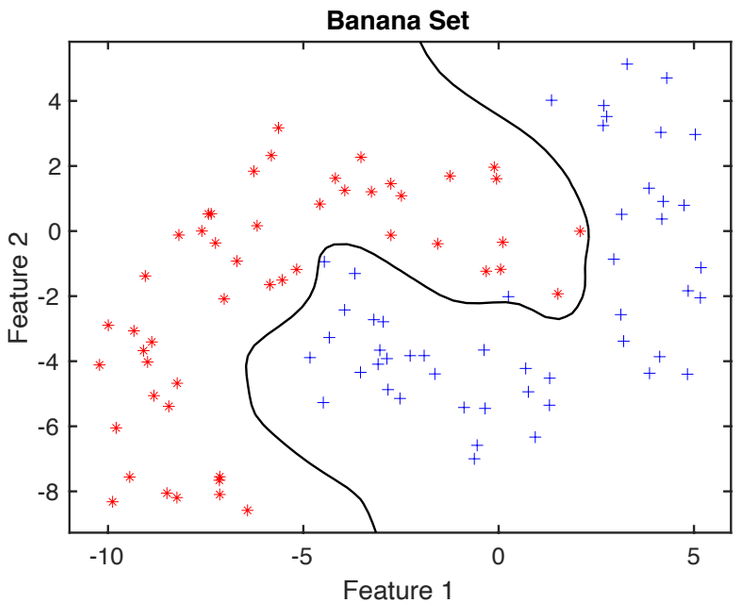
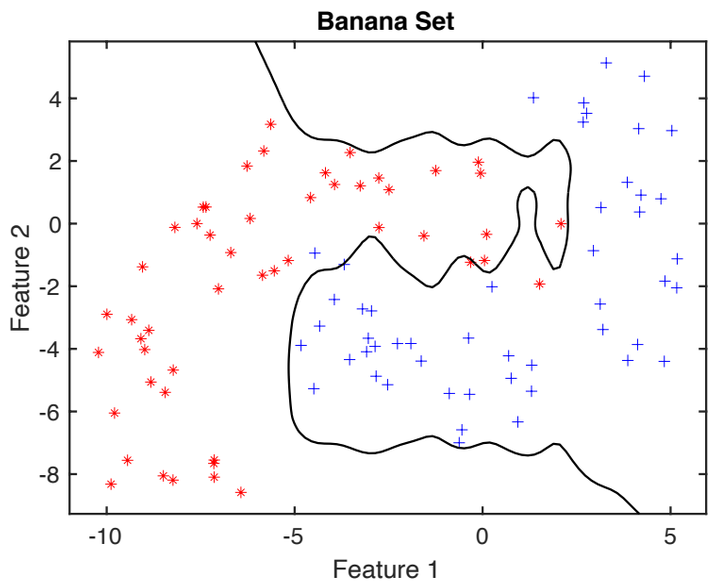
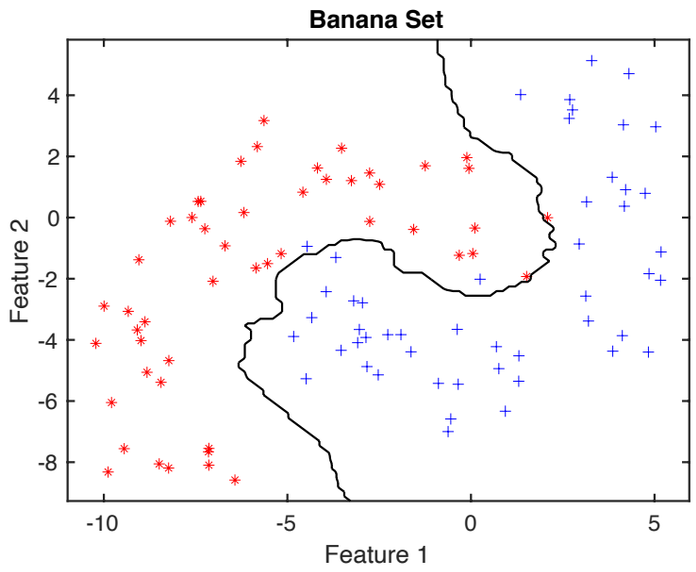
Parametric and Non-parametric Classification

David M.J. Tax

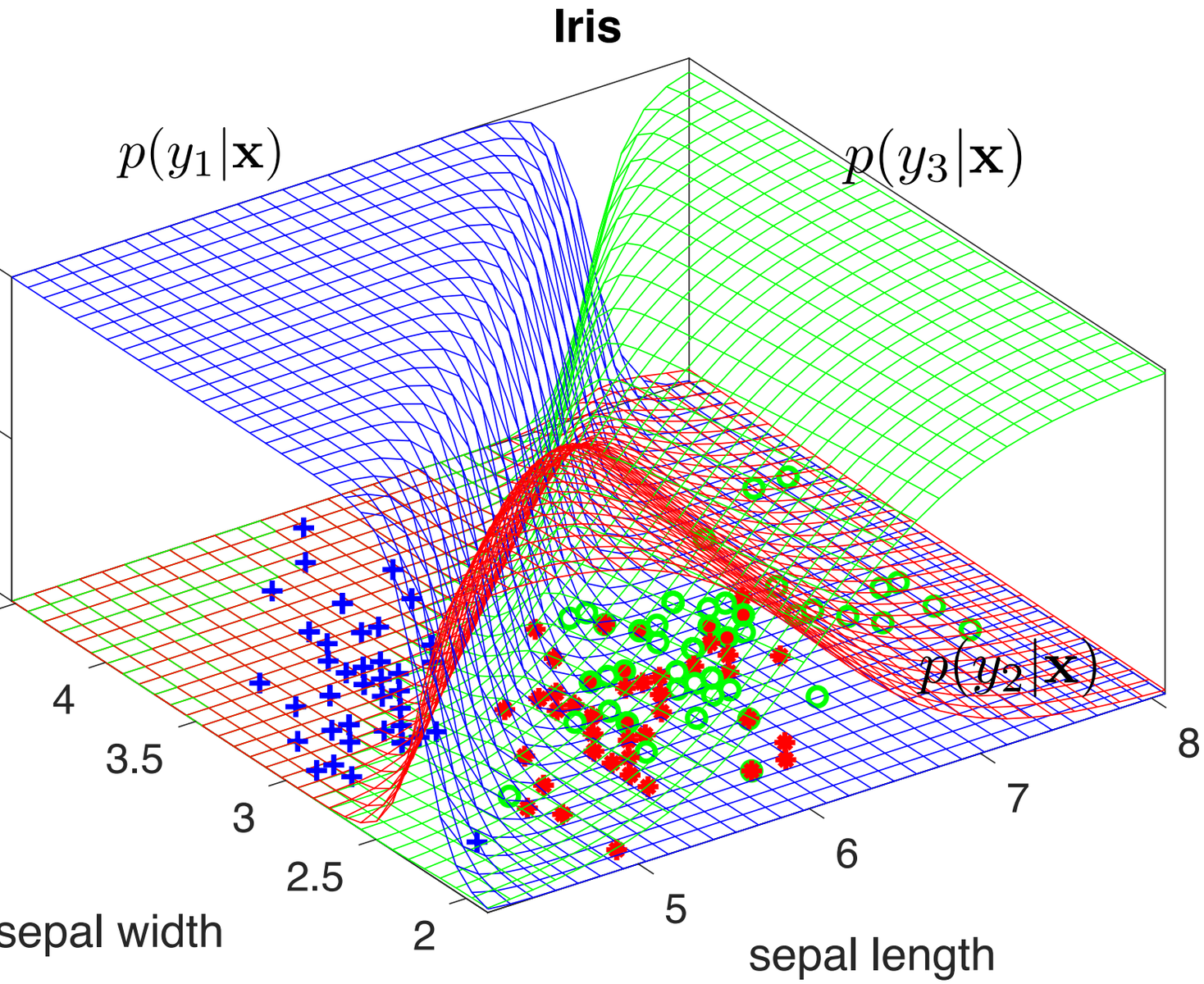






TU Delft
Delft University of Technology

Yesterday: the need for p(y|x)



- For each object in the feature space, we should find:

$$p(y|\mathbf{x})$$

- In practice, we approximate:

$$\hat{p}(y|\mathbf{x})$$

Bayes' theorem

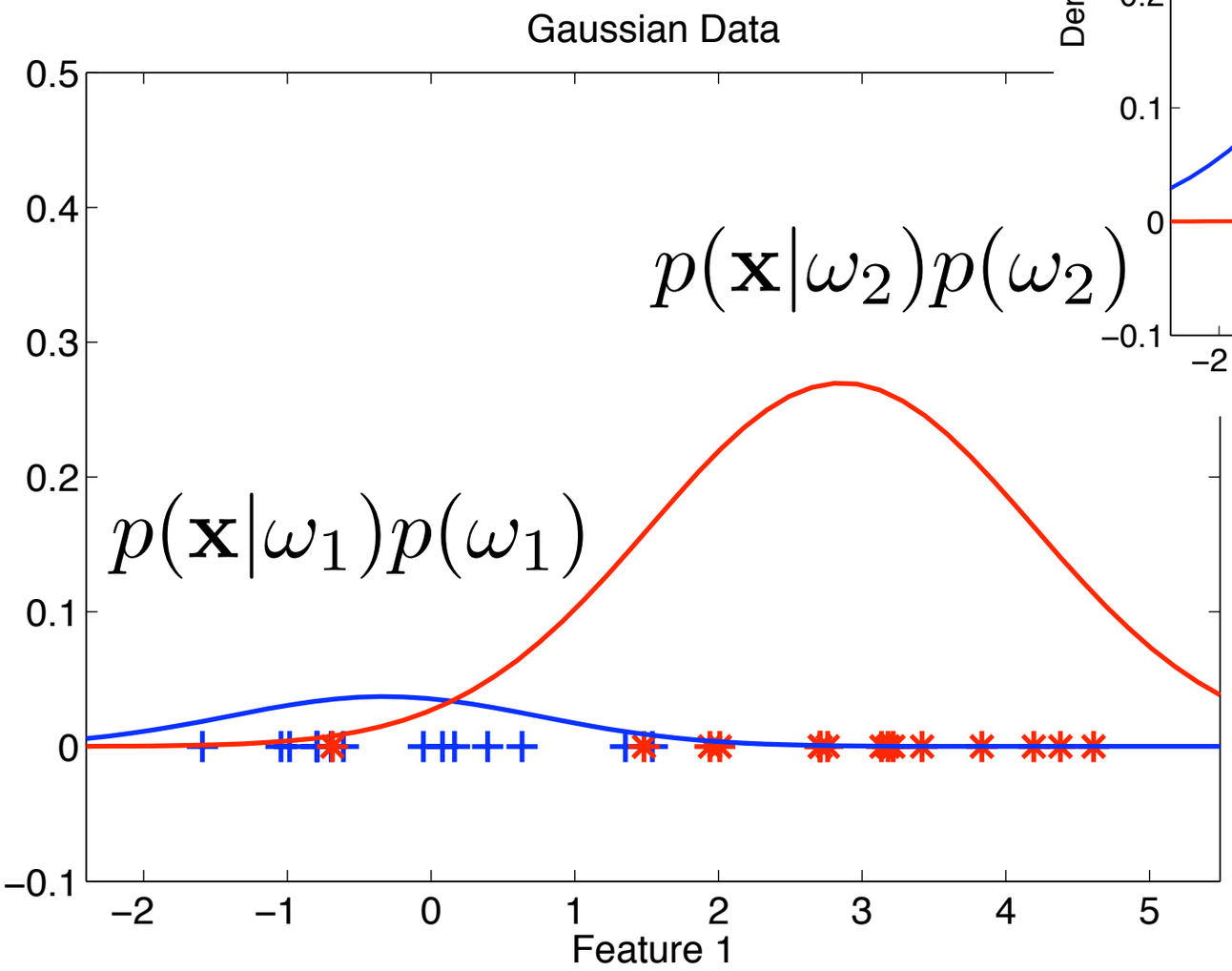
- In many cases the posterior is hard to estimate
- Often a functional form of the class distributions can be assumed
- Use Bayes' theorem to rewrite one into the other:

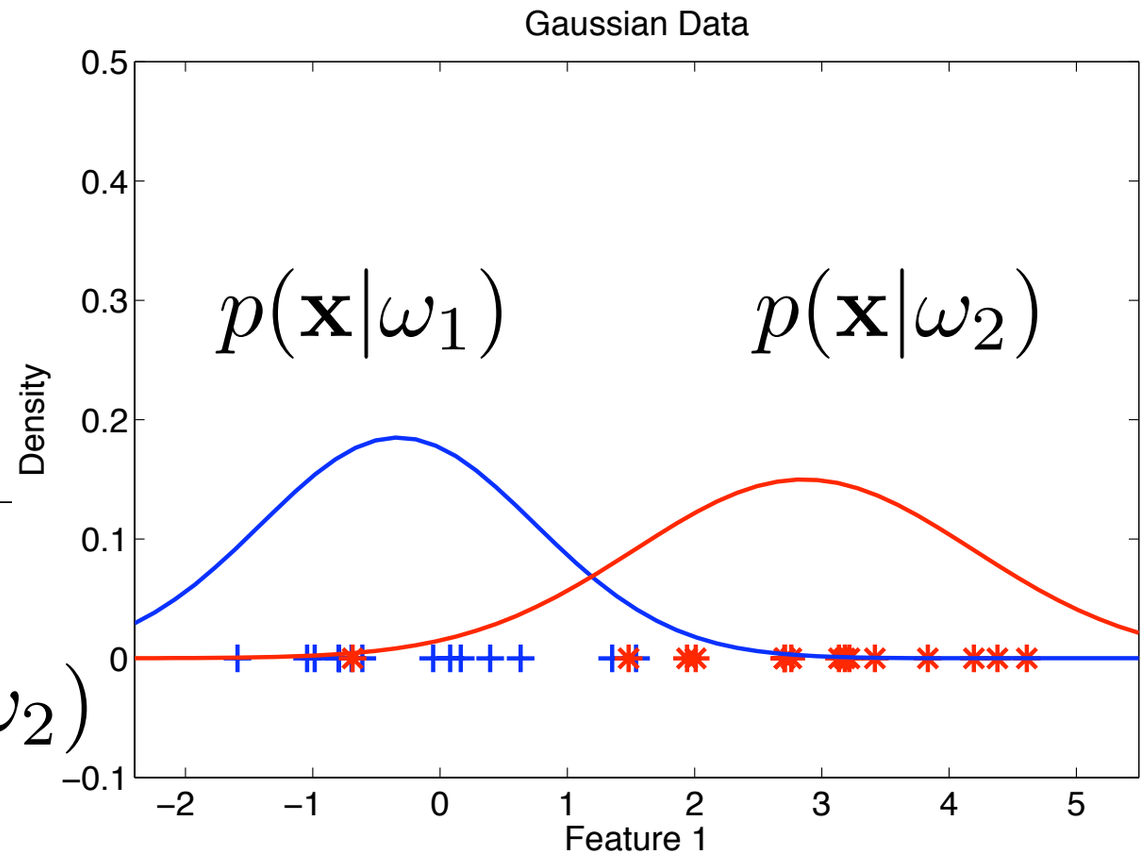
$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

class (conditional) distribution	$p(\mathbf{x}\|y)$
class prior	$p(y)$
(unconditional) data distribution	$p(\mathbf{x})$

Bayes rule

1. Estimate the class conditional probability



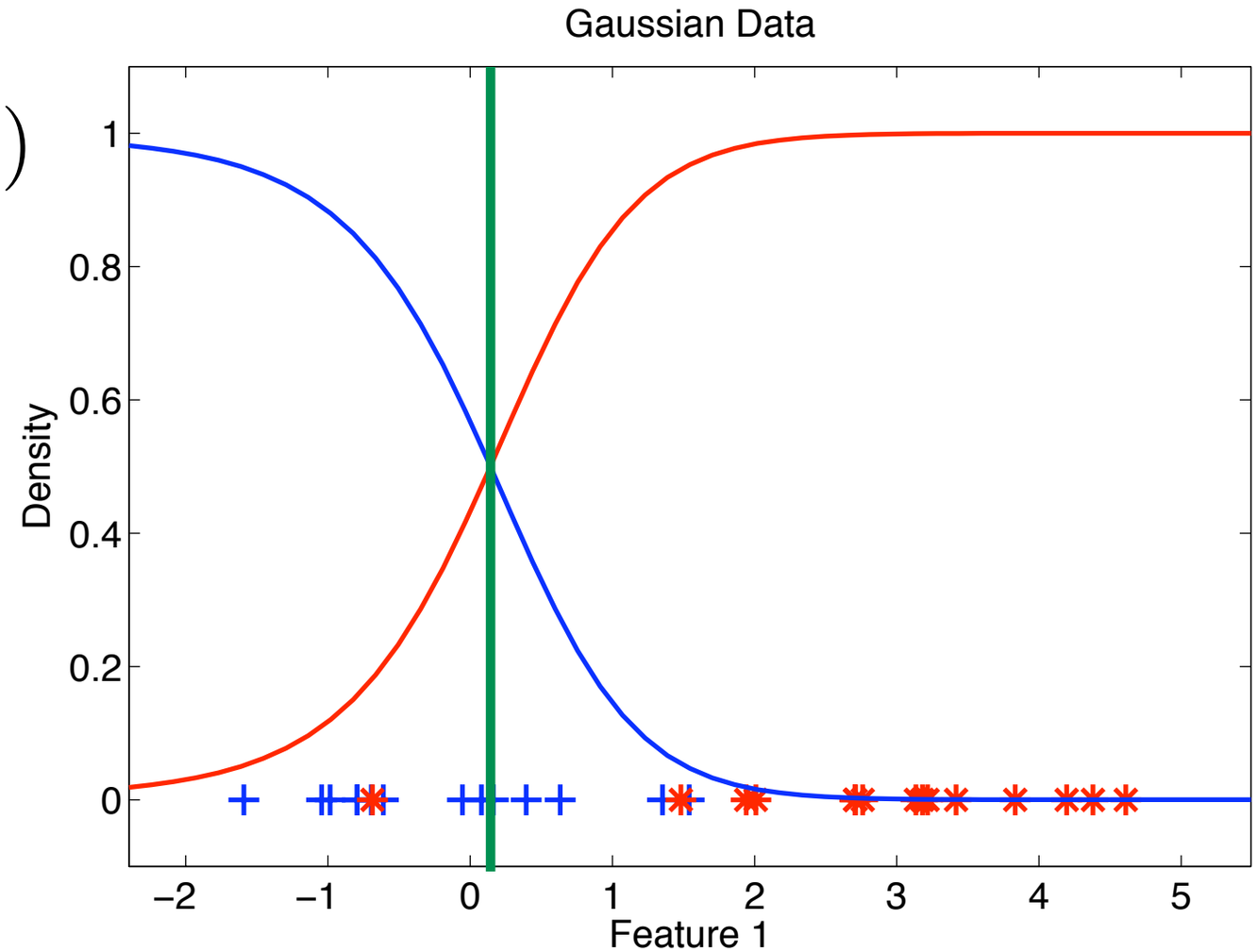


2. Incorporate the class priors

Bayes rule

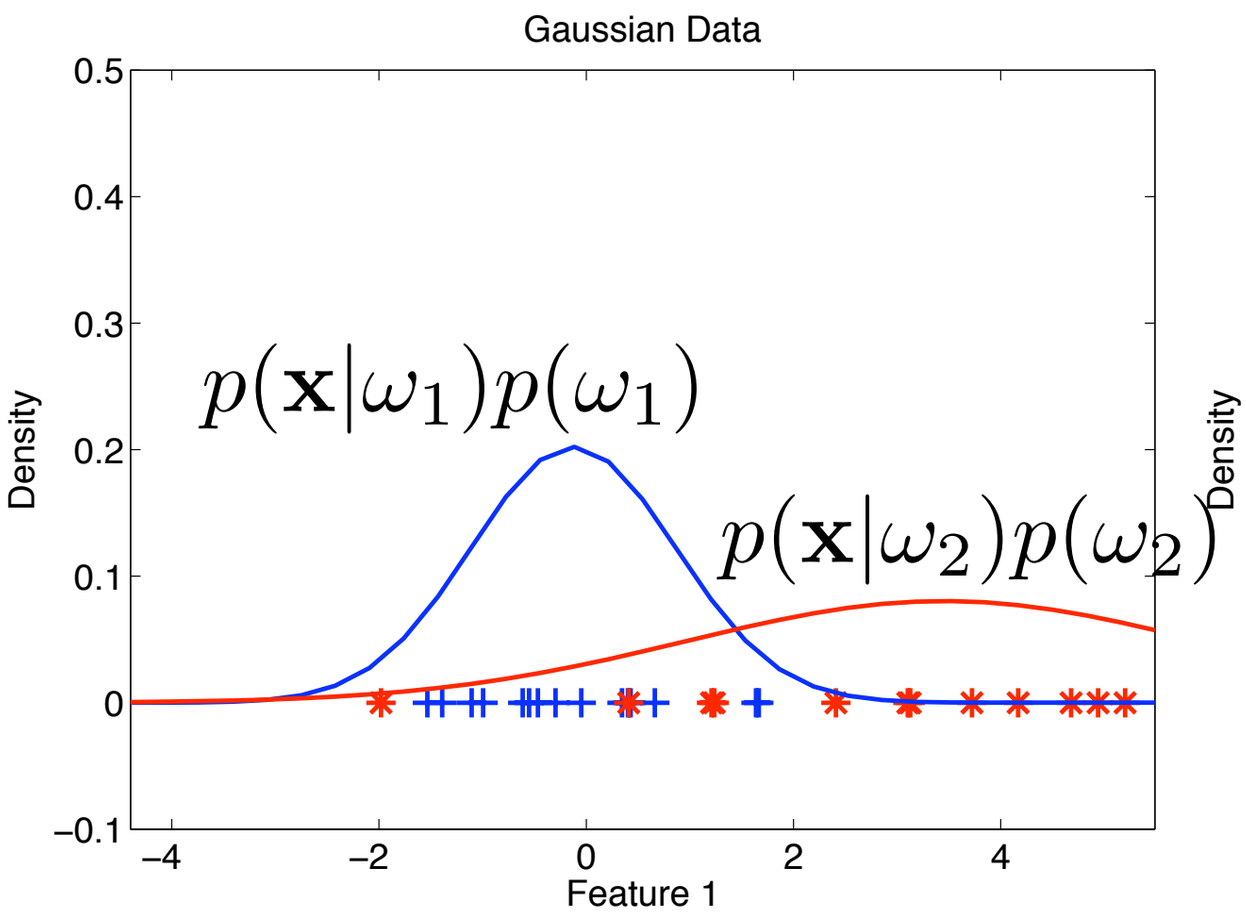
3. Compute the class posterior probabilities
4. Assign objects to the class with the highest posterior probability

