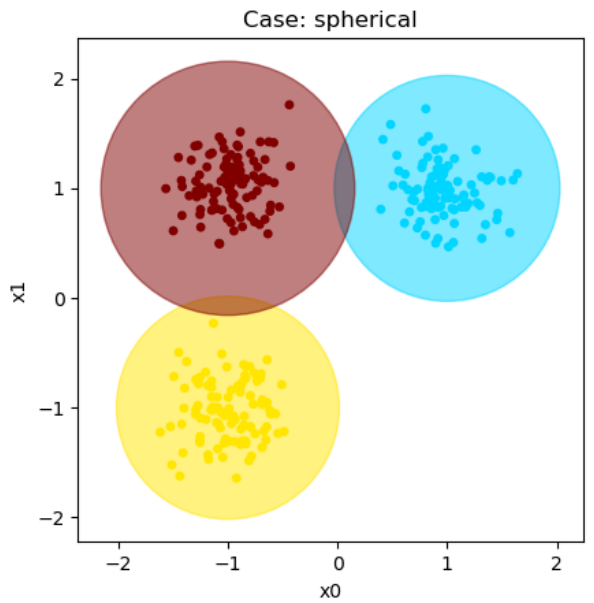
- Can show that GMM takes $O(M)$ operations as $M \rightarrow \infty$ (assumes that the number of iterations is bounded as number of data points increases)
- Memory requirement is $O(M)$ as $M \rightarrow \infty$
- Number of operations is proportional to d^3 (need to compute inverse and determinant of covariance matrices)

GMM Variations

- sklearn has 4 variations of GMM defined by the Covariance matrices
  - “full”: each component of mixture has unique covariance matrix (described in this section)
  - “tied”: same covariance matrix for all components, so elliptical regions have same orientation and proportions
  - “diag”: each component has own diagonal covariance matrix, so elliptical regions have width and height directions parallel to axes
  - “spherical”: each component has its own single variance value, so elliptical regions circles







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
- Note that there can be issues if dimension d is extremely large or if determinant $|\Sigma|$ is close to 0, as constant factor for normal pdf is

$$\frac{1}{\sqrt{(2\pi)^d |\Sigma|}}$$

and one must take inverse of Σ

- Regularization techniques deal with case when determinant is close to 0

Unsupervised Machine Learning with Python

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 7.2
- Stop video here, if you would like to do code design yourself

Gaussian Mixture Model Code Design

(1) Create function for computing multidimensional normal distribution pdf

(2) Create function for generating properties of contour ellipse of normal distribution pdf in 2d

(3) Derive gaussianmm class from clustering_base class

(4) Create Gaussian Mixture Model versions of plot_cluster and plot_cluster_animation to be able to plot both cluster assignments and normal distribution pdf elliptical regions

(5) Use numpy functionality (vectorization) to minimize explicit looping

Computing Conditional Probabilities

- Input: data points X_0, …,X_{M-1} means $\{\mu_k\}$, covariances $\{\Sigma_k\}$, weights $\{\phi_k\}$
- Need to compute

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

(1) Compute normals: $N(X_i, \mu_k, \Sigma_k)$ for k=0,…,K-1 and i=0,…,M-1

Example 5 datapoints and 2 clusters

$$N = \begin{bmatrix} 0.1 & 0.4 & 0.3 & 0.5 & 0.4 \\ 0.1 & 0.2 & 0.1 & 0.5 & 0.2 \end{bmatrix}$$

Cluster 0

Cluster 1

(2) Multiply each row by corresponding weight (Ex $\phi_0 = 0.4$, $\phi_1 = 0.6$)

$$weighted\ N = \begin{bmatrix} 0.04 & 0.16 & 0.12 & 0.20 & 0.16 \\ 0.06 & 0.12 & 0.06 & 0.30 & 0.12 \end{bmatrix}$$

Computing Conditional Probabilities

(3) Divide each entry of weighted N by sum of its column to get γ

$$weighted\ N = \begin{bmatrix} 0.04 & 0.16 & 0.12 & 0.20 & 0.16 \\ 0.06 & 0.12 & 0.06 & 0.30 & 0.12 \end{bmatrix}$$

$$\gamma = \begin{bmatrix} \frac{0.04}{0.04+0.06} & \frac{0.16}{0.16+0.12} & \frac{0.12}{0.12+0.06} & \frac{0.20}{0.20+0.30} & \frac{0.16}{0.16+0.12} \\ \frac{0.06}{0.04+0.06} & \frac{0.12}{0.16+0.12} & \frac{0.06}{0.12+0.06} & \frac{0.30}{0.20+0.30} & \frac{0.12}{0.16+0.12} \end{bmatrix}$$

Computing Log Likelihood and Cluster Assignments

Log Likelihood:

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

Implementation Details:

- Sum weighted N for each column
- Take log
- Sum result

Cluster assignment:

- Most likely cluster to which X_i belongs is k that gives the largest value of γ_{ki}

Implementation Details:

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

Maximization Step: Implementation Details

- Update estimated number of points in cluster k:

$M_k = \sum_{i=0}^{M-1} \gamma_{ki}$ Sum of row k of γ

- Weight for cluster k:

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

- Mean for cluster k:

$\mu_k = \frac{1}{M_k} \sum_{i=0}^{M-1} \gamma_{ki} X_i$ Data points weighted by row k of γ

- Covariance for cluster k:

$\Sigma_k = \frac{1}{M_k} \sum_{i=0}^{M-1} \gamma_{ki} (X_i - \mu_k)(X_i - \mu_k)^T$ Covariance (with mean μ_k) weighted by row k of γ

Class gaussianmm: Principal Variables

Variable	Type	Description
self.time_fit	float	Time for clustering
self.objectivesave	list	Value of the log likelihood function for each iteration Example with 3 iterations [-500,-400,-300]
self.X	2d numpy array	Dataset Number of rows = number of dimensions for data Number of cols = number of data points Example: 2 dimensions and 5 data points $\begin{bmatrix} 1 & 1.1 & 0.8 & 0.6 & 0.6 \\ 0.9 & 1.0 & 0.7 & 0.5 & 0.5 \end{bmatrix}$
self.clustersave	list of 1d numpy arrays	self.clustersave[i][j] is cluster assignment for iteration i, data point j Example: for 3 iterations: $[[-1 \quad -1 \quad -1 \quad -1 \quad -1], [0 \quad 1 \quad 1 \quad 0 \quad 1], [0 \quad 0 \quad 1 \quad 0 \quad 1]]$

Class gaussianmm: Principal Variables

Variable	Type	Description
self.meansave	list of list of means	self.meansave[i][j] is the mean for iteration i and cluster j Example with 3 iterations and 2 clusters $[[\begin{bmatrix}2\\1\end{bmatrix},\begin{bmatrix}3\\2\end{bmatrix}],[\begin{bmatrix}2\\2\end{bmatrix},\begin{bmatrix}3\\3\end{bmatrix}],[\begin{bmatrix}1\\4\end{bmatrix},\begin{bmatrix}4\\3\end{bmatrix}]]$
self.Covsave	list of list of covariance matrices	self.Covsave[i][j] is the covariance matrix for iteration i and cluster j Example with 3 iterations and 2 clusters $[[\begin{bmatrix}1 & 0.5\\0.5 & 2\end{bmatrix},\begin{bmatrix}3 & 1\\1 & 2\end{bmatrix}],[\begin{bmatrix}1.5 & 1\\1 & 1.5\end{bmatrix},\begin{bmatrix}2.5 & 1\\1 & 1.5\end{bmatrix}],[\begin{bmatrix}1.2 & 1.1\\1.1 & 1.3\end{bmatrix},\begin{bmatrix}1.5 & 1.1\\1.1 & 1.3\end{bmatrix}]]$
self.weightsave	list of list of weights	self.weightsave[i][j] is weight for iteration i and cluster j Example 3 iterations and 2 clusters [[0.7, 0.3], [0.5, 0.5], [0.6, 0.4]]
self.gamma	2d numpy array	Contains the conditional probabilities γ_{ki} computed in Expectation step

Class gaussianmm: Key Methods

Method	Input	Description
__init__	ncluster (integer) initialization (string)	Constructor for the class – input the number of clusters and initialization method (“random” or “kmeans++”)
initialize_algorithm		Initialize self.clustersave, self.objectivsave, self.Covsave, and self.weightsave. Initialize self.meansave using “random” or “kmeans++” initialization
fit	X (2d numpy array) max_iter (integer) tolerance (float) verbose (boolean)	Performs Gaussian Mixture Model approach until distance between current and previous means is less than tolerance. Take at most max_iter iterations Return: nothing
expectation		Compute the conditional probabilities γ_{ki} and log likelihood function Return: nothing (update self.gamma and self.objectivesave)
maximization		Update means $\{\mu_k\}$, covariances $\{\Sigma_k\}$, and weights $\{\phi_k\}$ Return: nothing (update self.meansave, self.Covsave, self.weightsave)

Class gaussianmm: Key Methods

Method	Input	Description
compute_distance2	list_cluster (list of cluster means)	Compute distance squared between each data point and point in list_cluster Return: 2d array containing squared distances
update_cluster_ assignment		Compute the cluster assignments based on self.gamma Return: nothing (update self.clustersave)
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 contour of the normal distributions for given iteration (level) Return: nothing See UnsupervisedML/Examples/Section02/MatplotlibAdvanced.ipynb and Section07/Normal.ipynb
plot_cluster_ animation	level (integer) interval (float) title,xlabel,ylabel(string)	Creates animation showing data points and evolution of cluster assignments and contours of Gaussian distributions Return: nothing See UnsupervisedML/Examples/Section02/MatplotlibAdvanced.ipynb and Section07/Normal.ipynb

Additional Functions

Method	Input	Description
normal_pdf_ vectorized	X (2d numpy array) mu (2d 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 using a vectorized approach Return: 2d numpy row array of normal pdf values See UnsupervisedML/Exercises/Section07/Exercises_7.1.2.ipynb
create_ellipse_patch _details	mu (numpy column array) Cov (2d numpy array) weight (float) contour (float)	This function determines matplotlib patch details of ellipse in 2d plane for which weighted normal pdf is equal to contour. Return: mean, width, height, and angle for ellipse See Unsupervised/Examples/Section07/Normal.ipynb

Unsupervised Machine Learning with Python

Section 7.4: Gaussian Mixture Model Code Walkthrough

Gaussian Mixture Model Code Walkthrough

Code located at:

- UnsupervisedML/Code/Programs

Files to Review	Description
gaussianmm.py	Class for Gaussian Mixture Model
normal.py	Functions for normal distribution pdf and creating elliptical contours in 2D
driver_gaussianmm.py	Driver for Gaussian Mixture Model

Course Resources at:

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

- Stop video if you would like to implement code yourself first