

# Unsupervised Machine Learning with Python

# Section 7.1: Normal Distribution

## Probability Density Function

# Why do we need Probability Density Functions

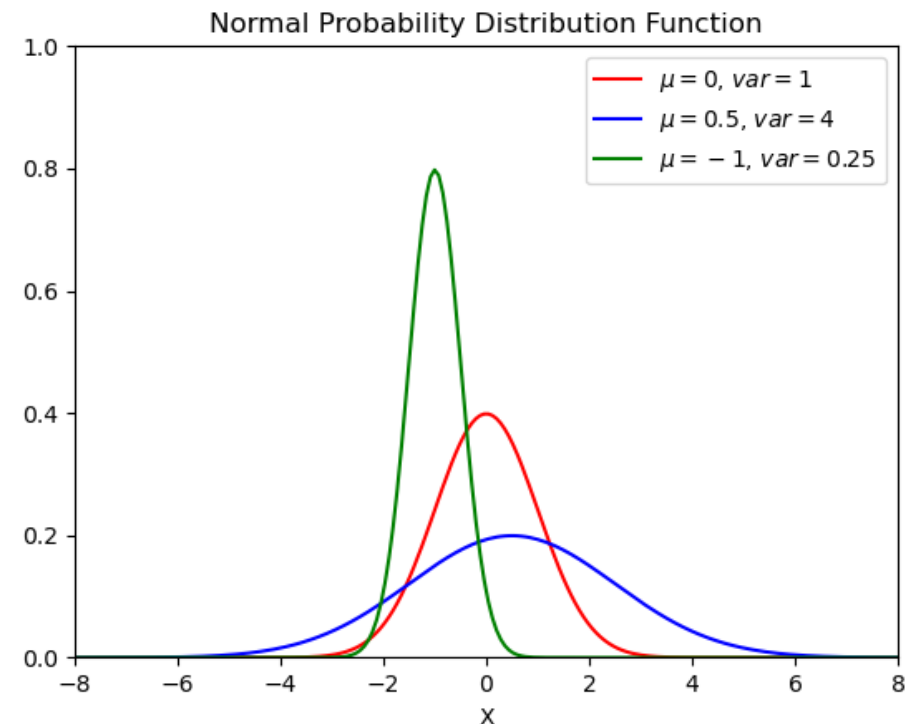
- Probability density functions (PDFs) are used for distribution-based clustering approaches
- Specifically for the Gaussian Mixture Model, need probability density function for the normal distribution

# Normal Distribution PDF 1D

Probability density function for 1 dimensional case

- Let  $X$  be a real number
- Assume mean  $\mu$  and variance  $v$  (both real numbers)
- Probability density function:

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



# Normal Distribution PDF Multi-Dimensional

Probability density function for d dimensional case

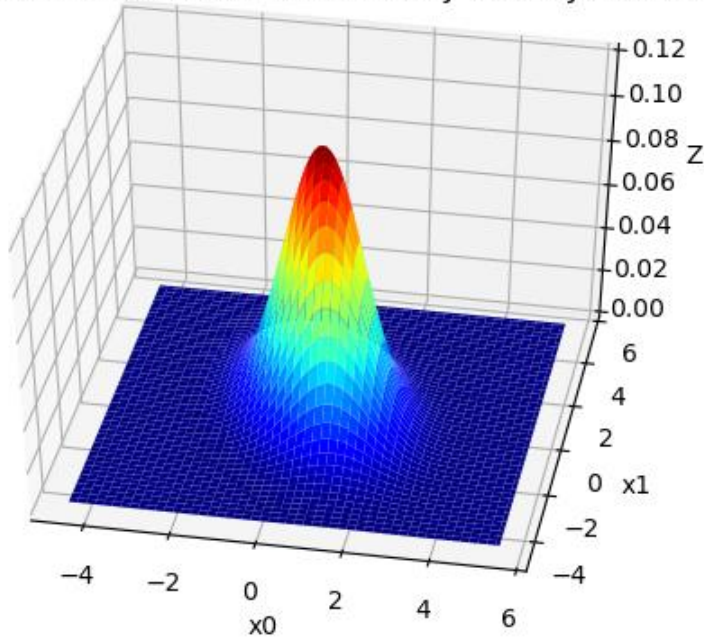
- Let  $X$  be a d-dimensional column vector
- Assume mean  $\mu$  (d-dimensional column vector)
- Assume covariance matrix  $\Sigma$  (dxd matrix)
  - Covariance matrix is symmetric
  - Covariance matrix is assumed to be positive-definite (eigenvalues  $> 0$ )
- Let  $|\Sigma|$  denote the determinant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

