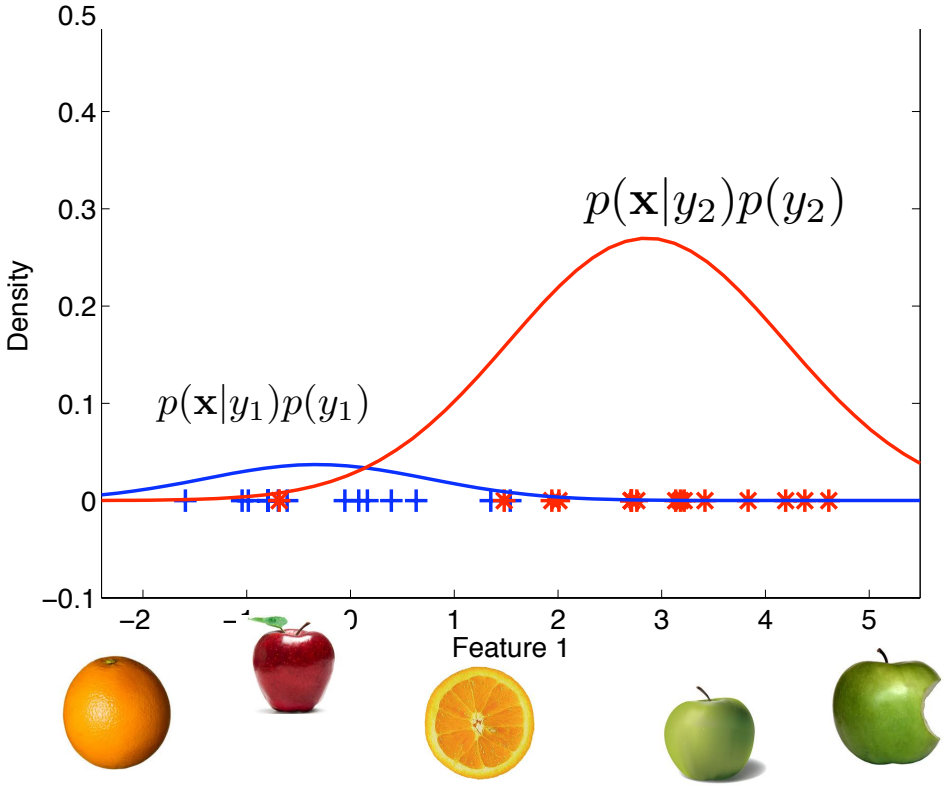
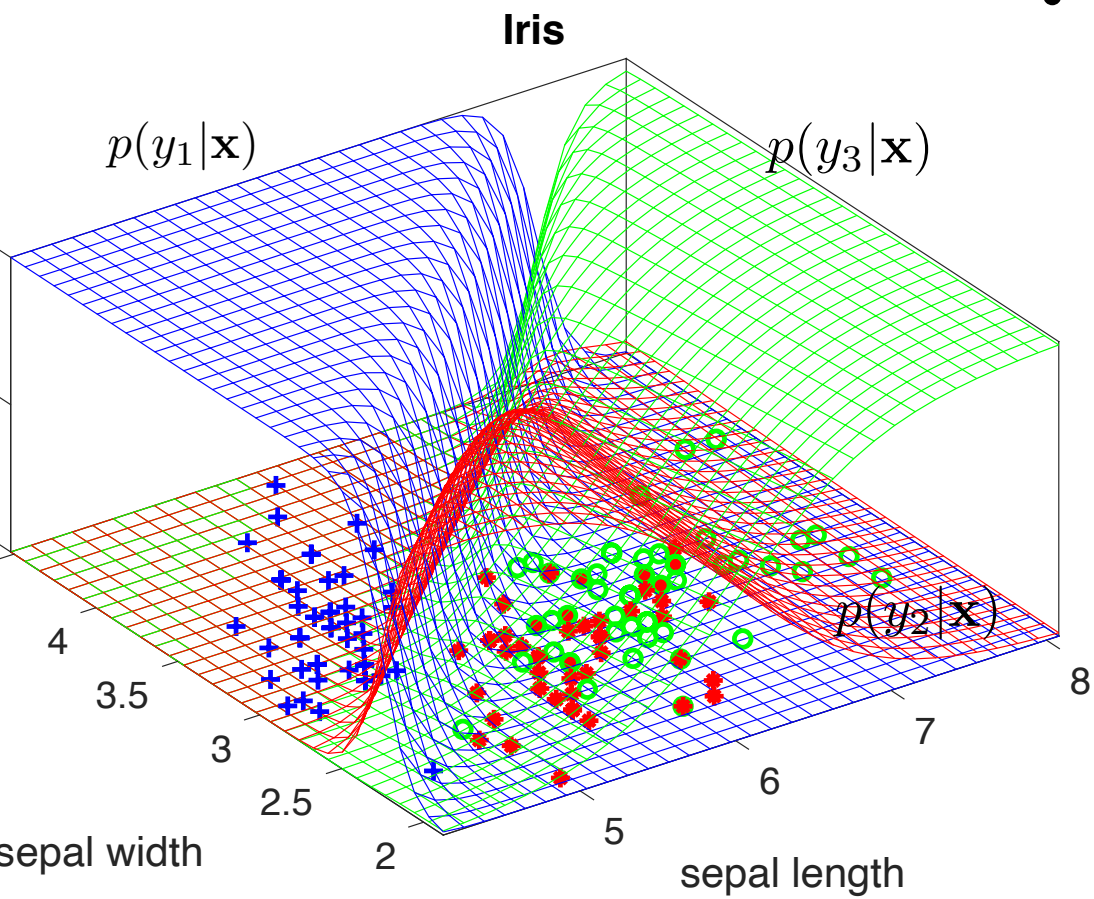
Gaussian Density-based Classifiers

CSE2510 Machine Learning



TU Delft

Modeling the posterior probability



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

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

In practice, we approximate: $\hat{p}(y|\mathbf{x})$

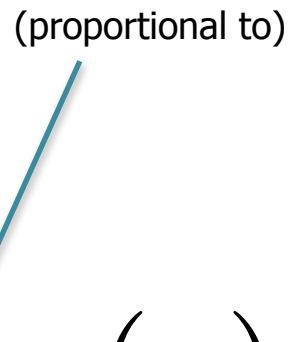
or we fit a function:

$$f(\mathbf{x})$$

TU Delft

Recapitulation

- Up to now:
  - Without models no generalization
  - Density estimation: core of statistical learning, really hard problem
  - Curse of dimensionality: adding features does not always help
  - Parametric density estimation: Gaussian
- Now:
  - Classification using Gaussian densities:
  - Quadratic classifier
  - Linear classifier
  - Nearest mean classifier



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

$$p(y_1|\mathbf{x}) > p(y_2|\mathbf{x})$$

Plug-in Bayes Rule

- Classify to class 1 when : $p(y_1|\mathbf{x}) > p(y_2|\mathbf{x})$
- May be easier to estimate the class-conditional probability density : $\hat{p}(\mathbf{x}|y_1),\ \hat{p}(\mathbf{x}|y_2)$
- Use Bayes' rule to transform one into the other :

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

- Putting just $\hat{p}(\mathbf{x}|y)$ in results in plug-in Bayes rule

Plug-in Bayes Rule

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

  we need to estimate:

- (1) the class priors $\hat{p}(y)$

- (2) the unconditional probabilities $\hat{p}(\mathbf{x})$

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

How to estimate the class priors?

- Given a training set, how can you estimate $\hat{p}(y)$?

How to estimate the class priors?

- Given a training set, how can you estimate $\hat{p}(y)$?
- The classes are discrete, $\hat{p}(y)$ is a true probability
- Often they are known/assumed
- Otherwise, just count!

$$\hat{p}(y_1) = \frac{N_1}{N}$$

$$\hat{p}(y_2) = \frac{N_2}{N}$$

How to estimate the unconditional pr.

- How to estimate the (unconditional) data distribution $\hat{p}(\mathbf{x})$?

