Learning Parameters of Bayesian networks
Lecture 12

David Sontag

New York University

Bayesian networks

- A **Bayesian network** is specified by a directed *acyclic* graph G=(V,E) with:
  - One node *i* for each random variable X_i
  - One conditional probability distribution (CPD) per node, $p(x_i \mid \mathbf{x}_{Pa(i)})$, specifying the variable's probability conditioned on its parents' values

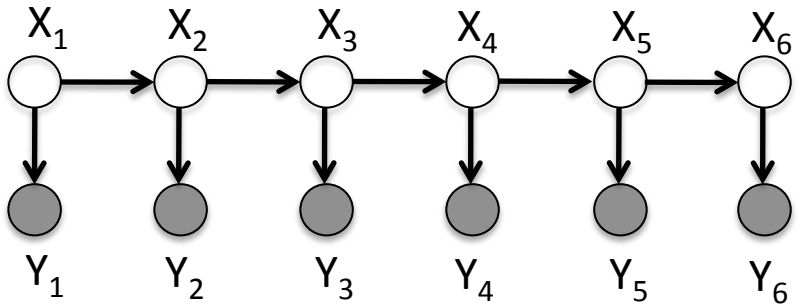
- Corresponds 1-1 with a particular factorization of the joint distribution:

$$p(x_1, \ldots x_n) = \prod_{i \in V} p(x_i \mid \mathbf{x}_{\mathrm{Pa}(i)})$$

- Powerful framework for designing *algorithms* to perform probability computations

HMMs as a *graphical model*

- We can represent a hidden Markov model with a graph:



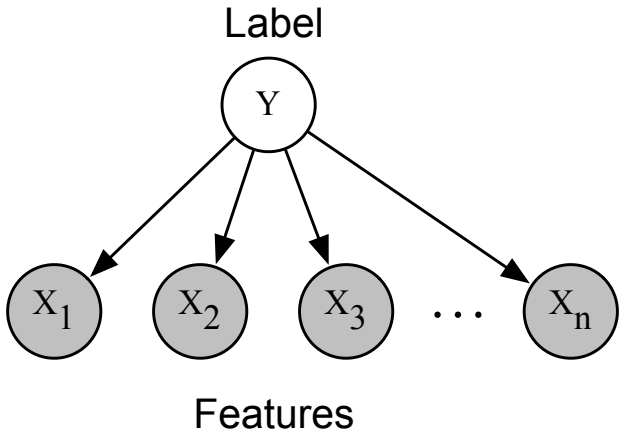
Shading in denotes *observed* variables (e.g. what is available at test time)

$$\Pr(x_1, \ldots x_n, y_1, \ldots, y_n) = \Pr(x_1) \Pr(y_1 \mid x_1) \prod_{t=2}^{n} \Pr(x_t \mid x_{t-1}) \Pr(y_t \mid x_t)$$

- There is a 1-1 mapping between the graph structure and the factorization of the joint distribution

Naïve Bayes as a *graphical model*

- We can represent a naïve Bayes model with a graph:



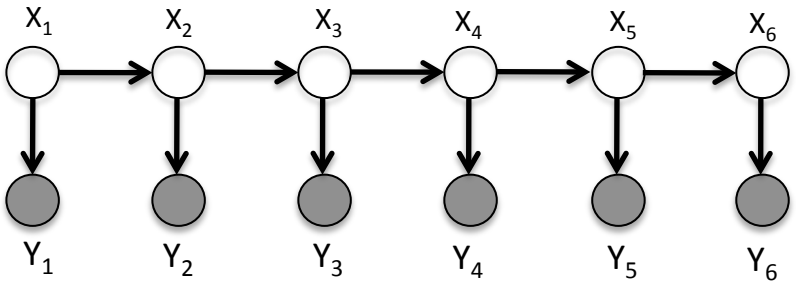
Shading in denotes *observed* variables (e.g. what is available at test time)

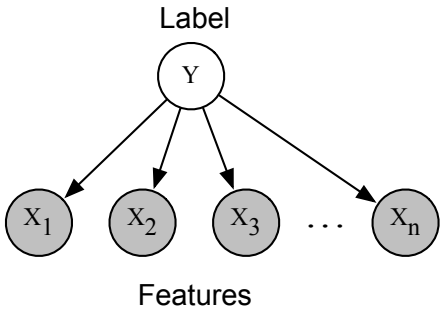
$$\Pr(y, x_1, \ldots, x_n) = \Pr(y) \prod_{i=1}^{n} \Pr(x_i \mid y)$$

- There is a 1-1 mapping between the graph structure and the factorization of the joint distribution

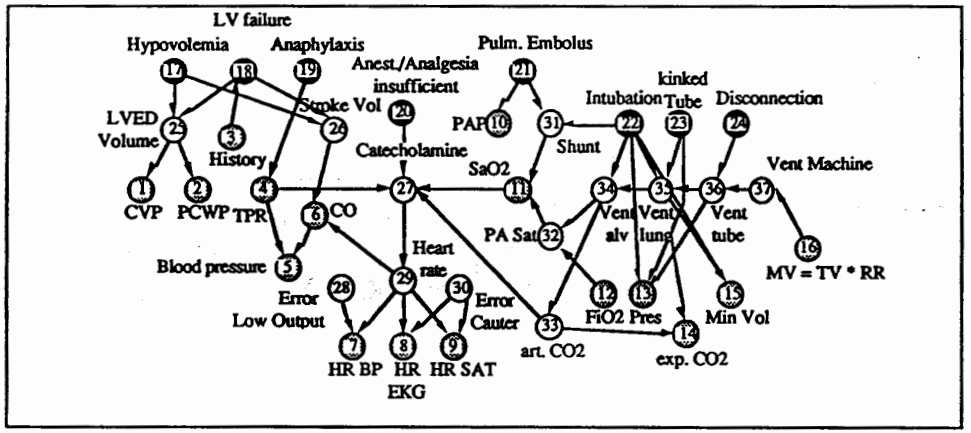
Inference in Bayesian networks

- Computing marginal probabilities in **tree** structured Bayesian networks is easy
  - The algorithm called "belief propagation" generalizes what we showed for hidden Markov models to arbitrary trees



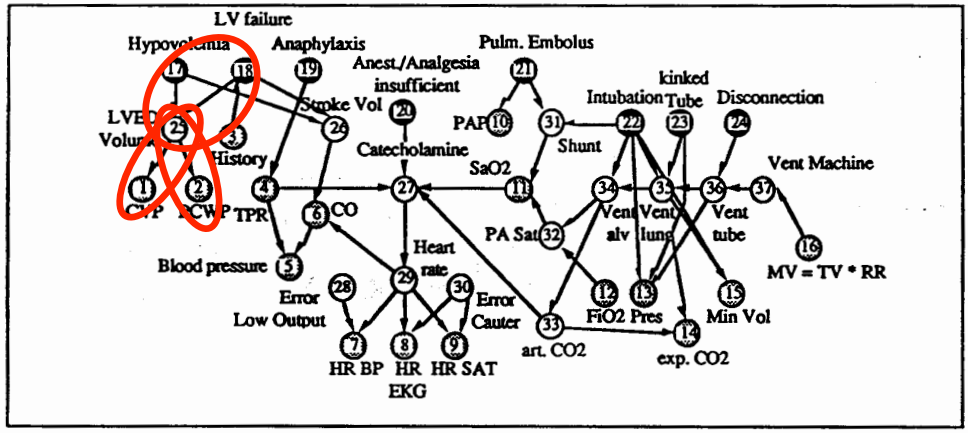


- Wait… this isn't a tree! What can we do?



