

# Week 7, Lecture 14 - Gaussian Mixture Models

Aaron Meyer

# Outline

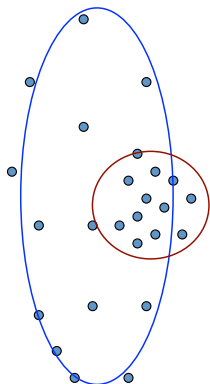
- ▶ Administrative Issues
- ▶ Gaussian mixtures
- ▶ Implementation

**Based on slides from David Sontag.**

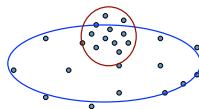
# Gaussian mixtures

# The Evils of “Hard Assignments”?

- ▶ Clusters may overlap
- ▶ Some clusters may be “wider” than others
- ▶ Distances can be deceiving!



# Probabilistic Clustering



- ▶ Try a probabilistic model!
  - ▶ Allows overlaps, clusters of different size, etc.
- ▶ Can tell a *generative story* for data
  - ▶  $P(X | Y)P(Y)$
- ▶ Challenge: we need to estimate model parameters without labeled Ys

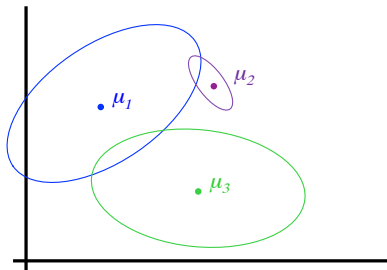
Y	X <sub>1</sub>	X <sub>2</sub>
??	0.1	2.1
??	0.5	-1.1
??	0.0	3.0
??	-0.1	-2.0
??	0.2	1.5
...	...	...

# The General GMM assumption

- ▶  $P(Y)$ : There are  $k$  components
- ▶  $P(X | Y)$ : Each component generates data from a *multivariate Gaussian* with mean  $\mu_i$  and covariance matrix  $\Sigma_i$

Each data point is sampled from a **generative process**:

1. Choose component  $i$  with probability  $P(y = i)$
2. Generate datapoint  $N(\mu_i, \Sigma_i)$



# What Model Should We Use?

- ▶ Depends on  $X$ .
- ▶ If we know which points are in a cluster, then we can define the best distribution for it.
  - ▶ Multinomial over clusters  $Y$
  - ▶ (Independent) Gaussian for each  $X_i$  given  $Y$

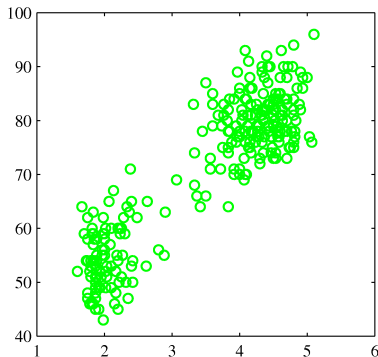
$$p(Y_i = y_k) = \theta_k$$

$$P(X_i = x \mid Y = y_k) = N(x \mid \mu_{ik}, \sigma_{ik})$$

# Could we make fewer assumptions?

- ▶ What if the  $X_i$  co-vary?
- ▶ What if there are multiple peaks?
- ▶ **Gaussian Mixture Models!**
  - ▶  $P(Y)$  still multinormal
  - ▶  $P(\mathbf{X} | Y)$  is a *multivariate* Gaussian distribution:

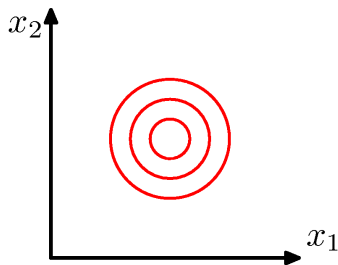
$$P(X = x_j | Y = i) = N(x_j, \mu_i, \Sigma_i)$$





# Multivariate Gaussians

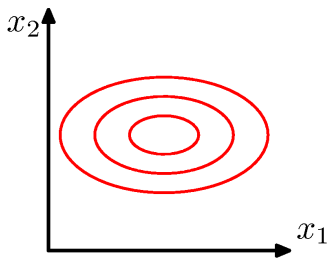
$$P(X=\mathbf{x}_j) = \frac{1}{(2\pi)^{m/2} \|\Sigma\|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}_j - \mu)^T \Sigma^{-1}(\mathbf{x}_j - \mu)\right]$$