- You can explicitly compute it:

$$p(\mathbf{x}) = p(\mathbf{x}|y_1)p(y_1) + p(\mathbf{x}|y_2)p(y_2)$$

- But if you just find the largest posterior, not needed:

$$\frac{p(\mathbf{x}|y_1)p(y_1)}{\cancel{p(\mathbf{x})}} > \frac{p(\mathbf{x}|y_2)p(y_2)}{\cancel{p(\mathbf{x})}}$$

Estimate the class conditional prob.

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

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

  - It models a 'blob'-like distribution, in a M-dimensional feature space
  - μ is the mean of the distribution
  - Σ is the (elliptical) shape of the distribution

Plug-in Gaussian Distribution

- For each class we have a Gaussian distribution

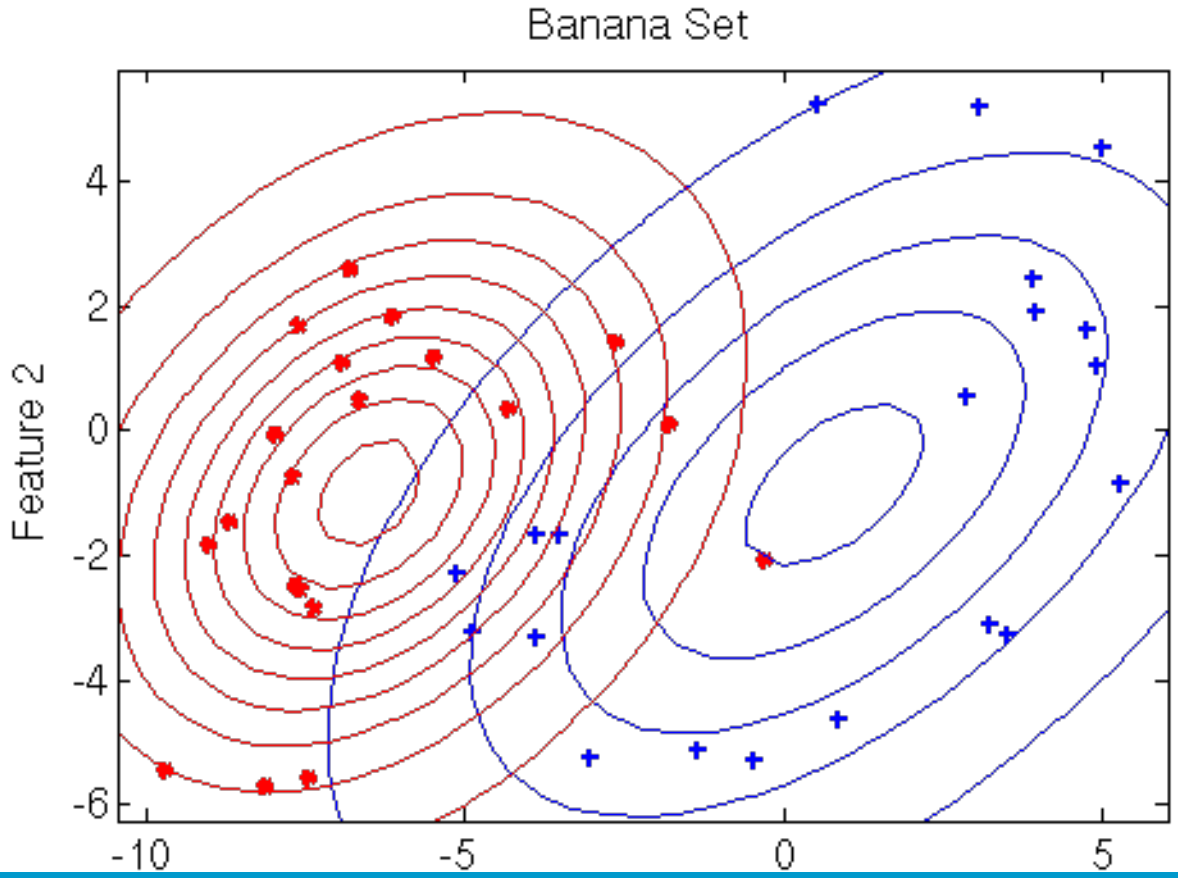
$$\hat{p}(\mathbf{x}|y) = \frac{1}{\sqrt{(2\pi)^M \det(\hat{\Sigma}_y)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\hat{\mu}_y)^T \hat{\Sigma}_y^{-1}(\mathbf{x}-\hat{\mu}_y)\right)$$

- We have to estimate the parameters, e.g. ML

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

Example on Banana Data

- A single Gaussian distribution on each class:



TU Delft

The Two-Class Case

- Define the discriminant

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

- Rewriting this, it appears that one gets

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

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

Class Posterior Probability Gaussian

- Combining

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

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

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

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

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

Normal-based Classifier

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

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

- You can work out the square term
- Classifier becomes :

Assign **x** to class y_i when for all $i \neq j$:

$$g_i(\mathbf{x}) - g_j(\mathbf{x}) > 0$$

Quadratic discriminant

- General form for a two-class classifier:

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

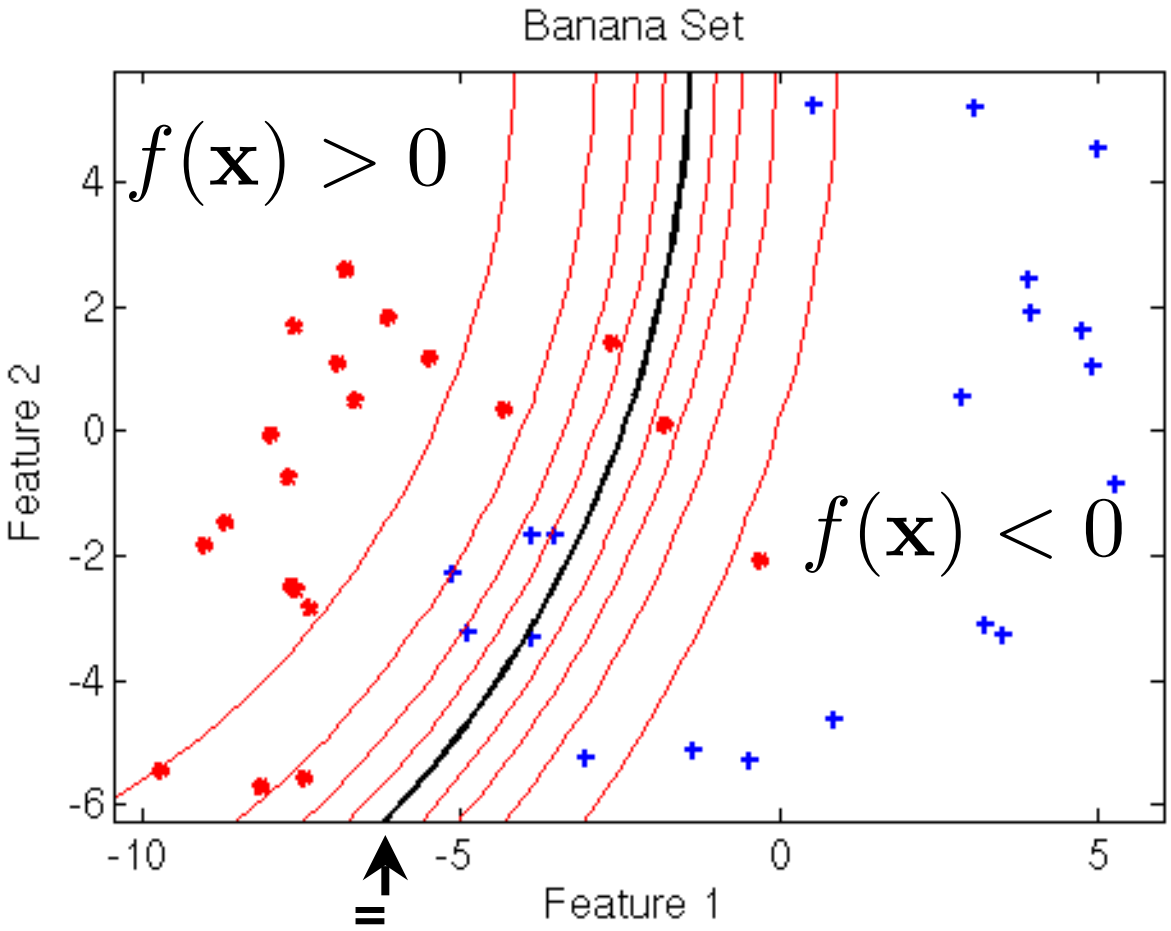
- with

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

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

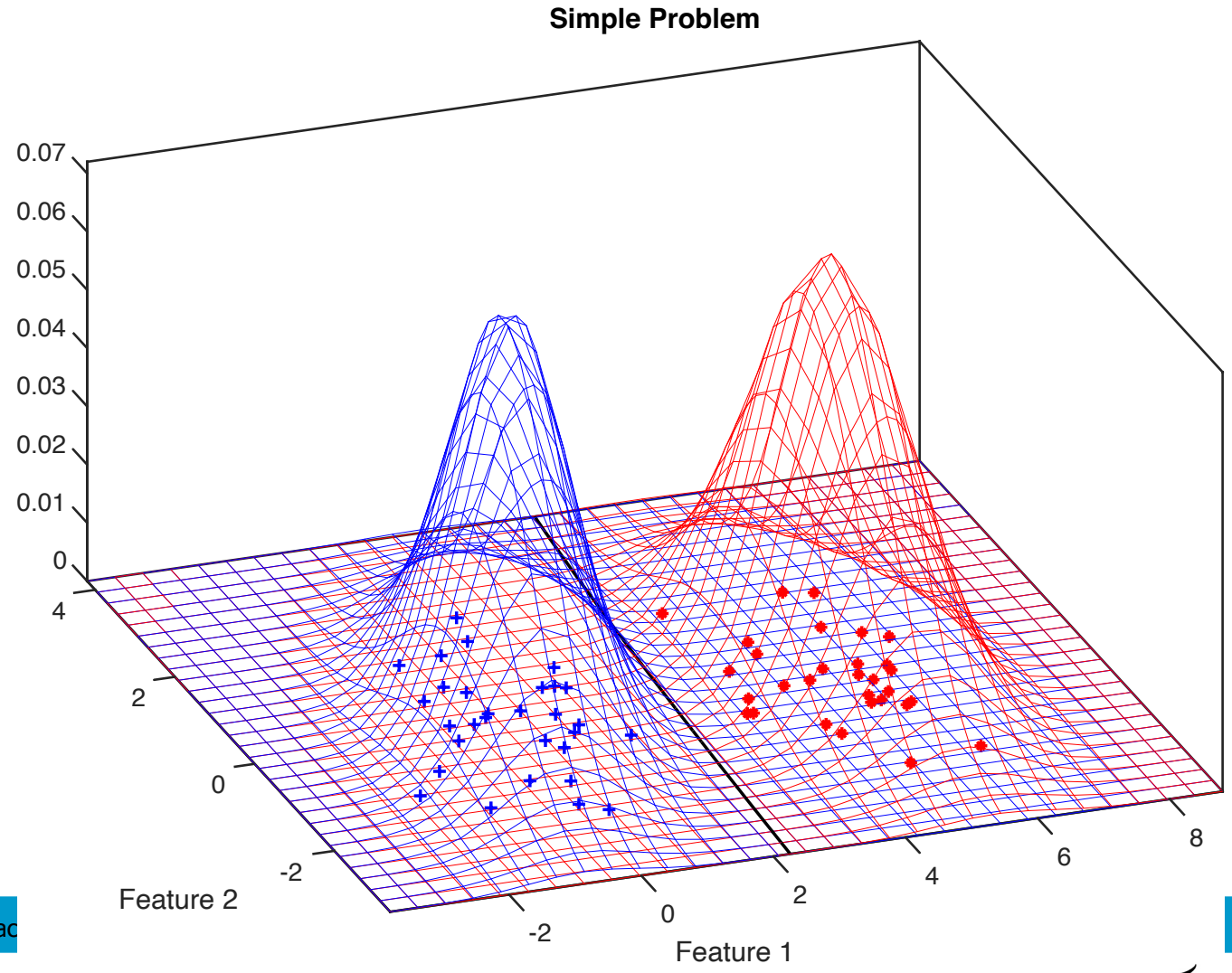
$$w_0 = -\frac{1}{2}\log\det\Sigma_1 - \frac{1}{2}\mu_1^T \Sigma_1^{-1}\mu_1 + \log p(y_1)$$
$$+\frac{1}{2}\log\det\Sigma_2 + \frac{1}{2}\mu_2^T \Sigma_2^{-1}\mu_2 - \log p(y_2)$$