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

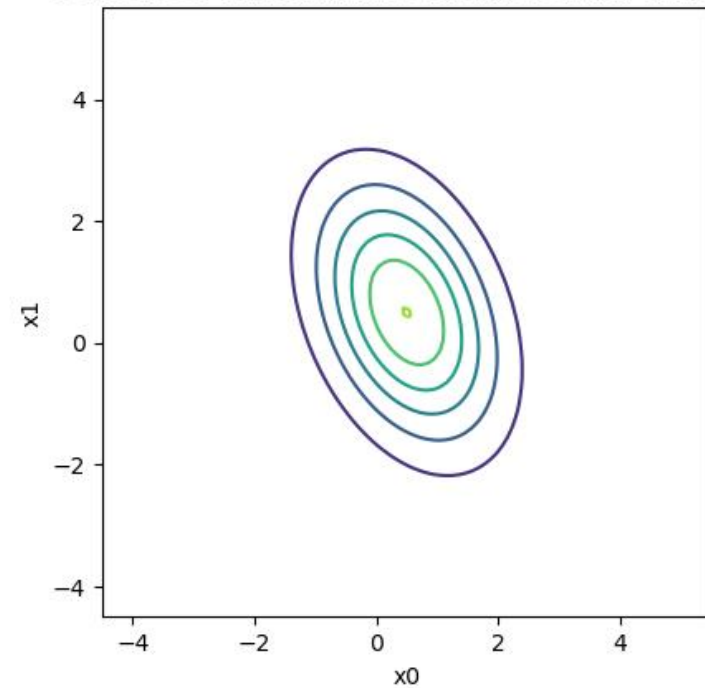
# Normal Probability Density Function in 2D

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

Surface Plot of 2d Normal Probability Density Function



Contours of 2d Normal Probability Density Function



# Normal Probability Density Function Contours

Details in UnsupervisedML/Resources/UnsupervisedML\_GMM.pdf

- Contours are curves in  $x_0$ - $x_1$  plane where  $N(X, \mu, \Sigma) = c$  is constant
- Neat connection between contours and SVD

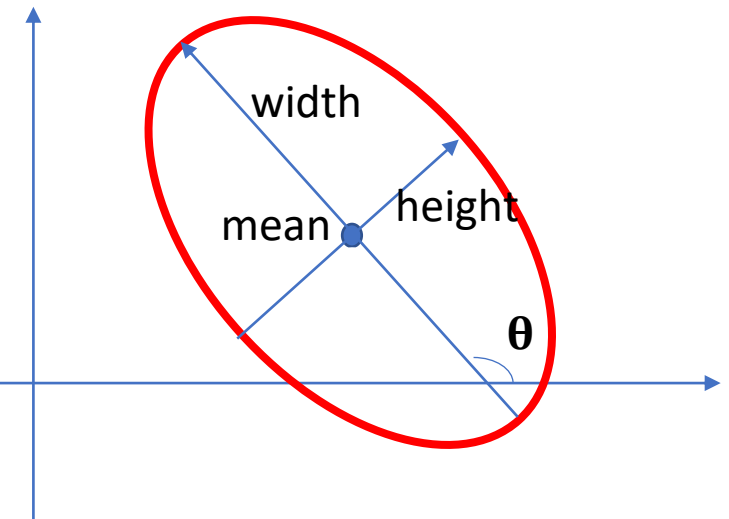
- In 2D: Mean:  $\begin{bmatrix} \mu_0 \\ \mu_1 \end{bmatrix}$  Covariance:  $\Sigma$  SVD:  $\Sigma = \begin{bmatrix} u_0 & u_1 \end{bmatrix} \begin{bmatrix} \sigma_0 & 0 \\ 0 & \sigma_1 \end{bmatrix} \begin{bmatrix} v_0^T \\ v_1^T \end{bmatrix}$

