

Gaussian Processes (cont.)

Implementation

Algorithm 1: GP regression

Data: training data (X, \mathbf{y}) , test data \mathbf{x}_*

Input: Hyper parameters $\sigma_f^2, l, \sigma_n^2$

$$K_{ij} \leftarrow k(\mathbf{x}_i, \mathbf{x}_j)$$

$$L \leftarrow \text{cholesky}(K + \sigma_n^2 I)$$

$$\boldsymbol{\alpha} \leftarrow L^T \backslash (L \backslash \mathbf{y})$$

$$\mathbb{E}[f_*] \leftarrow \mathbf{k}_*^T \boldsymbol{\alpha}$$

$$\mathbf{v} \leftarrow L \backslash \mathbf{k}_*$$

$$\text{var}[f_*] \leftarrow k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^T \mathbf{v}$$

$$\log p(\mathbf{y} \mid X) \leftarrow -\frac{1}{2}\mathbf{y}^T \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{N}{2} \log(2\pi)$$



Precomputed
during Training



Test Phase

- Cholesky decomposition is numerically stable
- Can be used to compute inverse efficiently



Estimating the Hyperparameters

To find optimal hyper parameters we need the **marginal likelihood**:

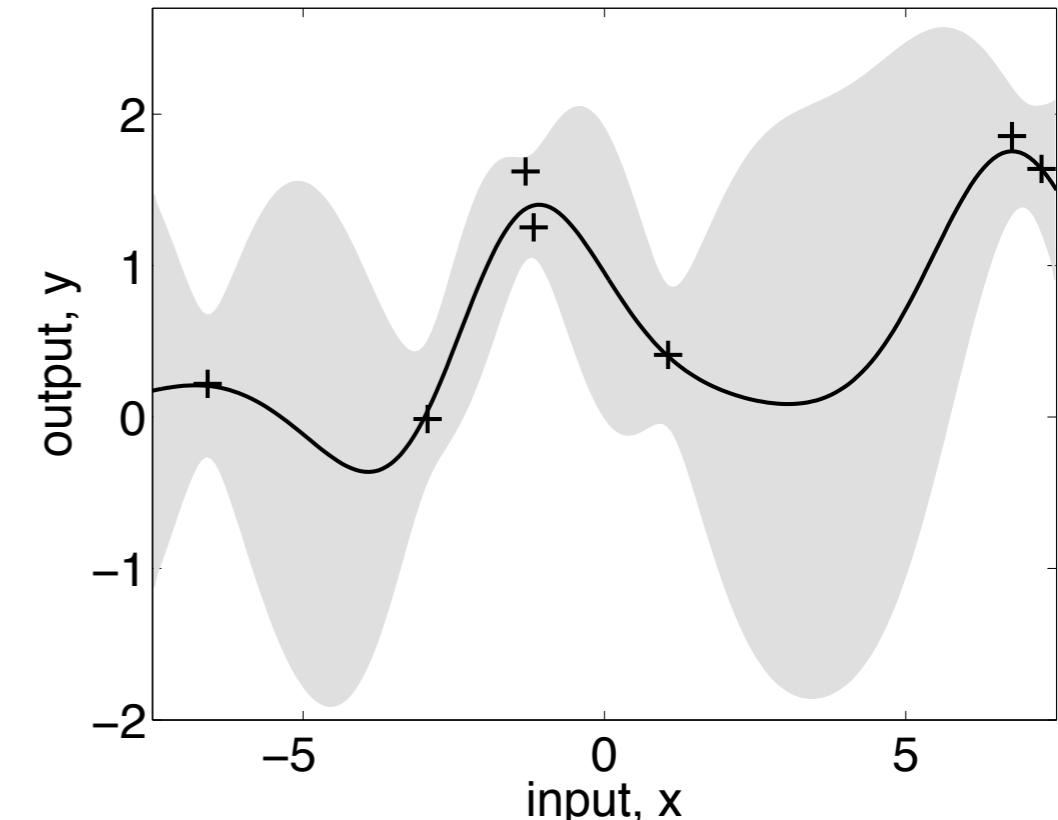
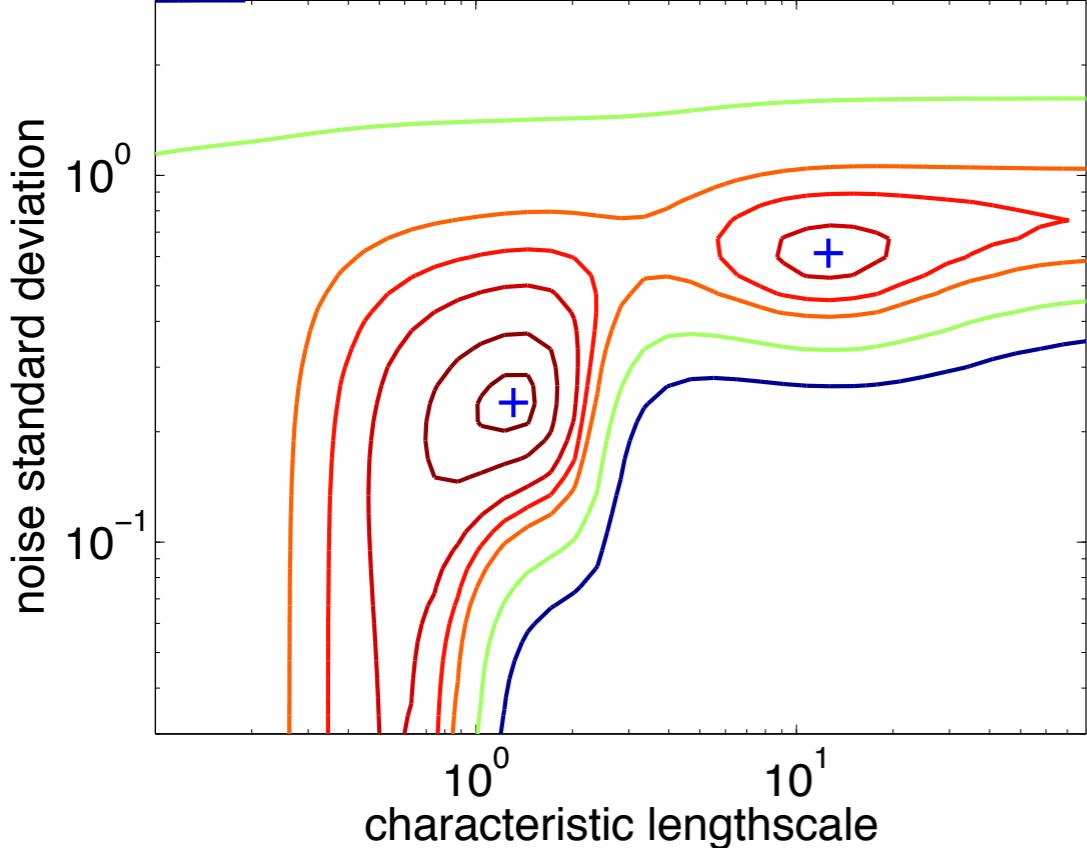
$$p(\mathbf{y} \mid X) = \int p(\mathbf{y} \mid \mathbf{f}, X)p(\mathbf{f} \mid X)d\mathbf{f}$$

This expression implicitly depends on the hyper parameters, but \mathbf{y} and X are given from the training data. It can be computed in closed form, as all terms are Gaussians.

We take the logarithm, compute the derivative and set it to 0. This is the **training step**.

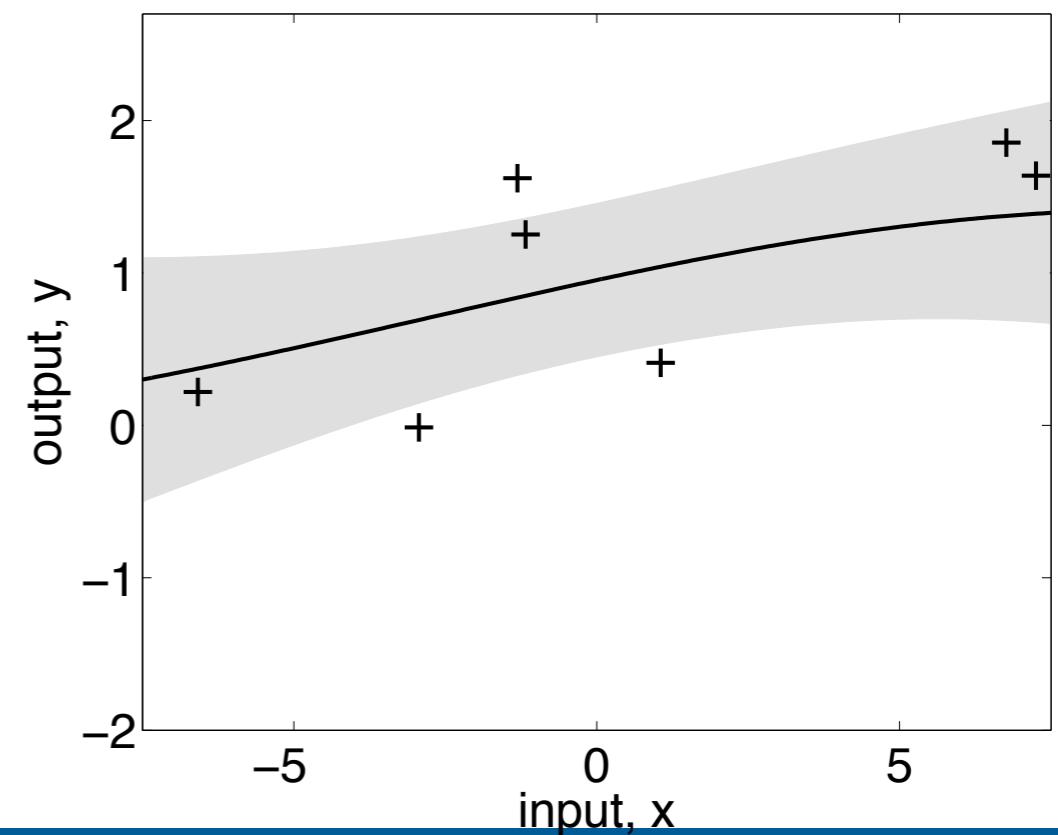


Estimating the Hyperparameters



The log marginal likelihood is not necessarily concave, i.e. it can have local maxima.

The local maxima can correspond to sub-optimal solutions.