Quadratic Classifier on Banana Data



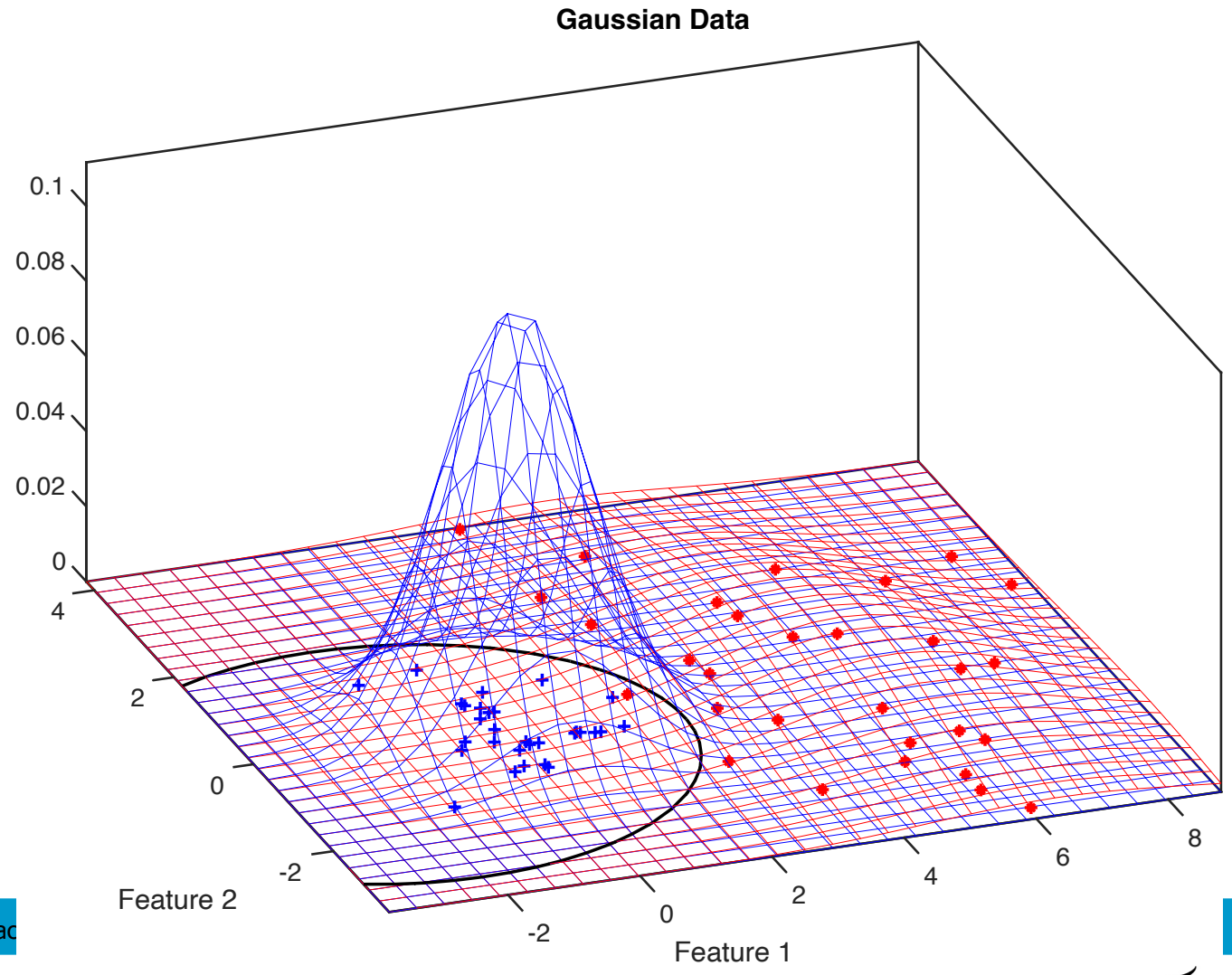
TU Delft

(Almost) linear decision boundary



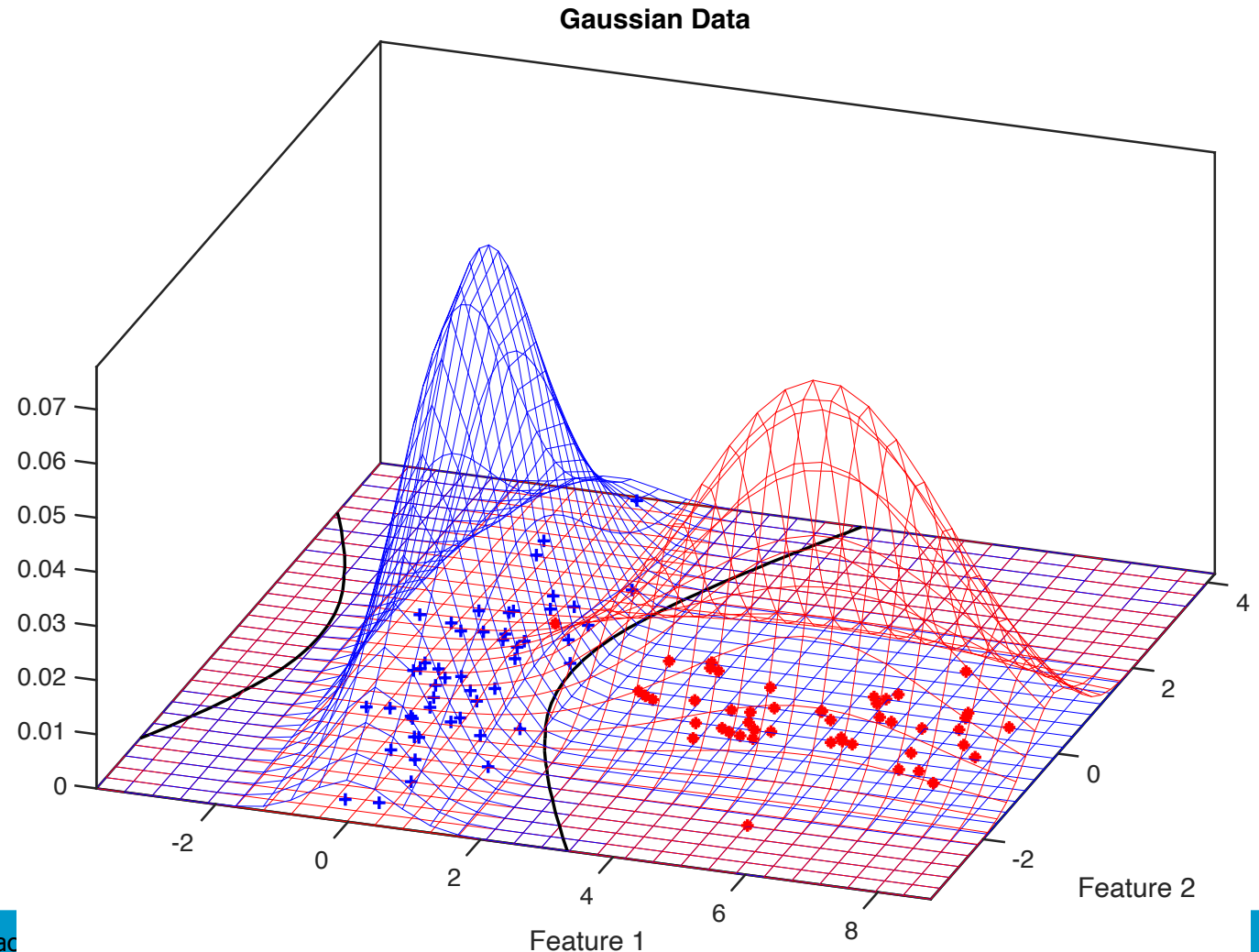
TU Delft

Circular decision boundary



TU Delft

Hyperbolic decision boundary

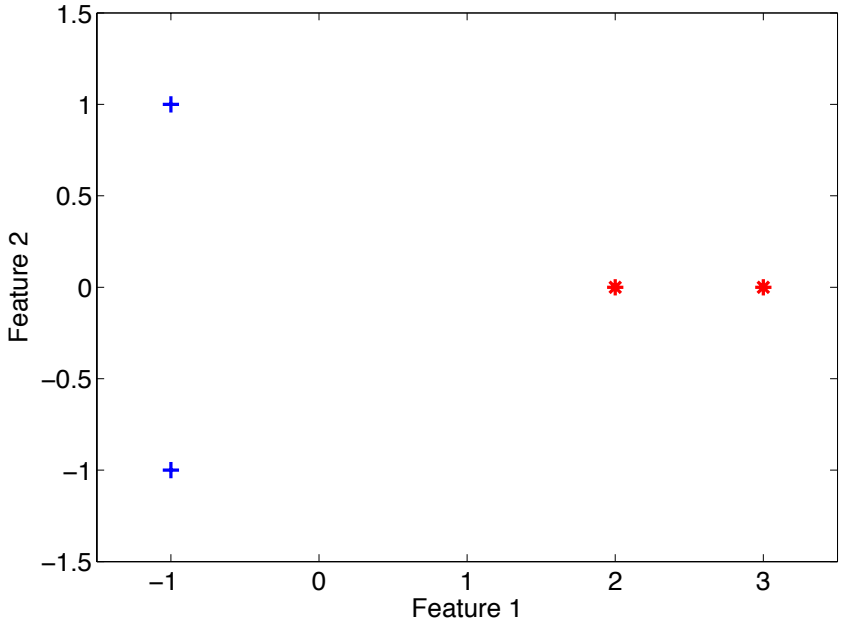


TU Delft

Estimating the covariance matrix

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

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



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

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?

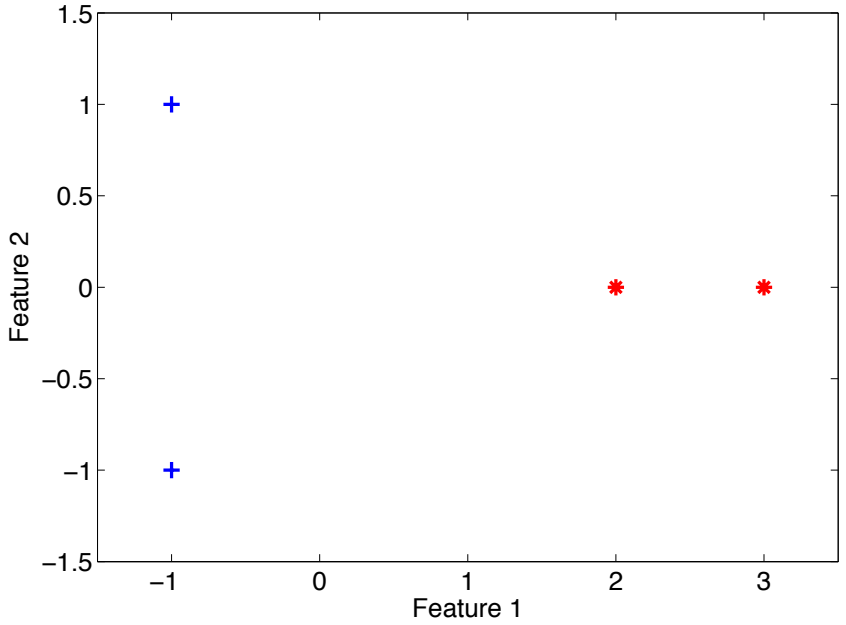
- Inverse of?: $\hat{\Sigma} = \begin{bmatrix} 0.25 & 0 \\ 0 & 0 \end{bmatrix}$
- It should hold that: $\hat{\Sigma}\hat{\Sigma}^{-1} = \mathbb{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
- Can you find a,b,c, such that?

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

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

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

Estimating covariance matrix

- For high-dimensional data, you need many samples to estimate the covariance matrix
- (How many parameters have to be estimated?)