Mean is the centre of ellipse

$\theta$  = angle made by  $u_0$  with horizontal axis

width =  $2\sqrt{-2\log[c2\pi\sqrt{|\Sigma|}]\sqrt{\sigma_0}}$  in the  $u_0$  direction

height =  $2\sqrt{-2\log[c2\pi\sqrt{|\Sigma|}]\sqrt{\sigma_1}}$  in the  $u_1$  direction



# Computational Complexity

- Determinant, inverse calculations require  $O(d^3)$  operations as  $d \rightarrow \infty$
- Operations to compute  $N(X, \mu, \Sigma)$  is  $O(d^3)$  as  $d \rightarrow \infty$
- Amount of memory for the calculation is  $O(d^2)$  as  $d \rightarrow \infty$
- If  $X$  is a dataset of  $M$  samples, then operations to compute  $N(X, \mu, \Sigma)$  is  $O(M)$  as  $M \rightarrow \infty$



# 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
- 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
- This approach can be used with other distributions
- See [UnsupervisedML\\_Resources.pdf](#) for links to additional resources

# GMM: Probability Density Function for Mixture

- Assume data points  $X_0, X_1, X_2, \dots, X_{M-1}$  in  $d$  dimensions
- Assume that there are  $K$  clusters:
  - Cluster  $k$ , denoted  $C_k$ , has mean  $\mu_k$ , covariance  $\Sigma_k$ , and weight  $\phi_k$
  - Note that weights satisfy  $\phi_0 + \dots + \phi_{K-1} = 1$
- 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 dataset

# 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  $X_0, \dots, X_{M-1}$  is given by likelihood function:

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

- 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  $\phi_0 + \dots + \phi_{K-1} = 1$
- In practice, maximize the log likelihood function:

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

# GMM: Expectation Maximization

- Use Lagrange multipliers approach to solve maximization problem
- Cannot solve maximization problem exactly:
  - Make an initial guesses for means  $\{\mu_k\}$ , covariances  $\{\Sigma_k\}$ , weights  $\{\phi_k\}$
  - Iteratively improve the guesses
  - Called Expectation Maximization algorithm
- See file UnsupervisedML/Resources/UnsupervisedML\_GMM.pdf for the derivation and additional mathematical details



# GMM: Expectation Step

- Input: data points  $X_0, X_1, X_2, \dots, X_{M-1}$
- Input: most recently computed means  $\{\mu_k\}$ , covariances  $\{\Sigma_k\}$ , weights  $\{\phi_k\}$
- Update conditional probabilities for cluster  $k=0, \dots, K-1$  and  $i=0, \dots, M-1$

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

# GMM: Cluster Assignment

- Recall that  $\gamma_{ki}$  is the probability that point  $X_i$  is in cluster  $C_k$
- Most likely cluster for  $X_i$  is  $k$  such that  $\gamma_{ki}$  is largest
- Can use numpy argmax command:  $Cluster(X_i) = \operatorname{argmax}_k(\gamma_{ki})$

# GMM: Maximization Step

- Input: data points  $X_0, X_1, X_2, \dots, X_{M-1}$
- 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:

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

# GMM: Initialization

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

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

- 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

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

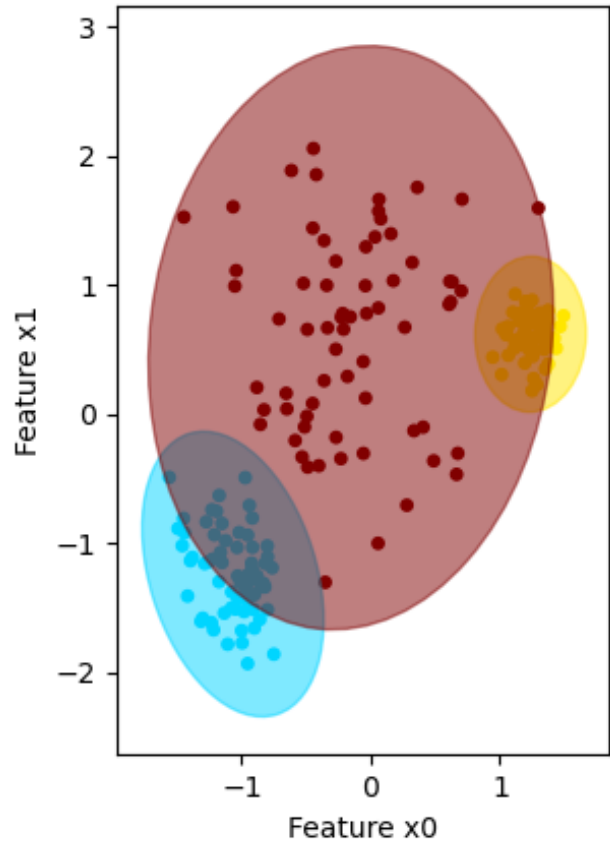
# GMM: Expectation Maximization Algorithm