Automatic Relevance Determination

- We have seen how the covariance function can be generalized using a matrix M
- If M is diagonal this results in the kernel function

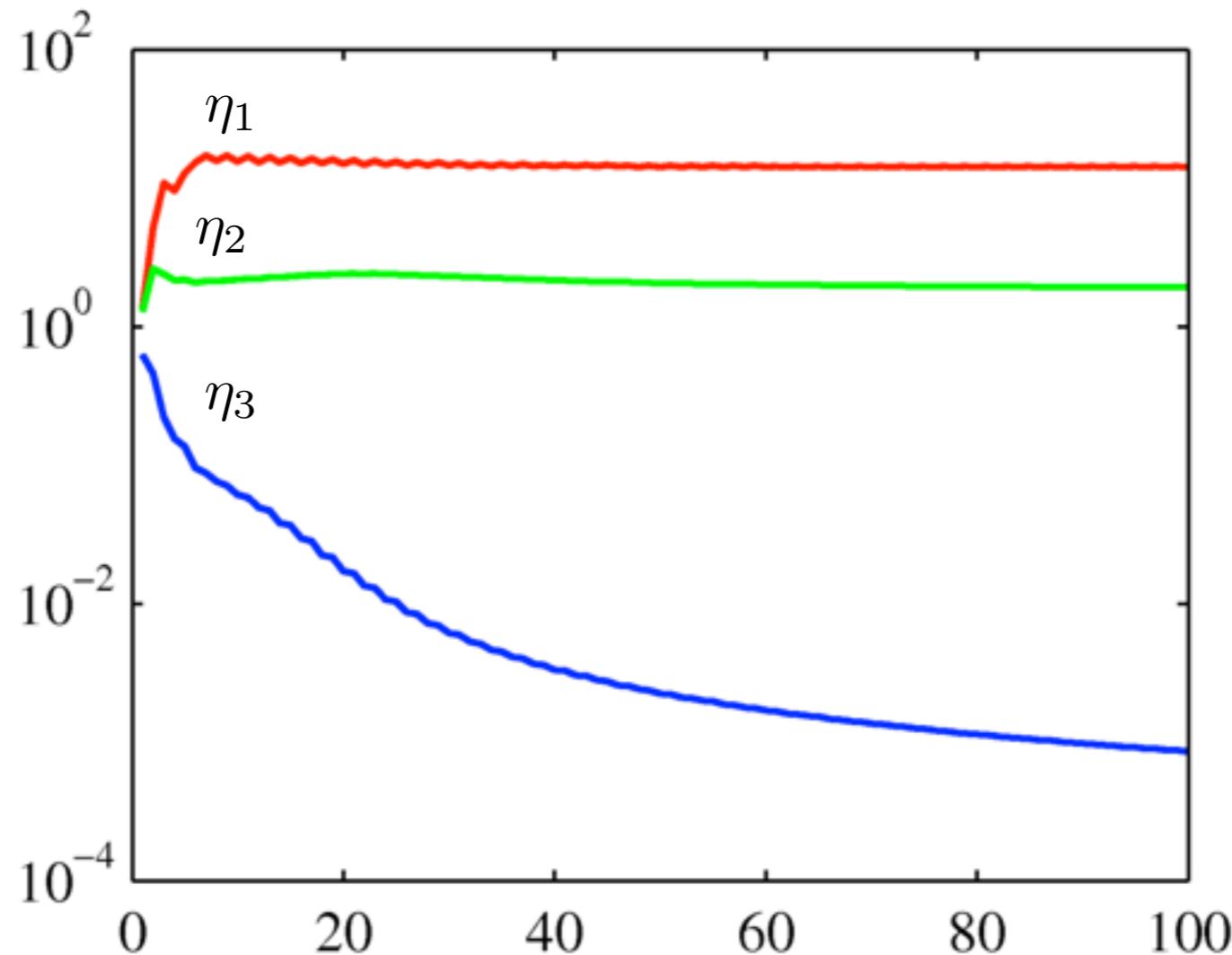
$$k(\mathbf{x}, \mathbf{x}') = \sigma_f \exp \left(\frac{1}{2} \sum_{i=1}^D \eta_i (x_i - x'_i)^2 \right)$$

- We can interpret the η_i as weights for each feature dimension
- Thus, if the length scale $l_i = 1/\eta_i$ of an input dimension is large, the input is less relevant
- During training this is done automatically



Automatic Relevance Determination

3-dimensional
data, parameters
 $\eta_1 \ \eta_2 \ \eta_3$ as they
evolve during
training



During the optimization process to learn the hyper-parameters, the reciprocal length scale for one parameter decreases, i.e.:

This hyper parameter is not very relevant!



Gaussian Processes - Classification

Gaussian Processes For Classification

In regression we have $y \in \mathbb{R}$, in binary classification we have $y \in \{-1; 1\}$

To use a GP for classification, we can apply a **sigmoid** function to the posterior obtained from the GP and compute the class probability as:

$$p(y = +1 \mid \mathbf{x}) = \sigma(f(\mathbf{x}))$$

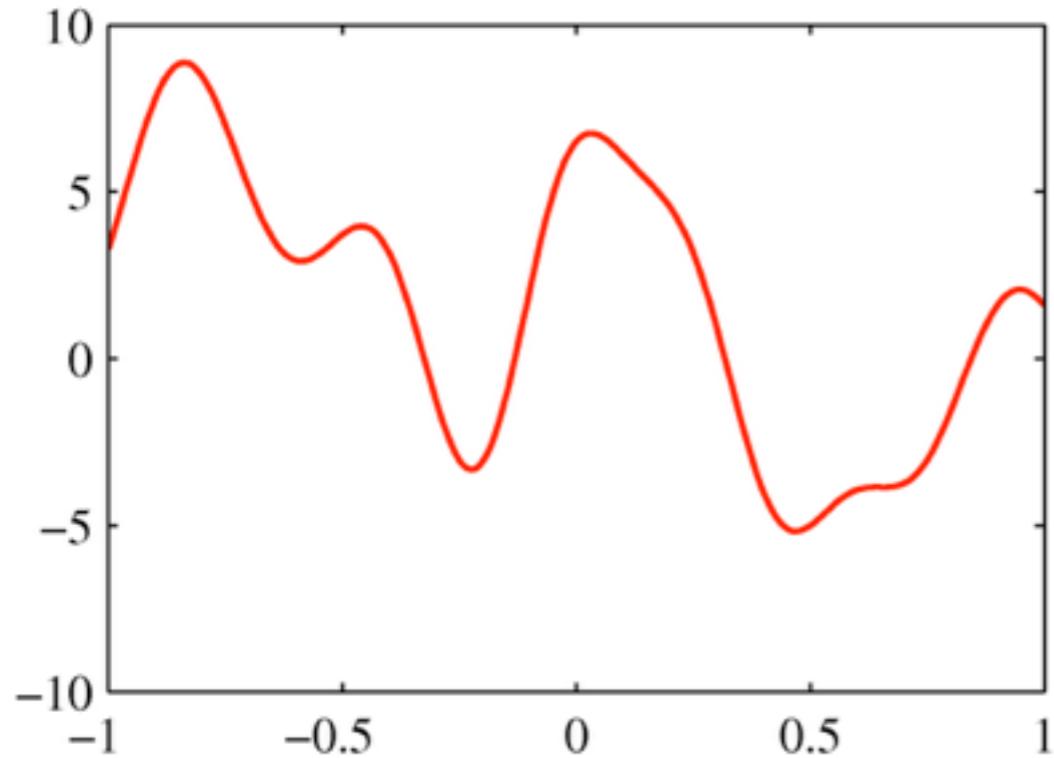
If the sigmoid function is symmetric: $\sigma(-z) = 1 - \sigma(z)$ then we have $p(y \mid \mathbf{x}) = \sigma(yf(\mathbf{x}))$.

A typical type of sigmoid function is the logistic sigmoid:

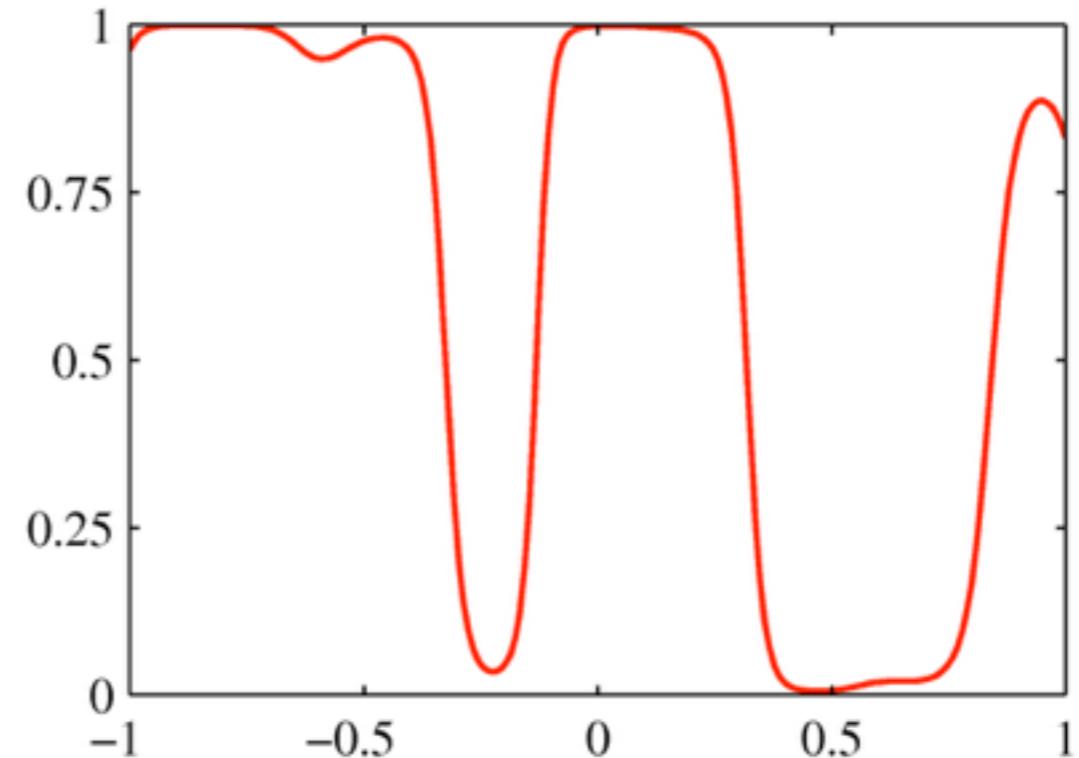
$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



Application of the Sigmoid Function



Function sampled from
a Gaussian Process



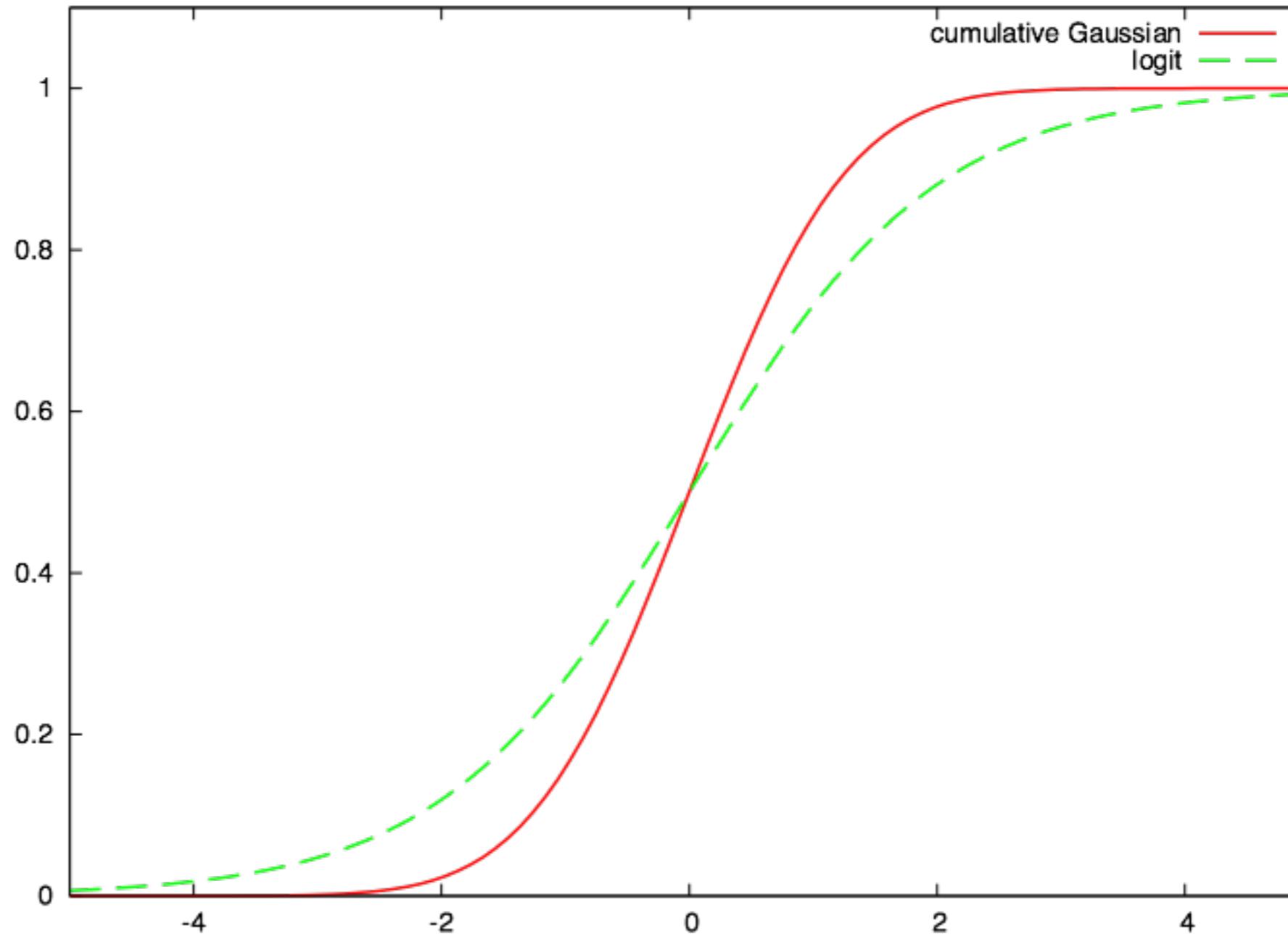
Sigmoid function applied to
the GP function