- Simplify:
- Assume all classes have **the same** covariance matrix

Average Covariance Matrix

- When using averaged covariance, we get

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

- I.e., first and quadratic term are always the same

- We end up with:

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

- This classifier is linear: linear normal-based classifier

The Two-Class Case: LDA

- 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

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

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

Linear discriminant

- Let us assume that the decision boundary can be described by:

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0$$

- Weight vector $\mathbf{w}$ and bias term (offset) w_0
- Classify:

$$\text{classify } \mathbf{x} \text{ to } \begin{cases} y_1 & \text{if } \mathbf{w}^T\mathbf{x} + w_0 \geq 0 \\ y_2 & \text{if } \mathbf{w}^T\mathbf{x} + w_0 < 0 \end{cases}$$

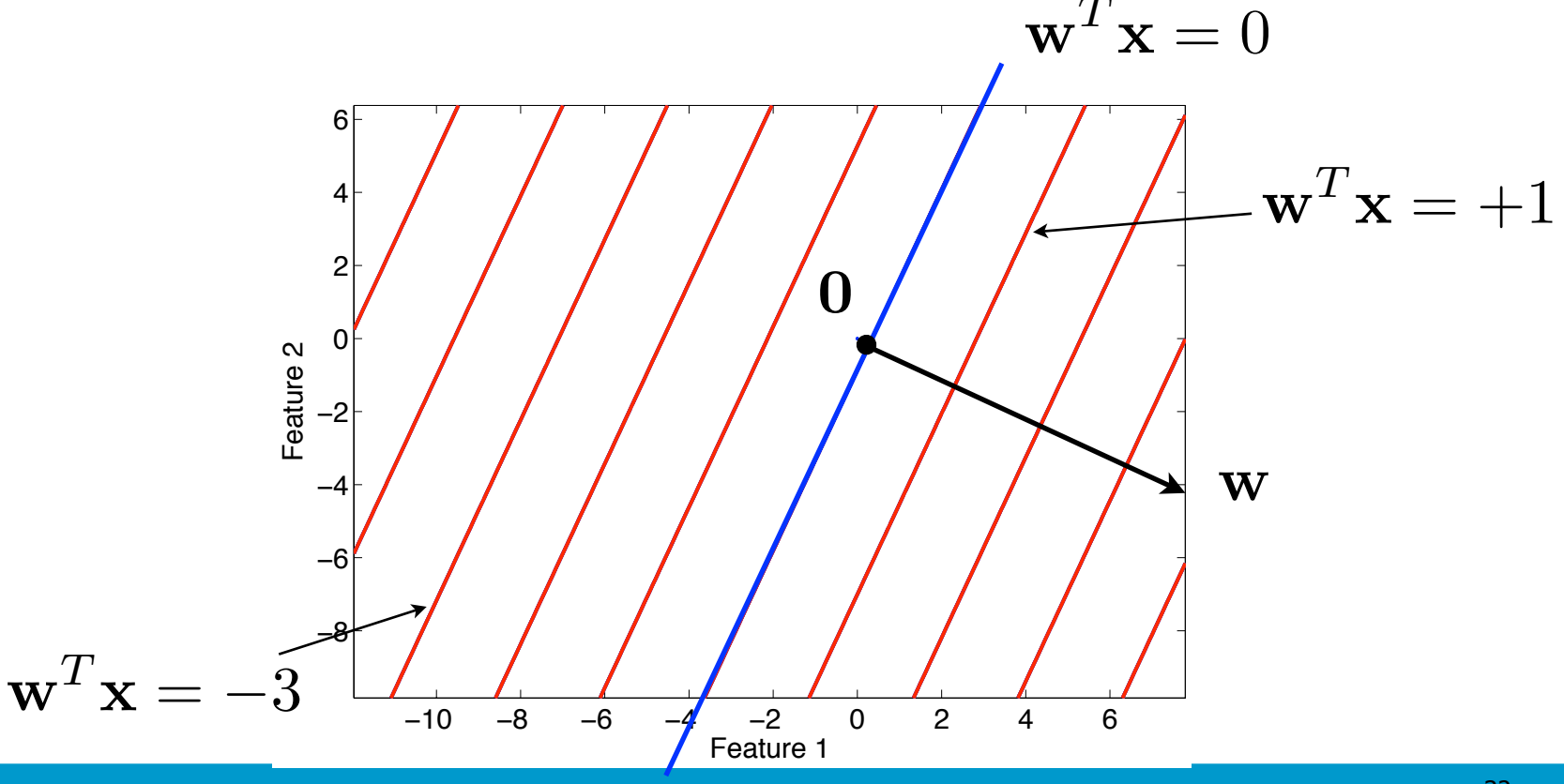
- In the most general sense, this is called linear discriminant analysis

Linear function?

- What does $\mathbf{w}^T\mathbf{x}$ mean?

Linear function?

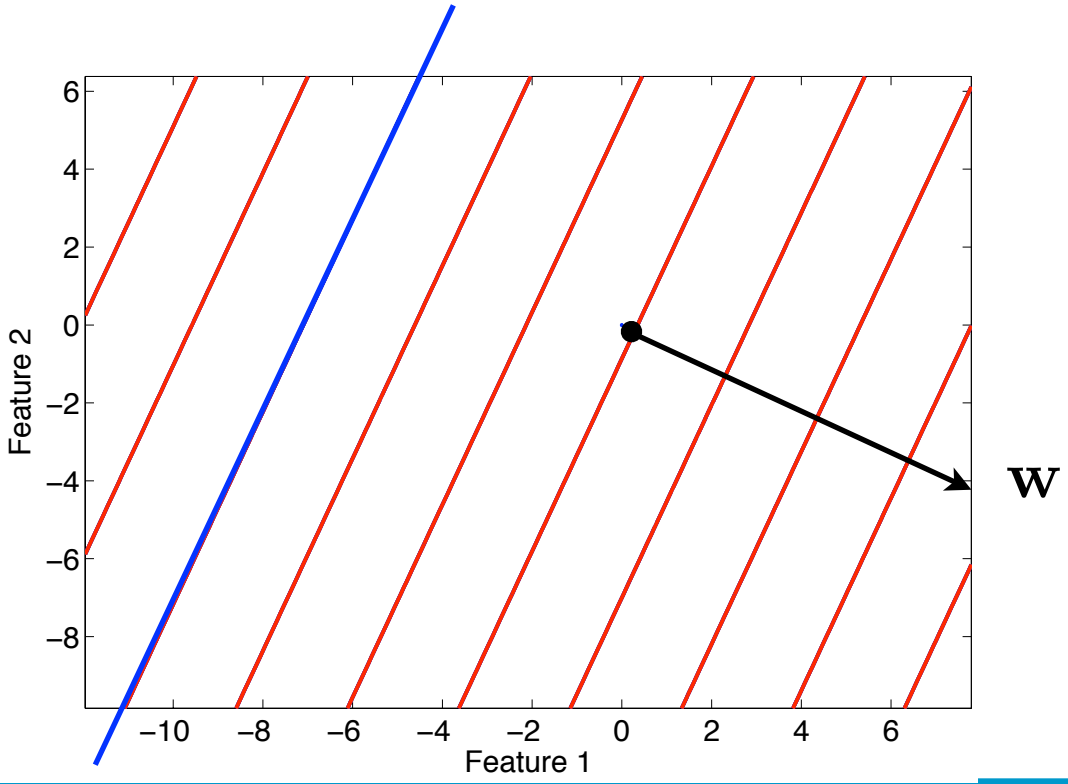
- What does $\mathbf{w}^T\mathbf{x}$ mean?
  Assume I have a 2-dimensional $\mathbf{w}$