$\Sigma \propto$  identity matrix

## Multivariate Gaussians

$$P(X=\mathbf{x}_j)=\frac{1}{(2\pi)^{m/2} \|\Sigma\|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}_j-\boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x}_j-\boldsymbol{\mu})\right]$$

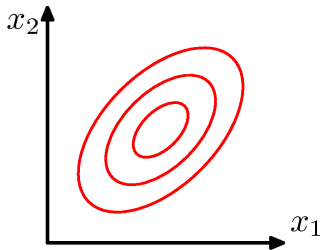


$\Sigma$  = diagonal matrix

$X_i$  are independent *ala* Gaussian NB

# Multivariate Gaussians

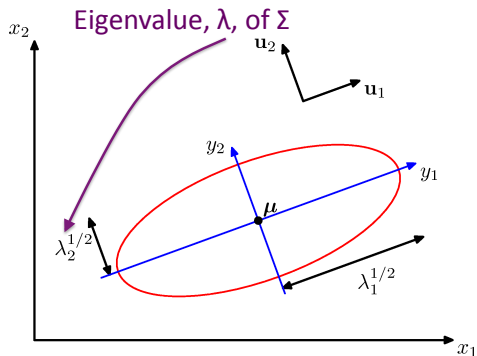
$$P(X=\mathbf{x}_j)=\frac{1}{(2\pi)^{m/2} \|\Sigma\|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}_j-\boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x}_j-\boldsymbol{\mu})\right]$$



$\Sigma$  = arbitrary (semidefinite) matrix:

- specifies rotation (change of basis)
- eigenvalues specify relative elongation

# Multivariate Gaussians

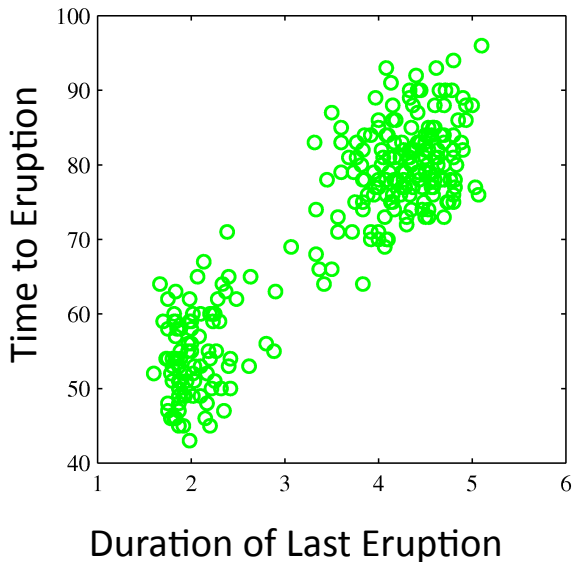


Covariance matrix,  $\Sigma$ , =  
degree to which  $x_i$  vary  
together

$$P(X=\mathbf{x}_j) = \frac{1}{(2\pi)^{m/2} \|\Sigma\|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}_j - \mu)^T \Sigma^{-1}(\mathbf{x}_j - \mu)\right]$$

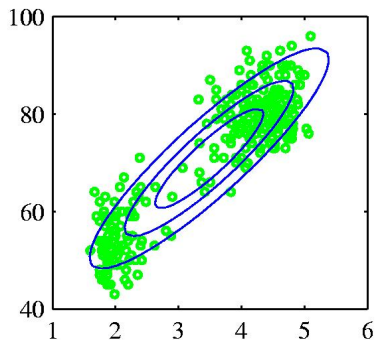
# Mixtures of Gaussians (1)

Old Faithful Data Set

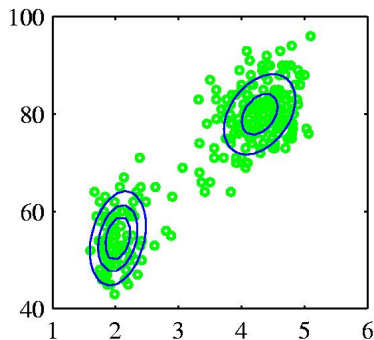


# Mixtures of Gaussians (1)

Old Faithful Data Set



Single Gaussian



Mixture of two Gaussians

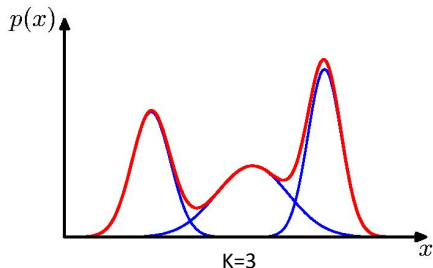
# Mixtures of Gaussians (2)