Another symmetric sigmoid function is the
cumulative Gaussian:

$$\Phi(z) = \int_{-\infty}^z \mathcal{N}(x \mid 0, 1) dx$$



Visualization of Sigmoid Functions



The cumulative Gaussian is slightly steeper than the logistic sigmoid



The Latent Variables

In regression, we directly estimated f as

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

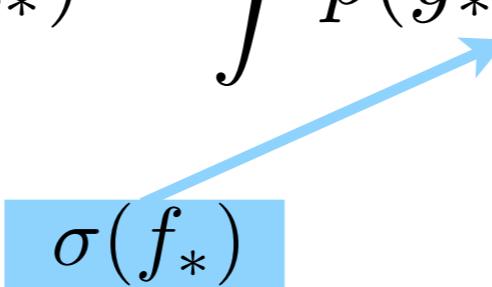
and values of f were observed in the training data. Now only labels +1 or -1 are observed and f is treated as a set of **latent variables**.

A major advantage of the Gaussian process classifier over other methods is that it **marginalizes** over all latent functions rather than maximizing some model parameters.



Class Prediction with a GP

The aim is to compute the predictive distribution

$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$


$\sigma(f_*)$



Class Prediction with a GP

The aim is to compute the predictive distribution

$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

we marginalize over the latent variables from the training data:

$$p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f} \mid X, \mathbf{y}) d\mathbf{f}$$

predictive distribution of the
latent variable (from regression)



Class Prediction with a GP

The aim is to compute the predictive distribution

$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

we marginalize over the latent variables from the training data:

$$p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f} \mid X, \mathbf{y}) d\mathbf{f}$$

we need the posterior over the latent variables:

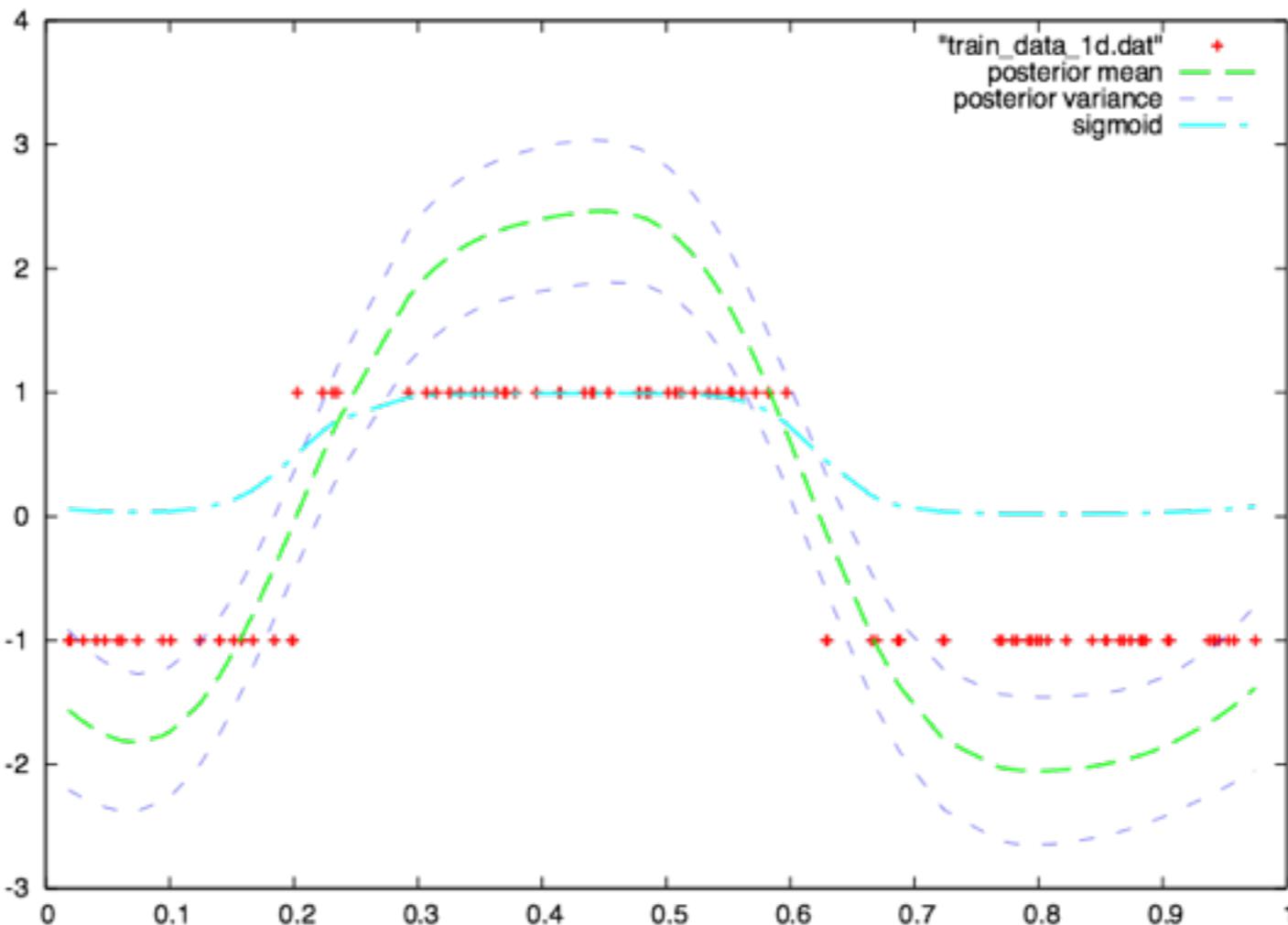
$$p(\mathbf{f} \mid X, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f}) p(\mathbf{f} \mid X)}{p(\mathbf{y} \mid X)}$$

Diagram illustrating the components of the posterior distribution:

- likelihood (sigmoid)**: Represented by a blue box on the left.
- prior**: Represented by a blue box on the right.
- normalizer**: Represented by a blue box at the bottom right.
- Blue arrows indicate the flow of information: one arrow points from the likelihood box to the numerator term $p(\mathbf{y} \mid \mathbf{f})$, another arrow points from the prior box to the numerator term $p(\mathbf{f} \mid X)$, and a third arrow points from the normalizer box to the denominator term $p(\mathbf{y} \mid X)$.



A Simple Example



- Red: Two-class training data
- Green: mean function of $p(f | X, y)$
- Light blue: sigmoid of the mean function



But There Is A Problem...

$$p(\mathbf{f} \mid X, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f})p(\mathbf{f} \mid X)}{p(\mathbf{y} \mid X)}$$

- The likelihood term is not a Gaussian!
- This means, we can not compute the posterior in closed form.
- There are several different solutions in the literature, e.g.:
 - Laplace approximation
 - Expectation Propagation
 - Variational methods



Laplace Approximation

$$p(\mathbf{f} \mid X, \mathbf{y}) \approx q(\mathbf{f} \mid X, \mathbf{y}) = \mathcal{N}(\mathbf{f} \mid \hat{\mathbf{f}}, H^{-1})$$

where $\hat{\mathbf{f}} = \arg \max_{\mathbf{f}} p(\mathbf{f} \mid X, \mathbf{y})$

and $H = -\nabla \nabla \log p(\mathbf{f} \mid X, \mathbf{y})|_{\mathbf{f}=\hat{\mathbf{f}}}$

second-order
Taylor expansion

To compute $\hat{\mathbf{f}}$ an iterative approach using Newton's method has to be used.

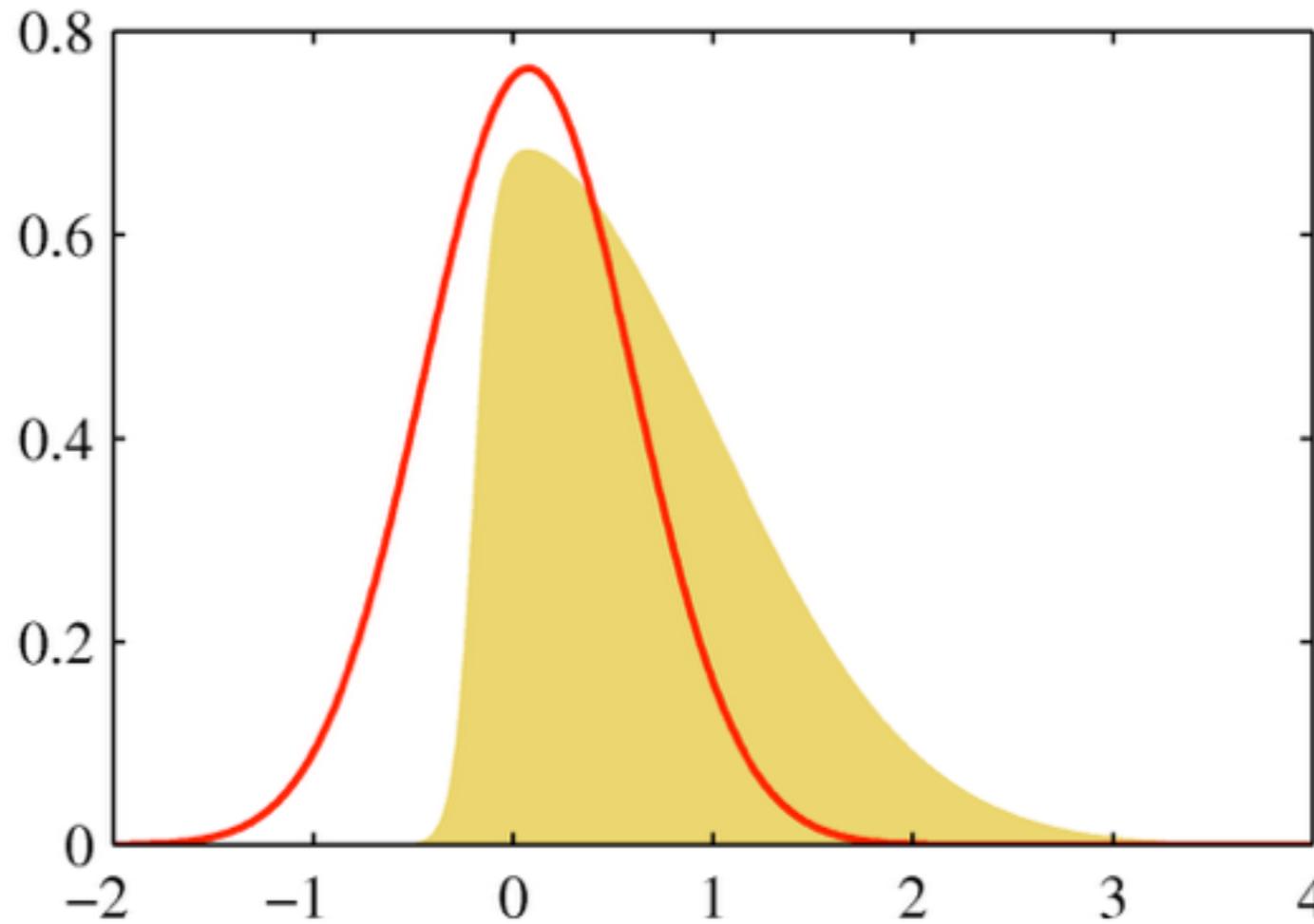
The Hessian matrix H can be computed as

$$H = K^{-1} + W$$

where $W = -\nabla \nabla \log p(\mathbf{y} \mid \mathbf{f})$ is a diagonal matrix which depends on the sigmoid function.