TU Delft

Linear function, the bias

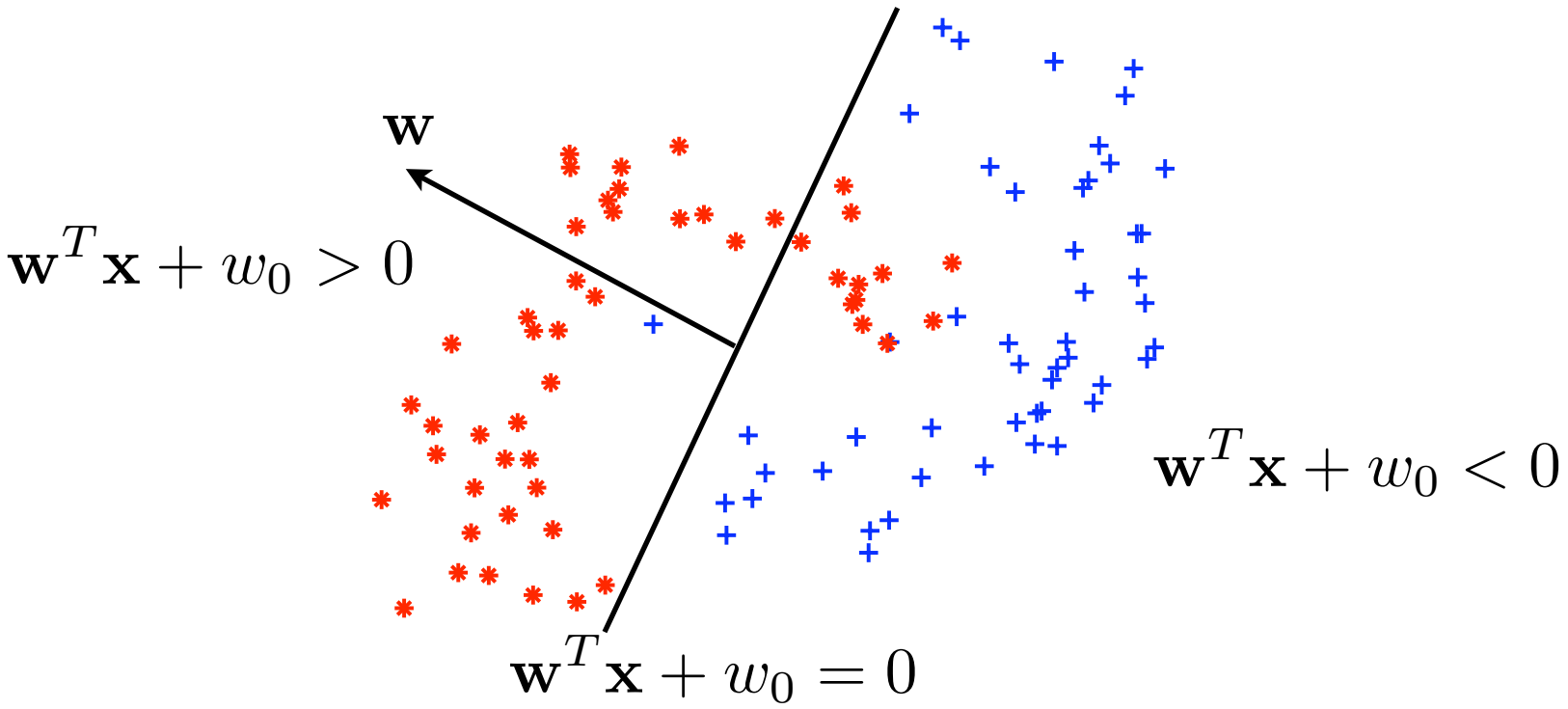
- What does $\mathbf{w}^T\mathbf{x} + w_0$ mean?
  Assume I have $\mathbf{w}^T\mathbf{x} + w_0 = 0$



$w_0 = ?$

TU Delft

Linear discriminant



- Classifier is a linear function of the features
- The classification depends if the weighted sum of the features is above or below 0

Incorporate the bias term

- Quite often you see $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} > 0$

  instead of $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 > 0$

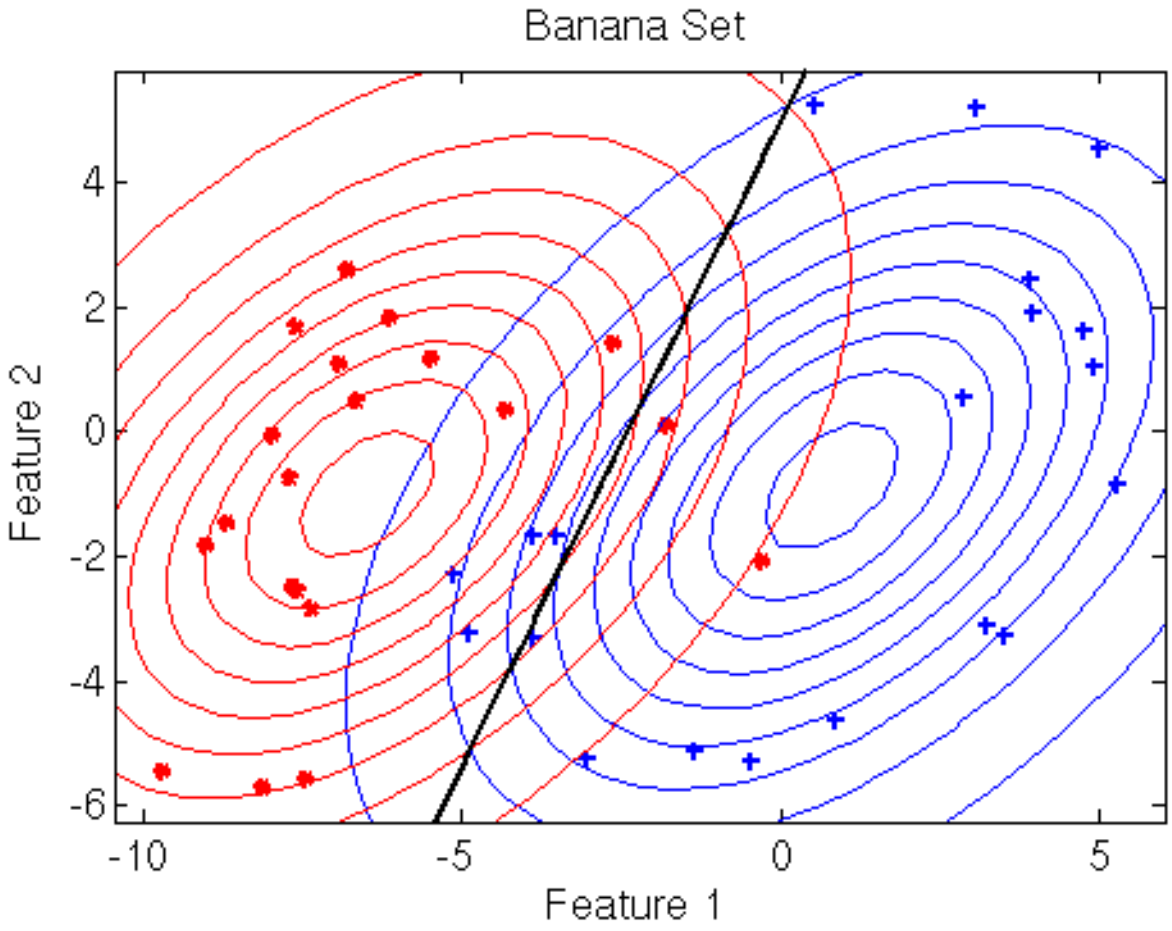
- No problem, if you (re-)define the feature vector as:

  (homogeneous coordinates)

- Then: $\tilde{\mathbf{x}} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$

$$g(\mathbf{x}) = [\mathbf{w}^T \; w_0] \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}$$

Linear Classifier on Banana Data



TU Delft

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 :

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

  i.e. the classes are circular.