Combine simple models into a complex model:

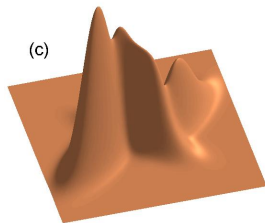
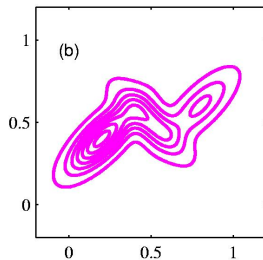
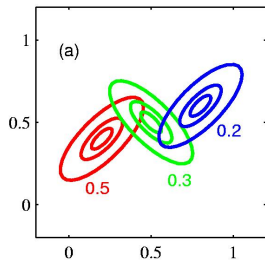
$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \underbrace{\mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}_{\text{Component}}$$

Mixing coefficient

$$\forall k : \pi_k \geq 0 \quad \sum_{k=1}^K \pi_k = 1$$



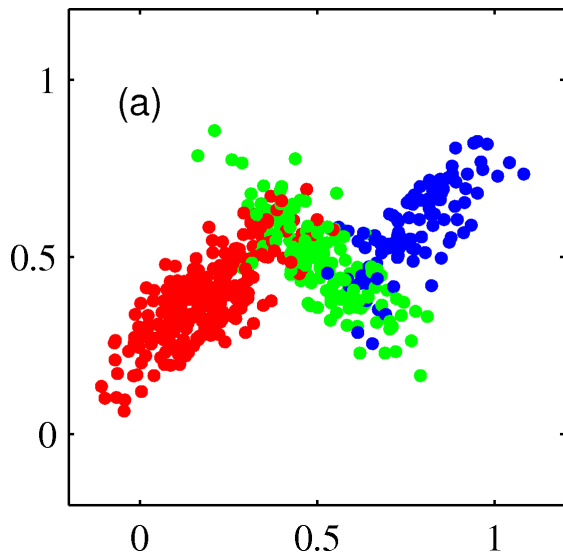
# Mixtures of Gaussians (3)





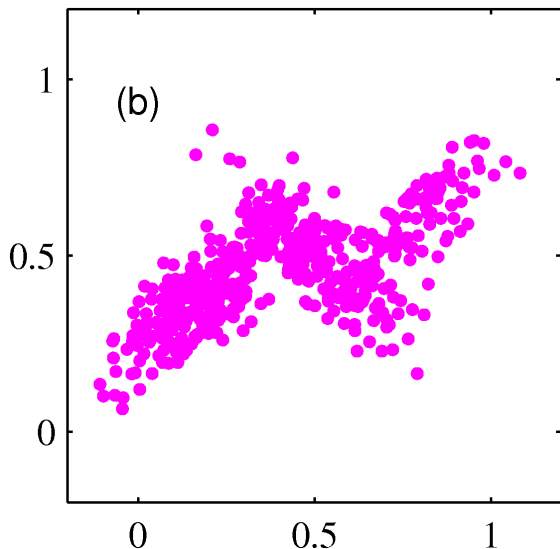
# Eliminating Hard Assignments to Clusters

Model data as mixture of multivariate Gaussians



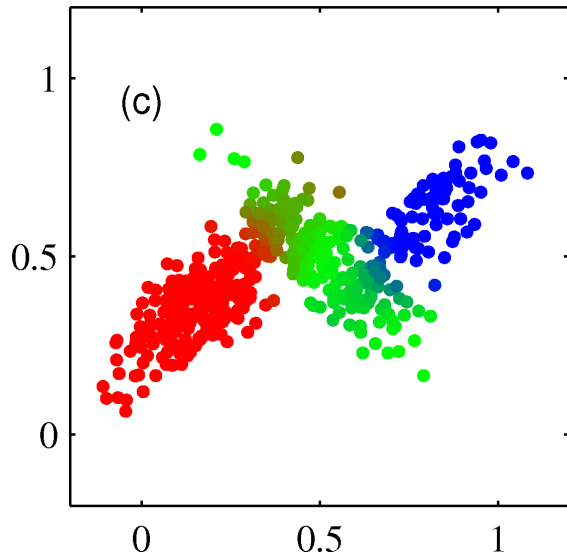
# Eliminating Hard Assignments to Clusters