- Input: data points  $X_0, X_1, X_2, \dots, X_{M-1}$
  - Specify tolerance  $\varepsilon$  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  $\varepsilon$ 
    - 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

Typically, an implementation will stop if convergence is achieved or if a maximum number of iterations is hit in step (2)

# GMM: Visualization of Results

Gaussian Mixture Model Dataset: varied\_blobs1

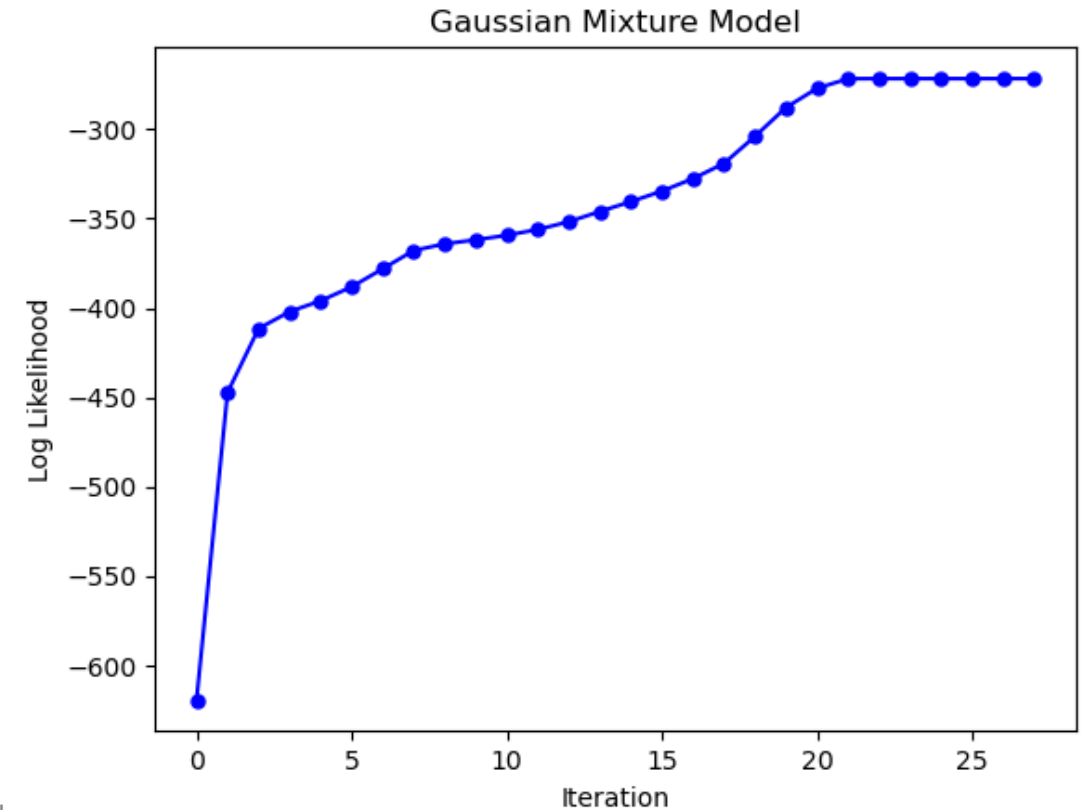
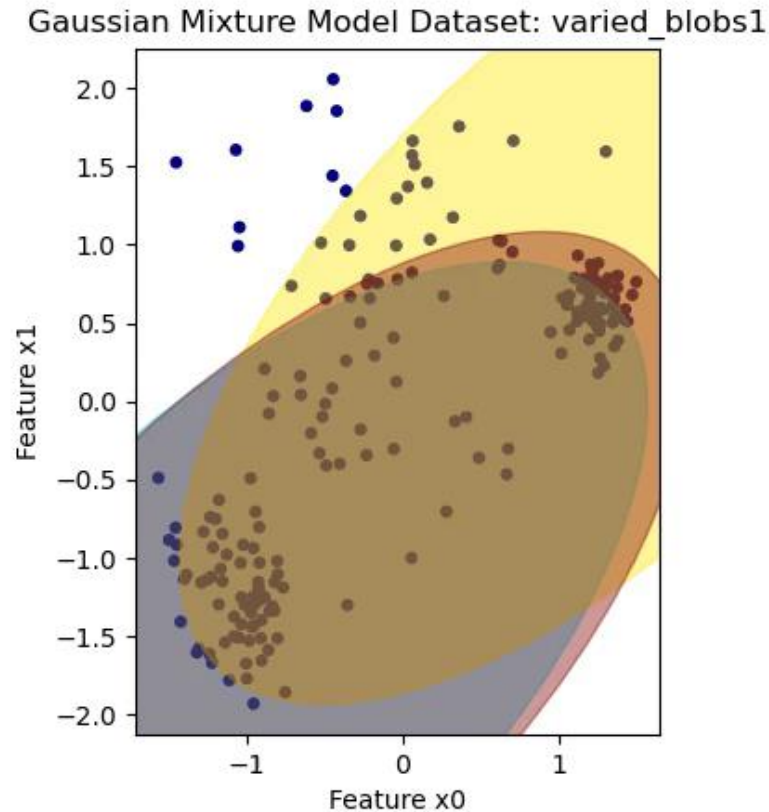