Inference in Bayesian networks

- In some cases (such as this) we can *transform* this into what is called a "junction tree", and then run belief propagation



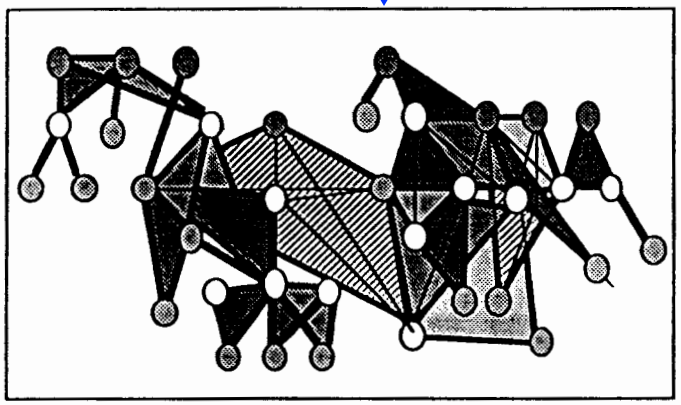
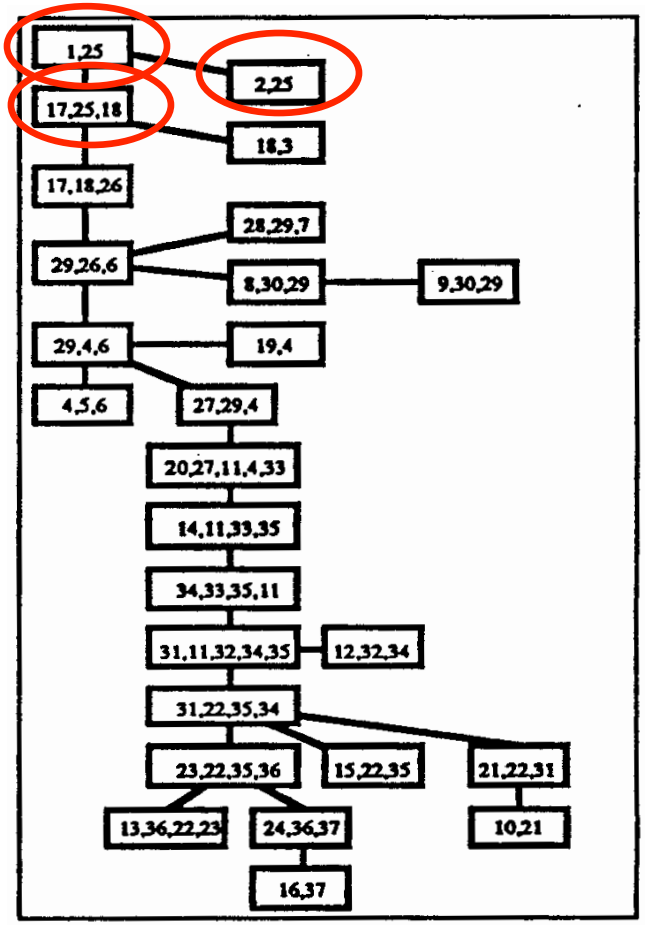


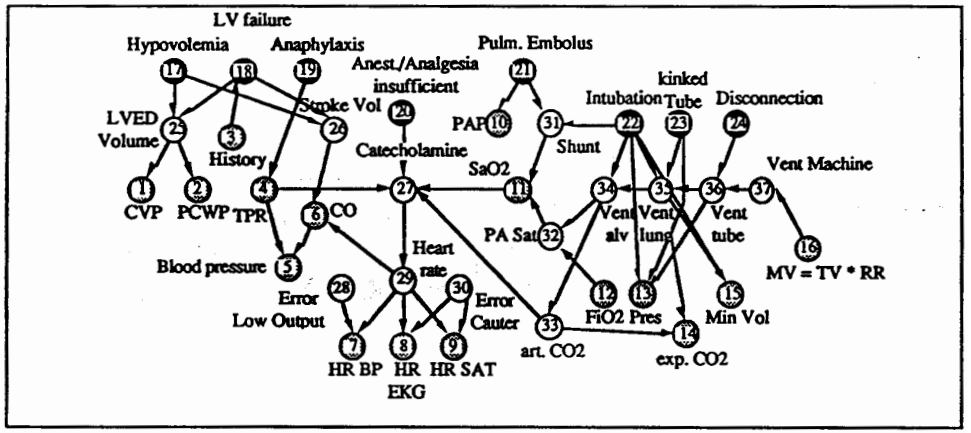
Fig. 7

Spiegelhalter's algorithm re-arranges the ALARM network by triangulation and clique formation. The cliques are shaded differently to make them visible.



Approximate inference – more next week

- There is also a wealth of **approximate** inference algorithms that can be applied to Bayesian networks such as these



- Markov chain Monte Carlo algorithms repeatedly sample assignments for estimating marginals
- Variational inference algorithms (which are deterministic) attempt to fit a simpler distribution to the complex distribution, and then computes marginals for the simpler distribution

Maximum likelihood estimation in Bayesian networks

- Suppose that we know the Bayesian network structure G
- Let $\theta_{x_i|\mathbf{x}_{pa(i)}}$ be the parameter giving the value of the CPD $p(x_i \mid \mathbf{x}_{pa(i)})$
- Maximum likelihood estimation corresponds to solving:

$$\max_{\theta} \frac{1}{M} \sum_{m=1}^{M} \log p(\mathbf{x}^M; \theta)$$

subject to the non-negativity and normalization constraints

- This is equal to:

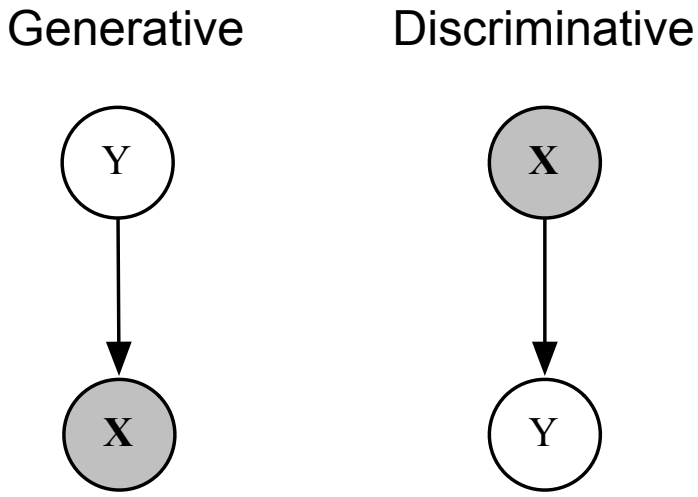
$$\begin{aligned} \max_{\theta} \frac{1}{M} \sum_{m=1}^{M} \log p(\mathbf{x}^M; \theta) &= \max_{\theta} \frac{1}{M} \sum_{m=1}^{M} \sum_{i=1}^{N} \log p(x_i^M \mid \mathbf{x}_{pa(i)}^M; \theta) \\ &= \max_{\theta} \sum_{i=1}^{N} \frac{1}{M} \sum_{m=1}^{M} \log p(x_i^M \mid \mathbf{x}_{pa(i)}^M; \theta) \end{aligned}$$

- The optimization problem decomposes into an independent optimization problem for each CPD! Has a simple closed-form solution.