Model data as mixture of multivariate Gaussians



# Eliminating Hard Assignments to Clusters

Model data as mixture of multivariate Gaussians



# ML estimation in **supervised** setting

- ▶ Univariate Gaussian

$$\mu_{MLE} = \frac{1}{N} \sum_{i=1}^N x_i \quad \sigma_{MLE}^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{\mu})^2$$

- ▶ **Mixture of Multivariate Gaussians**

- ▶ ML estimate for each of the Multivariate Gaussians is given by:

$$\mu_{ML}^k = \frac{1}{n} \sum_{j=1}^n x_j \quad \Sigma_{ML}^k = \frac{1}{n} \sum_{j=1}^n (\mathbf{x}_j - \mu_{ML}^k) (\mathbf{x}_j - \mu_{ML}^k)^T$$

Just sums over  $x$  generated from the  $k$ 'th Gaussian

# But what if unobserved data?

- ▶ MLE:

- ▶  $\arg \max_{\theta} \prod_j P(y_j, x_j)$

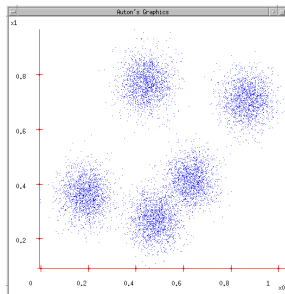
- ▶  $\theta$ : all model parameters

- ▶ eg, class probs, means, and variances

- ▶ But we don't know  $y_j$ 's!

- ▶ Maximize **marginal likelihood**:

- ▶  $\arg \max_{\theta} \prod_j P(x_j) = \arg \max_{\theta} \prod_j \sum_{k=1}^K P(Y_j = k, x_j)$



# How do we optimize? Closed Form?

- ▶ Maximize **marginal likelihood**:

$$\arg \max_{\theta} \prod_j P(x_j) = \arg \max \prod_j \sum_{k=1}^K P(Y_j = k, x_j)$$

- ▶ Almost always a hard problem!
  - ▶ Usually no closed form solution
  - ▶ Even when  $\lg P(X, Y)$  is convex,  $\lg P(X)$  generally isn't. . .
  - ▶ For all but the simplest  $P(X)$ , we will have to do gradient ascent, in a big messy space with lots of local optima. . .

# Learning general mixtures of Gaussians

$$P(y = k | \mathbf{x}_j) \propto \frac{1}{(2\pi)^{m/2} \|\Sigma_k\|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}_j - \mu_k)^T \Sigma_k^{-1}(\mathbf{x}_j - \mu_k)\right] P(y = k)$$

- Marginal likelihood:

$$\begin{aligned} \prod_{j=1}^m P(\mathbf{x}_j) &= \prod_{j=1}^m \sum_{k=1}^K P(\mathbf{x}_j, y = k) \\ &= \prod_{j=1}^m \sum_{k=1}^K \frac{1}{(2\pi)^{m/2} \|\Sigma_k\|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x}_j - \mu_k)^T \Sigma_k^{-1}(\mathbf{x}_j - \mu_k)\right] P(y = k) \end{aligned}$$

- ▶ Need to differentiate and solve for  $\mu_k$ ,  $\Sigma_k$ , and  $P(Y=k)$  for  $k=1..K$
- ▶ There will be no closed form solution, gradient is complex, lots of local optimum
- ▶ **Wouldn't it be nice if there was a better way!?!**

# EM

## Expectation Maximization



# The EM Algorithm

- ▶ A clever method for maximizing marginal likelihood:
  - ▶  $\arg \max_{\theta} \prod_j P(x_j) = \arg \max_{\theta} \prod_j \sum_{k=1}^K P(Y_j = k, x_j)$
  - ▶ A type of gradient ascent that can be easy to implement
    - ▶ e.g. no line search, learning rates, etc.
- ▶ Alternate between two steps:
  - ▶ Compute an expectation
  - ▶ Compute a maximization
- ▶ Not magic: **still optimizing a non-convex function with lots of local optima**
  - ▶ The computations are just easier (often, significantly so!)