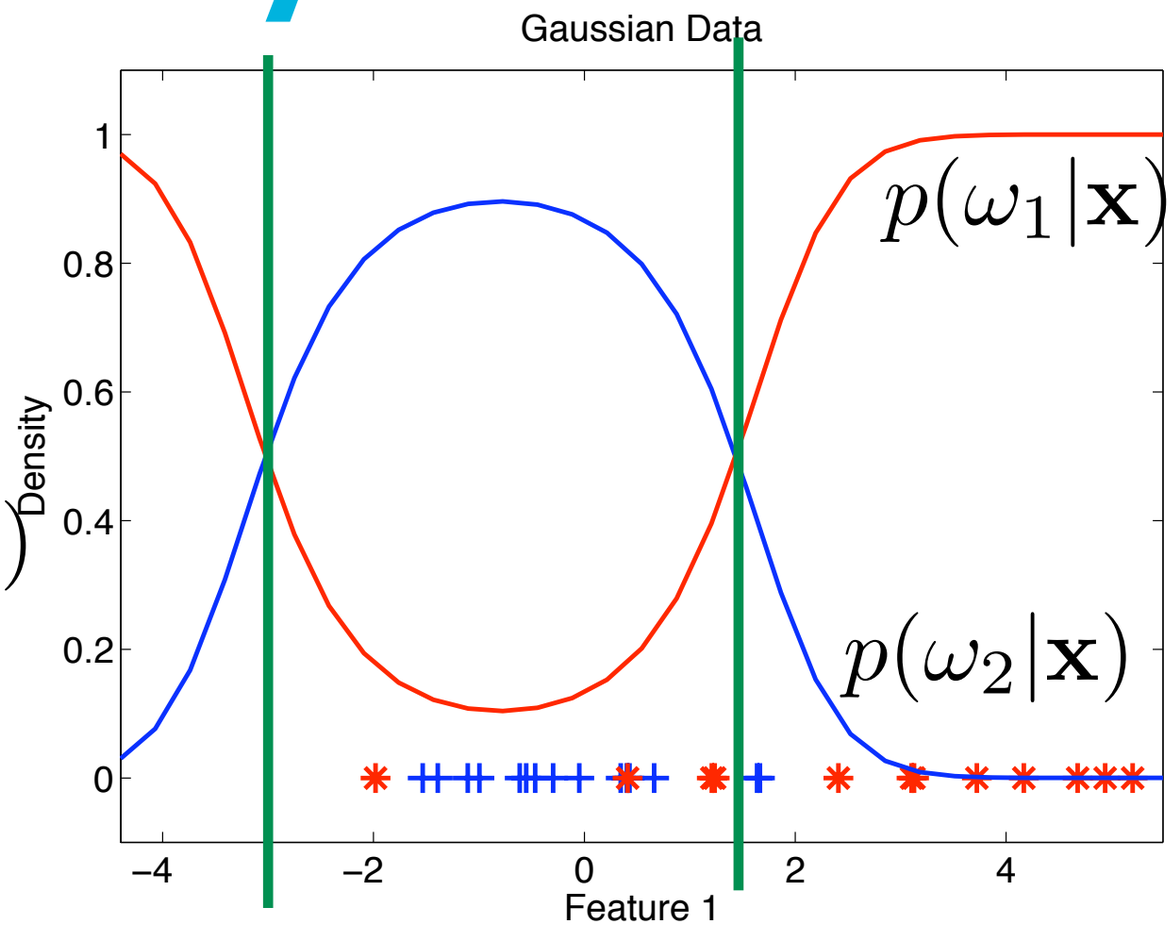
$p(\omega_1|\mathbf{x})$



$p(\omega_2|\mathbf{x})$

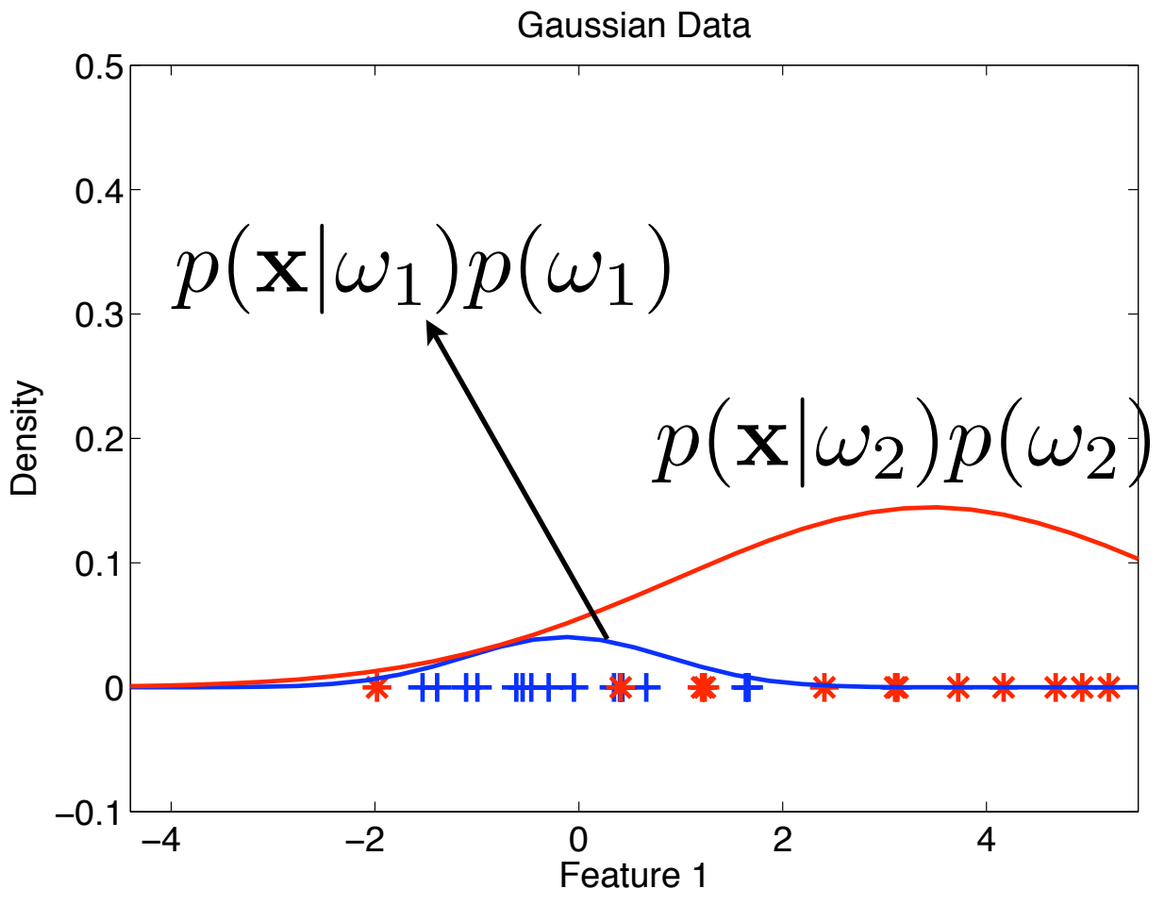
Complicated decision boundary

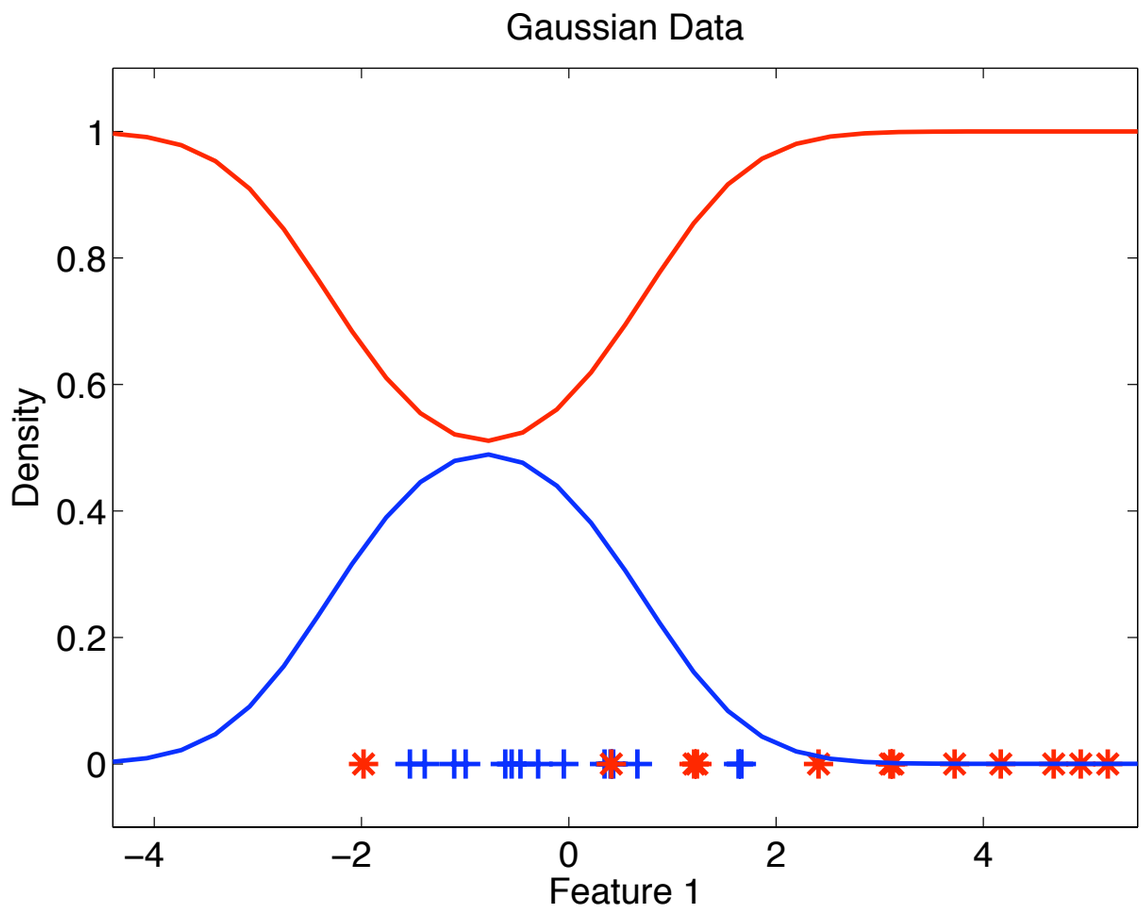




- Depending on the class-conditional probability densities, complicated decision boundaries can appear

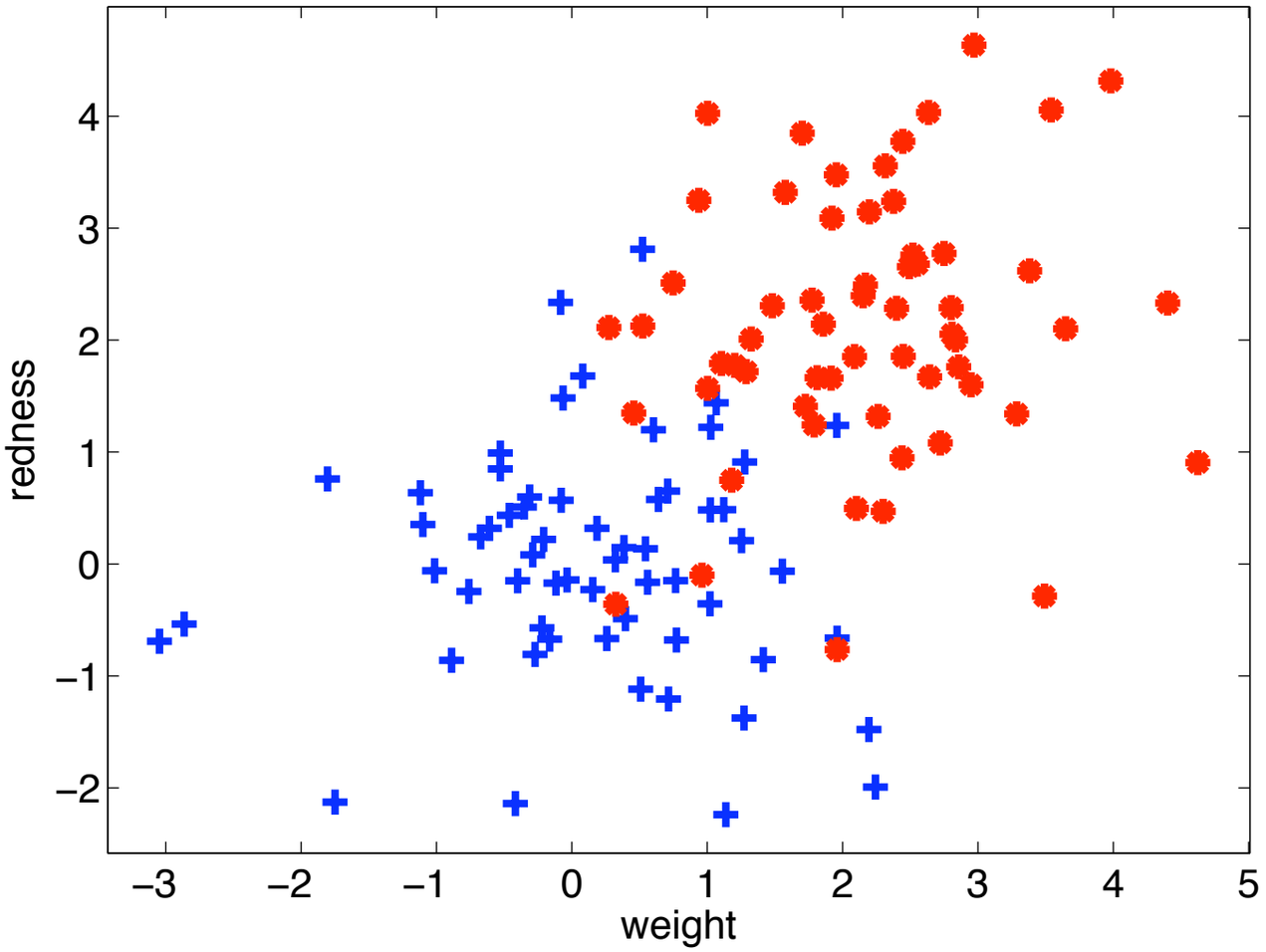
Missing decision boundary



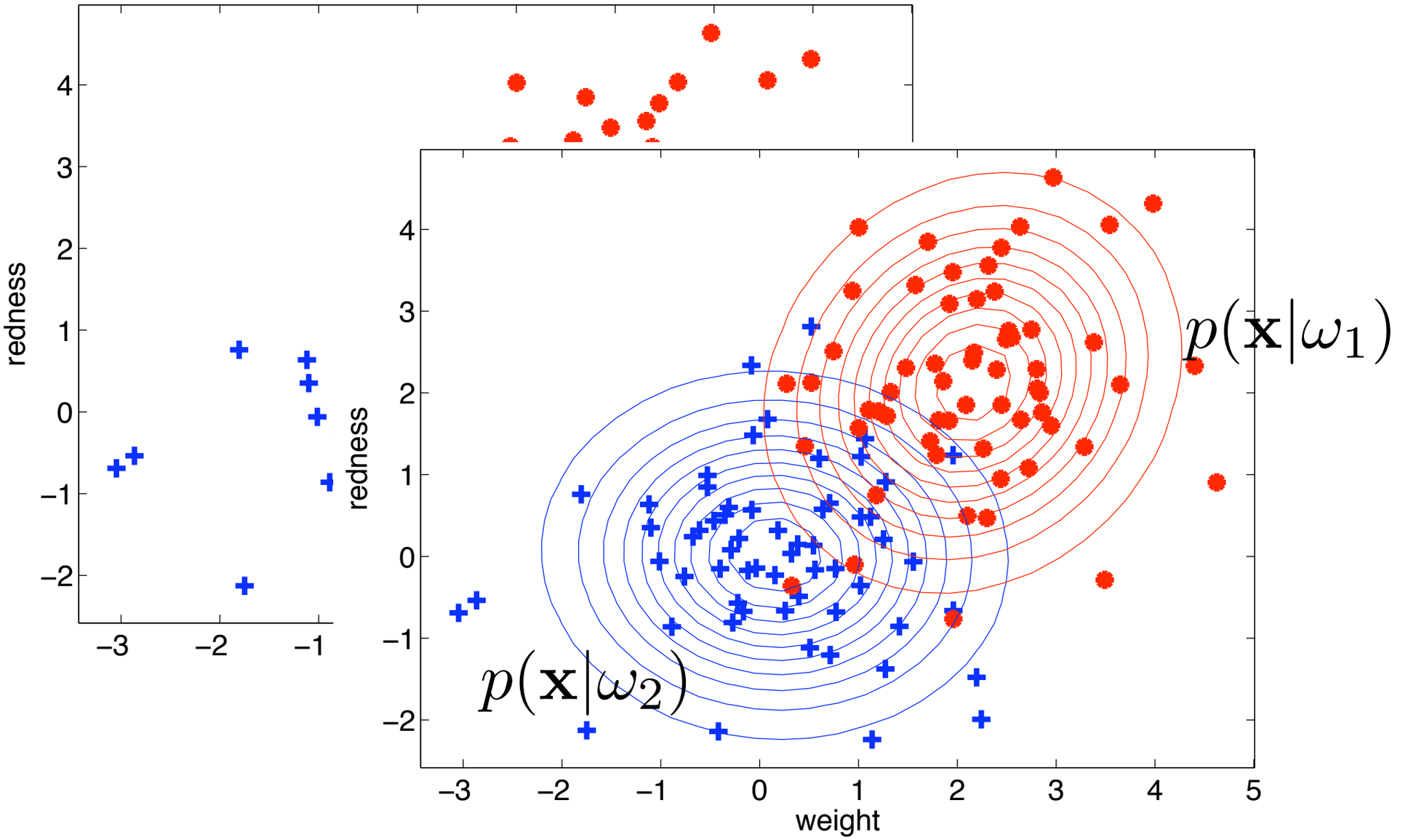


- A class can be too small (class prior is low) or too dispersed, that no objects are assigned to that class

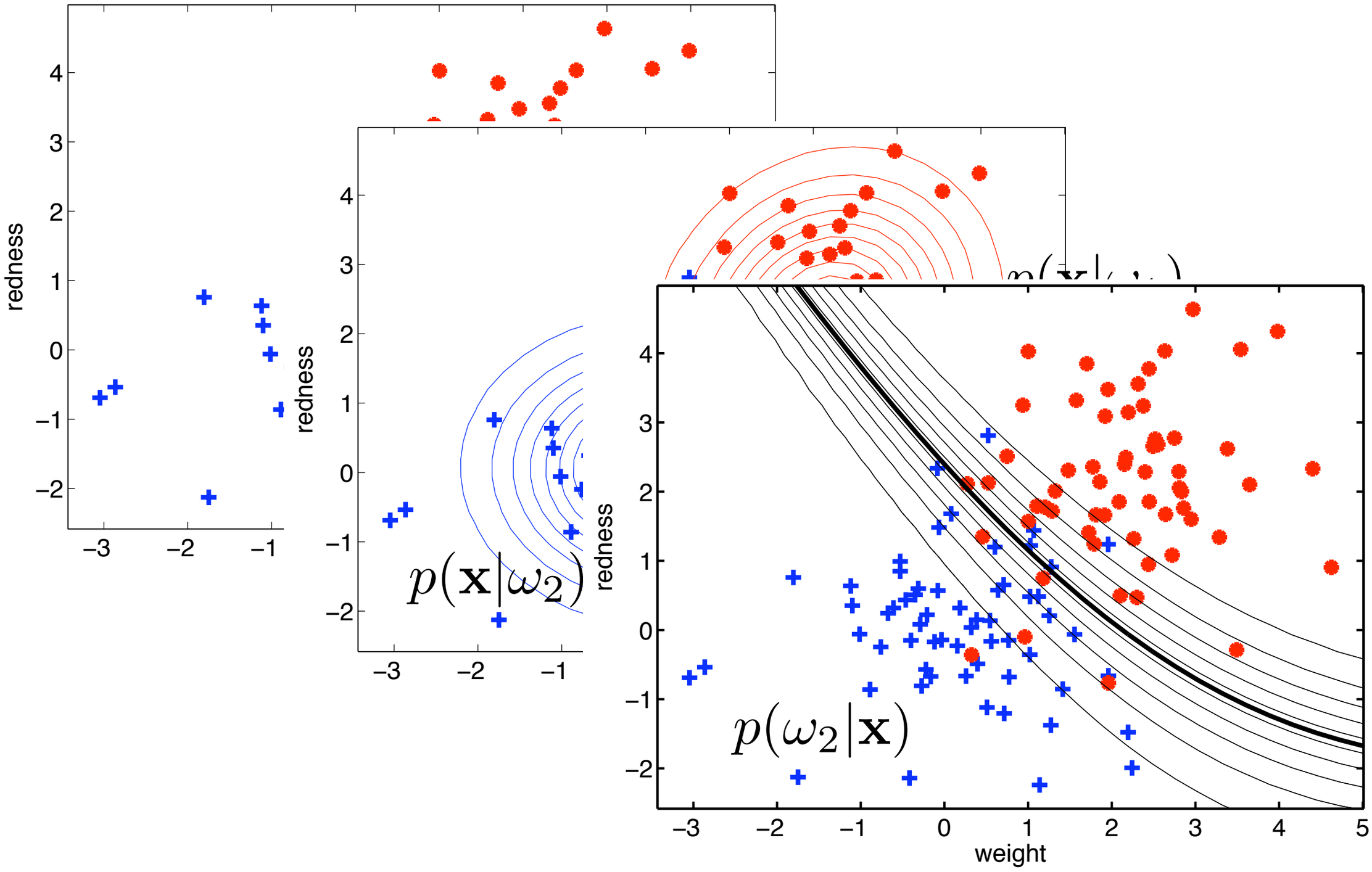
2-dimensional feature space



2-dimensional feature space



2-dimensional feature space



Contents today

- Parametric classifiers: Gaussian-based
  - Quadratic discriminant
  - Linear discriminant (LDA)
  - Nearest Mean
- Non-parametric classifiers:
  - Histogram method
  - Parzen classifier
  - Nearest neighbor classifier
  - Scaling