## Example with 3 Clusters

- Show “filled” elliptical contour for each Gaussian in mixture
- Each ellipse shows region where  $\phi_k N(X, \mu_k, \Sigma_k) \geq 0.002$
- Color of data point indicates cluster to which it most likely belongs

# 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

- Expectation step: compute  $M \cdot K$  conditional probabilities
- Maximization step: compute  $K$  means,  $K$  covariances, and  $K$  weights
- 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)



# 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

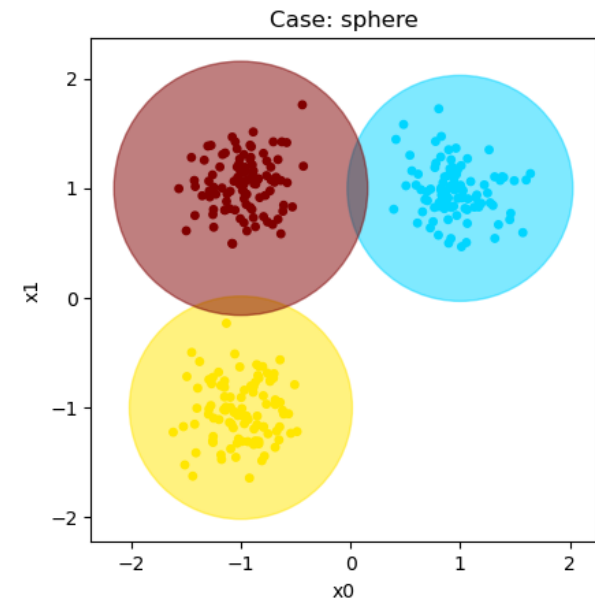
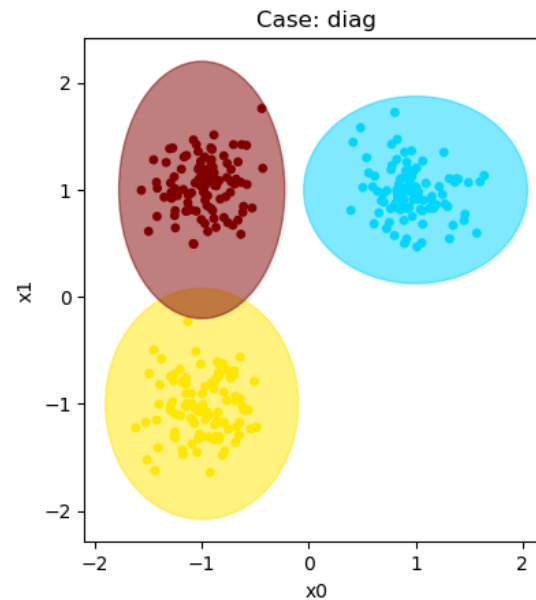
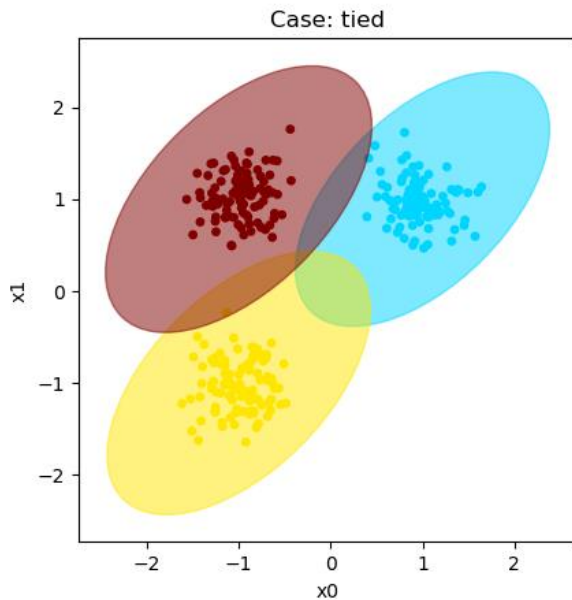
$$\frac{1}{\sqrt{(2\pi)^d |\Sigma|}}$$