## EM: Two Easy Steps

**Objective:**  $\operatorname{argmax}_{\theta} \lg \prod_j \sum_{k=1}^K P(Y_j=k, x_j | \theta) = \sum_j \lg \sum_{k=1}^K P(Y_j=k, x_j | \theta)$

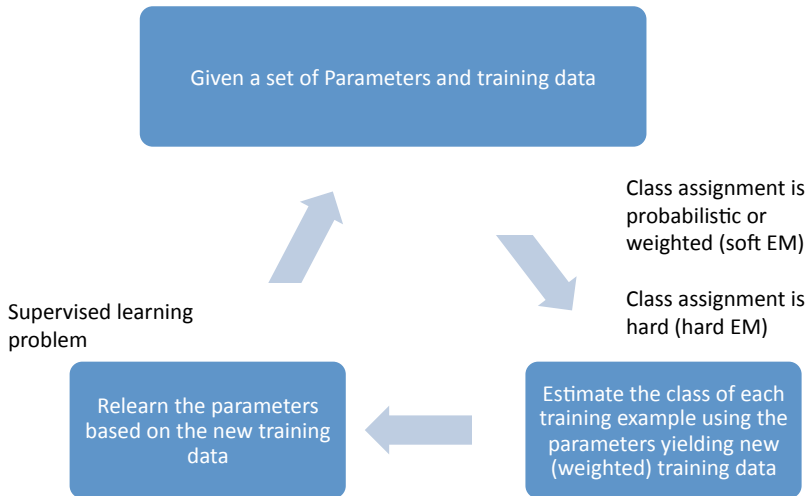
**Data:**  $\{x_j \mid j=1 \dots n\}$

Notation a bit inconsistent  
Parameters =  $\theta = \lambda$

- **E-step:** Compute expectations to “fill in” missing y values according to current parameters,  $\theta$ 
  - For all examples  $j$  and values  $k$  for  $Y_j$ , compute:  $P(Y_j=k \mid x_j, \theta)$
- **M-step:** Re-estimate the parameters with “weighted” MLE estimates
  - Set  $\theta = \operatorname{argmax}_{\theta} \sum_j \sum_k P(Y_j=k \mid x_j, \theta) \log P(Y_j=k, x_j \mid \theta)$

Especially useful when the E and M steps have closed form solutions!!!

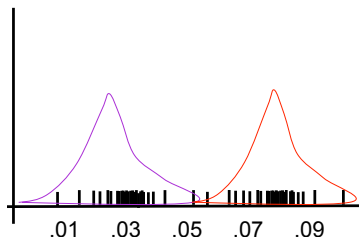
# EM algorithm: Pictorial View



# Simple example: learn means only!

## Consider:

- 1D data
- Mixture of  $k=2$  Gaussians
- Variances fixed to  $\sigma=1$
- Distribution over classes is uniform
- Just need to estimate  $\mu_1$  and  $\mu_2$



$$\prod_{j=1}^m \sum_{k=1}^K P(x, Y_j = k) \propto \prod_{j=1}^m \sum_{k=1}^{K=2} \exp\left[-\frac{1}{2\sigma^2} \|x - \mu_k\|^2\right] P(Y_j = k)$$

## EM for GMMs: only learning means

**Iterate:** On the  $t'$ th iteration let our estimates be

$$\lambda_t = \{ \mu_1^{(t)}, \mu_2^{(t)} \dots \mu_K^{(t)} \}$$

### E-step

Compute “expected” classes of all datapoints

$$P(Y_j = k | x_j, \mu_1 \dots \mu_K) \propto \exp\left(-\frac{1}{2\sigma^2} \|x_j - \mu_k\|^2\right) P(Y_j = k)$$

### M-step

Compute most likely new  $\mu$ s given class expectations

$$\mu_k = \frac{\sum_{j=1}^m P(Y_j = k | x_j) x_j}{\sum_{j=1}^m P(Y_j = k | x_j)}$$

# E.M. for General GMMs

**Iterate:** On the  $t$ 'th iteration let our estimates be

$$\lambda_t = \{ \mu_1^{(t)}, \mu_2^{(t)} \dots \mu_K^{(t)}, \Sigma_1^{(t)}, \Sigma_2^{(t)} \dots \Sigma_K^{(t)}, p_1^{(t)}, p_2^{(t)} \dots p_K^{(t)} \}$$

$p_k^{(t)}$  is shorthand for  
estimate of  $P(y=k)$  on  
 $t$ 'th iteration

## E-step

Compute “expected” classes of all datapoints for each class

$$P(Y_j = k | x_j, \lambda_t) \propto p_k^{(t)} P(x_j | \mu_k^{(t)}, \Sigma_k^{(t)})$$