Plug-in Bayes Rule

- So, for Bayes rule $p(y|\mathbf{x}) = \dfrac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$

  we need to estimate:
- (1) the class priors $\hat{p}(y) = \dfrac{N_y}{N}$
- (2) the unconditional probabilities

$$\hat{p}(\mathbf{x}) = \sum_{i=1}^{C} \hat{p}(\mathbf{x}|y_i)\hat{p}(y_i)$$

- (3) the class-conditional probabilities $\hat{p}(\mathbf{x}|y)$

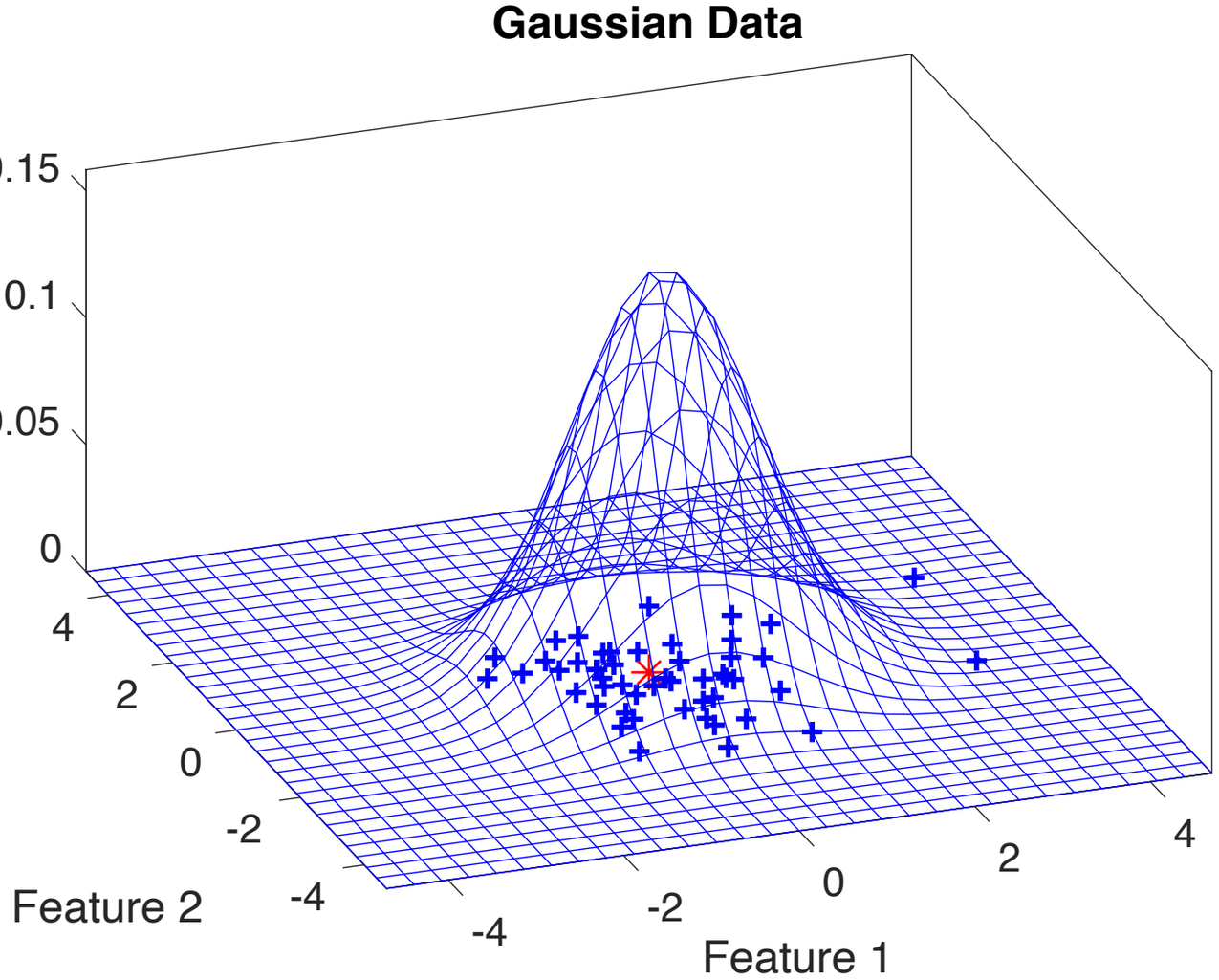
Estimate the class conditional prob.

- The model for the class conditional probability is now the crucial choice
- Very common: Gaussian distribution

$$p(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^p \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right)$$

- It models a 'blob'-like distribution, in a p-dimensional feature space!
- μ is the mean of the distribution
- $\sum$ is the (elliptical) shape of the distribution

Gaussian Distribution 2D



(column vector)

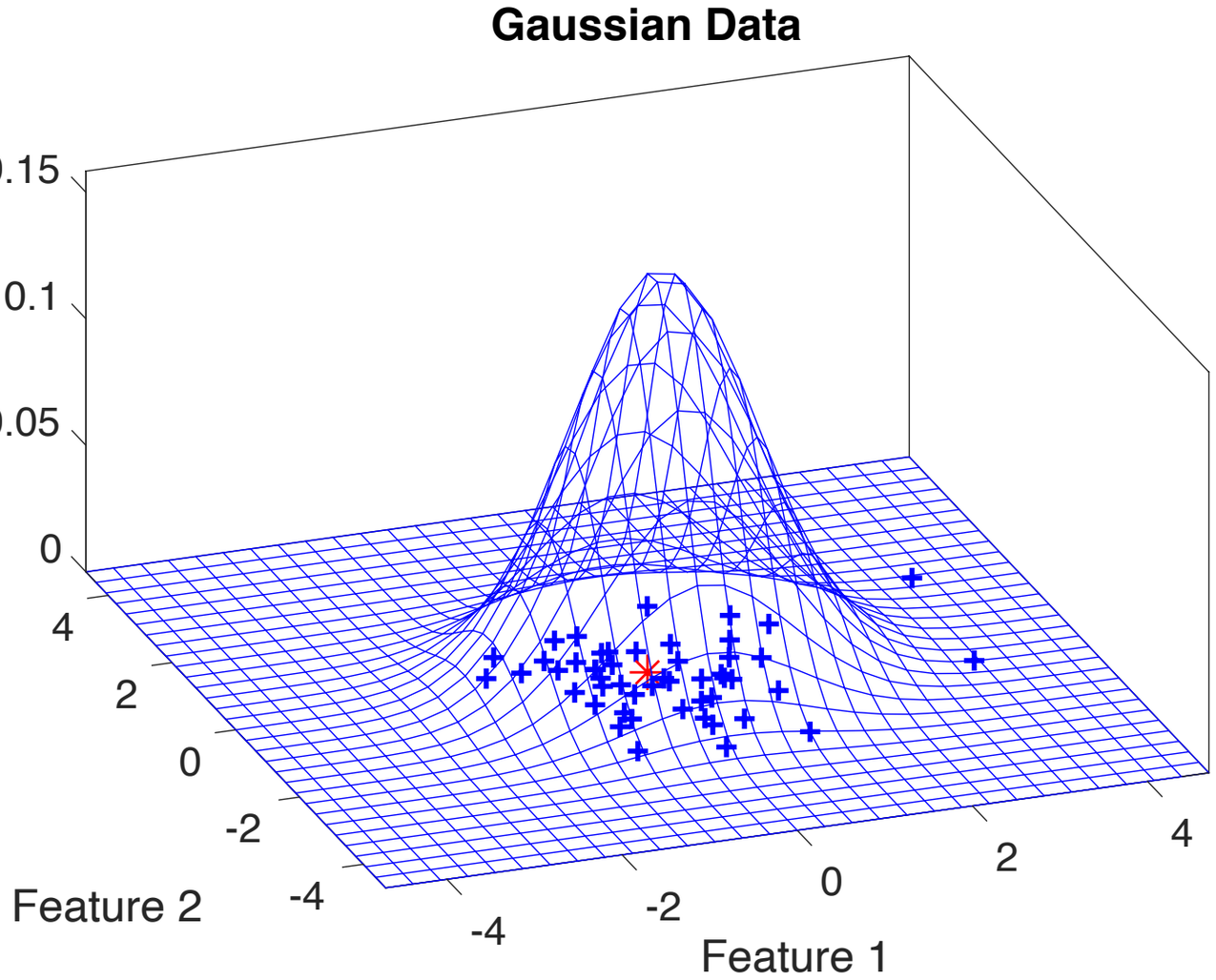
$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \rho_{12}\sigma_1\sigma_2 \\ \rho_{12}\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$$

$$p(\mathbf{x}) = \frac{1}{\sqrt{2\pi^p \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

TU Delft

Gaussian Distribution 2D



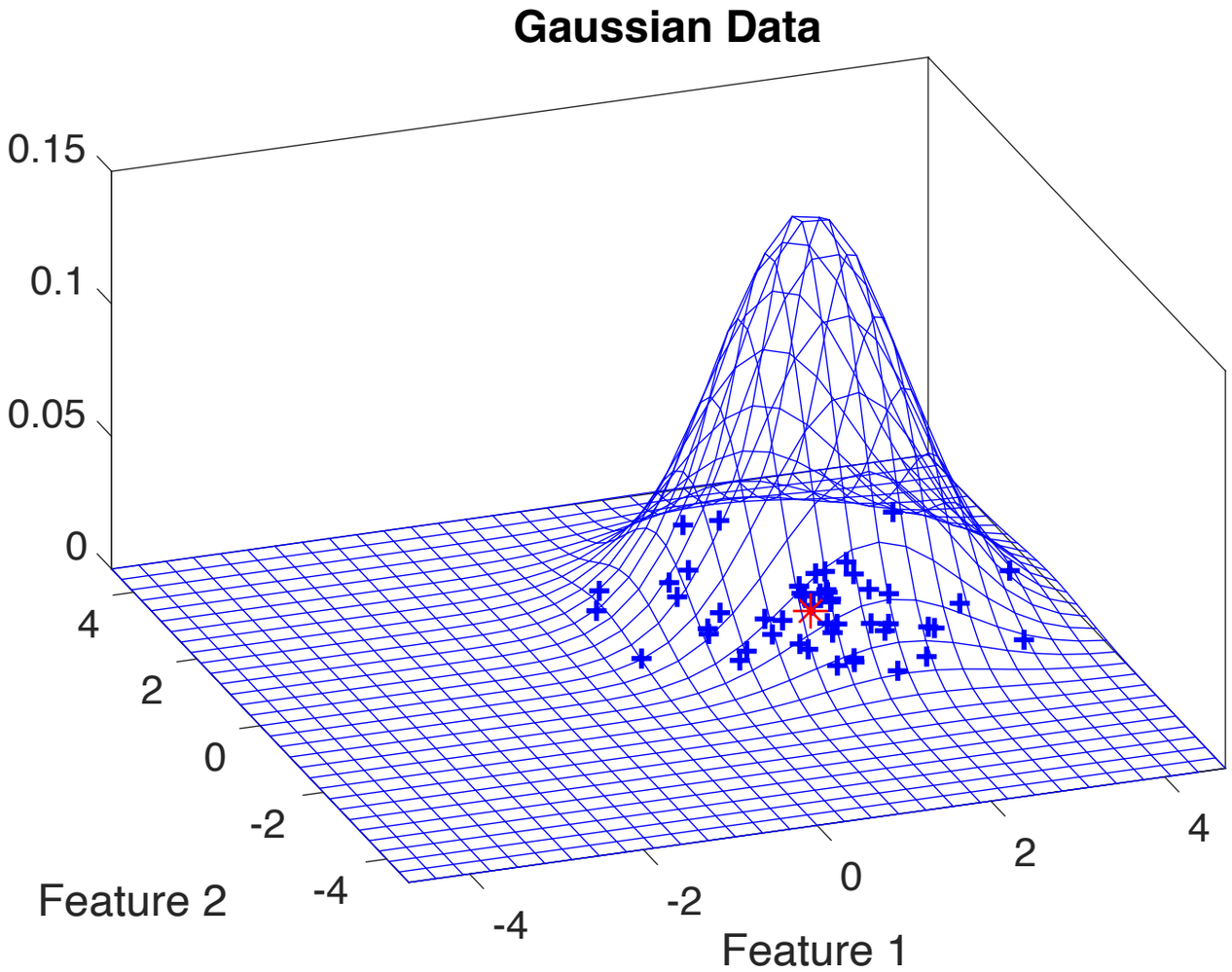
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$p(\mathbf{x}) = \frac{1}{\sqrt{2\pi^p \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right)$$

TU Delft

Gaussian Distribution 2D



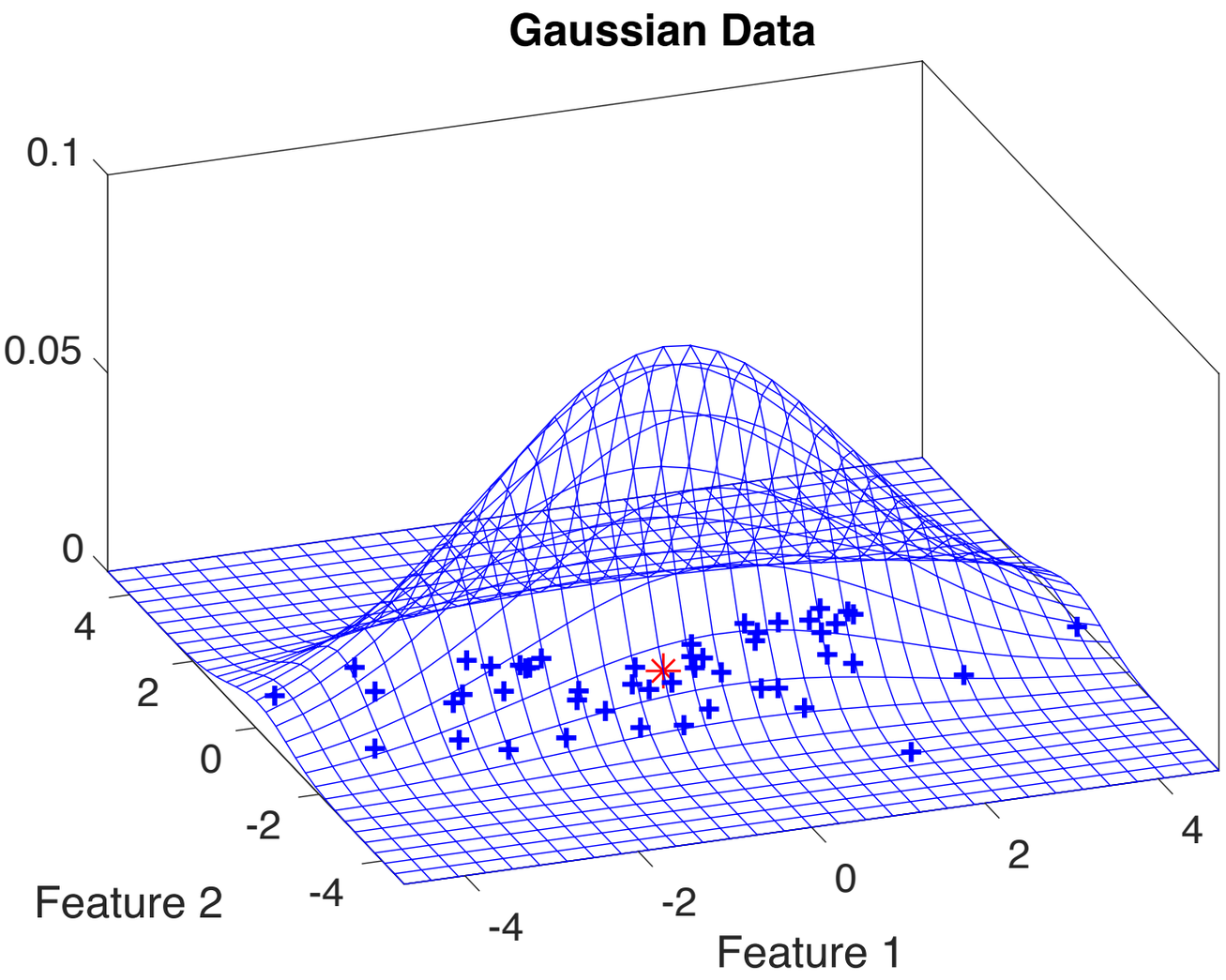
$$\mu = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$p(\mathbf{x}) = \frac{1}{\sqrt{2\pi^p \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right)$$

TU Delft

Gaussian Distribution 2D



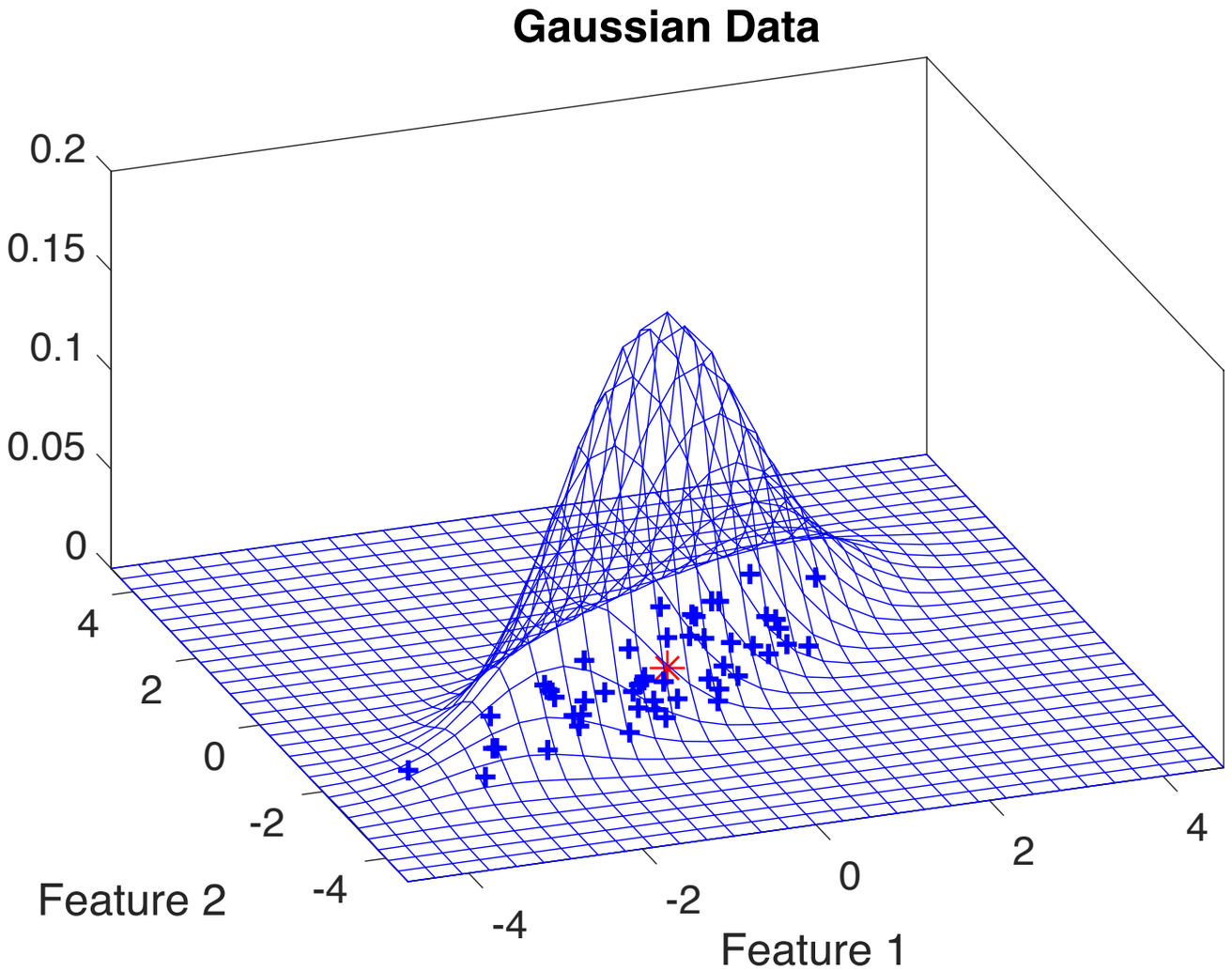
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$$

$$p(\mathbf{x}) = \frac{1}{\sqrt{2\pi^p \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right)$$

TU Delft

Gaussian Distribution 2D



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}$$

$$p(\mathbf{x}) = \frac{1}{\sqrt{2\pi^p \det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \Sigma^{-1}(\mathbf{x}-\mu)\right)$$