Discriminative versus generative classifiers

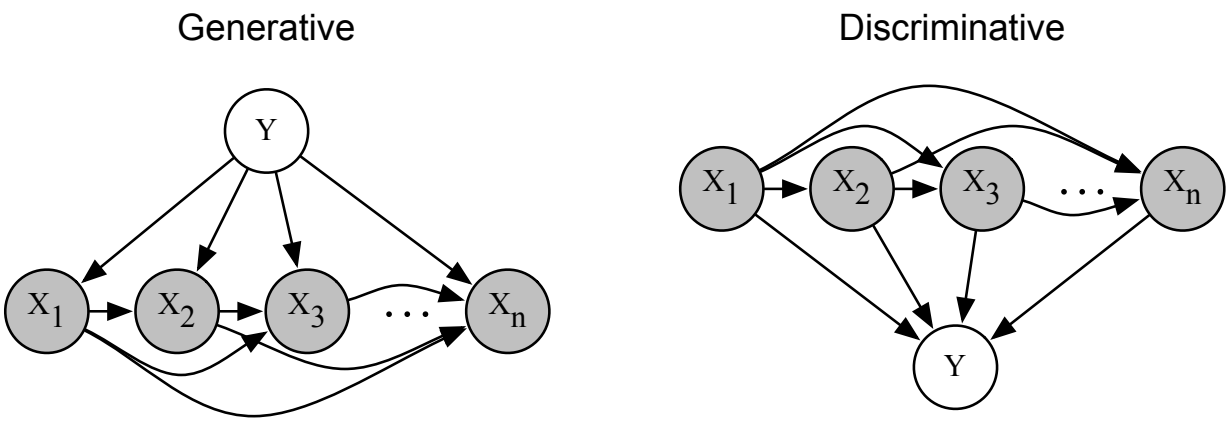
- There is often significant flexibility in choosing the structure and parameterization of a Bayesian network

- Without further constraints, these are **equivalent** models of p(Y, **X**):



Discriminative versus generative classifiers

- Let's go a bit deeper to understand what are the trade-offs inherent in each approach
- Since $\mathbf{X}$ is a random vector, for $Y \rightarrow \mathbf{X}$ to be equivalent to $\mathbf{X} \rightarrow Y$, we must have:



We must make the following choices:

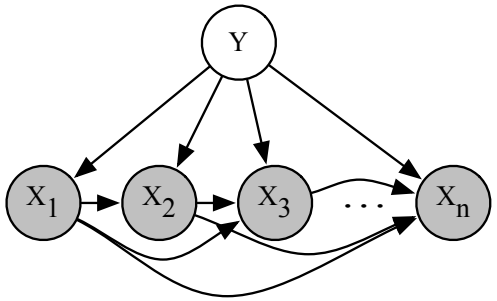
1. In the generative model, how do we parameterize $p(X_i \mid \mathbf{X}_{pa(i)}, Y)$?
2. In the discriminative model, how do we parameterize $p(Y \mid \mathbf{X})$?

Discriminative versus generative classifiers

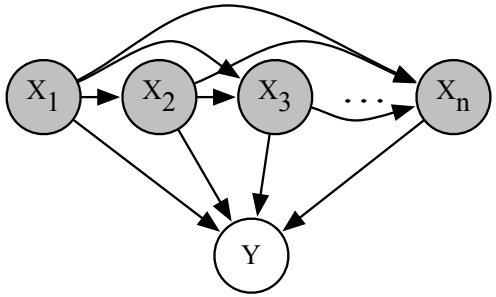
We must make the following choices:

1. In the generative model, how do we parameterize $p(X_i \mid \mathbf{X}_{pa(i)}, Y)$?
2. In the discriminative model, how do we parameterize $p(Y \mid \mathbf{X})$?

Generative

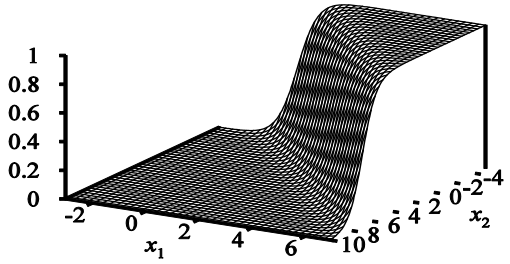


Discriminative



1. For the generative model, assume that $X_i \perp \mathbf{X}_{-i} \mid Y$ (**naive Bayes**)
2. For the discriminative model, assume that

$$p(Y = 1 \mid \mathbf{x}; \alpha) = \frac{1}{1 + e^{-\alpha_0 - \sum_{i=1}^{n} \alpha_i x_i}}$$



logistic regression

Discriminative versus generative classifiers

1. For the generative model, assume that $X_i \perp \mathbf{X}_{-i} \mid Y$ (**naive Bayes**)
2. For the discriminative model, assume that

$$p(Y = 1 \mid \mathbf{x}; \alpha) = \frac{e^{\alpha_0 + \sum_{i=1}^{n} \alpha_i x_i}}{1 + e^{\alpha_0 + \sum_{i=1}^{n} \alpha_i x_i}} = \frac{1}{1 + e^{-\alpha_0 - \sum_{i=1}^{n} \alpha_i x_i}}$$

- In problem set , you show **assumption** 1 $\Rightarrow$ **assumption** 2
- Thus, every conditional distribution that can be represented using naive Bayes can *also* be represented using the logistic model

- What can we conclude from this?

With a large amount of training data, logistic regression will perform at least as well as naive Bayes!

Logistic regression for discrete classification

Logistic regression in more general case, where set of possible Y is $\{y_1,\ldots,y_R\}$

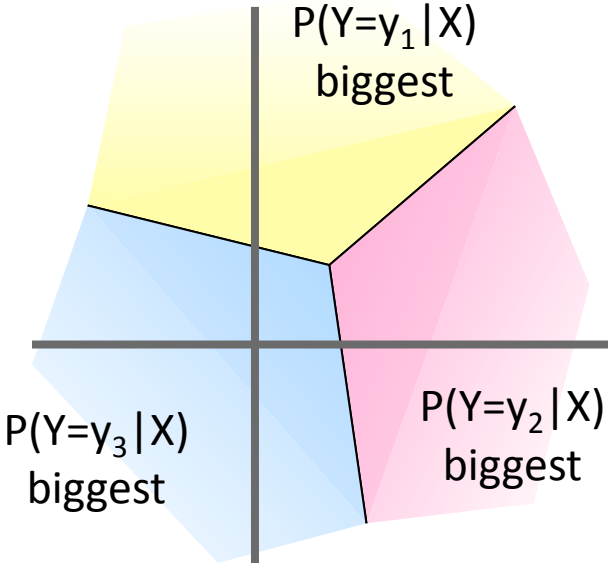
- Define a weight vector w_i for each y_i, i=1,…,R-1

$$P(Y=1|X) \propto \exp(w_{10} + \sum_i w_{1i} X_i)$$

$$P(Y=2|X) \propto \exp(w_{20} + \sum_i w_{2i} X_i)$$

…

$$P(Y=r|X) = 1 - \sum_{j=1}^{r-1} P(Y=j|X)$$



Logistic regression for discrete classification

- Logistic regression in more general case, where *Y* is in the set $\{y_1,\ldots,y_R\}$

  for *k<R*

$$P(Y = y_k|X) = \frac{\exp(w_{k0} + \sum_{i=1}^{n} w_{ki}X_i)}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}$$

  for *k=R* (normalization, so no weights for this class)

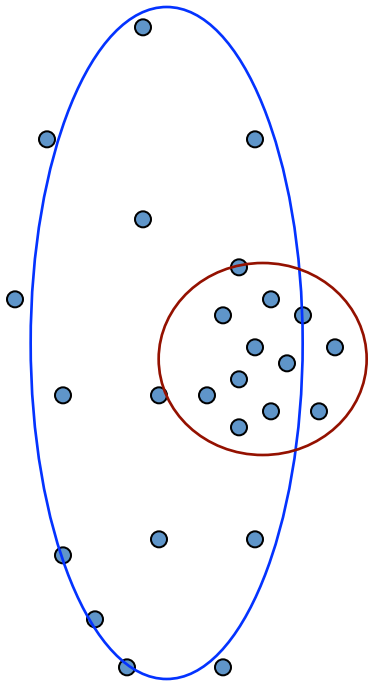
$$P(Y = y_R|X) = \frac{1}{1 + \sum_{j=1}^{R-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}$$

Features can be discrete or continuous!