Just evaluate a  
Gaussian at  $x_j$

## M-step

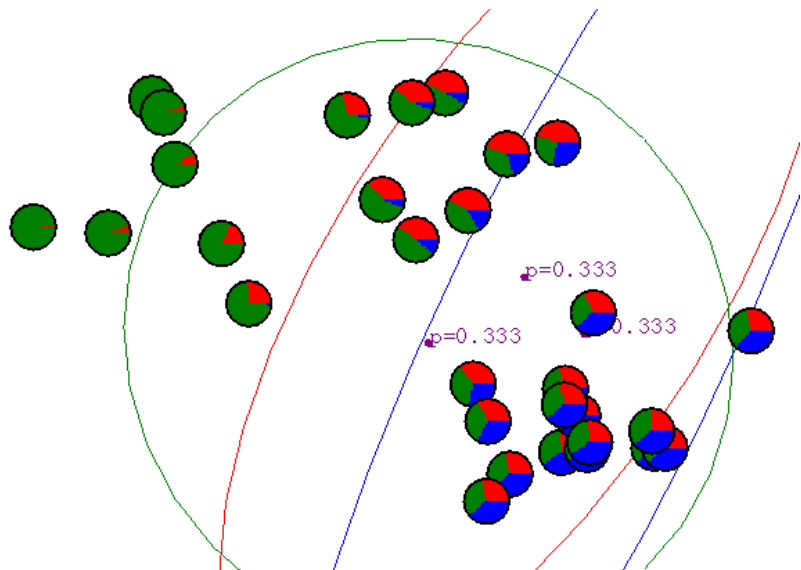
Compute weighted MLE for  $\mu$  given expected classes above

$$\mu_k^{(t+1)} = \frac{\sum_j P(Y_j = k | x_j, \lambda_t) x_j}{\sum_j P(Y_j = k | x_j, \lambda_t)} \quad \Sigma_k^{(t+1)} = \frac{\sum_j P(Y_j = k | x_j, \lambda_t) [x_j - \mu_k^{(t+1)}][x_j - \mu_k^{(t+1)}]^T}{\sum_j P(Y_j = k | x_j, \lambda_t)}$$

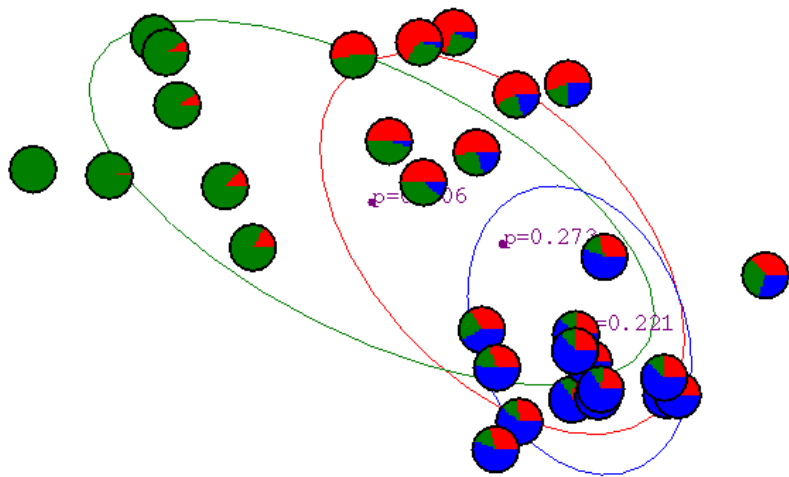
$$p_k^{(t+1)} = \frac{\sum_j P(Y_j = k | x_j, \lambda_t)}{m}$$