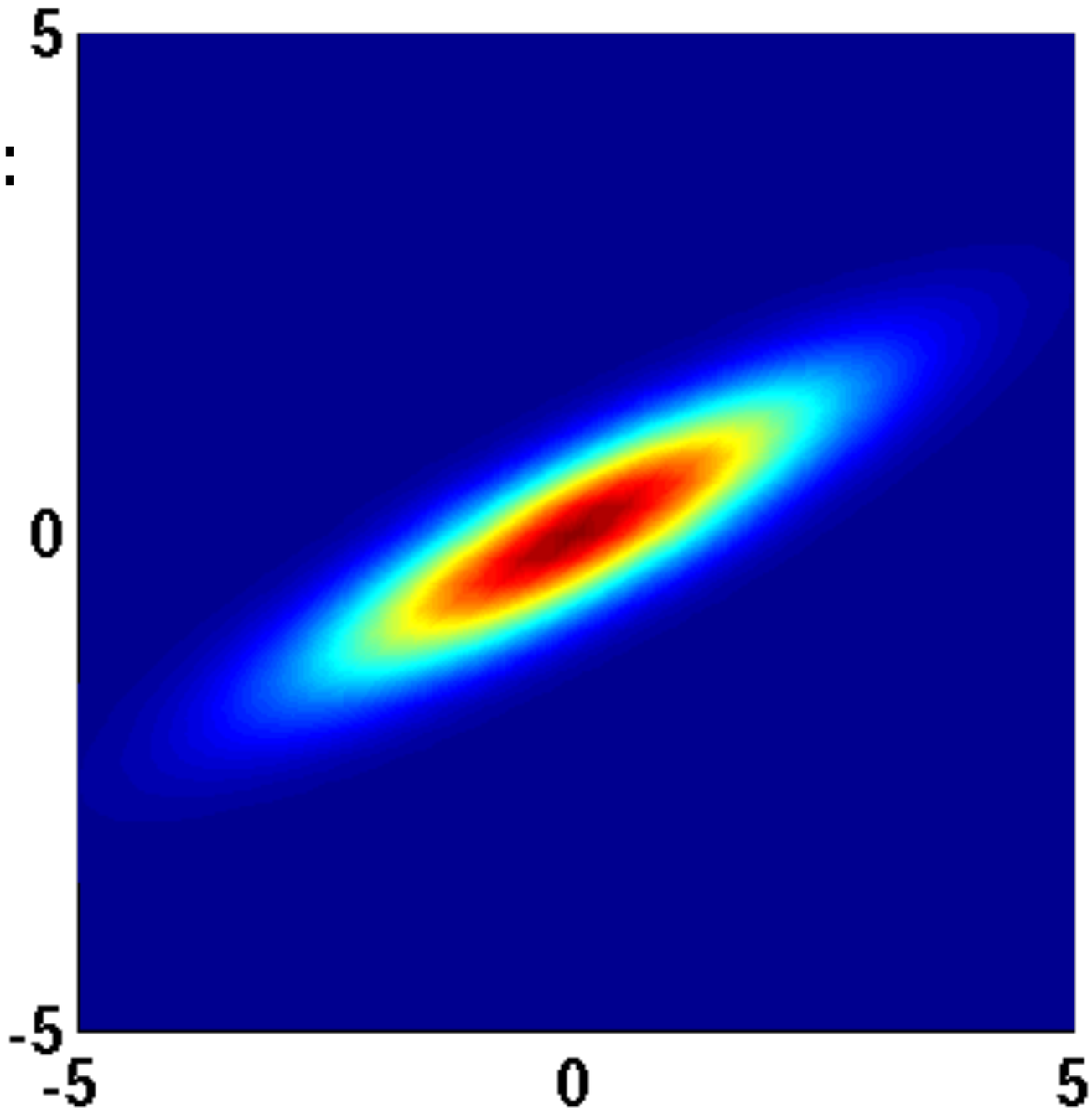
TU Delft

Example 2D Gaussian

- Top view on an example 2D Gaussian:

$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 3 & 1.5 \\ 1.5 & 2 \end{bmatrix}$$



Plug-in Gaussian Distribution

- Now we use the Gaussian distribution for each class:

$$\hat{p}(\mathbf{x}|y) = \frac{1}{\sqrt{2\pi^p \det(\hat{\Sigma}_y)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\hat{\mu}_y)^T \hat{\Sigma}_y^{-1}(\mathbf{x}-\hat{\mu}_y)\right)$$

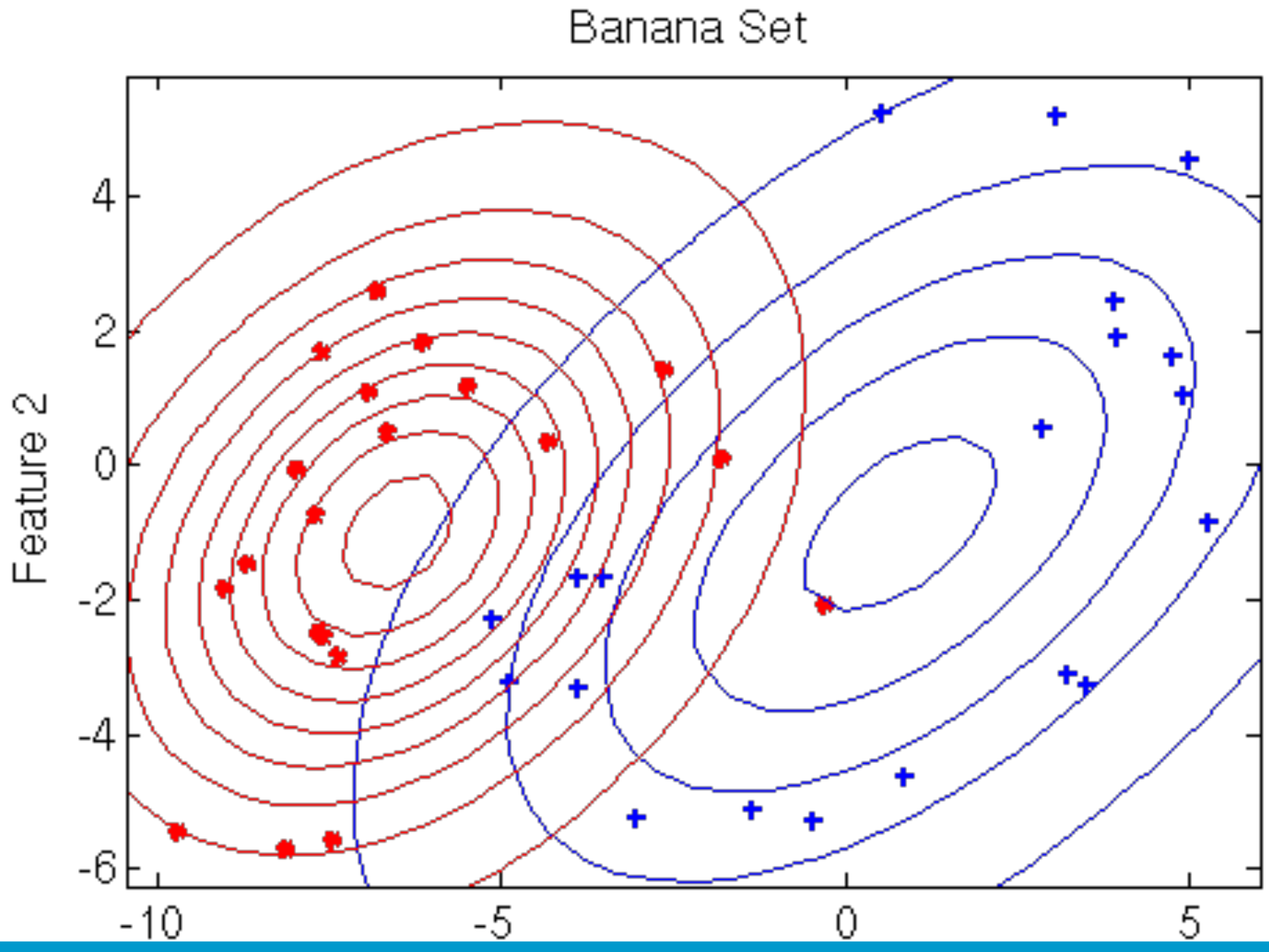
- We have to estimate the parameters on some training set, $\{\mathbf{x}_1, .., \mathbf{x}_N\}$ e.g. using ML:

(Outer product)

$$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i \qquad \hat{\Sigma} = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T$$

Example on Banana Data

- A single Gaussian distribution on each class:



The Two-Class Case

- Define the discriminant

$$f(\mathbf{x}) = \log p(y_1|\mathbf{x}) - \log p(y_2|\mathbf{x})$$

- Rewriting this, one gets

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{W}\mathbf{x} + \mathbf{w}^T\mathbf{x} + w_0$$

- How does **W** and **w** look like?
- This is called a quadratic classifier because the decision boundary is a quadratic function of **x**

Class Posterior Probability Gaussian

- Combining

$$\hat{p}(\mathbf{x}|y) = \frac{1}{\sqrt{(2\pi)^p \det(\hat{\Sigma}_y)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\hat{\mu}_y)^T \hat{\Sigma}_y^{-1}(\mathbf{x}-\hat{\mu}_y)\right)$$

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$$

We can derive for class y_i the log(p(y|**x**)) :

$$\log(\hat{p}(y_i|\mathbf{x})) = -\frac{p}{2}\log(2\pi) - \frac{1}{2}\log(\det \Sigma_i)$$

$$-\frac{1}{2}(\mathbf{x}-\mu_i)^T \Sigma_i^{-1}(\mathbf{x}-\mu_i) + \log p(y_i) - \log p(\mathbf{x})$$

Normal-based Classifier

- p(**x**) is independent of the classes, it can be dropped:

$$g_i(\mathbf{x}) = -\frac{1}{2}\log(\det \Sigma_i) - \frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma_i^{-1}(\mathbf{x} - \mu_i) + \log p(y_i)$$

- Classifier becomes :

  Assign **x** to class y_i when for all $i \neq j$: $g_i(\mathbf{x}) > g_j(\mathbf{x})$

Quadratic discriminant

- General form:

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{W} \mathbf{x} + \mathbf{w}^T \mathbf{x} + w_0$$

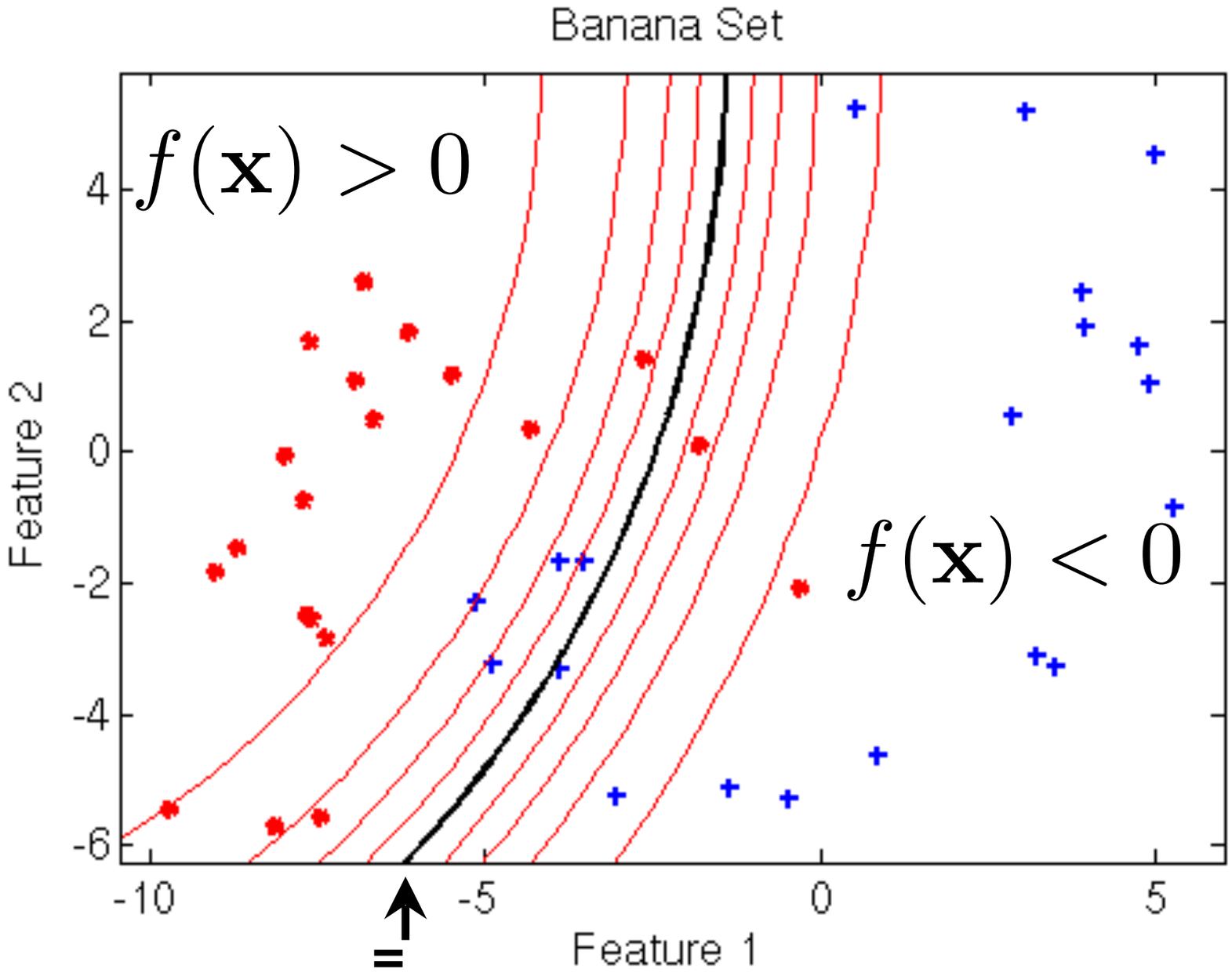
- with

$$\mathbf{W} = \frac{1}{2}\left(\Sigma_2^{-1} - \Sigma_1^{-1}\right)$$

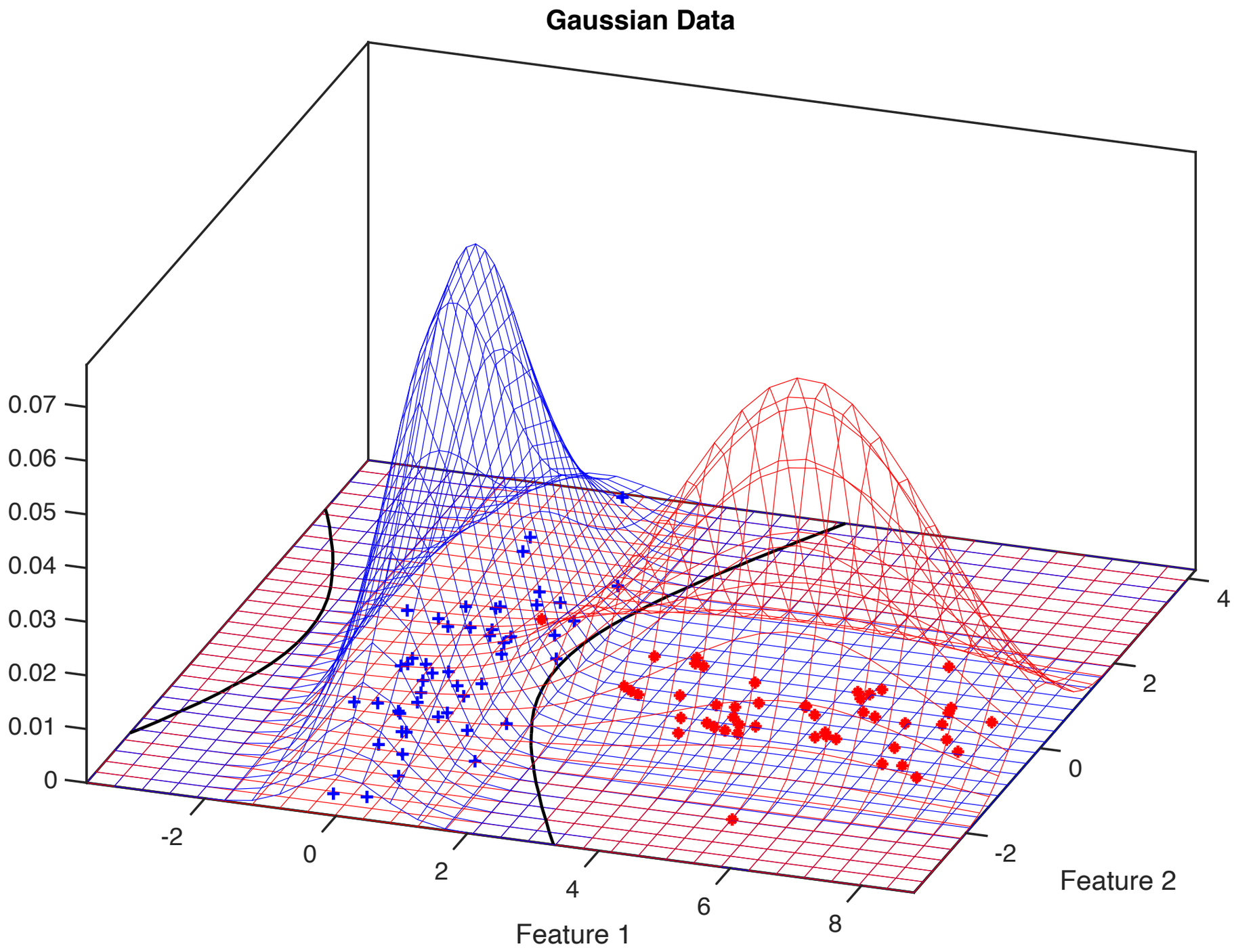
$$\mathbf{w} = \mu_1^T \Sigma_1^{-1} - \mu_2^T \Sigma_2^{-1}$$

$$\begin{aligned} w_0 = &- \frac{1}{2}\log\det\Sigma_1 - \frac{1}{2}\mu_1^T \Sigma_1^{-1}\mu_1 + \log p(y_1) \\ &+ \frac{1}{2}\log\det\Sigma_2 + \frac{1}{2}\mu_2^T \Sigma_2^{-1}\mu_2 - \log p(y_2) \end{aligned}$$

Quadratic Classifier on Banana Data



Hyperbolic decision boundary

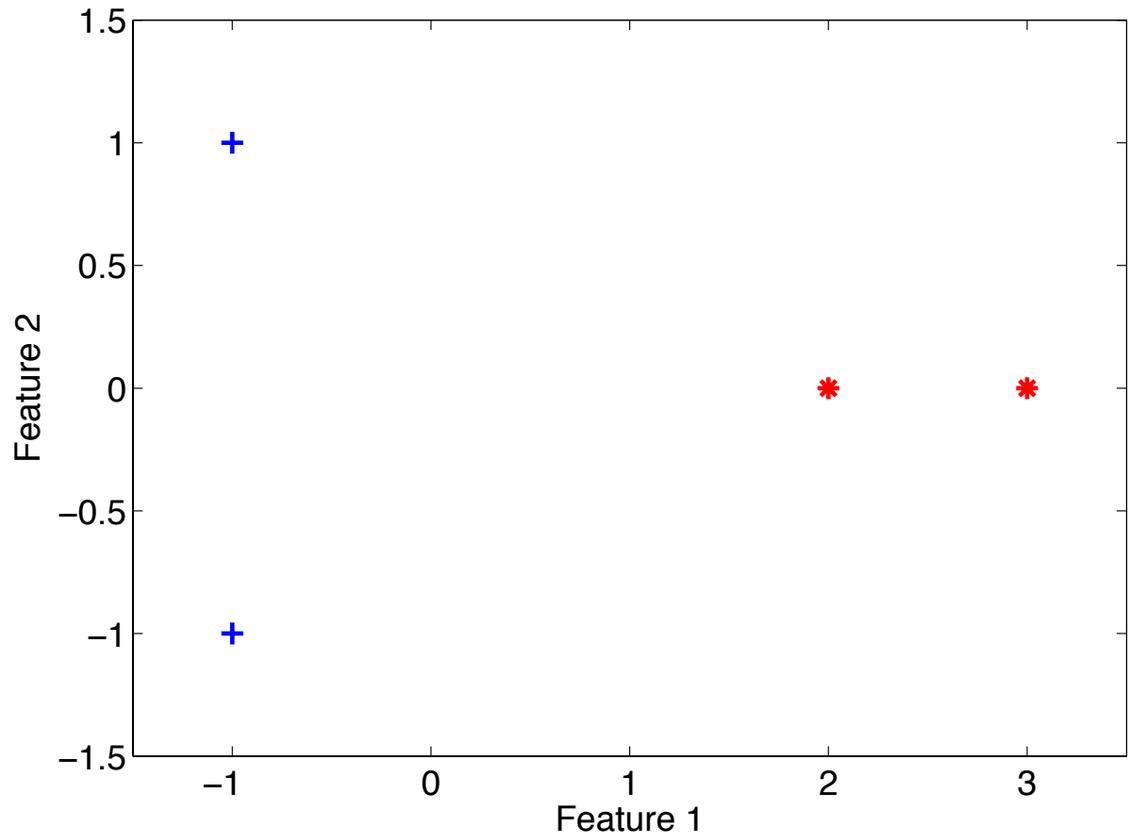


TU Delft

Estimating the covariance matrix

$$X = \begin{bmatrix} -1 & -1 \\ -1 & +1 \\ 2 & 0 \\ 3 & 0 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} -1 \\ +1 \\ +1 \end{bmatrix}$$



- Mean of class +1?
- Covariance matrix of class +1? And its inverse?

TU Delft

Estimating the covariance matrix

- Make new, centered, dataset:

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T = \frac{1}{2}\sum_{i=1}^{n}\tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_i^T$$

- Then:

$$\tilde{\mathbf{x}}_1 = \begin{bmatrix} -0.5 \\ 0 \end{bmatrix} \quad \tilde{\mathbf{x}}_2 = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}$$

- And:

$$\tilde{\mathbf{x}}_1\tilde{\mathbf{x}}_1^T = \begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix}$$

Estimating the covariance matrix

- The same for $\tilde{\mathbf{x}}_2\tilde{\mathbf{x}}_2^T = \begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix}$

- So in total:

$$\hat{\Sigma} = \frac{1}{2}\sum_{i=1}^{n} \tilde{\mathbf{x}}_i\tilde{\mathbf{x}}_i^T = \frac{1}{2}\left(\begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix}\right)$$

$$= \begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix}$$

Inverse of the covariance matrix?