Mixture Models & EM algorithm

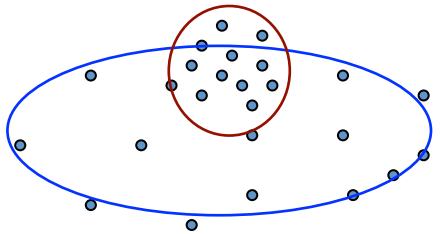
Slides adapted from Carlos Guestrin, Dan Klein, Luke Zettlemoyer, Dan Weld, Vibhav Gogate, and Andrew Moore

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(Y)P(X|Y)
- Challenge: we need to estimate model parameters without labeled Ys

Y	X_1	X_2
??	0.1	2.1
??	0.5	-1.1
??	0.0	3.0
??	-0.1	-2.0
??	0.2	1.5
…	…	…

Gaussian Mixture Models

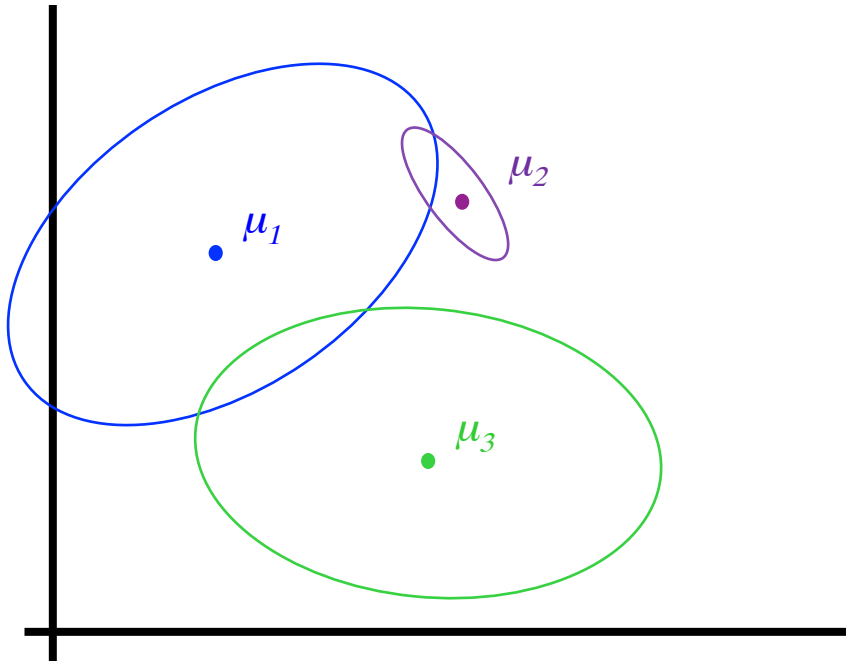
- P(Y): There are k components
- P(X|Y): Each component generates data from a **multivariate Gaussian** with mean μ_i and covariance matrix Σ_i

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

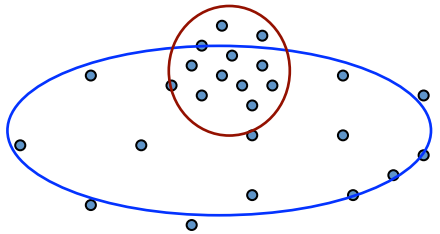
1. Choose component i with probability *P(y=i)* *[Multinomial]*
2. Generate datapoint ~ N(m_i, Σ_i)

$$P(X = \mathbf{x}_j \mid Y = i) =$$

$$\frac{1}{(2\pi)^{m/2} \|\Sigma_i\|^{1/2}} \exp\left[-\frac{1}{2}\left(\mathbf{x}_j - \mu_i\right)^T \Sigma_i^{-1}\left(\mathbf{x}_j - \mu_i\right)\right]$$



What Model Should We Use?



- Depends on X!
- Here, maybe Gaussian Naïve Bayes?
  - Multinomial over clusters Y
  - (Independent) Gaussian for each X_i given Y

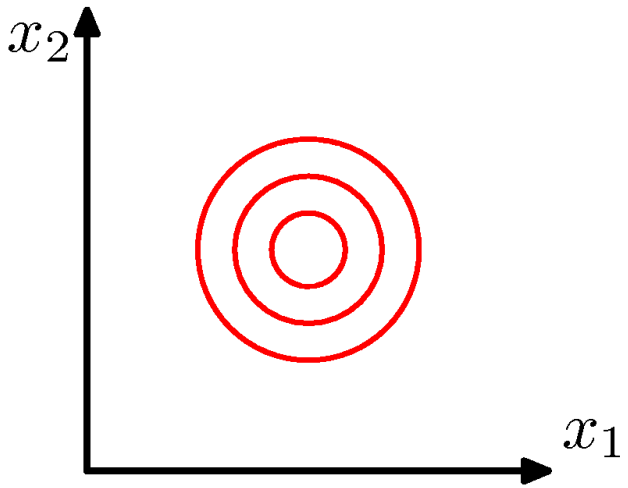
Y	X_1	X_2
??	0.1	2.1
??	0.5	-1.1
??	0.0	3.0
??	-0.1	-2.0
??	0.2	1.5
	…	…

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

$$P(X_i = x \mid Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2\pi}} \; e^{\frac{-(x-\mu_{ik})^2}{2\sigma_{ik}^2}}$$

Multivariate Gaussians

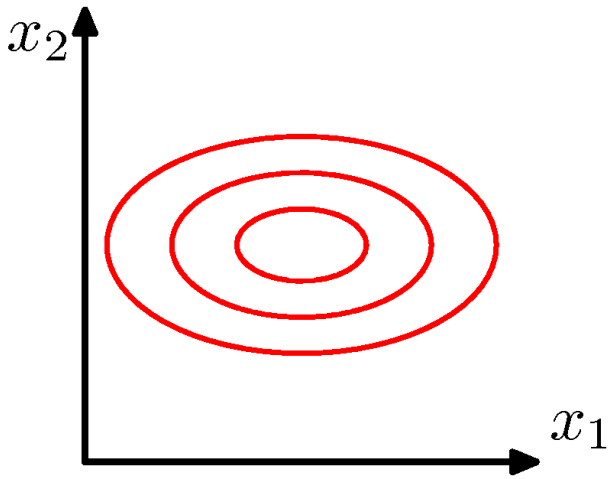
$$P(X=\mathbf{x}_j) = \frac{1}{(2\pi)^{m/2}\,\|\Sigma\|^{1/2}} \exp\left[-\frac{1}{2}\left(\mathbf{x}_j-\mu\right)^T \Sigma^{-1}\left(\mathbf{x}_j-\mu\right)\right]$$