Laplace Approximation



- Yellow: a non-Gaussian posterior
- Red: a Gaussian approximation, the mean is the mode of the posterior, the variance is the negative second derivative at the mode



Predictions

Now that we have $p(\mathbf{f} \mid X, \mathbf{y})$ we can compute:

$$p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) p(\mathbf{f} \mid X, \mathbf{y}) d\mathbf{f}$$

From the regression case we have:

$$p(f_* \mid X, \mathbf{x}_*, \mathbf{f}) = \mathcal{N}(f_* \mid \mu_*, \Sigma_*)$$

where $\mu_* = \mathbf{k}_*^T K^{-1} \mathbf{f}$ $\Sigma_* = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T K^{-1} \mathbf{k}_*$

Linear in \mathbf{f}

This reminds us of a property of Gaussians that we saw earlier!



Gaussian Properties (Rep.)

If we are given this:

I. $p(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}_1)$

II. $p(\mathbf{y} \mid \mathbf{x}) = \mathcal{N}(\mathbf{y} \mid A\mathbf{x} + \mathbf{b}, \boldsymbol{\Sigma}_2)$

Then it follows (properties of Gaussians):

III. $p(\mathbf{y}) = \mathcal{N}(\mathbf{y} \mid A\boldsymbol{\mu} + \mathbf{b}, \boldsymbol{\Sigma}_2 + A\boldsymbol{\Sigma}_1 A^T)$

IV. $p(\mathbf{x} \mid \mathbf{y}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\Sigma}(A^T \boldsymbol{\Sigma}_2^{-1} (\mathbf{y} - \mathbf{b}) + \boldsymbol{\Sigma}_1^{-1} \mathbf{y}), \boldsymbol{\Sigma})$

where

$$\boldsymbol{\Sigma} = (\boldsymbol{\Sigma}_1^{-1} + A^T \boldsymbol{\Sigma}_s^{-1} A)^{-1}$$



Applying this to Laplace

$$\mathbb{E}[f_* \mid X, \mathbf{y}, \mathbf{x}_*] = \mathbf{k}(\mathbf{x}_*)^T K^{-1} \hat{\mathbf{f}}$$

$$\mathbb{V}[f_* \mid X, \mathbf{y}, \mathbf{x}_*] = k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T (K + W^{-1})^{-1} \mathbf{k}_*$$

It remains to compute

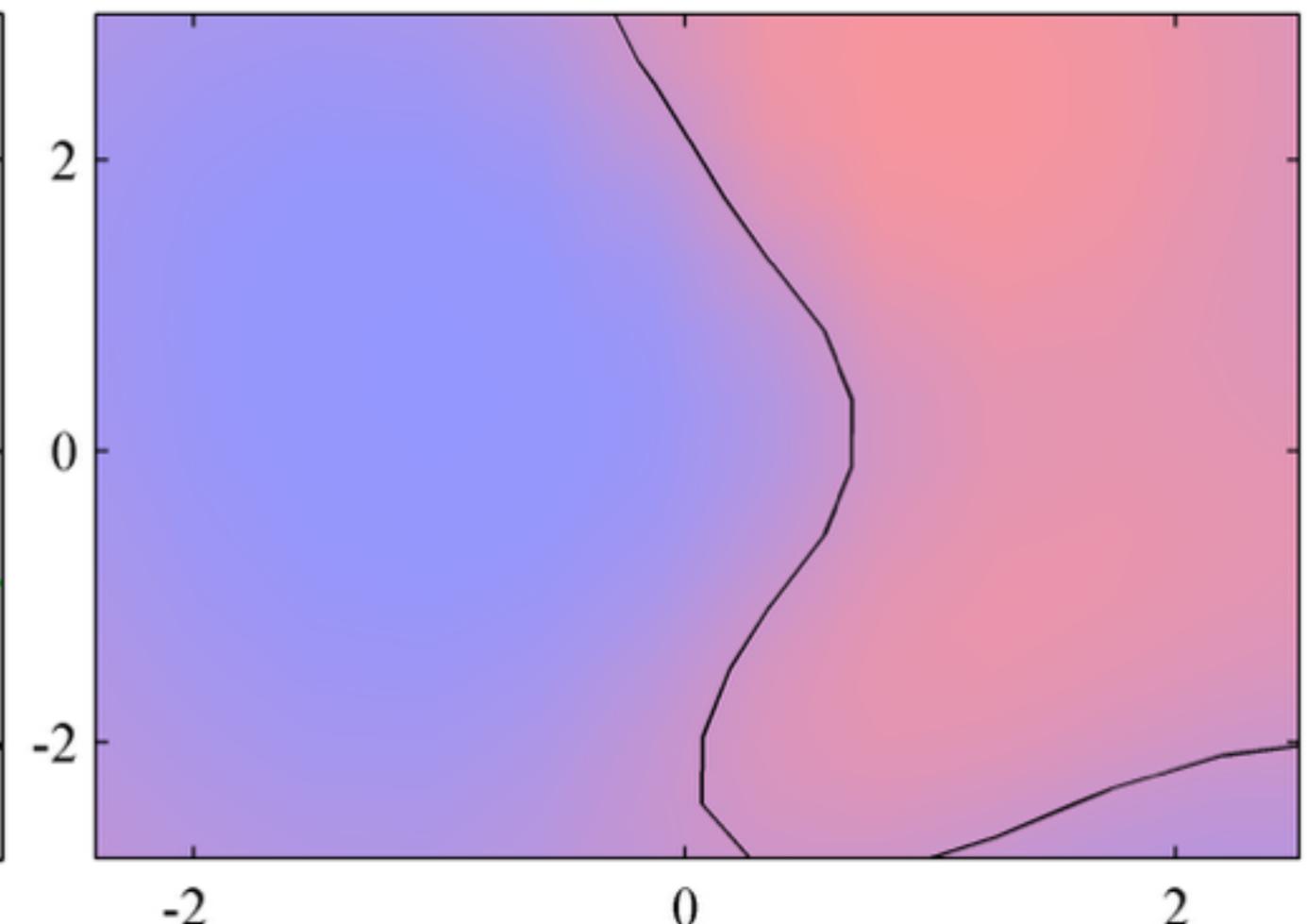
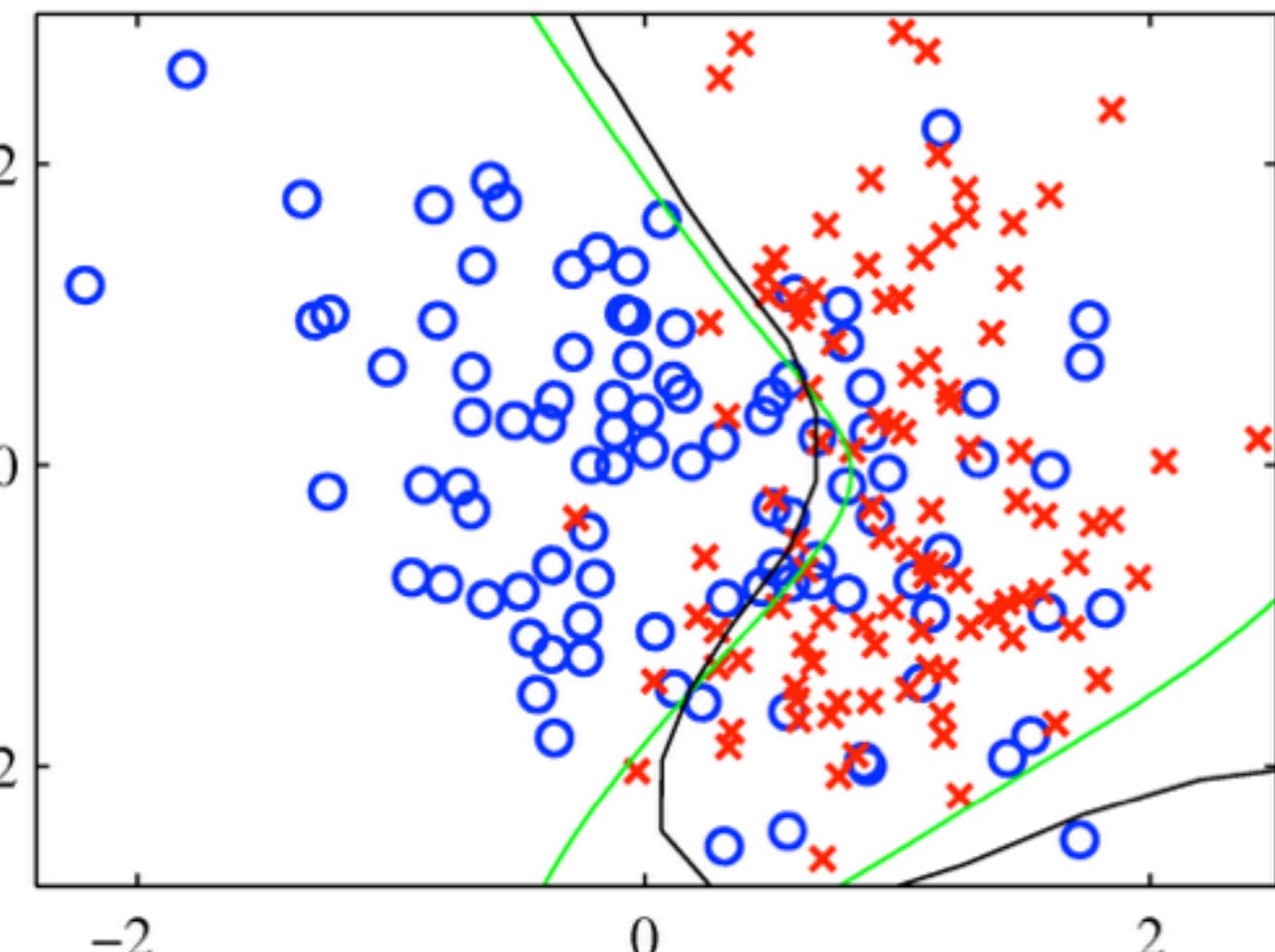
$$p(y_* = +1 \mid X, \mathbf{y}, \mathbf{x}_*) = \int p(y_* \mid f_*) p(f_* \mid X, \mathbf{y}, \mathbf{x}_*) df_*$$

Depending on the kind of sigmoid function we

- can compute this in closed form (cumulative Gaussian sigmoid)
- have to use sampling methods or analytical approximations (logistic sigmoid)