- Inverse of?:

$$\hat{\Sigma} = \begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix}$$

- It should hold that:

$$\hat{\Sigma}\hat{\Sigma}^{-1} = \mathbb{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

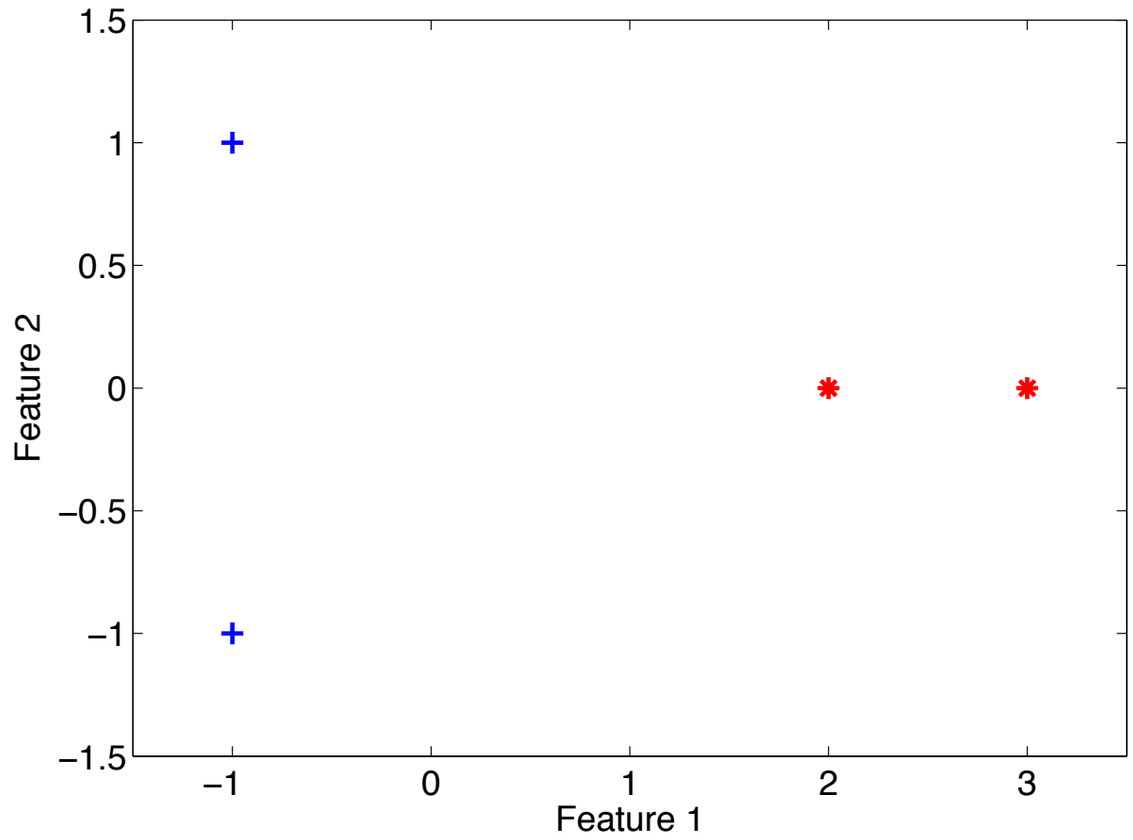
- Can you find a,b,c, such that?

$$\begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

No inverse…

- One of the variances is 0
- We don’t have enough data
- The Gaussian density is not defined

- Boohoo



TU Delft

Estimating Covariance Matrices

- For quadratic classifier need to estimate covariances, e.g., by

$$\hat{\Sigma}_k = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \hat{\mu}_k)(\mathbf{x}_i - \hat{\mu}_k)^T$$

  for each of the classes
- When there is insufficient data, this covariance matrix cannot be inverted
- Alternative : average over all class covariance matrices

$$\hat{\Sigma}_k = \frac{1}{C}\sum_{k=1}^{C}\hat{\Sigma}_k$$

The Two-Class Case: Linear Discriminant

- Define the discriminant

$$f(\mathbf{x}) = \log p(y_1|\mathbf{x}) - \log p(y_2|\mathbf{x})$$

- We get

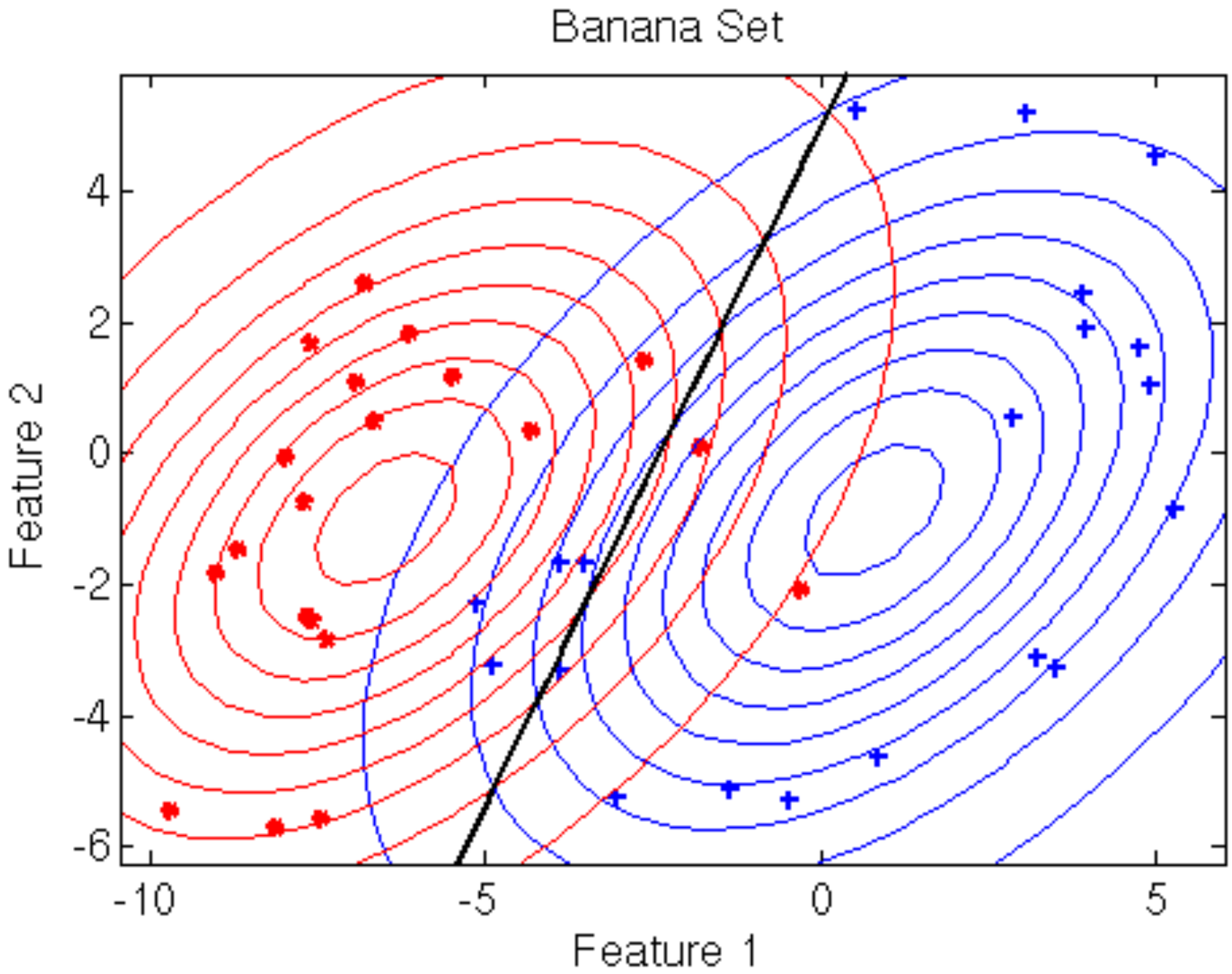
$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

with

$$\mathbf{w} = \hat{\Sigma}^{-1}(\hat{\mu}_2 - \hat{\mu}_1)$$

$$w_0 = \frac{1}{2}\hat{\mu}_2^T\hat{\Sigma}^{-1}\hat{\mu}_2 - \frac{1}{2}\hat{\mu}_1^T\hat{\Sigma}^{-1}\hat{\mu}_1 + \log\frac{p(y_1)}{p(y_2)}$$

Linear Classifier on Banana Data



No Estimated Covariance Matrix

- In some cases even a full averaged covariance matrix is too much to estimate
- As simplification one could assume that all features have the same variance, and are uncorrelated :

$$\hat{\Sigma} = \sigma^2 \mathbb{I}$$

- Obviously, decision rule becomes even simpler :

$$g_i(\mathbf{x}) = -\frac{1}{\sigma^2}\left(\frac{1}{2}\mu_i^T\mu_i - \mu_i^T\mathbf{x}\right) + \log(p(y_i))$$

Nearest Mean Classifier

- Define the discriminant :

$$f(\mathbf{x}) = \log p(y_1|\mathbf{x}) - \log p(y_2|\mathbf{x})$$

- We get

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

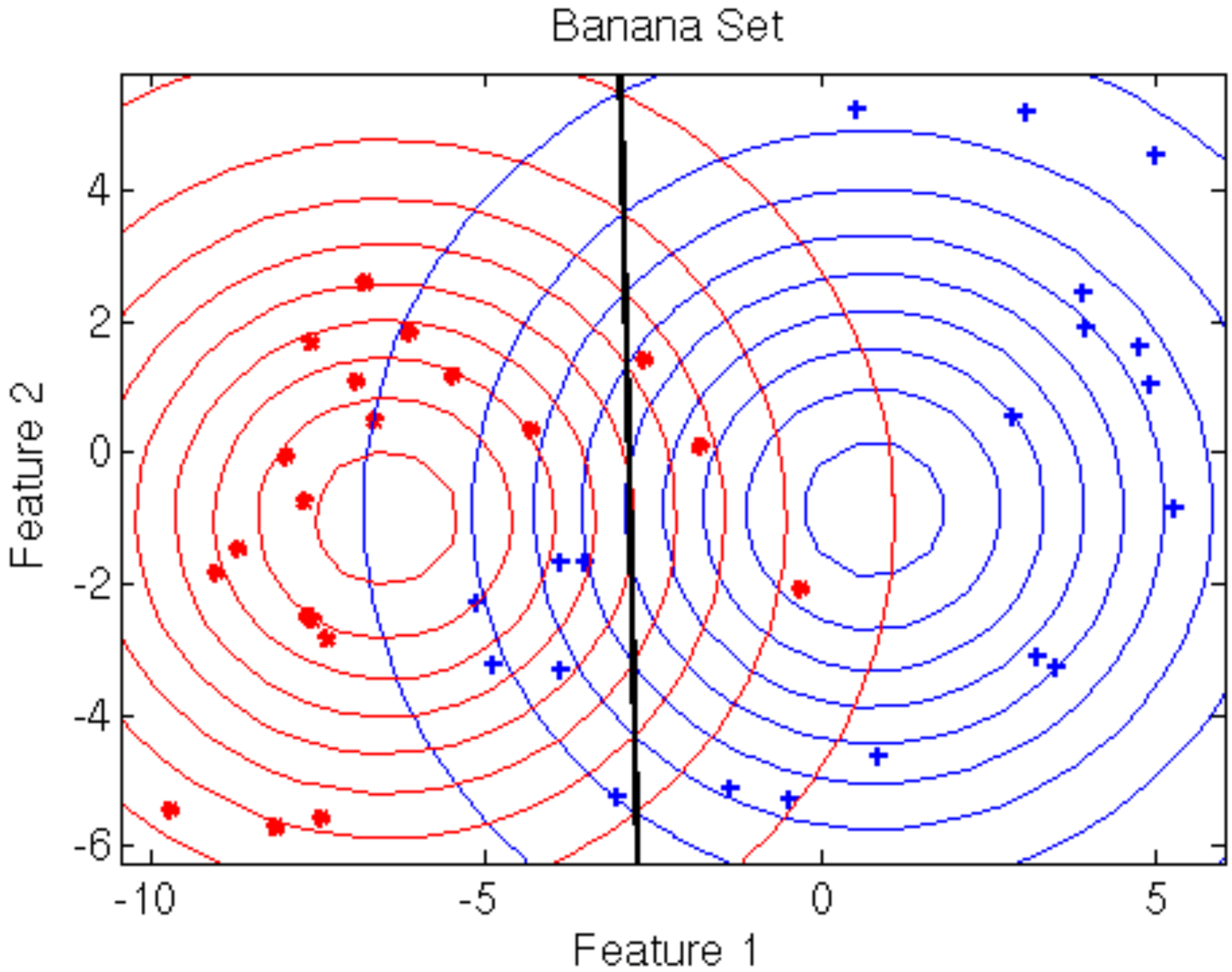
with

$$\mathbf{w} = \hat{\mu}_2 - \hat{\mu}_1$$

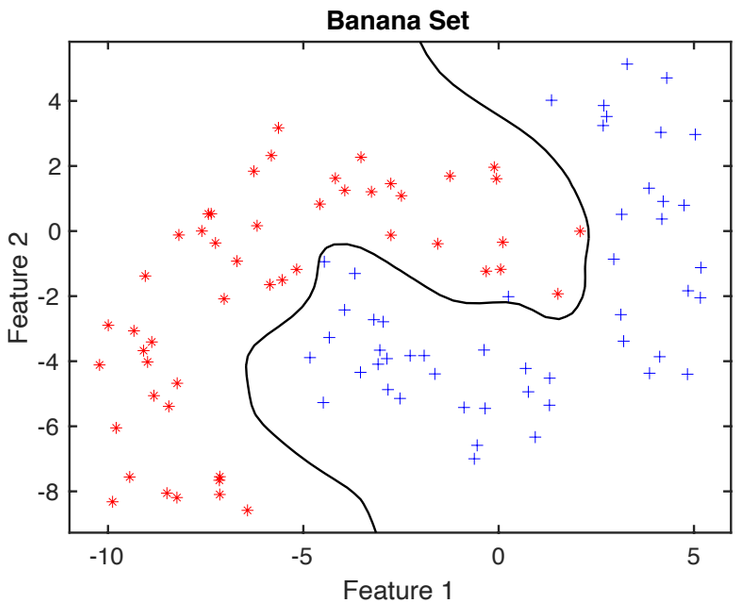
$$w_0 = \frac{1}{2}\hat{\mu}_2^T\hat{\mu}_2 - \frac{1}{2}\hat{\mu}_1^T\hat{\mu}_1 + \sigma^2 \log\frac{p(y_1)}{p(y_2)}$$

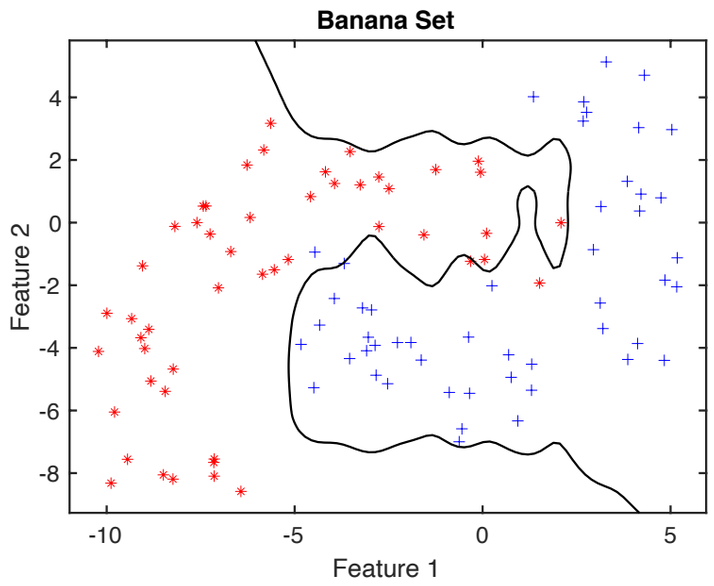
- Again a linear classifier, but it only uses distance to the mean of each of the classes : nearest mean classifier

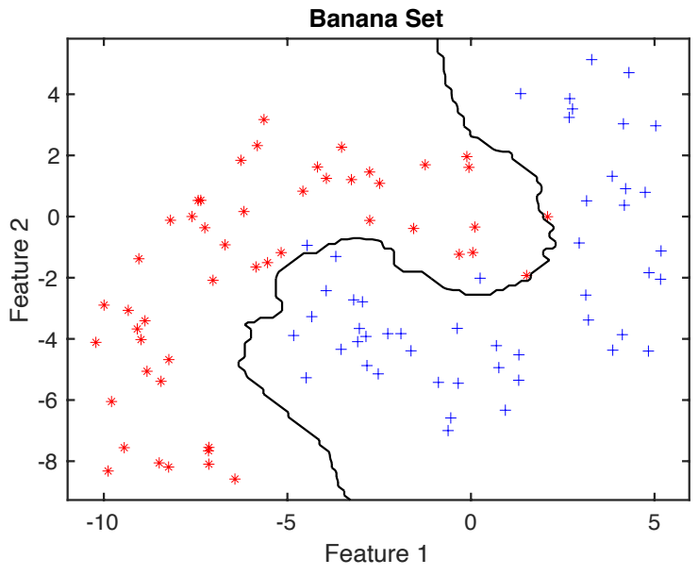
Nearest Mean on Banana Data



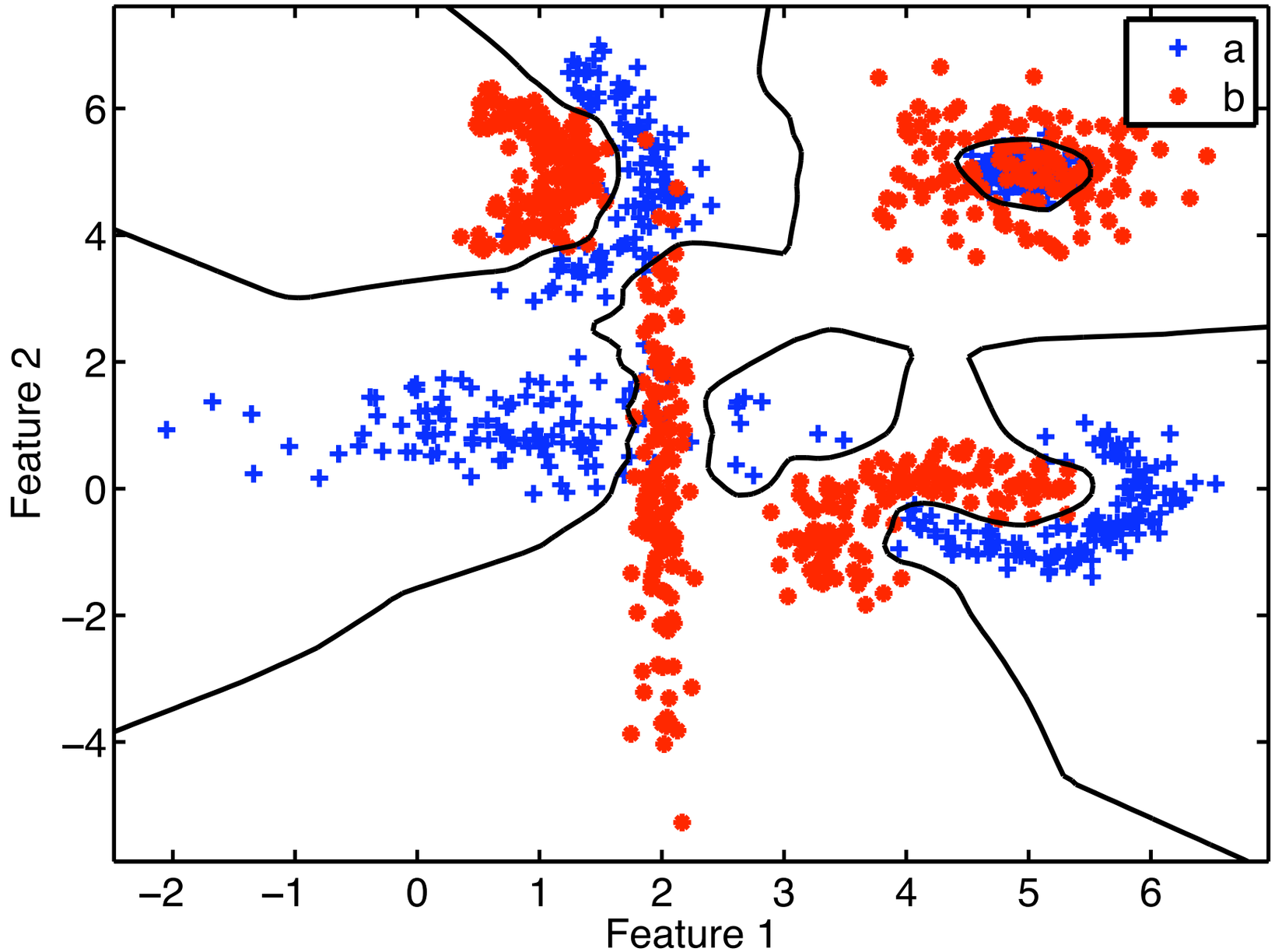
Non-parametric Classification







Multi-modal distributions

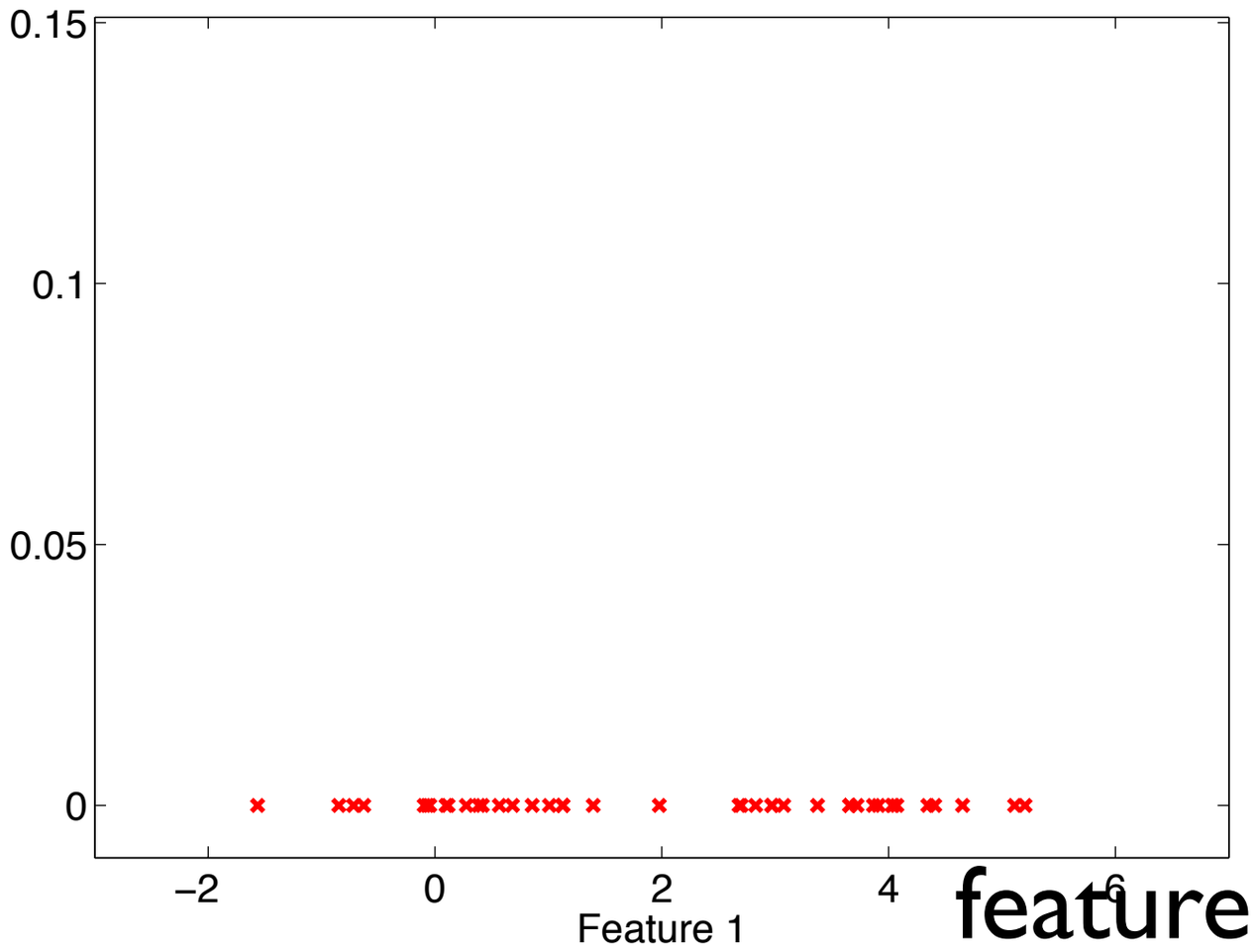


- Depending on the class distributions, the decision boundary can have arbitrary shapes