$\Sigma \propto$ identity matrix

Multivariate Gaussians

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

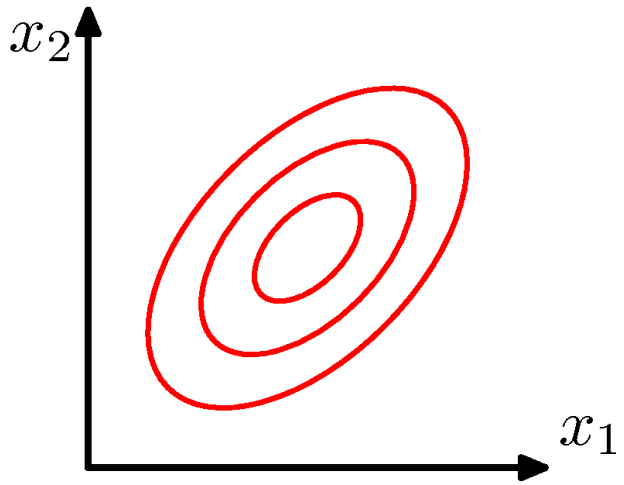


Σ = diagonal matrix

X_i are independent *ala* Gaussian NB

Multivariate Gaussians

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

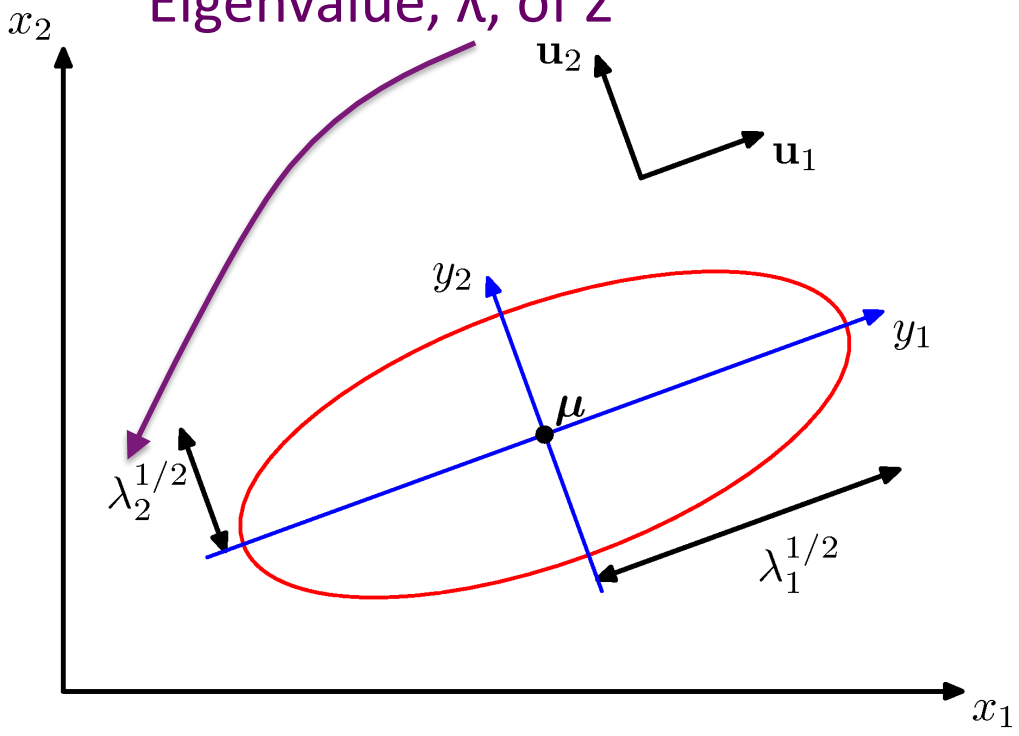


Σ = arbitrary (semidefinite) matrix:

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

Multivariate Gaussians

Eigenvalue, λ, of Σ

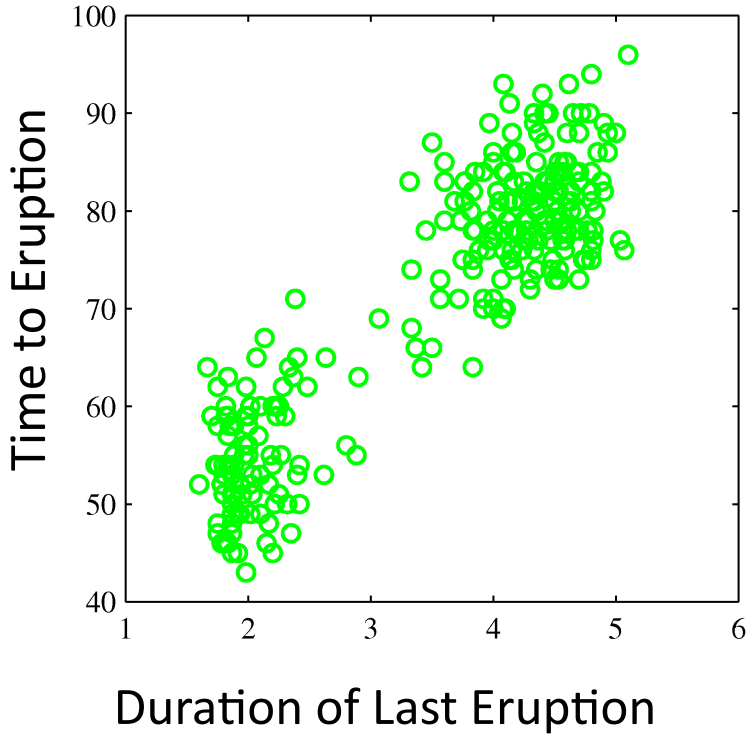


Covariance matrix, Σ, = degree to which x_i vary together

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

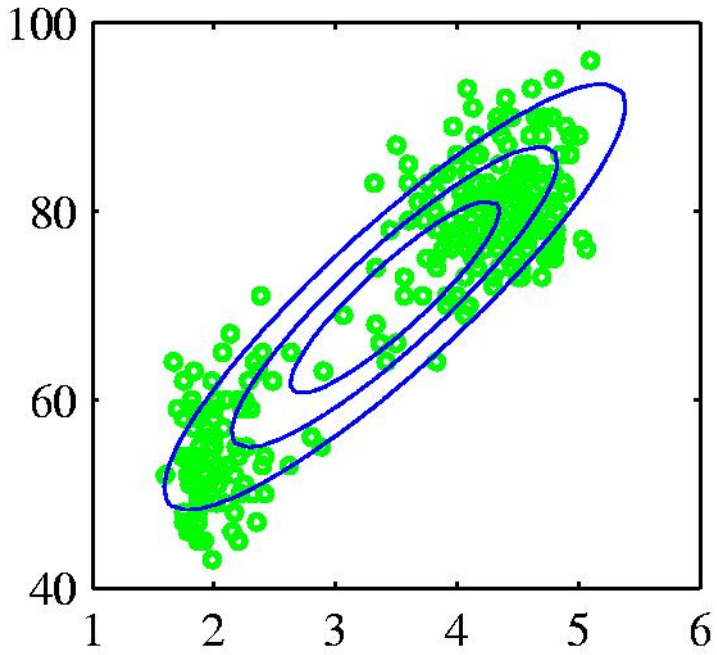
Mixtures of Gaussians (1)