A Simple Example



- Two-class problem (training data in red and blue)
- Green line: optimal decision boundary
- Black line: GP classifier decision boundary
- Right: posterior probability



Summary

- Kernel methods solve problems by implicitly mapping the data into a (high-dimensional) feature space
- The feature function itself is not used, instead the algorithm is expressed in terms of the kernel
- Gaussian Processes are Normal distributions over functions
- To specify a GP we need a covariance function (kernel) and a mean function
- More on Gaussian Processes:
http://videolectures.net/epsrcws08_rasmussen_lgp/



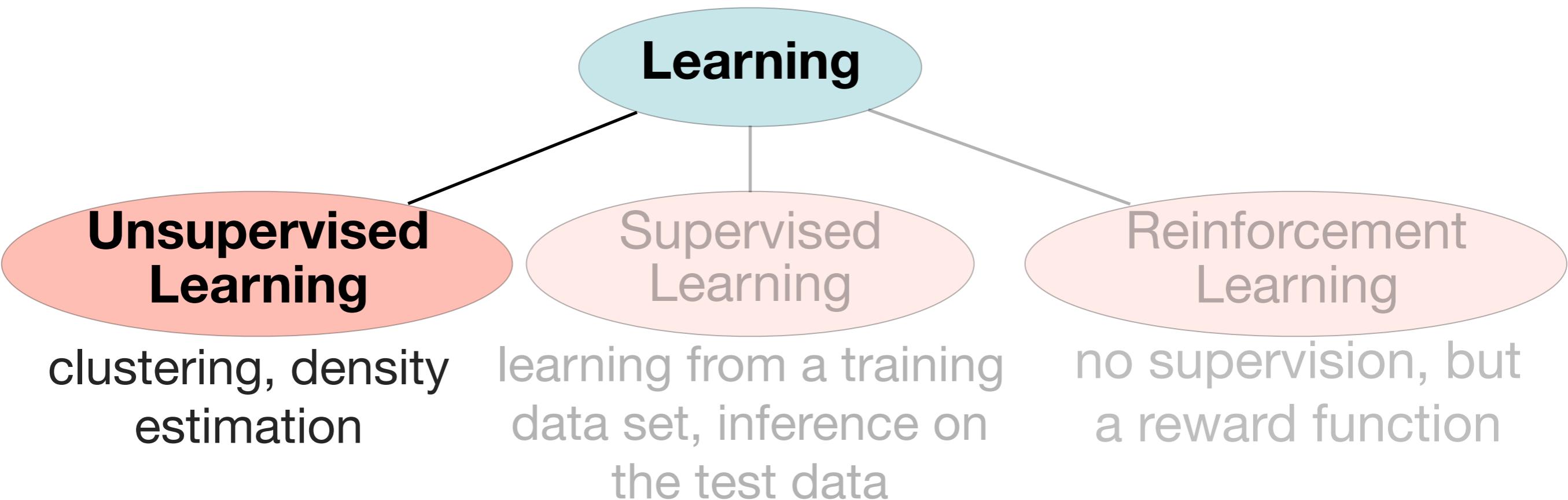
10. Clustering

Motivation

- Supervised learning is good for interaction with humans, but labels from a supervisor are sometimes hard to obtain
- Clustering is **unsupervised** learning, i.e. it tries to learn only from the data
- Main idea: find a similarity measure and group similar data objects together
- Clustering is a very old research field, many approaches have been suggested
- Main problem in most methods: how to find a good number of clusters



Categories of Learning



In unsupervised learning, there is no **ground truth** information given.

Most Unsupervised Learning methods are based on **Clustering**.



K-means Clustering

- Given: data set $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$, number of clusters K
- Goal: find cluster centers $\{\mu_1, \dots, \mu_K\}$ so that

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \mu_k\|^2$$

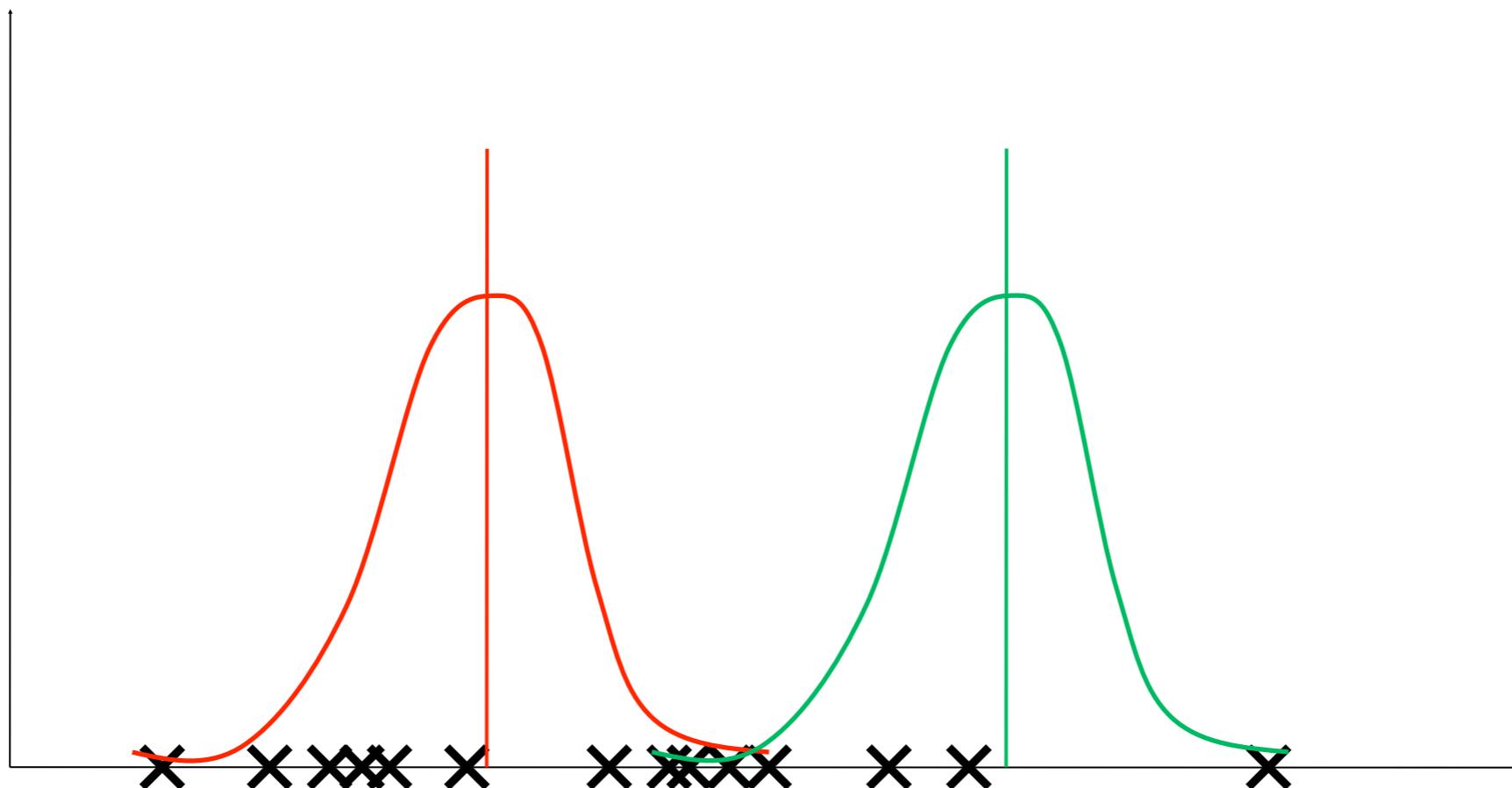
is minimal, where $r_{nk} = 1$ if \mathbf{x}_n is assigned to μ_k

- Idea: compute r_{nk} and μ_k iteratively
- Start with some values for the cluster centers
- Find optimal assignments r_{nk}
- Update cluster centers using these assignments
- Repeat until assignments or centers don't change



K-means Clustering

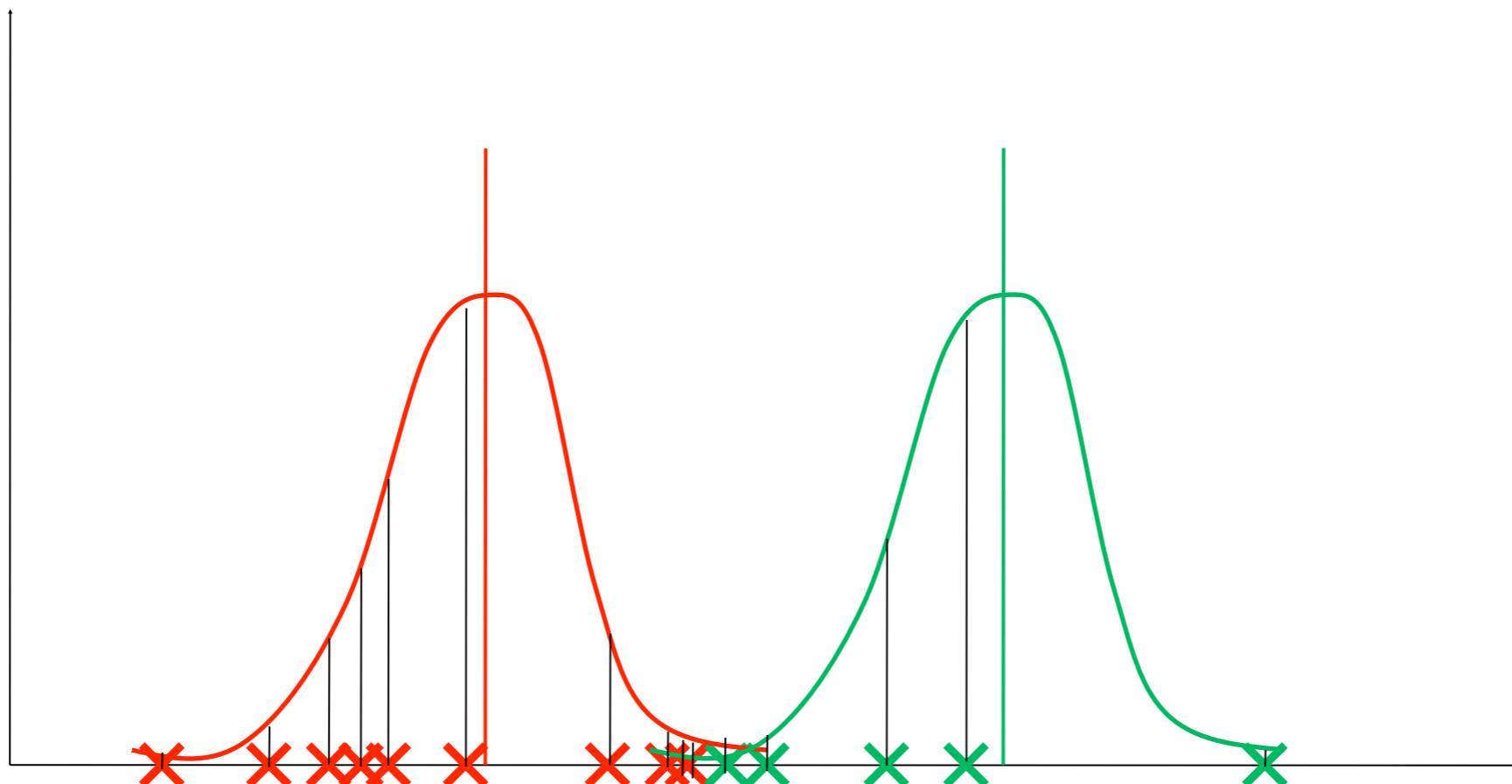
Initialize cluster means: $\{\mu_1, \dots, \mu_K\}$