Contents Non-Parametric

- Histogram method
- Parzen classifier
- Nearest neighbor classifier
- Scaling
- Curse of Dimensionality

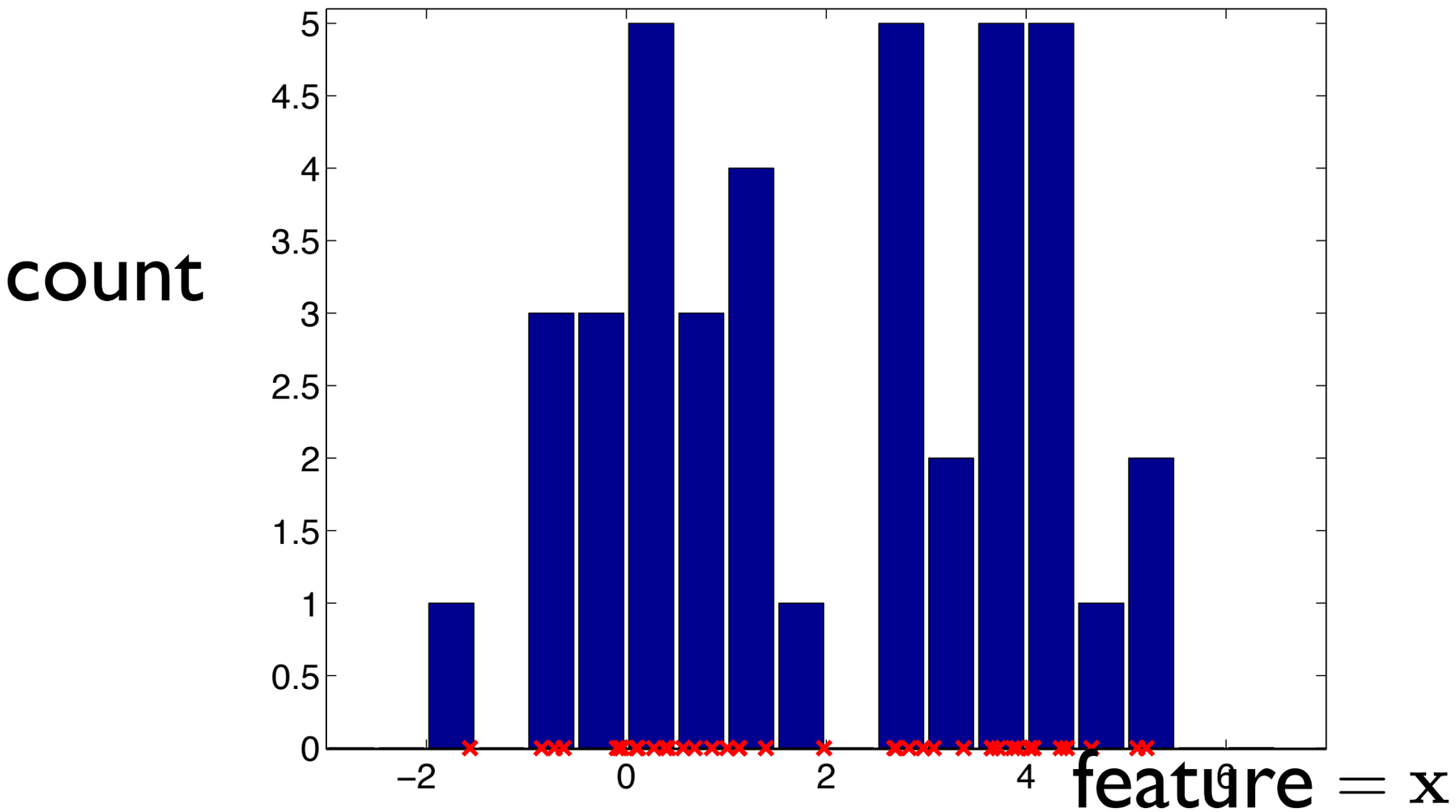
Histogram method



feature $= \mathbf{x}$

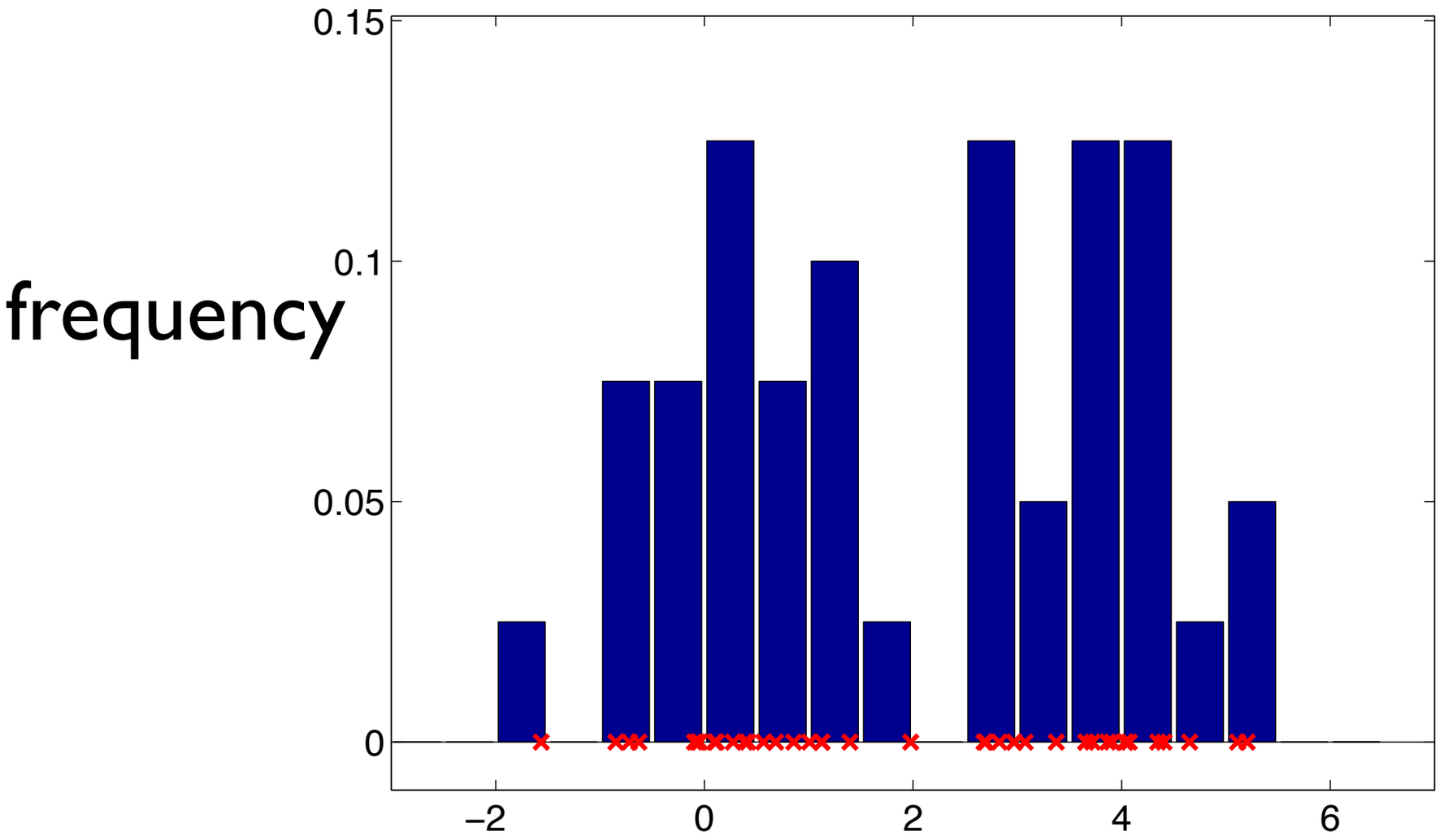
- Example: we have one feature, 40 examples
- How to estimate the probability density (fraction of objects per volume)?

Histogram method



- Split the feature in subregions: width h
- Count the number of objects in each region: k_N

Histogram method



Probability density estimate (frac.of points/volume):

$$\hat{p}(\mathbf{x}) = \hat{p}(\hat{\mathbf{x}}) \approx \frac{1}{h}\frac{k_N}{N}, \quad |\mathbf{x} - \hat{\mathbf{x}}| \leq h/2$$

Histogram method

- How many bins???
- For a reasonable precise approximation, h cannot be too large
- For a stable estimate, h cannot be too small
- Depends on the number of training examples
- (Also the offset can have an influence!)

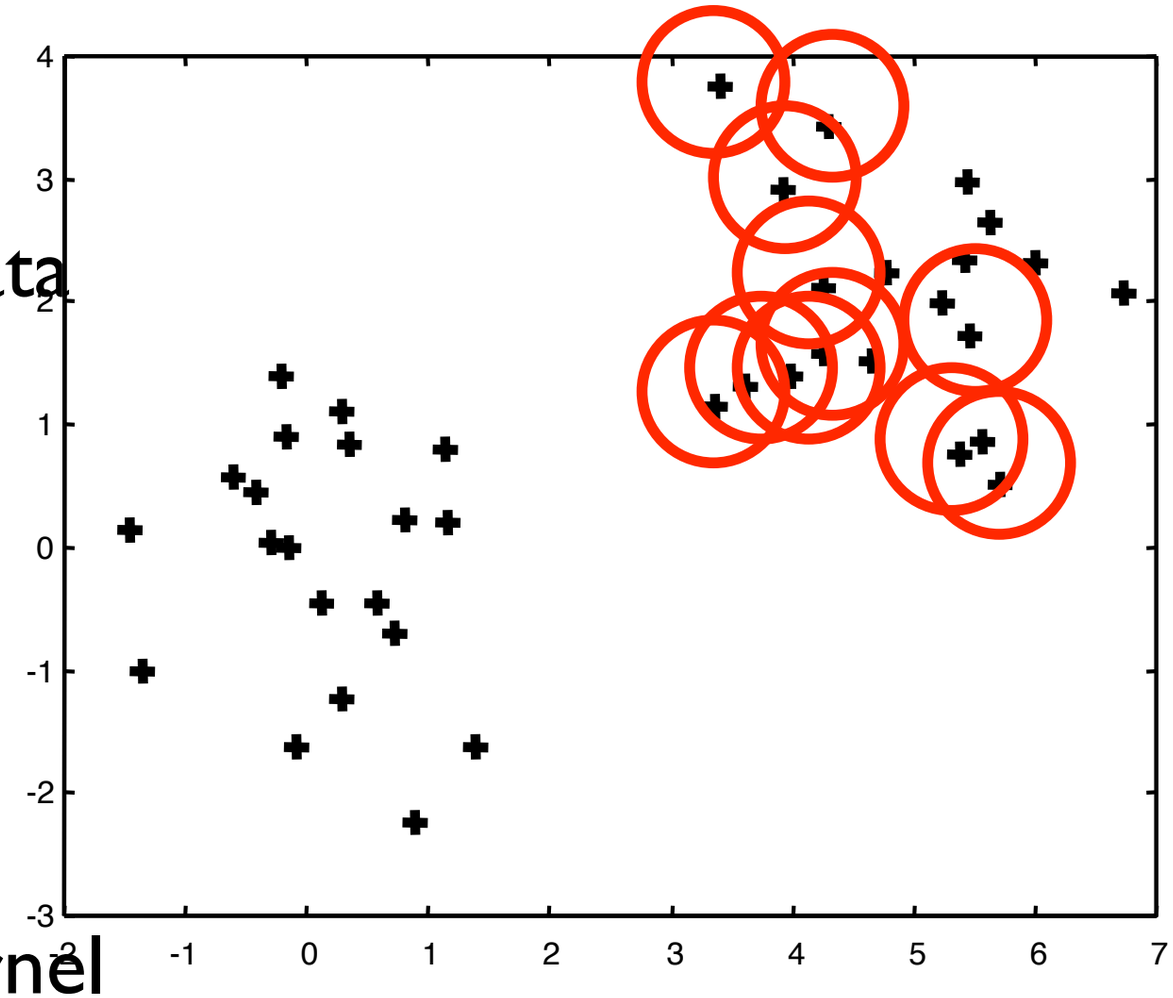
- In practice, two very related methods are used:
  - Parzen density estimate
  - k-Nearest-neighbor density estimate

Parzen density estimation

- Procedure:
  - Fix volume of cell
  - Locate cells on training data
  - Add contributions of cells
- Define cell shape (*kernel*)
  e.g. uniform:

$$K(r,h) = \begin{cases} 0 & \text{if } |r| > h \\ 1/V & \text{if } |r| \leq h \end{cases}$$

with V the volume of the kernel



- For test object **z** sum all cells:

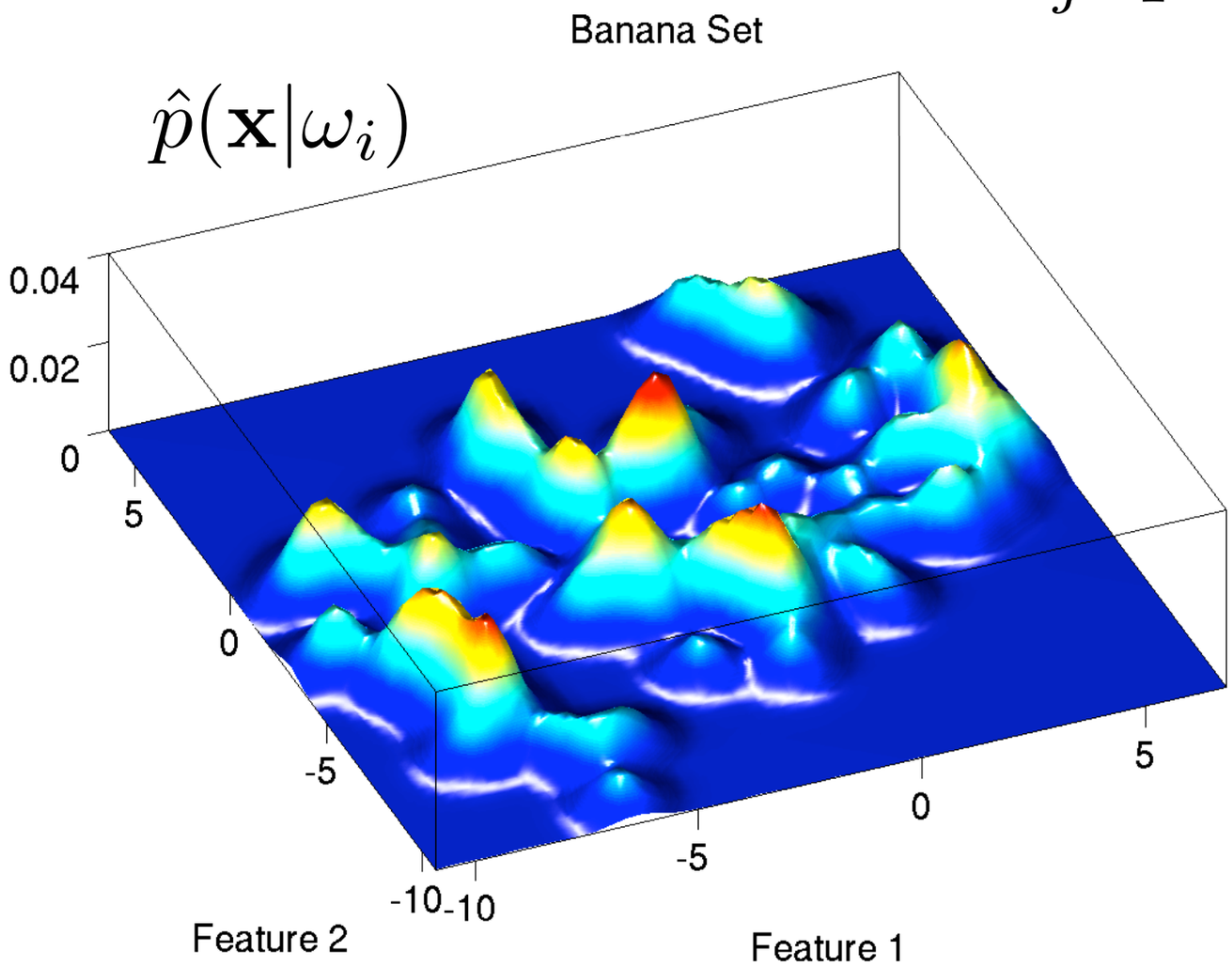
$$\hat{p}(\mathbf{z}|h) = \frac{1}{n}\sum_{i=1}^{n} K(\|\mathbf{z} - \mathbf{x}_i\|, h)$$

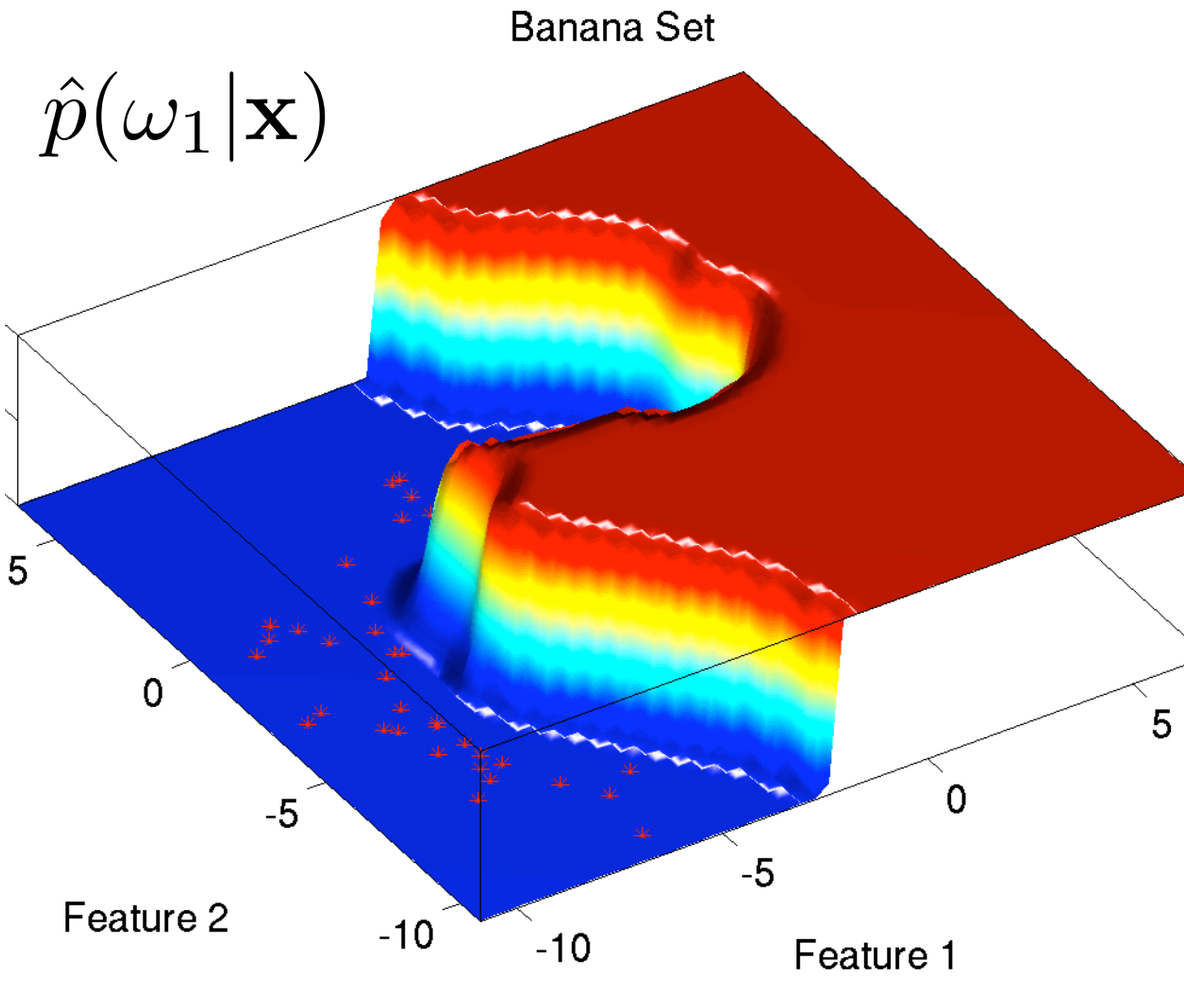
Parzen classifier

parzenc

- Plug in the Gaussian density:

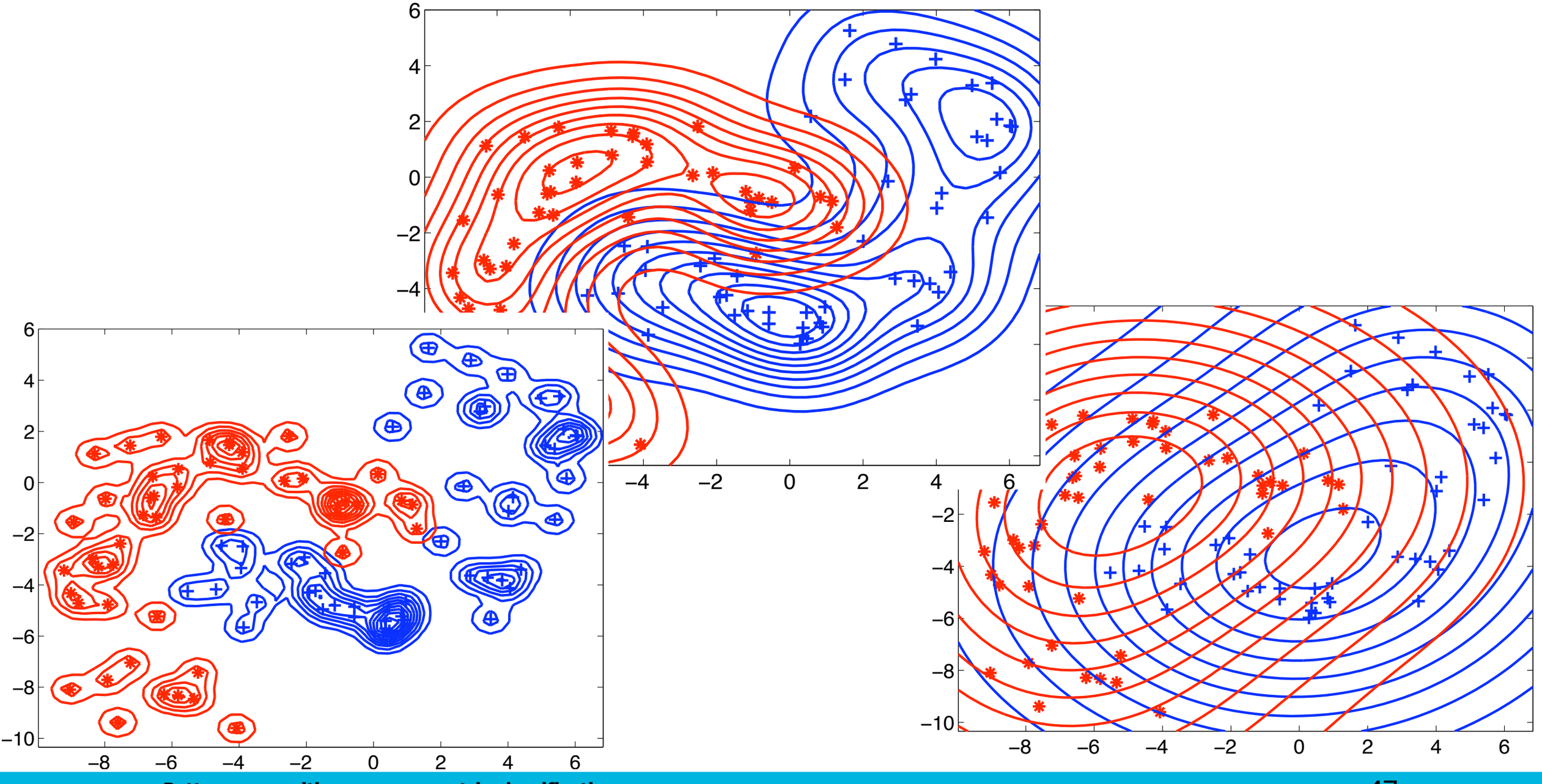
$$\hat{p}(\mathbf{x}|\omega_i) = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathcal{N}(\mathbf{x}|\mathbf{x}_j^{(i)}, h\mathcal{I})$$





Parzen width parameter

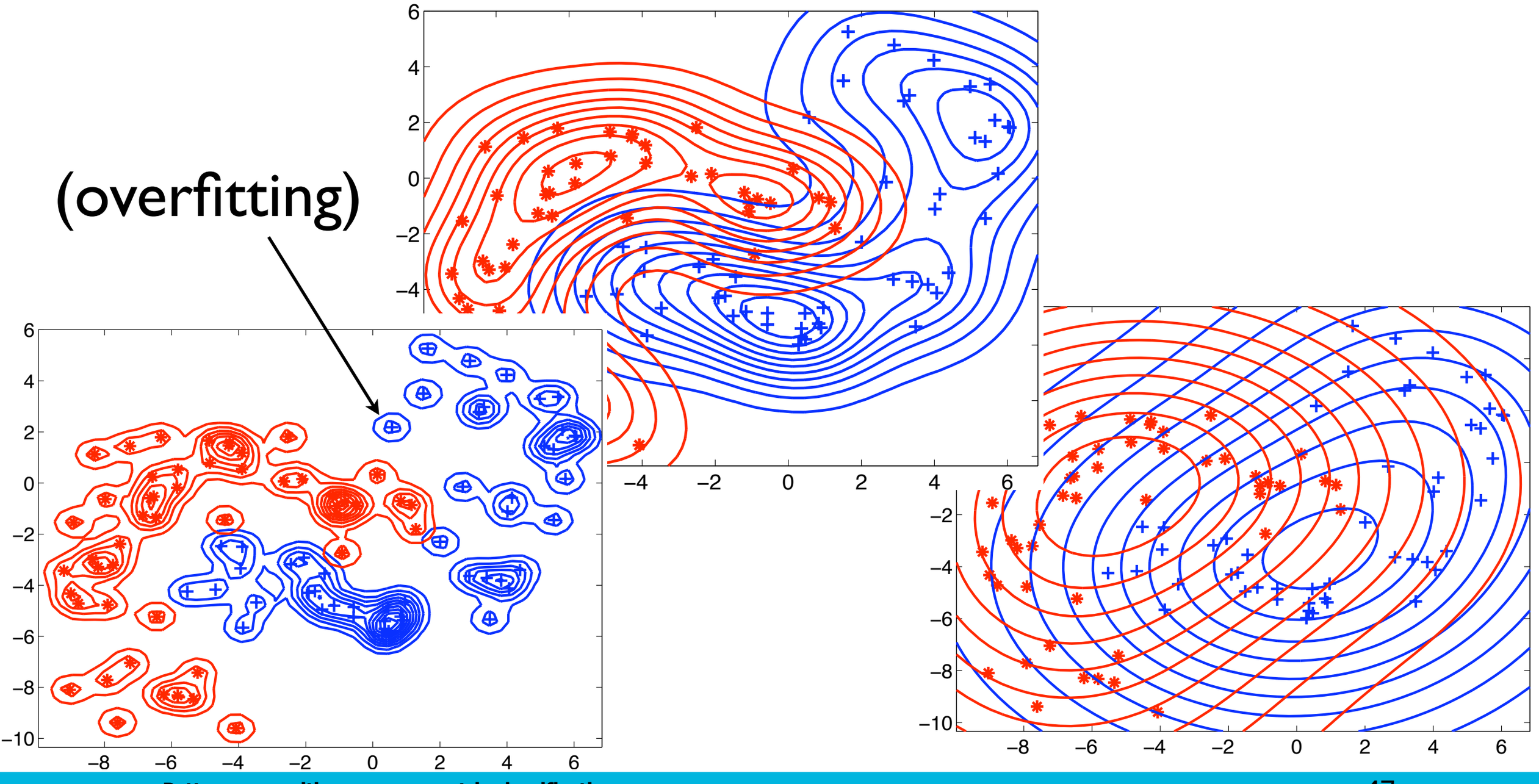
- The choice of h is important

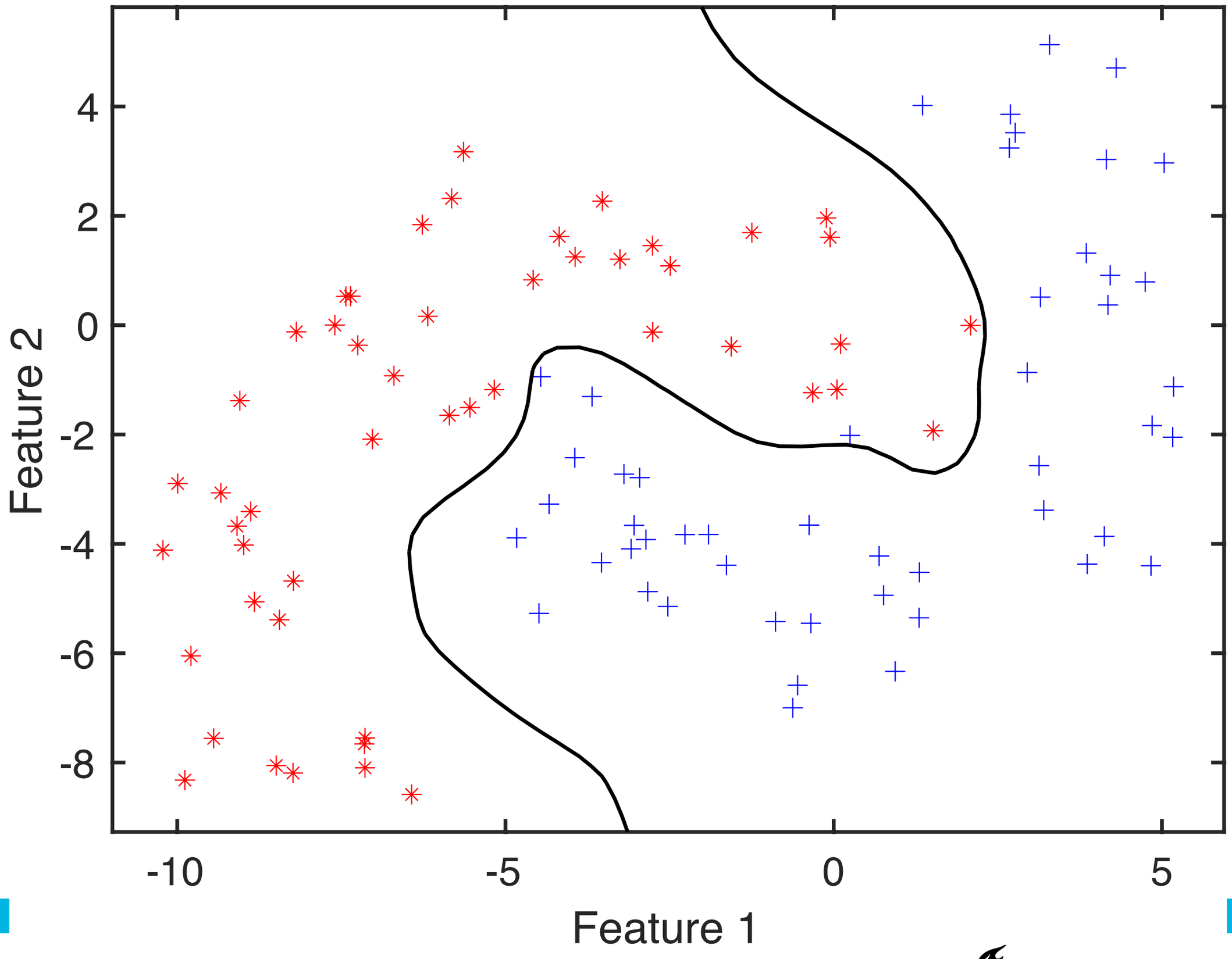


Parzen width parameter

- The choice of h is important

(overfitting)





Delft University of Technology

Optimization of h

- Use the average k-nearest neighbor distance (k=10 is suggested...)
- Use a heuristic

$$h = \sigma \left(\frac{4}{p+2} \right)^{\frac{1}{p+4}} n^{\frac{-1}{p+4}}$$

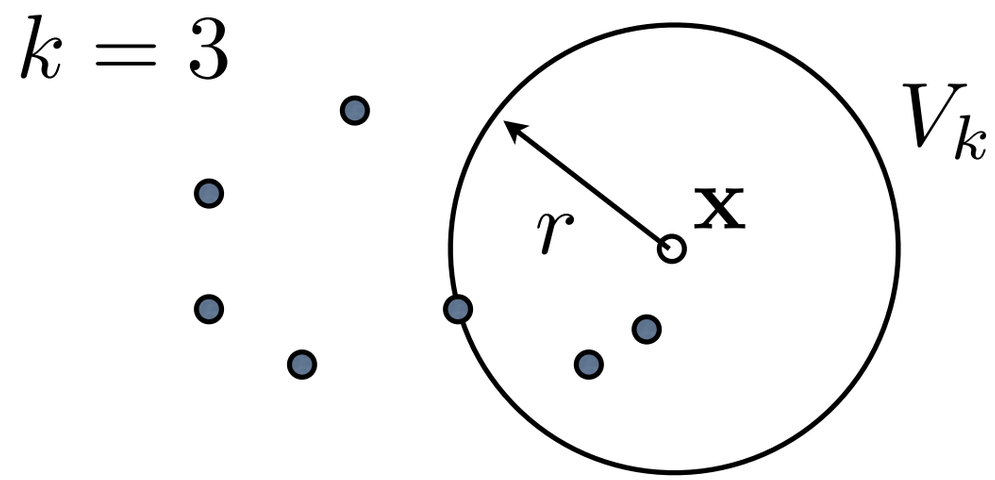
$$\sigma^2 = \frac{1}{p} \sum_{i=1}^{p} s_{ii}$$

- Optimize the likelihood using leave-one-out

$$\prod_{i=1}^{n} \hat{p}(\mathbf{x}_i)$$

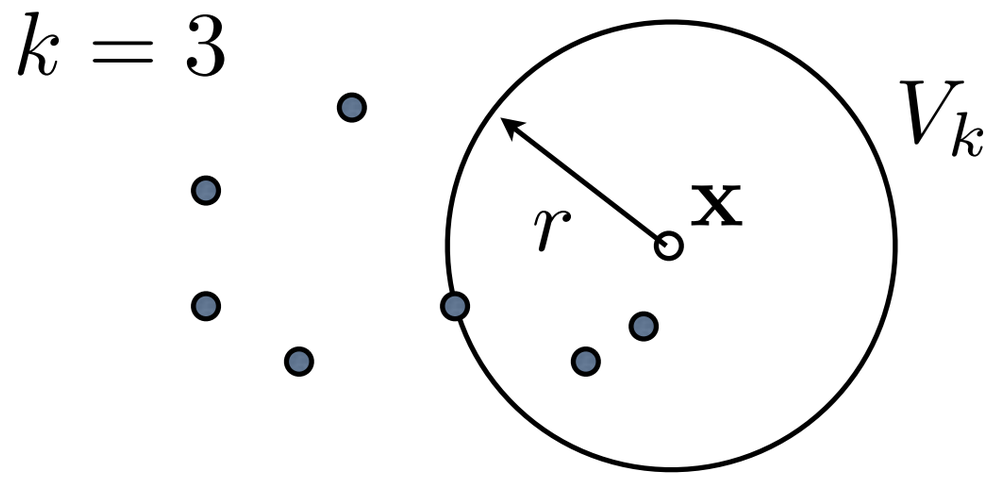
- and more...
- See lab course.

Nearest neighbor classification



- Locate the cell on the test object $\mathbf{x}$
- Do **not** fix the volume of the cell: grow the cell until it covers k objects: find the k-th neighbor
- Need one cell per test object

Nearest neighbor classification

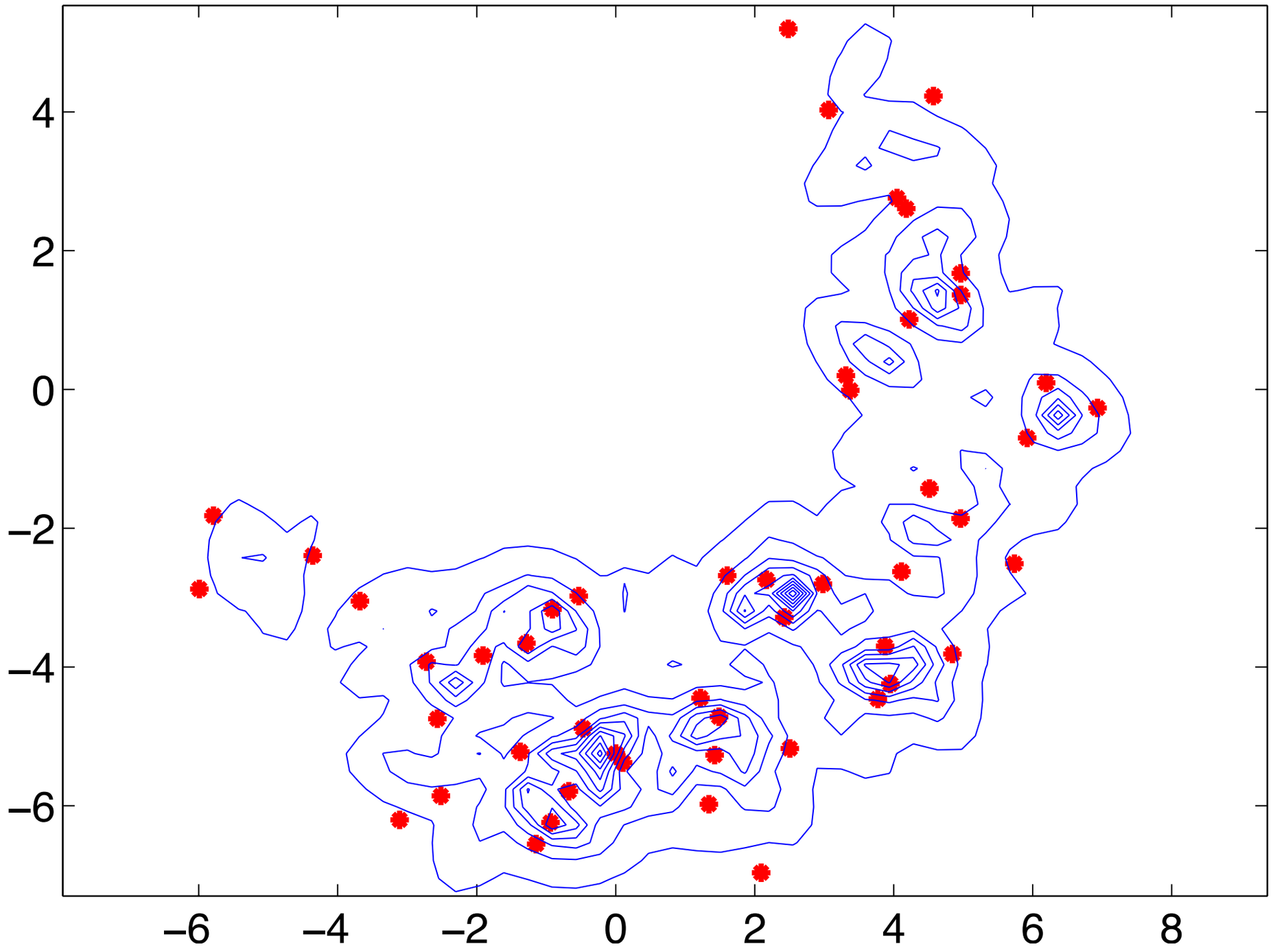


- For the k-nearest neighbor density estimate

$$\hat{p}(\mathbf{x}) = \frac{k}{nV_k}$$

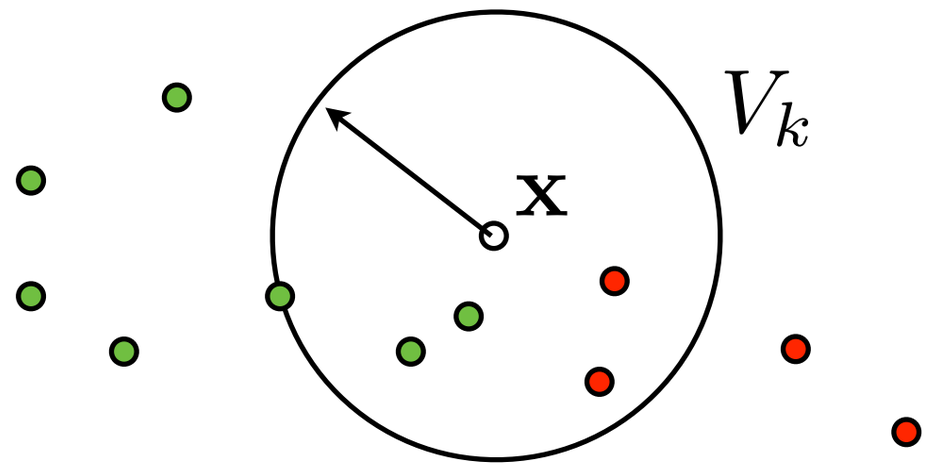
where V_k is the volume of the sphere centered at $\mathbf{x}$ with radius r (the distance to the k-th nearest neighbor)

k-NN density estimate



k-Nearest neighbor classification

knnc



$k = 5$

$k_1 = 3$

$k_2 = 2$

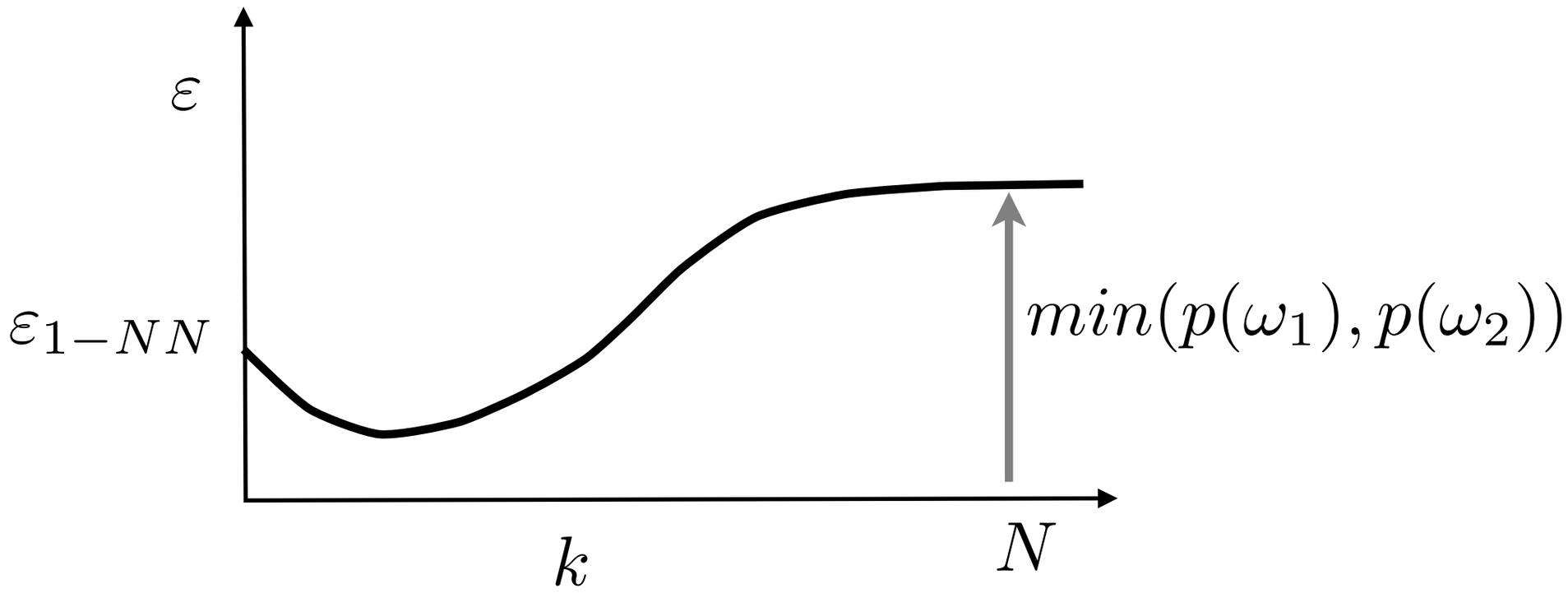
- When more classes are present, count how many objects of each of the classes are member of the k neighbors: $\hat{p}(\mathbf{x}|\omega_m) = \dfrac{k_m}{n_m V_k}$
- Class priors: $\hat{p}(\omega_m) = \dfrac{n_m}{n}$
- Bayes: $\hat{p}(\mathbf{x}|\omega_m)\hat{p}(\omega_m) > \hat{p}(\mathbf{x}|\omega_i)\hat{p}(\omega_i) \rightarrow k_m > k_i$

The choice of k

- We can still choose k.
- What is the influence of k?
  When does the classifier become more smooth?
  When more ragged?