Old Faithful Data Set

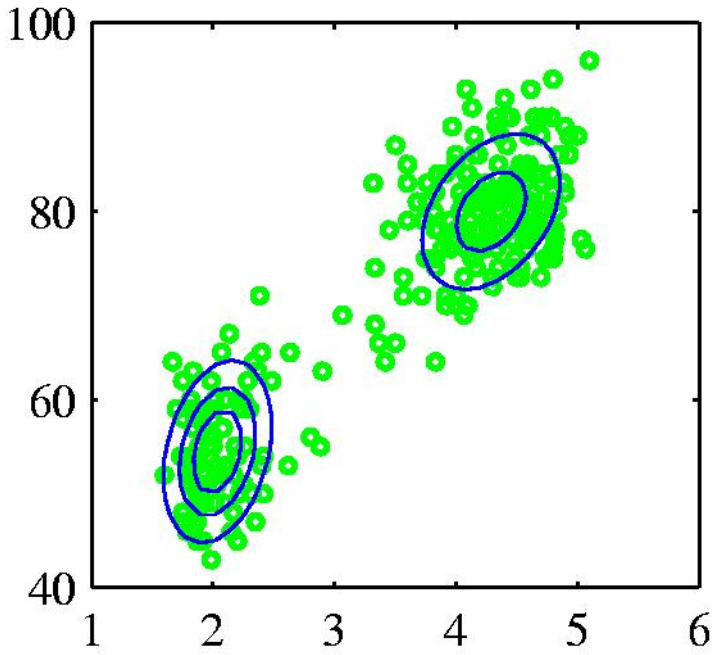


Mixtures of Gaussians (1)

Old Faithful Data Set



Single Gaussian



Mixture of two Gaussians

Mixtures of Gaussians (2)

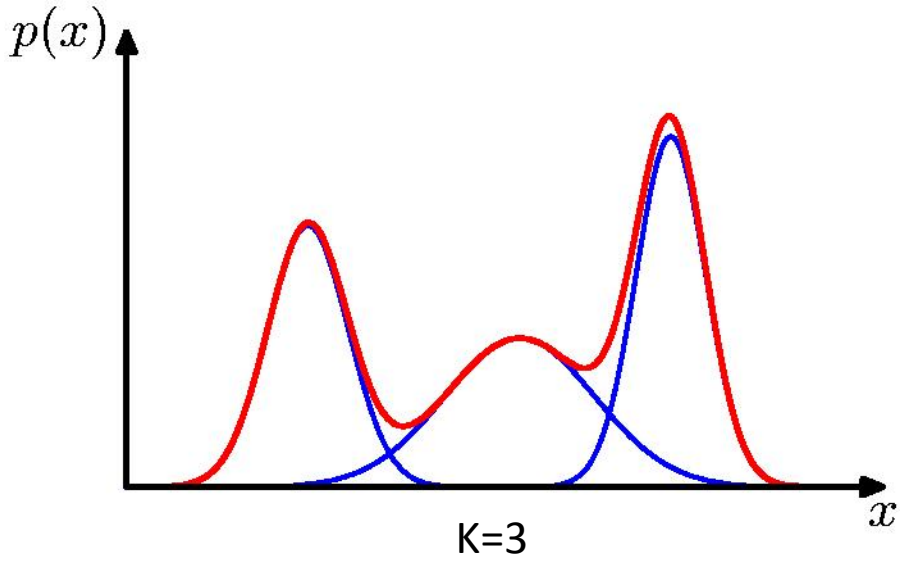
Combine simple models into a complex model:

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

Component

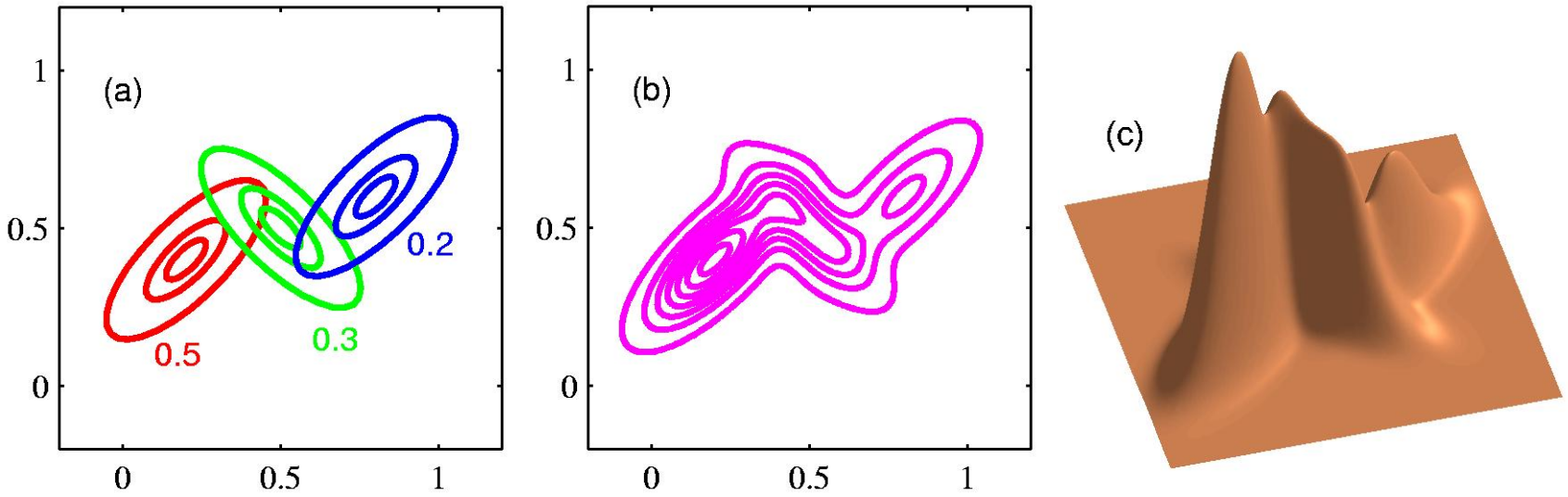
Mixing coefficient

$$\forall k : \pi_k \geqslant 0 \qquad \sum_{k=1}^{K} \pi_k = 1$$



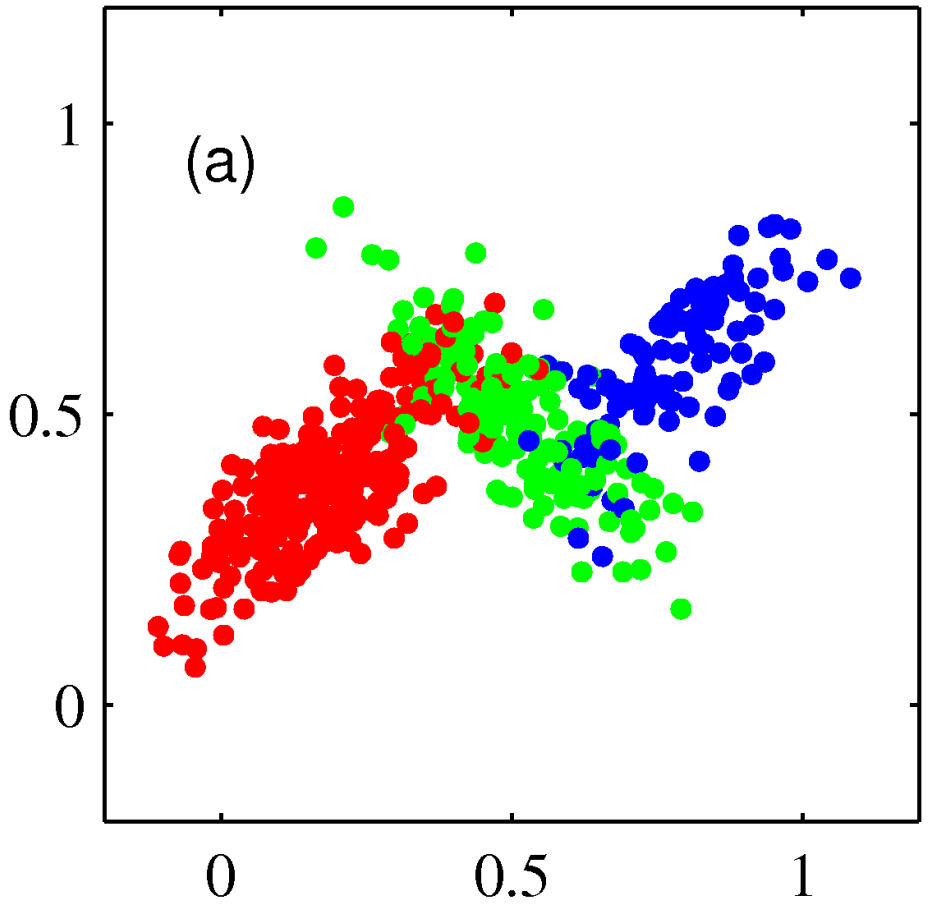
K=3

Mixtures of Gaussians (3)