K-means Clustering

Find optimal assignments:

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg \min_j \| \mathbf{x}_n - \boldsymbol{\mu}_j \| \\ 0 & \text{otherwise} \end{cases}$$

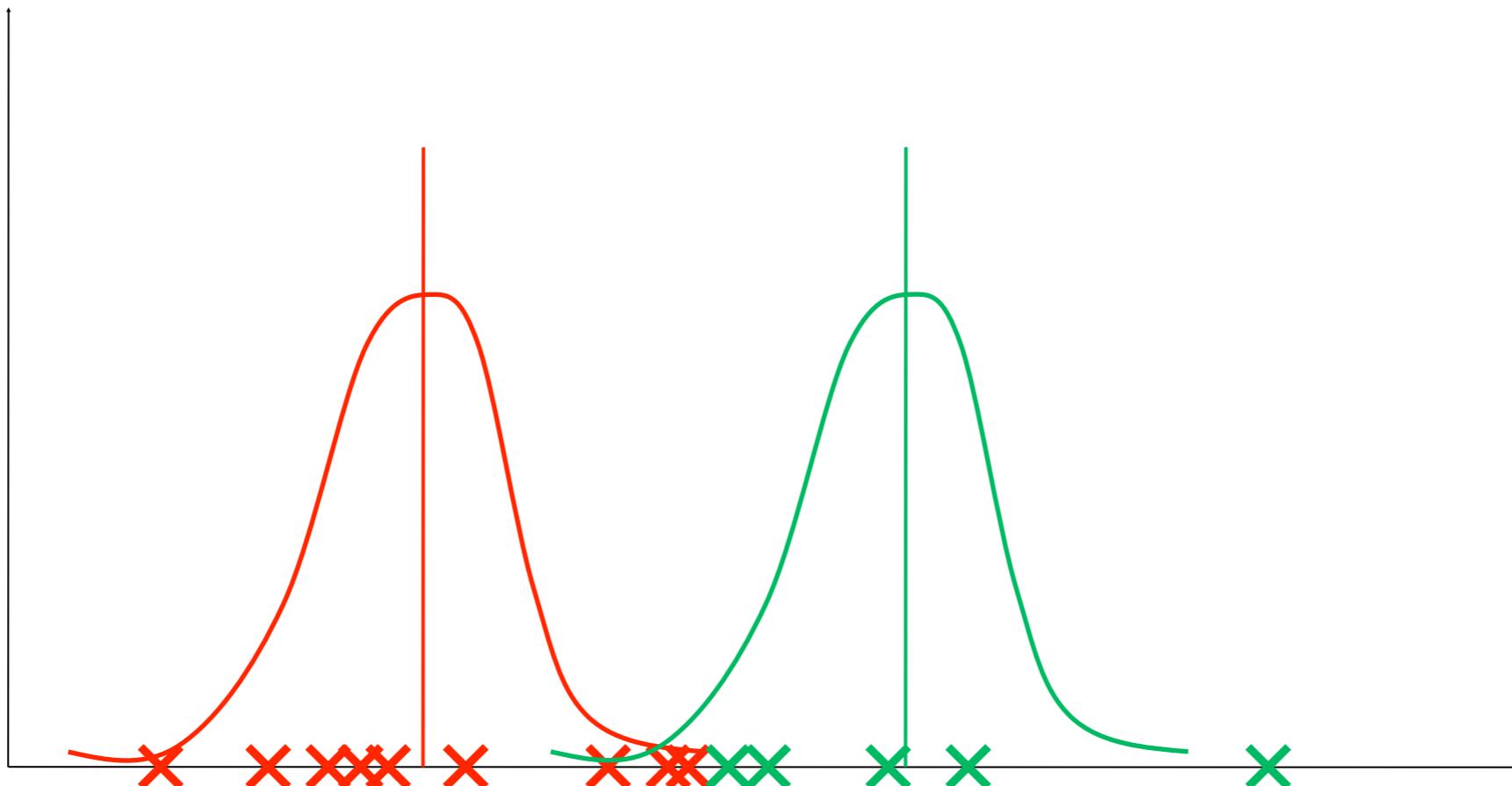


K-means Clustering

Find new optimal means:

$$\frac{\partial J}{\partial \mu_k} = 2 \sum_{n=1}^N r_{nk} (\mathbf{x}_n - \mu_k) \stackrel{!}{=} 0$$

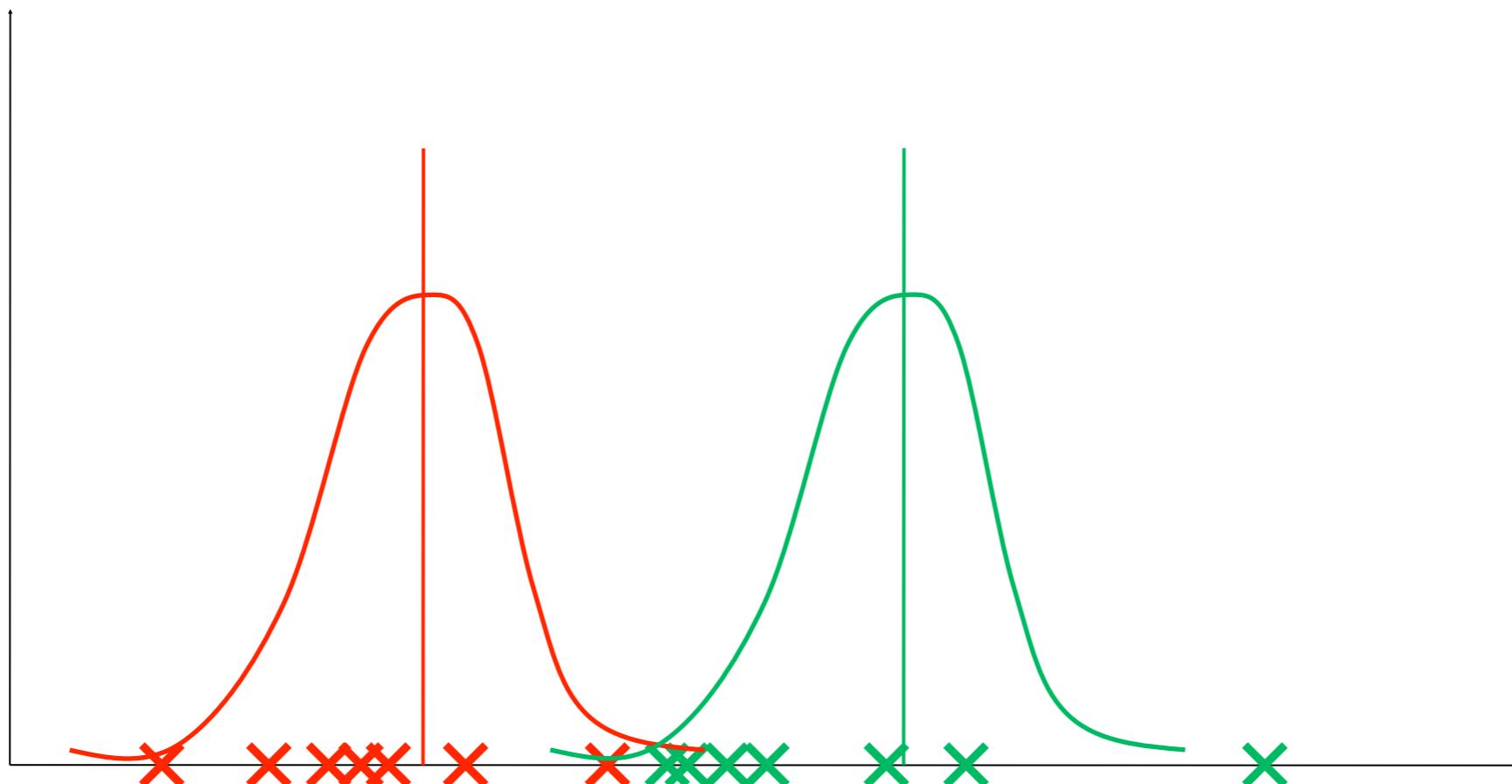
$$\Rightarrow \mu_k = \frac{\sum_{n=1}^N r_{nk} \mathbf{x}_n}{\sum_{n=1}^N r_{nk}}$$



K-means Clustering

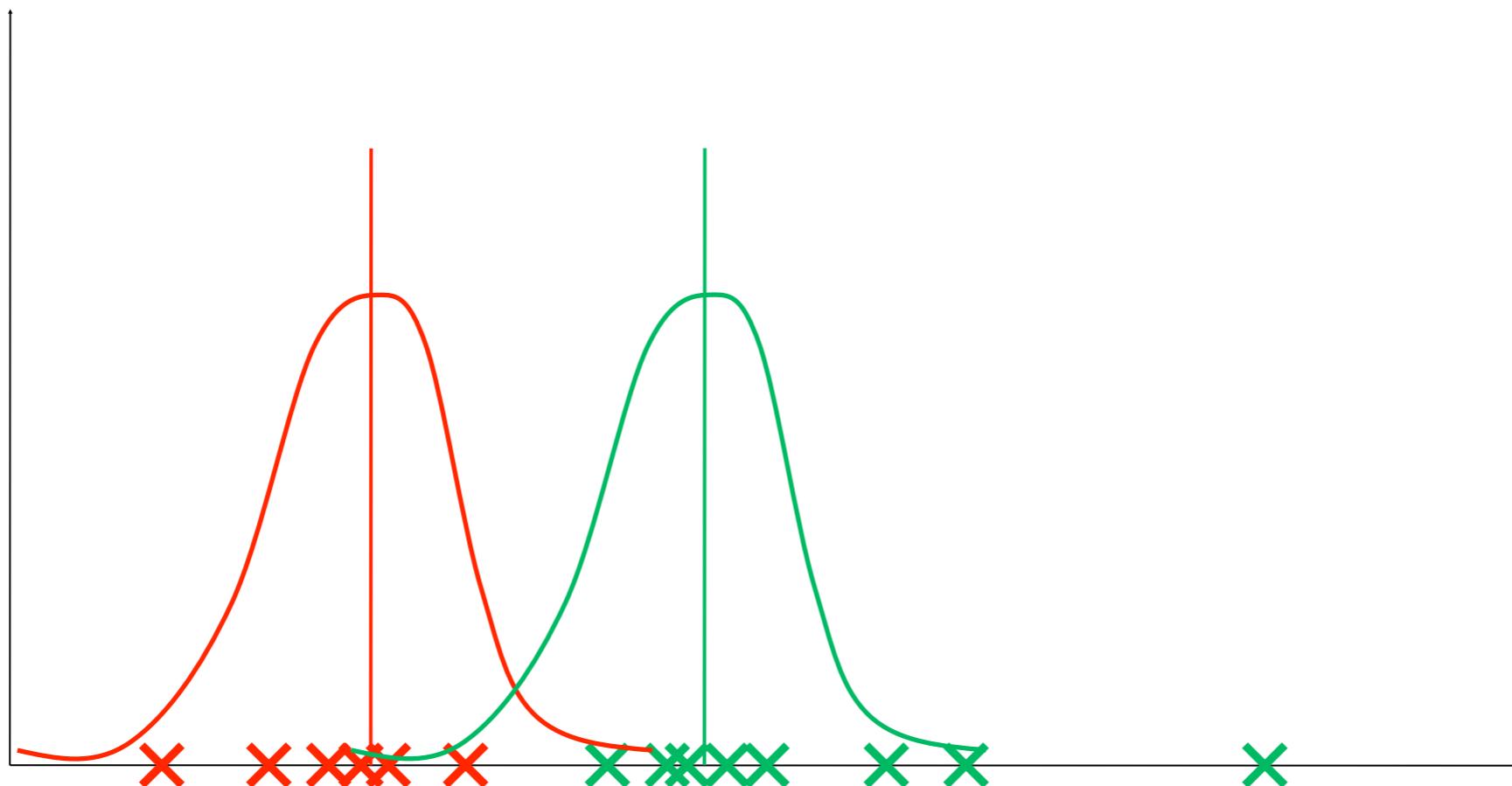
Find new optimal assignments:

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg \min_j \| \mathbf{x}_n - \boldsymbol{\mu}_j \| \\ 0 & \text{otherwise} \end{cases}$$

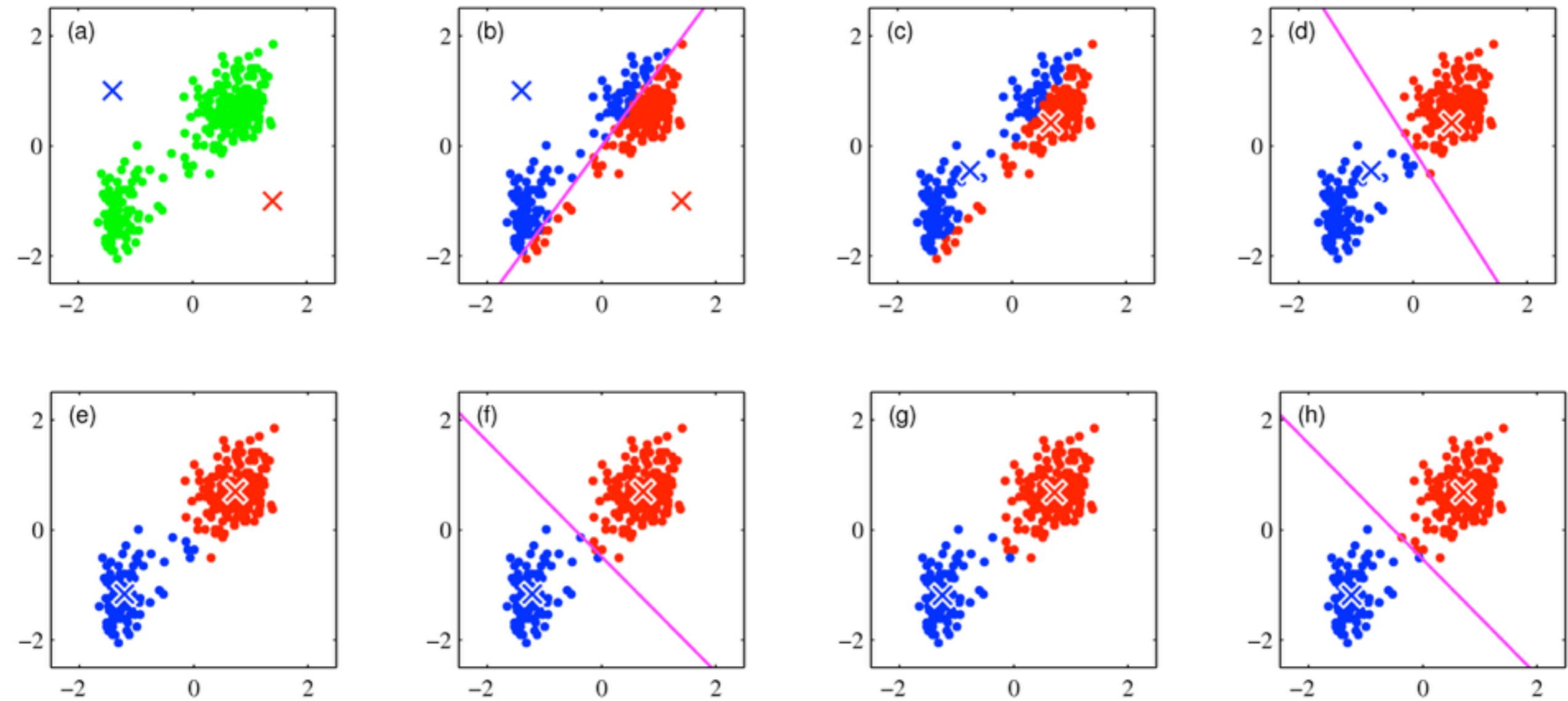


K-means Clustering

Iterate these steps until means and assignments do not change any more



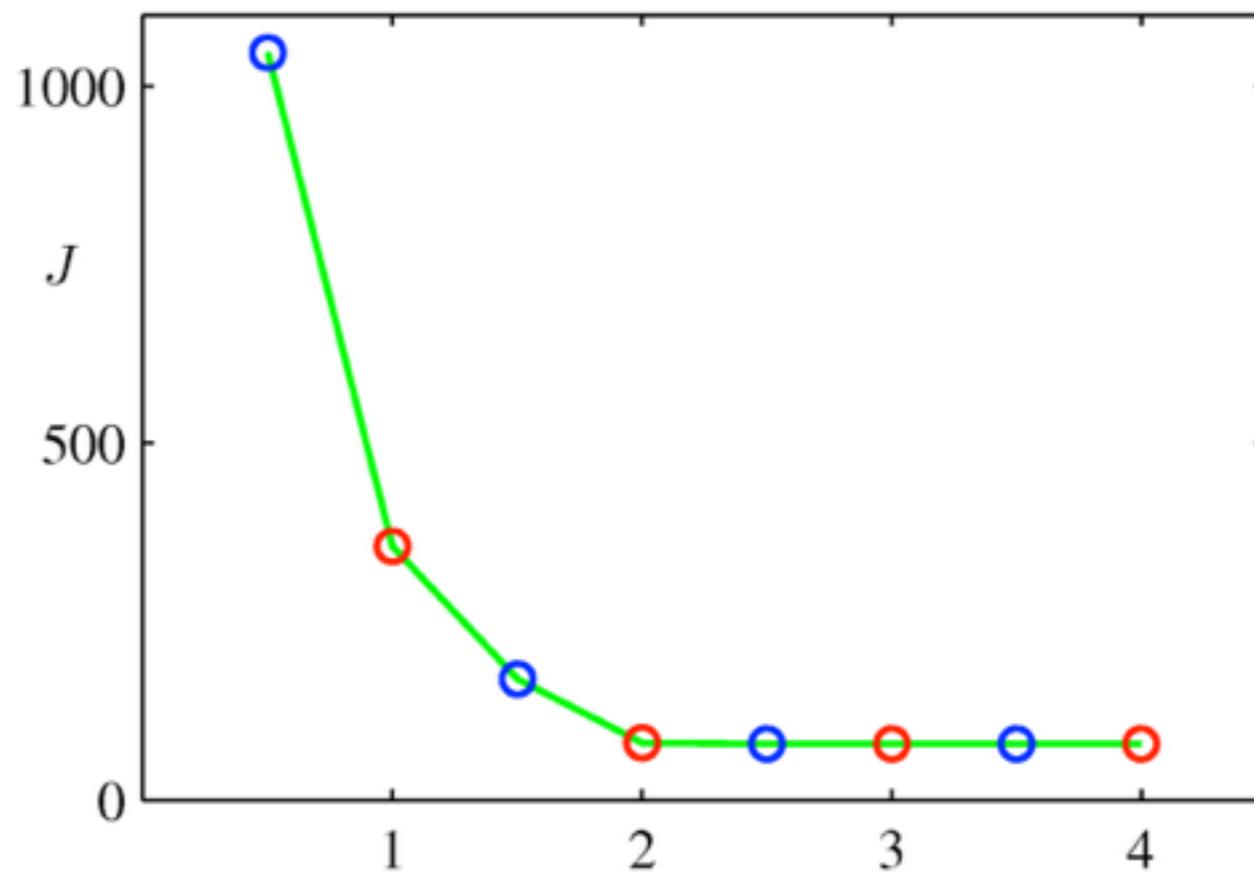
2D Example



- Real data set
- Random initialization
- Magenta line is “decision boundary”



The Cost Function



- After every step the cost function J is minimized
- Blue steps: update assignments
- Red steps: update means
- Convergence after 4 rounds



K-means for Segmentation

$K = 2$



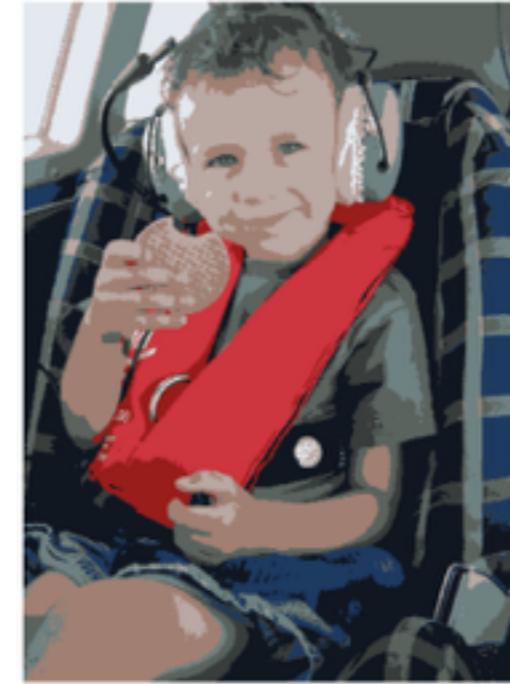
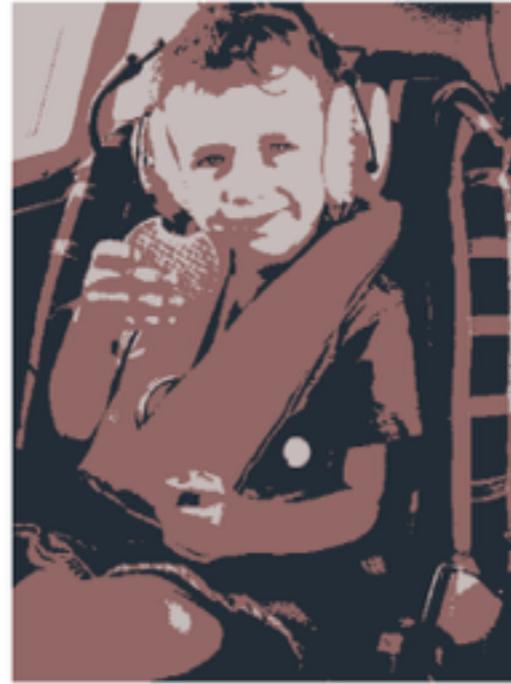
$K = 3$



$K = 10$



Original image



K-Means: Additional Remarks

- K-means converges always, but the minimum is not guaranteed to be a global one
- There is an **online** version of *K*-means
 - After each addition of \mathbf{x}_n , the nearest center μ_k is updated:
- The ***K*-medoid** variant:
 - Replace the Euclidean distance by a general measure V .

$$\tilde{J} = \sum_{n=1}^N \sum_{k=1}^K r_{nk} V(\mathbf{x}_n, \mu_k)$$



Mixtures of Gaussians

- Assume that the data consists of K clusters
- The data within each cluster is Gaussian
- For any data point \mathbf{x} we introduce a K -dimensional binary random variable \mathbf{z} so that:

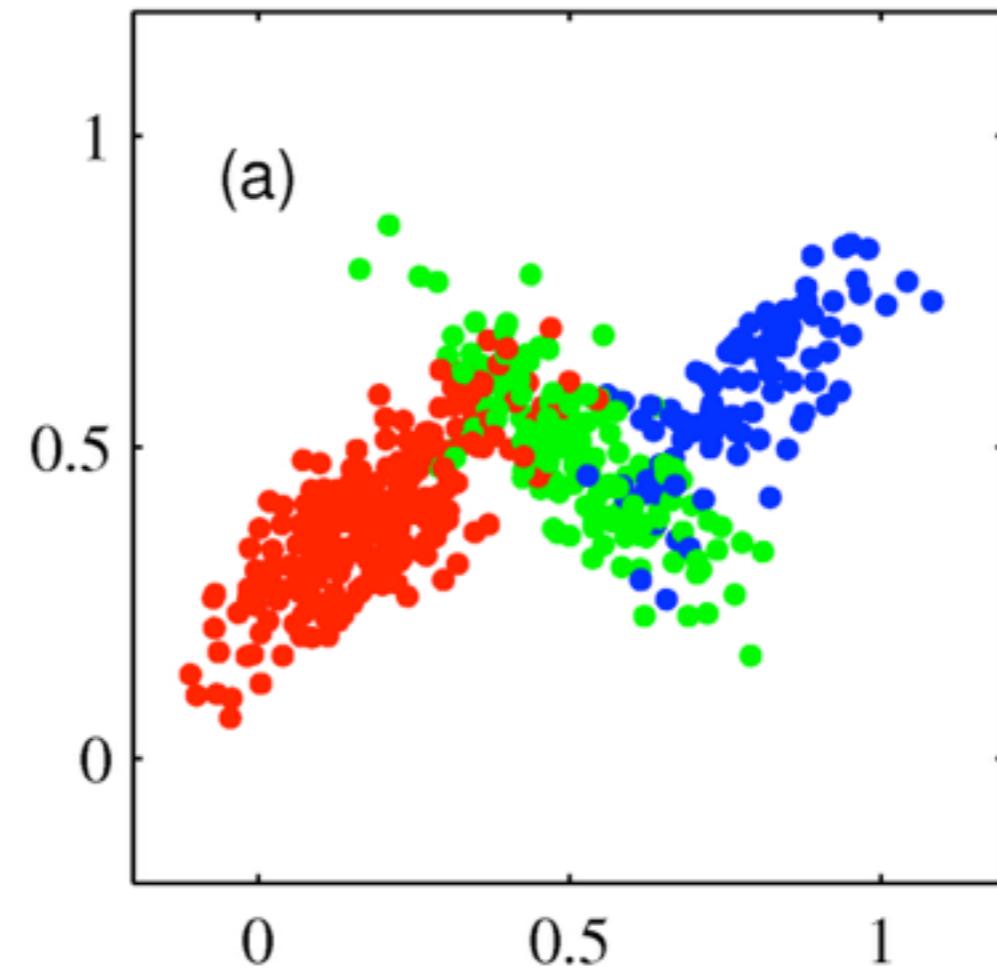
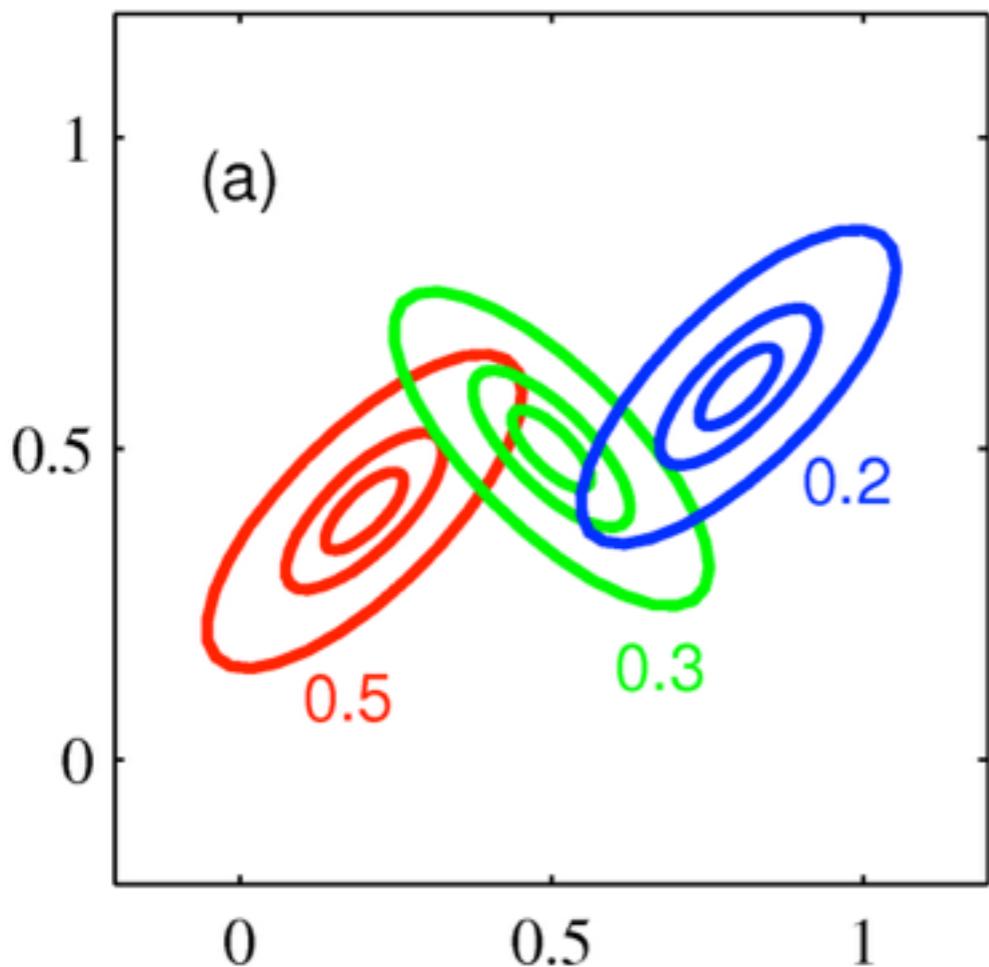
$$p(\mathbf{x}) = \sum_{k=1}^K \underbrace{p(z_k = 1)}_{=: \pi_k} \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

where

$$z_k \in \{0, 1\}, \quad \sum_{k=1}^K z_k = 1$$



A Simple Example



- Mixture of three Gaussians with mixing coefficients
- Left: all three Gaussians as contour plot
- Right: samples from the mixture model, the red component has the most samples



Parameter Estimation

- From a given set of training data $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ we want to find parameters $(\pi_{1,\dots,K}, \mu_{1,\dots,K}, \Sigma_{1,\dots,K})$ so that the likelihood is maximized (MLE):

$$p(\mathbf{x}_1, \dots, \mathbf{x}_N \mid \pi_{1,\dots,K}, \mu_{1,\dots,K}, \Sigma_{1,\dots,K}) = \prod_{n=1}^N \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n \mid \mu_k, \Sigma_k)$$

or, applying the logarithm:

$$\log p(X \mid \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \log \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n \mid \mu_k, \Sigma_k)$$

- However: this is not as easy as maximum-likelihood for single Gaussians!

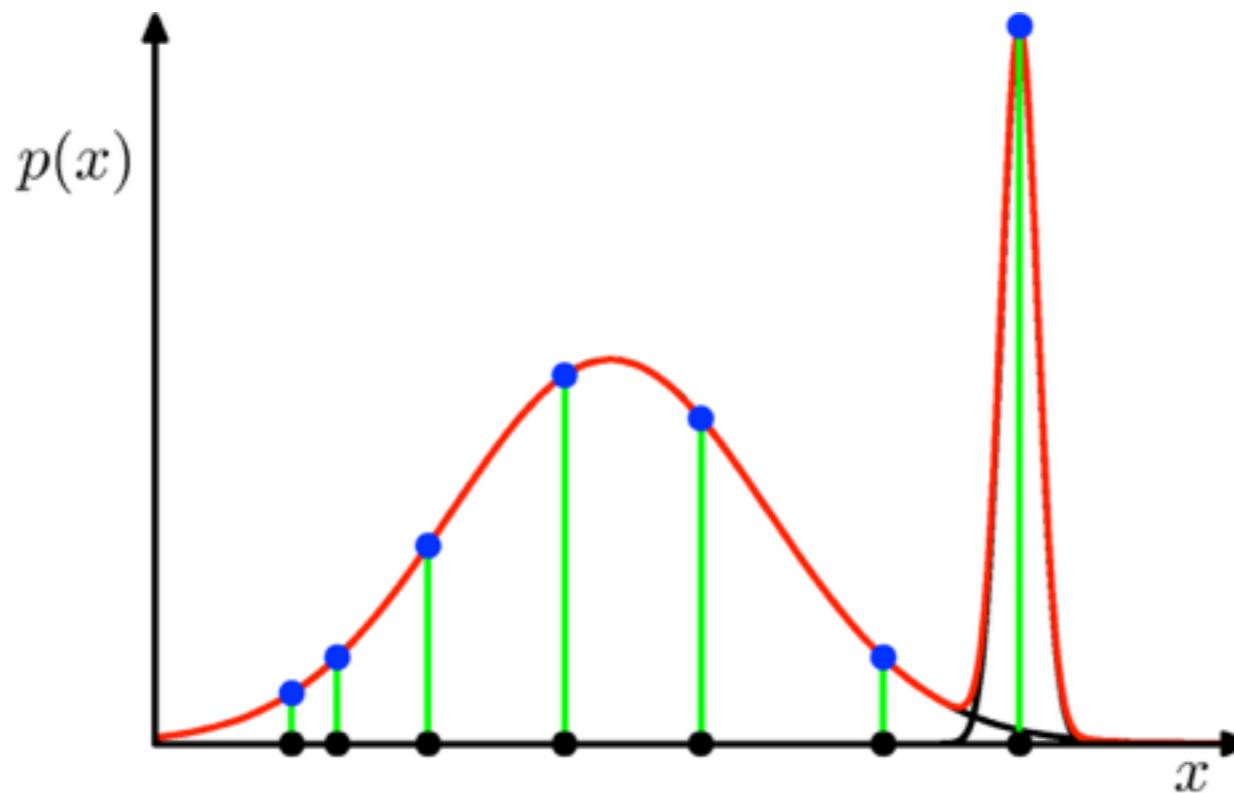


Problems with MLE for Gaussian Mixtures

- Assume that for one k the mean μ_k is exactly at a data point \mathbf{x}_n
 - For simplicity: assume that $\Sigma_k = \sigma_k^2 I$
 - Then:
$$\mathcal{N}(\mathbf{x}_n \mid \mathbf{x}_n, \sigma_k^2 I) = \frac{1}{\sqrt{2\pi}\sigma_k^D}$$
 - This means that the overall log-likelihood can be maximized arbitrarily by letting $\sigma_k \rightarrow 0$ (**overfitting**)
- Another problem is the **identifiability**:
 - The order of the Gaussians is not fixed, therefore:
 - There are $K!$ equivalent solutions to the MLE problem



Overfitting with MLE for Gaussian Mixtures



- One Gaussian fits exactly to one data point
- It has a very small variance, i.e. contributes strongly to the overall likelihood
- In standard MLE, there is no way to avoid this!



Expectation-Maximization

- EM is an elegant and powerful method for MLE problems with latent variables
- Main idea: model parameters and latent variables are estimated iteratively, where average over the latent variables (expectation)
- A typical example application of EM is the Gaussian Mixture model (GMM)
- However, EM has many other applications
- First, we consider EM for GMMs



Expectation-Maximization for GMM

- First, we define the **responsibilities**:

$$\gamma(z_{nk}) = p(z_{nk} = 1 \mid \mathbf{x}_n) \quad z_{nk} \in \{0, 1\}$$

$$\sum_k z_{nk} = 1$$



Expectation-Maximization for GMM

- First, we define the **responsibilities**:

$$\begin{aligned}\gamma(z_{nk}) &= p(z_{nk} = 1 \mid \mathbf{x}_n) \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}\end{aligned}$$