$m = \# \text{training examples}$

## Gaussian Mixture Example: Start

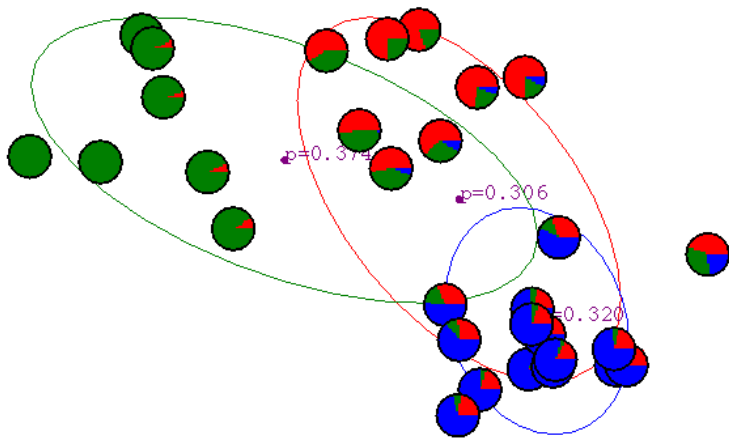


## After First Iteration

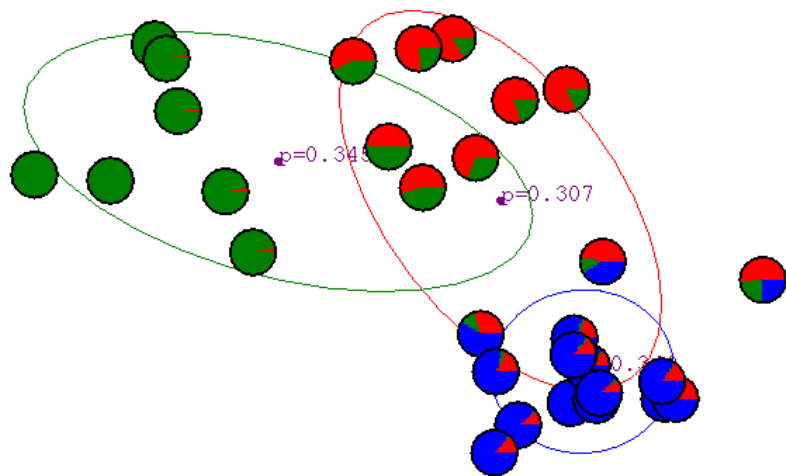




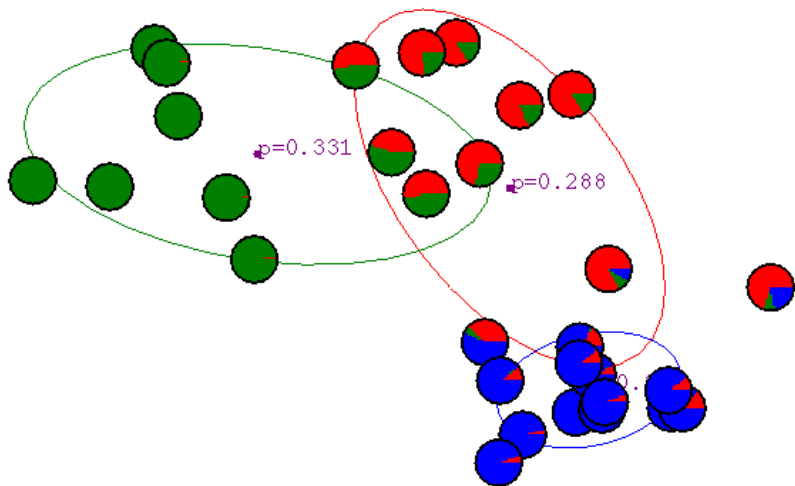
## After 2nd Iteration



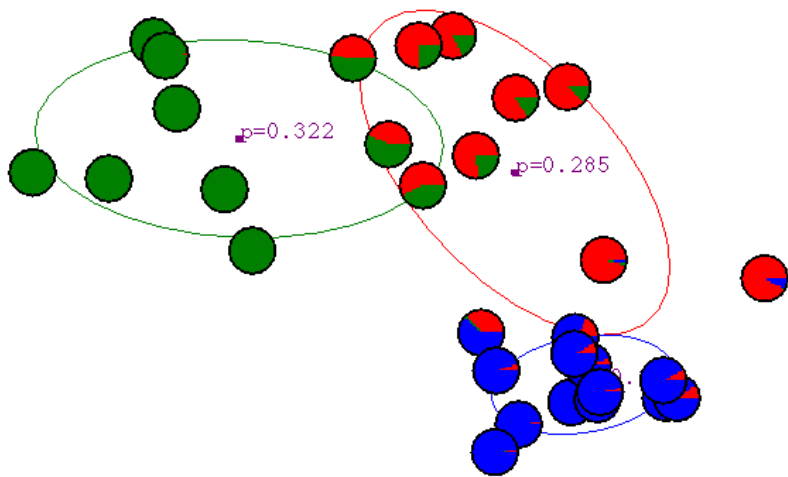
## After 3rd iteration



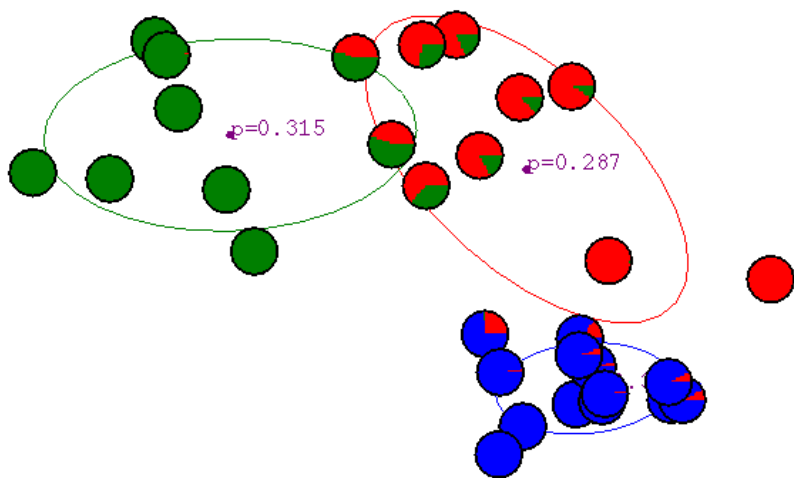
After 4th iteration



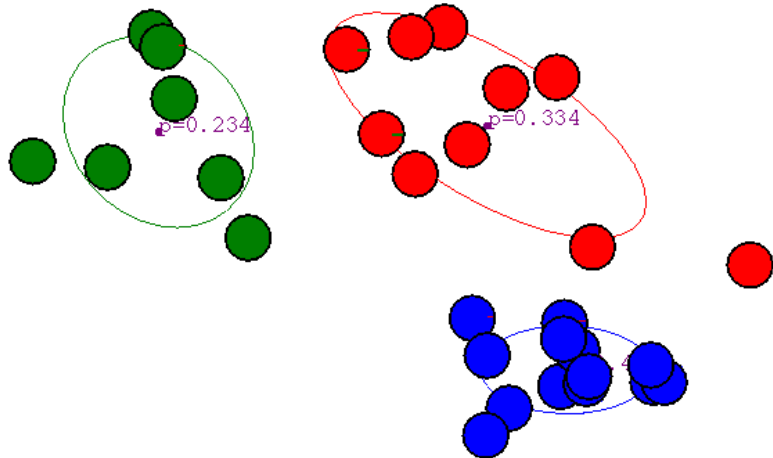
After 5th iteration



After 6th iteration



After 20th iteration



# What if we do hard assignments?

**Iterate:** On the  $t$ 'th iteration let our estimates be

$$\lambda_t = [\mu_1^{(t)}, \mu_2^{(t)}, \dots, \mu_3^{(t)}]$$

## E-step

Compute “expected” classes of all datapoints

$$P(Y_j = k | x_j, \mu_1 \dots, \mu_K) \propto \exp\left(-\frac{1}{2\sigma^2} \|x_j - \mu_k\|^2\right) P(Y_j = k)$$

## M-step

Compute most likely new  $\mu$ s given class expectations

~~$$\mu_k = \frac{\sum_{j=1}^m P(Y_j = k | x_j) x_j}{\sum_{j=1}^m P(Y_j = k | x_j)}$$~~

$$\mu_k = \frac{\sum_{j=1}^m \delta(Y_j = k, x_j) x_j}{\sum_{j=1}^m \delta(Y_j = k, x_j)}$$

$\delta$  represents hard assignment to “most likely” or nearest cluster

Equivalent to k-means clustering algorithm!!!

# Implementation

`sklearn.mixture.GaussianMixture` implements GMMs within `sklearn`.

- ▶ `GaussianMixture` creates the class
  - ▶ `n_components` indicates the number of Gaussians to use.
  - ▶ `covariance_type` is type of covariance
    - ▶ `full`
    - ▶ `spherical`
    - ▶ `diag`
    - ▶ `tied` means all components share the same covariance matrix
  - ▶ `max_iter` is EM iterations to use
- ▶ Functions
  - ▶ `M.fit(X)` fits using the EM algorithm
  - ▶ `M.predict_proba(X)` is the posterior probability of each component given the data
  - ▶ `M.predict(X)` predict the class labels of each data point



## Further Reading

- ▶ `sklearn.mixture.GaussianMixture`
- ▶ Python Data Science Handbook: GMMs