and one must take inverse of  $\Sigma$

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

# 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 contours have same orientation and proportions
  - “diag”: each component has own diagonal covariance matrix, so elliptical contours have width and height directions parallel to axes
  - “spherical”: each component has its own single variance value, so elliptical contours are in fact circles



# 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) Derive gaussianmm class from clustering\_base class
- (2) Create functions computing multidimensional normal distribution pdf and for plotting contours of normal distribution pdf in 2d
- (3) Create Gaussian Mixture Model versions of plot\_cluster and plot\_cluster\_animation to be able to plot both cluster assignments and normal distribution contours

# Computing Conditional Probabilities

- Input: data points  $X_0, \dots, X_{M-1}$  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  $k=0, \dots, K-1$  and  $i=0, \dots, M-1$

Example 5 datapoints and 2 clusters

$$N = \begin{bmatrix} 0.1 & 0.4 & 0.3 & 0.5 & 0.4 \\ 0.1 & 0.2 & 0.1 & 0.5 & 0.2 \end{bmatrix} \quad \begin{array}{l} \leftarrow \text{Cluster 0} \\ \leftarrow \text{Cluster 1} \end{array}$$

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

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

# Computing Conditional Probabilities

(3) Divide each entry of weighted N by sum of its column to get  $\gamma$

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

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



# 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  $\gamma$

- Update Weights:

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

- Update Means:

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

Data points weighted by row k of  $\gamma$

- Update Covariances:

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

Covariance (with mean  $\mu_k$ ) weighted by row k of  $\gamma$

# 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 -1 -1 -1 -1], [0 1 1 0 1], [0 0 1 0 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 $[[[2], [3]], [[2], [3]], [[1], [4]]]$
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 $[[[1 \ 0.5], [3 \ 1]], [[1.5 \ 1], [2.5 \ 1]], [[1.2 \ 1.1], [1.5 \ 1.1]]]$
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 $\gamma_{ki}$ computed in Expectation step

# Class gaussianmm: Key Methods

Method	Input	Description
<code>__init__</code>	ncluster (integer) initialization (string)	Constructor for the class – input the number of clusters and initialization method (“random” or “kmeans++”)
<code>initialize_algorithm</code>		Initialize self.clustersave, self.objectivesave, self.Covsave, and self.weightsave. Initialize self.meansave using “random” or “kmeans++” initialization
<code>fit</code>	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
<code>expectation</code>		Compute the conditional probabilities $\gamma_{ki}$ and log likelihood function Return: nothing (update self.gamma and self.objectivesave)
<code>maximization</code>		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 (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: 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



# GMM Clustering: Code Walkthrough

Code located at:

- UnsupervisedML/Code/Programs

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

Course Resources at:

- <https://github.com/satishchandrareddy/UnsupervisedML/>
- Stop video if you would like to implement code yourself first