- What happens for $k = 1$ and $k = n$?

The choice of k

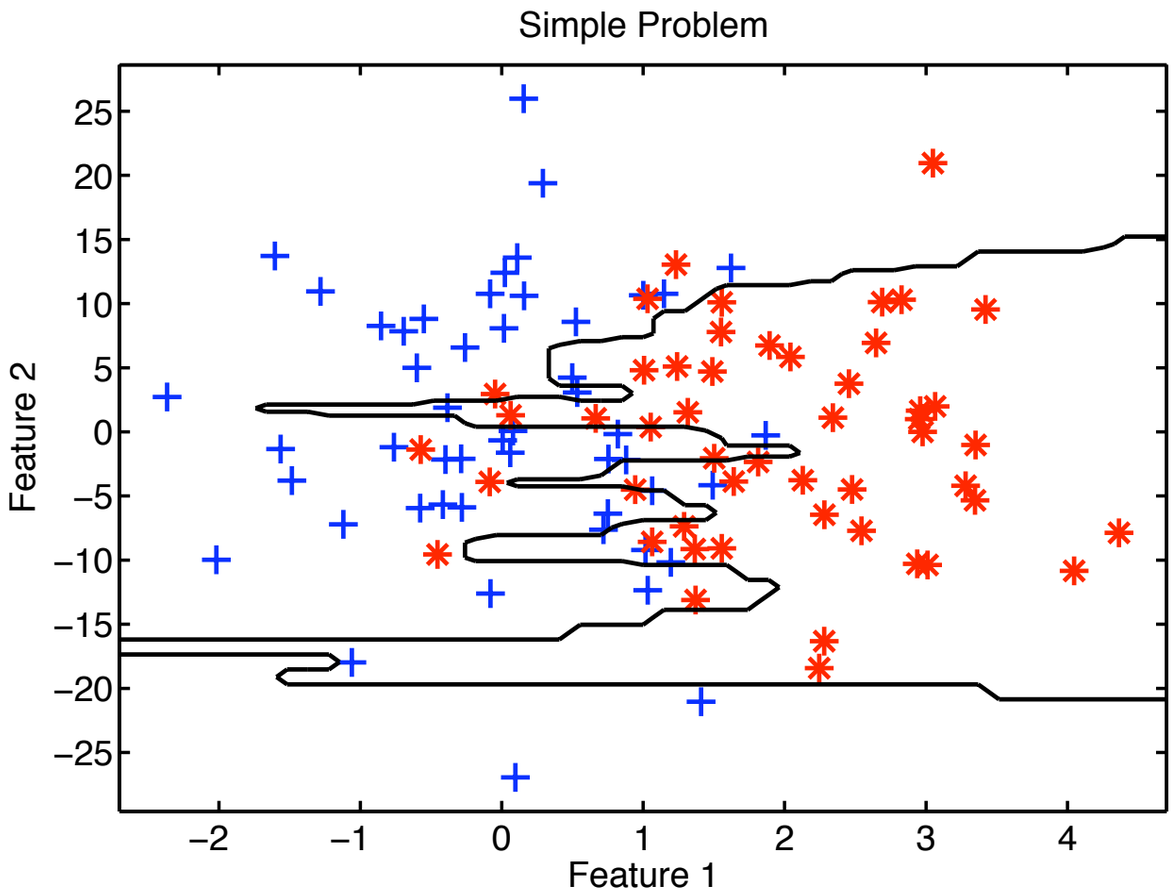


$$\lim_{k \to N} \varepsilon_{k-NN} = \min(p(\omega_1), p(\omega_2))$$

At some optimum k the error is minimum.

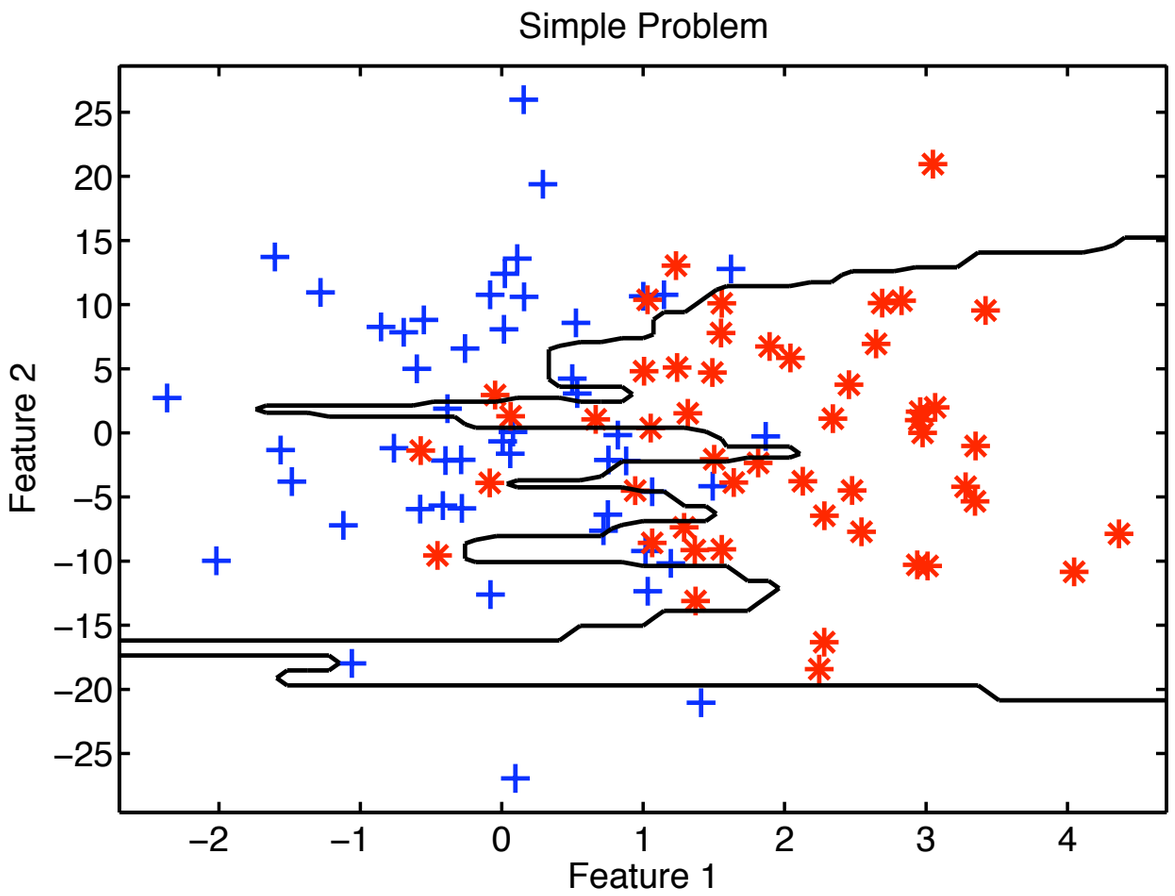
Sometimes strange results:

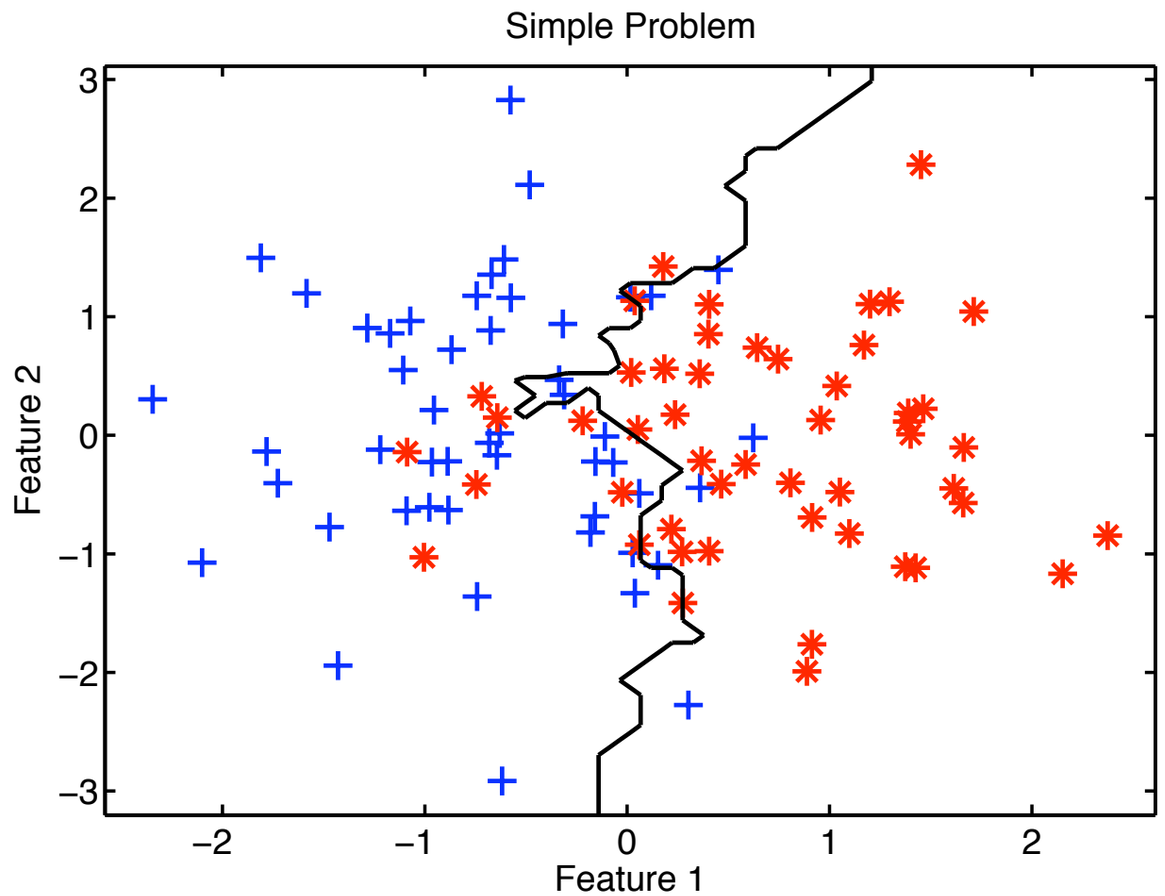
- How is this possible?



Sometimes strange results:

- How is this possible?



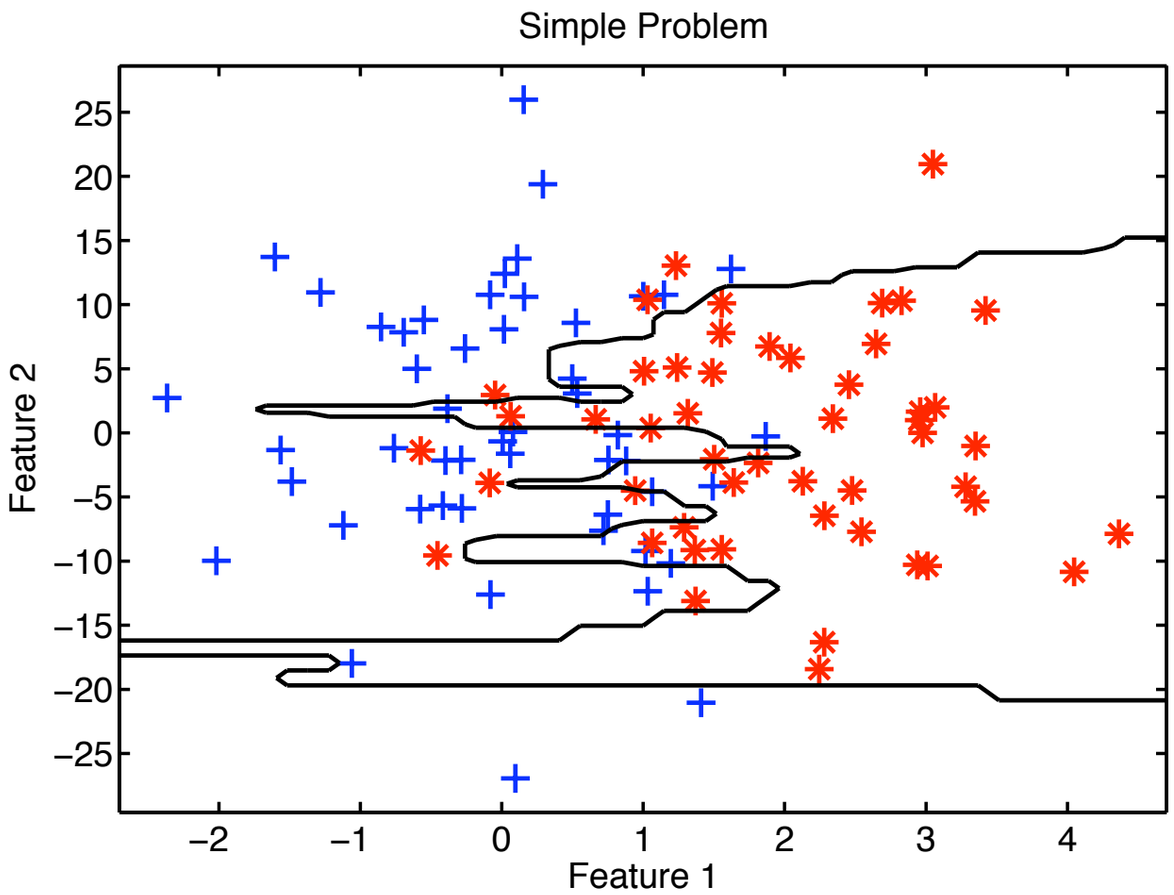


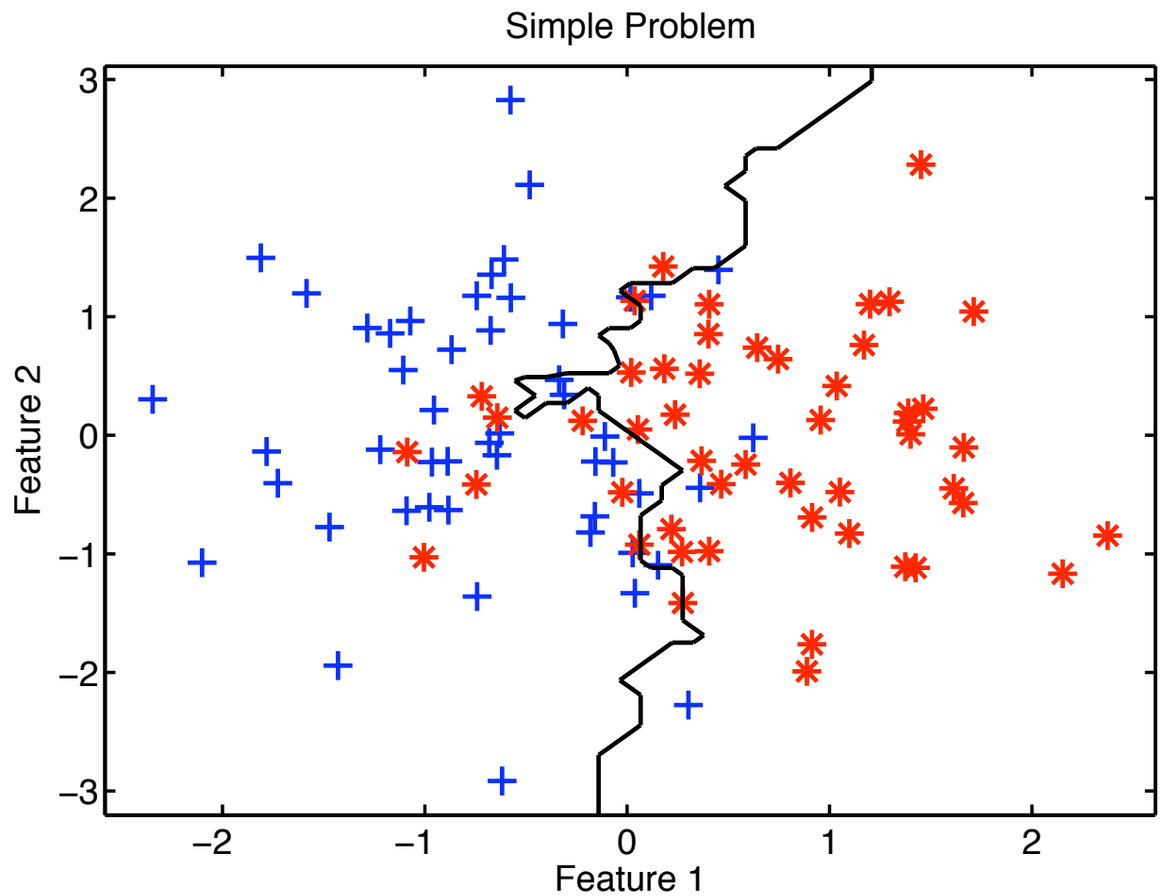
Scale your features!

```
b=a*scalem(a,'variance')
```

Sometimes strange results:

- How is this possible?





Scale your features!

```
u=scalem([],'variance')*knnc([],3)
w = a*u
```

Advantages/disadvantages

- simple and flexible classifiers
- often a very good classification performance
- it is simple to adapt the complexity of the classifier

- relatively large training sets are needed
- the complete training set has to be stored
- distances to all training objects have to be computed
- the features have to be scaled sensibly
- the value for k or h has to be optimized

Conclusions

- For classification we want $p(\omega|\mathbf{x})$
- We can use Bayes theorem when we only can estimate $p(\mathbf{x}|\omega)$ and $p(\omega)$
- Using the Parzen density and the nearest neighbor density we can derive the Parzen classifier and the nearest neighbor classifier
- Both methods contain a parameter that influences the complexity (flexibility)
- Both methods are sensitive to the scaling of the features

The curse of Dimensionality

- Typically, we have more than one or two features.
  It seems, **more features** provide **more information**, so use more features!

The curse of Dimensionality

- Typically, we have more than one or two features.
  It seems, **more features** provide **more information**, so use more features!

Really.....?

The curse of Dimensionality

- Typically, we have more than one or two features.
  It seems, **more features** provide **more information**, so use more features!
  Really.....?

- Unfortunately, this is not really true:
  the **curse of dimensionality**

  The number of examples you need, may increase exponentially with the number of features!

Learning in high dim.

- A large part of the work in Pattern Recognition is to deal with high dimensional data

- Try to use all information, without overfitting

- Reduce features
- Simplify models
- Careful optimization

Avoid density estimation

- From the k-nearest neighbor we saw already that we don't need to explicitly estimate a density!
- Estimating densities is hard, in particular when we have a high number of features (high dimensional feature space, curse of dimensionality)
- Alternatively:
  - Assume we have a function to describe the decision boundary,
  - Optimize the free parameters of this function directly
  - No Bayes' theorem, no density estimates!

Conclusions

- You can make classifiers using parametric density estimations + Bayes' rule: Quadratic, linear classifier, nearest mean classifier
- You can make classifiers using non-parametric density estimations + Bayes' rule: Parzen classifier, nearest neighbor classifier

- When you have many features, beware of the curse of dimensionality

Questions to consider...

- Do nonparametric classifiers have parameters?
- If I increase the size of the training set, will the optimal h in the Parzen classifier grow, or shrink?
- If I increase the size of the training set, will the optimal k in the k-nearest neighbor grow/ shrink?
- How will the optimal k/h grow/shrink when I add more features?
- How do I find out in practice if a feature is conditionally independent of another feature?