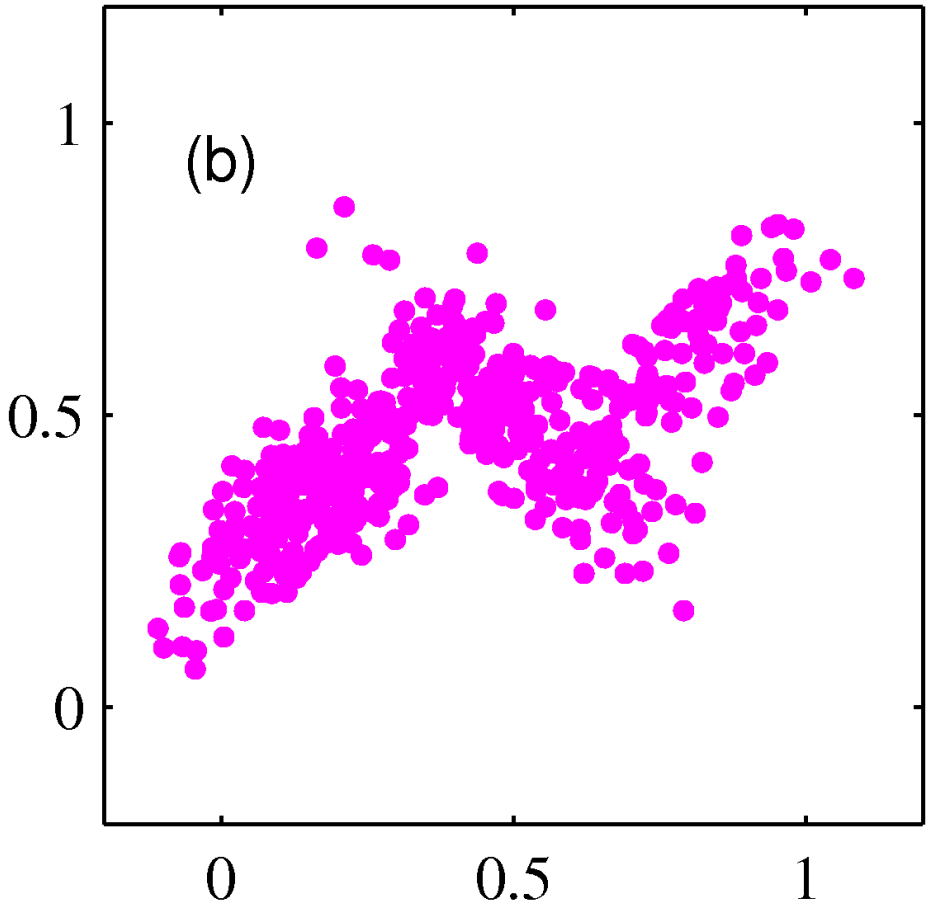
Eliminating Hard Assignments to Clusters

Model data as mixture of multivariate Gaussians



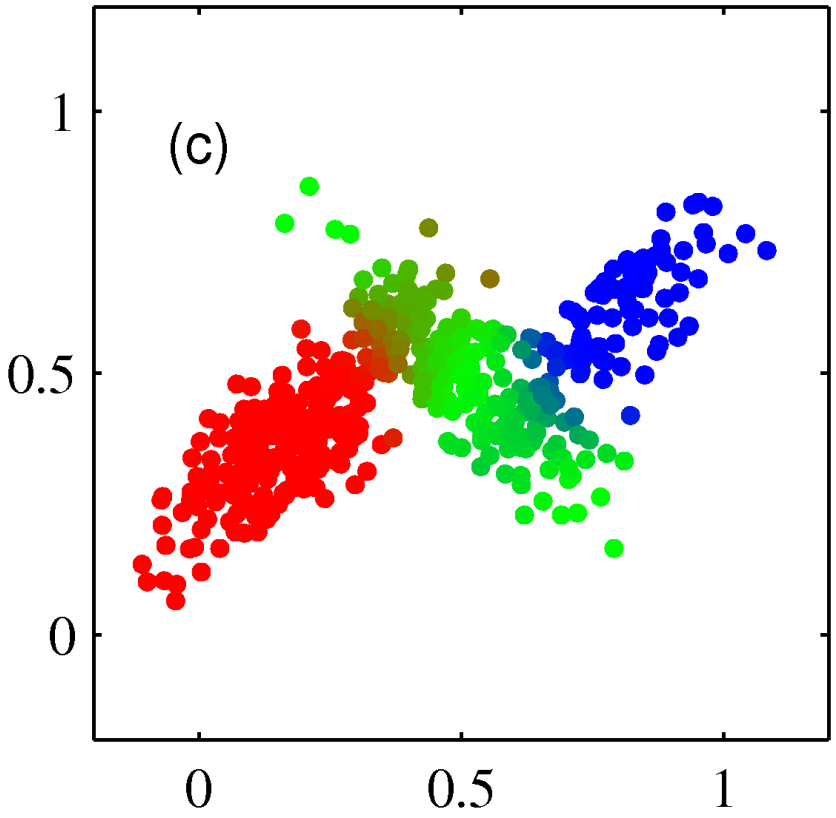
Eliminating Hard Assignments to Clusters

Model data as mixture of multivariate Gaussians



Eliminating Hard Assignments to Clusters

Model data as mixture of multivariate Gaussians



Shown is the *posterior probability* that a point was generated from ith Gaussian: $\Pr(Y = i \mid x)$

ML estimation in **supervised** setting

- Univariate Gaussian

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

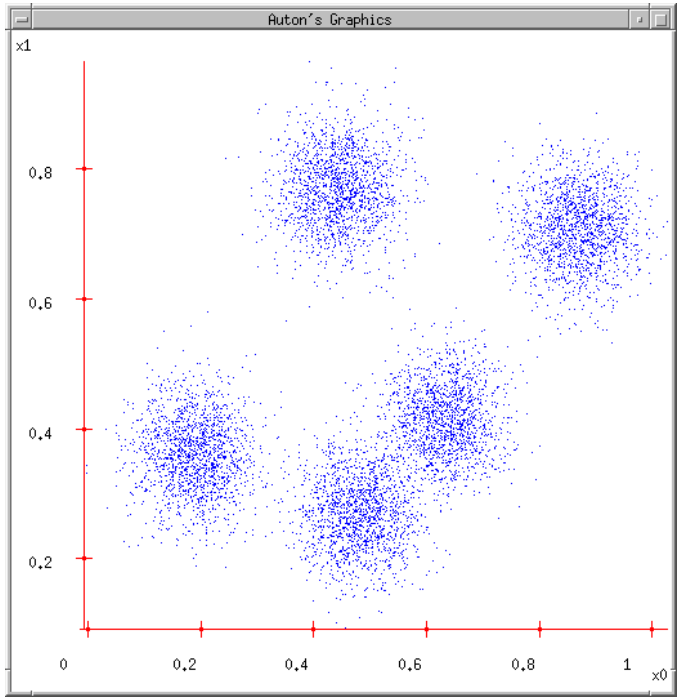
- ***Mixture*** of ***Multi***variate Gaussians