- 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

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

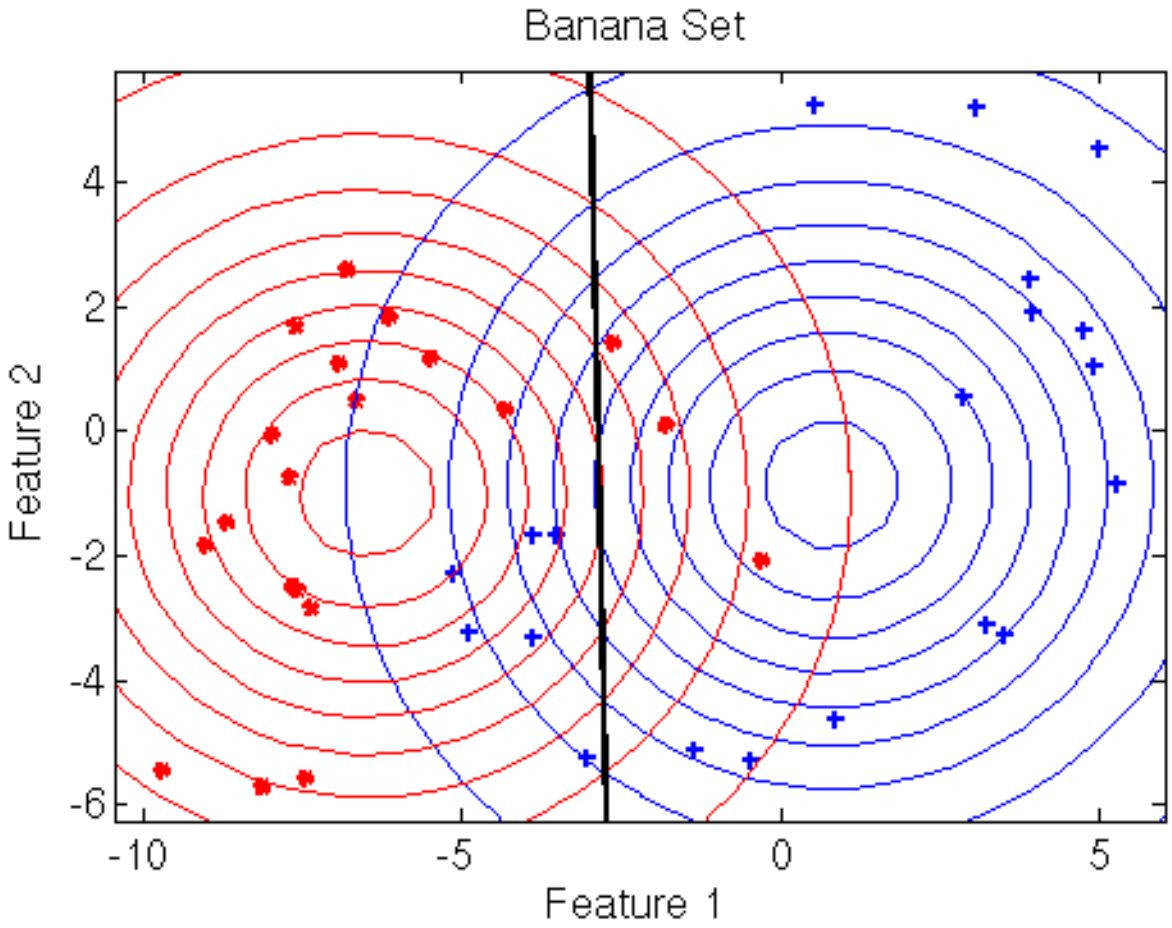
with

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

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

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

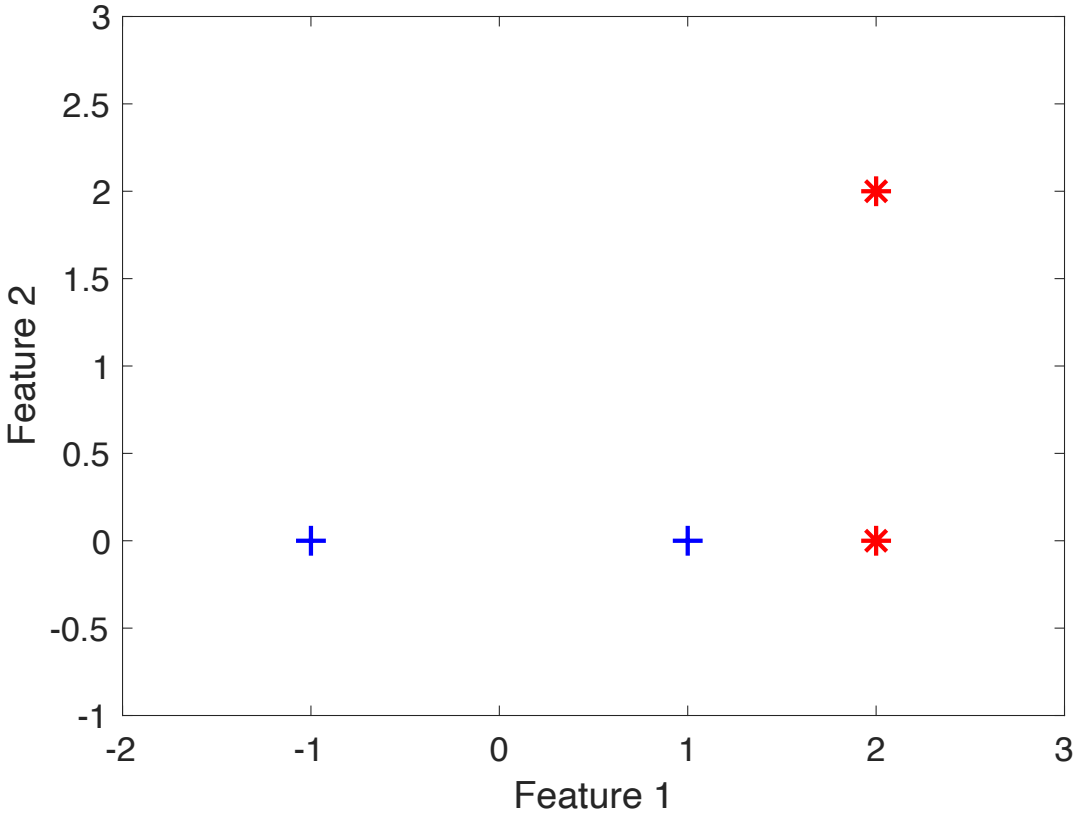
Nearest Mean on Banana Data



TU Delft

Nearest mean on simple data

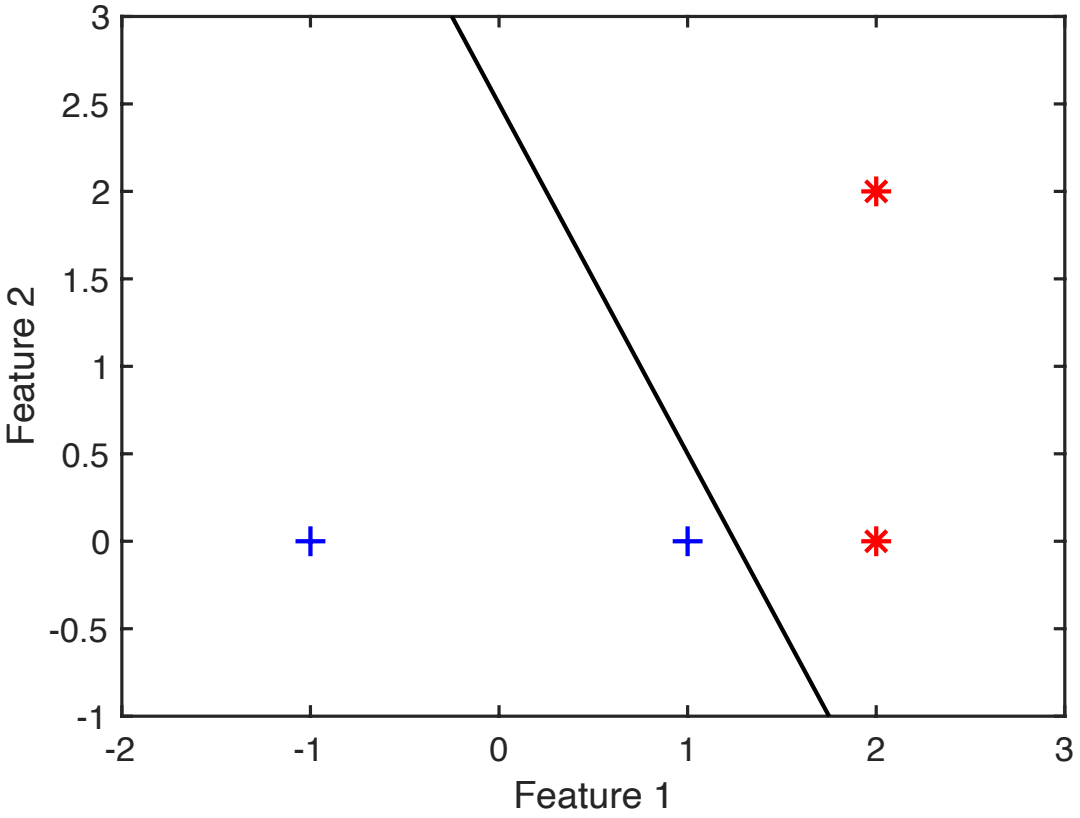
- How does the NMC look like for:



TU Delft

Nearest mean on simple data

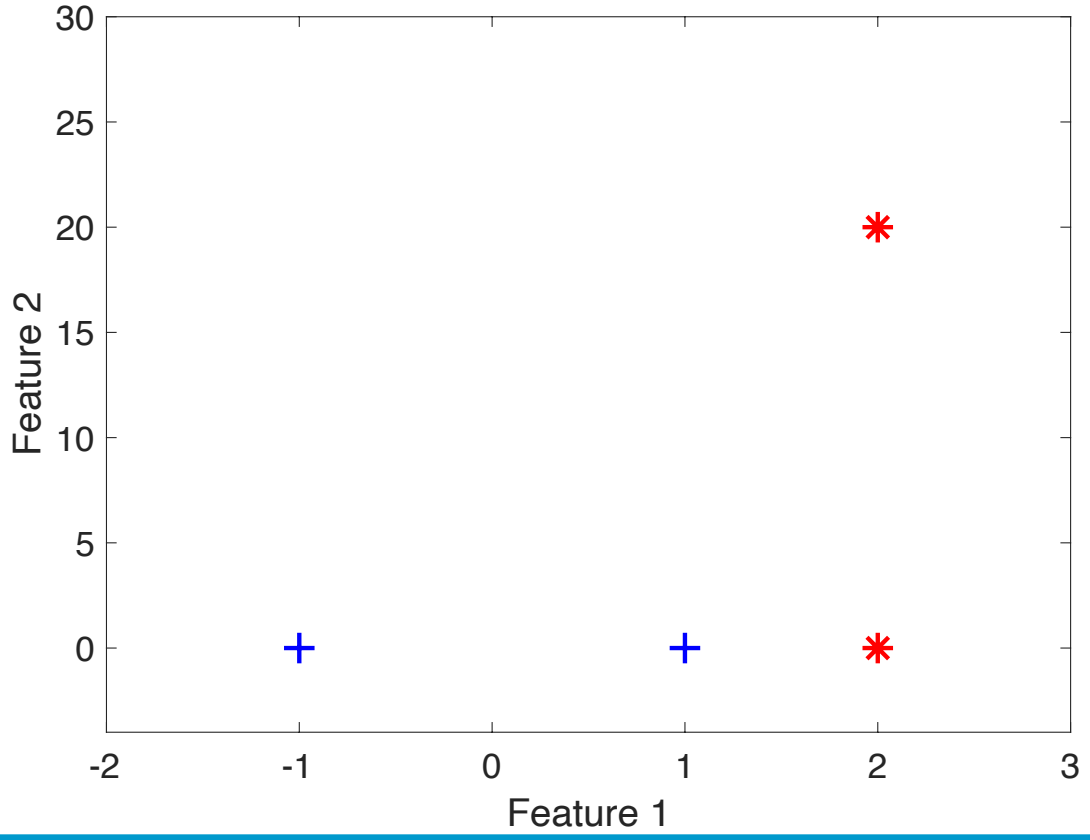
- How does the NMC look like for:



TU Delft

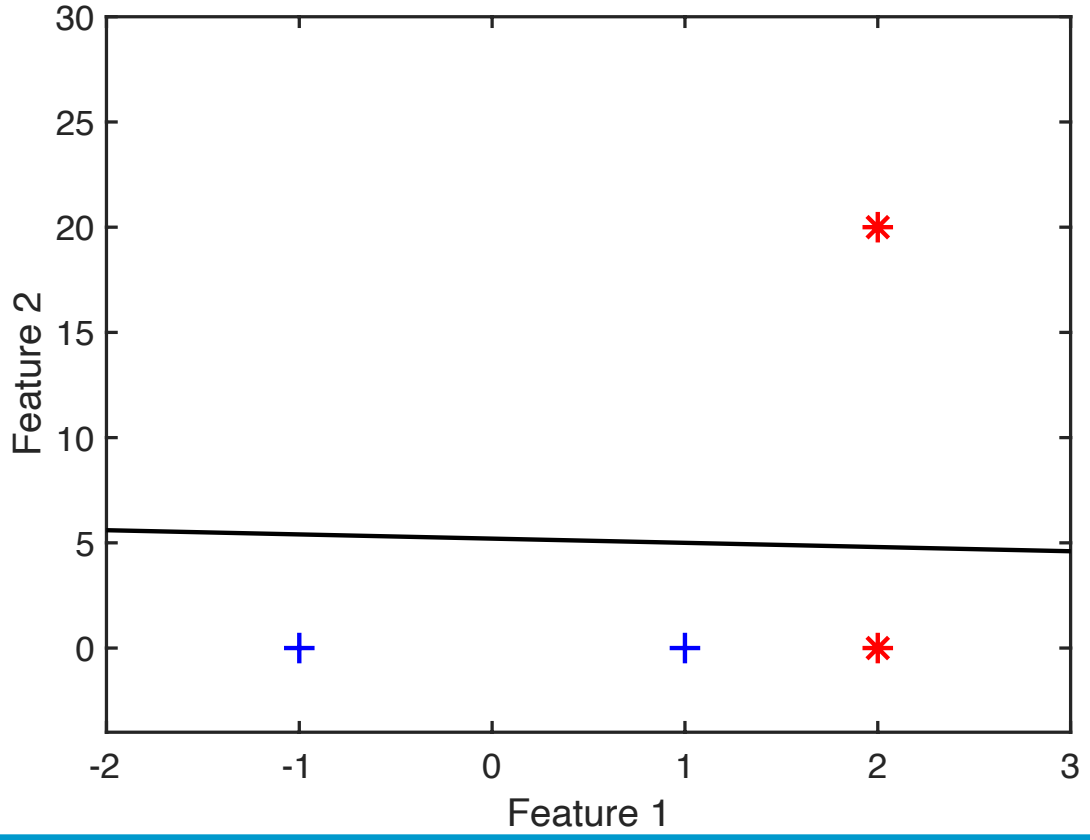
Nearest mean on simple data

- And now?



Nearest mean on simple data

- And now?



Feature scaling

- When a classifier depends on (euclidean) distances, scaling of the features matter
- Changing the scaling can improve/deteriorate the classifier: check the variances of all features
- If you don't know anything, rescale your features beforehand
- Fairly standard is zero-mean, unit-variance scaling:

$$\tilde{x} = \frac{x - \mu}{\sigma}$$

Concluding Remarks

- Using plug-in Bayes' rule with normal distribution for every class can give rise to different classifiers
  - Separate mean and covariance matrix per class gives the **quadratic classifier**
  - Separate mean, equal covariance matrix per class gives the **linear classifier**
  - Separate mean, identity covariance matrix per class gives the **nearest mean classifier**
- More flexible classifier needs more training data
- Simple classifiers still perform quite well in practice

Concluding Remarks

- This curse of dimensionality is really a nasty one
  - One of the major issues in the pattern recognition course is to learn ways to detect and understand it, and possibly deal with it

One may Wonder…

- Is ML estimation optimal for classification?
- So how many free parameters are there really for a linear classifier?
- When underlying class distributions are truly Gaussian, do our Gaussian-based classifiers result in Bayes optimal classifiers?

After this week you should be able to:

- explain how you obtain a classifier using a Gaussian (multivariate) distribution for each class
- implement a simple univariate classifier in Python
- explain what the 'curse of dimensionality' is

- explain the advantages and disadvantages are of the Quadratic classifier, the LDA and the nearest mean classifier
- identify when scaling of the features is important and how to cope with feature scaling