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

$$\mu^k_{ML} = \frac{1}{n}\sum_{j=1}^{n} x_n \qquad \Sigma^k_{ML} = \frac{1}{n}\sum_{j=1}^{n}\left(\mathbf{x}_j - \mu^k_{ML}\right)\left(\mathbf{x}_j - \mu^k_{ML}\right)^T$$

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

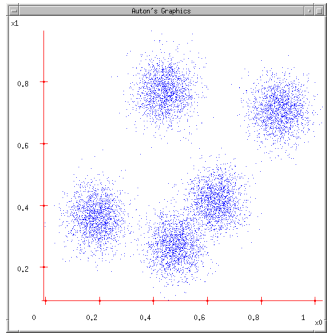
That was easy!
But what if ***unobserved data***?



- MLE:
  - $\text{argmax}_\theta \prod_j P(y_j, x_j)$
  - θ: all model parameters
    - eg, class probs, means, and variances
- But we don't know y_j's!!!
- Maximize ***marginal likelihood***:
  - $\text{argmax}_\theta \prod_j P(x_j) = \text{argmax} \prod_j \sum_{k=1}^{K} P(Y_j=k, x_j)$

How do we optimize? Closed Form?

- Maximize ***marginal likelihood***:
  - $\text{argmax}_\theta \prod_j P(x_j) = \text{argmax} \prod_j \sum_{k=1}^{K} P(Y_j=k, x_j)$
- Almost always a hard problem!
  - Usually no closed form solution
  - Even when lgP(X,Y) is convex, lgP(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 optimum…



Learning general mixtures of Gaussian

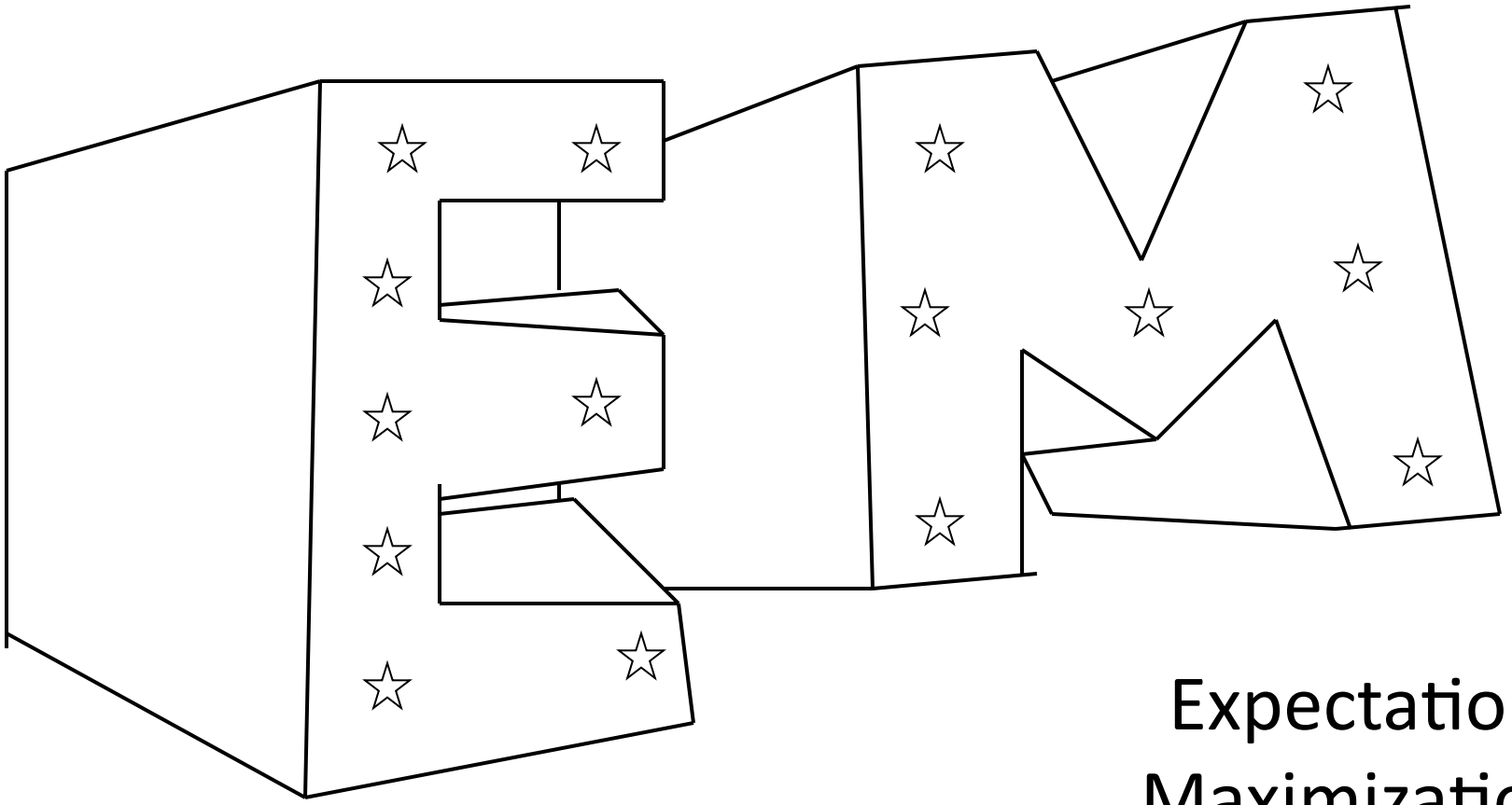
$$P(\mathbf{x}_j, y = k) = \frac{1}{(2\pi)^{m/2} \|\Sigma_k\|^{1/2}} \exp\left[-\frac{1}{2}\left(\mathbf{x}_j - \mu_k\right)^T \Sigma_k^{-1}\left(\mathbf{x}_j - \mu_k\right)\right] P(y = k)$$

- Marginal likelihood:

$$\prod_{j=1}^{m} P(\mathbf{x}_j) = \prod_{j=1}^{m} \sum_{k=1}^{K} P(\mathbf{x}_j, y = k)$$

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

- Need to differentiate and solve for μ_k, Σ_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!?!***



Expectation
Maximization

1977: Dempster, Laird, & Rubin

The EM Algorithm

- A clever method for maximizing marginal likelihood:
  - $\text{argmax}_\theta \prod_j P(x_j) = \text{argmax}_\theta \prod_j \sum_{k=1}^{K} P(Y_j=k, x_j)$
  - Based on coordinate descent. Easy to implement (eg, 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: $\text{argmax}_\theta \lg \prod_j \sum_{k=1}^{K} P(Y_j=k, x_j ; \theta) = \sum_j \lg \sum_{k=1}^{K} P(Y_j=k, x_j; \theta)$

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

- **E-step**: Compute expectations to "fill in" missing y values according to current parameters, θ
  - 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 $\theta^{new} = \text{argmax}_\theta \sum_j \sum_k P(Y_j=k \mid x_j ;\theta^{old}) \log P(Y_j=k, x_j ; \theta)$

Particularly useful when the E and M steps have closed form solutions