



# *CS545 - Machine Learning for Signal Processing*

# Decision Theory and Simple Classifiers

24 September 2023

# Today's lecture

- Decision theory
- Linear and quadratic classifiers
  - Gaussian models, the perceptron

# Classification

- In detection we had one template
  - Decision: was it there?
    - Or rather, how much of it is there?
- In classification we have many templates
  - Decision: Do any of them appear?

# Using training data

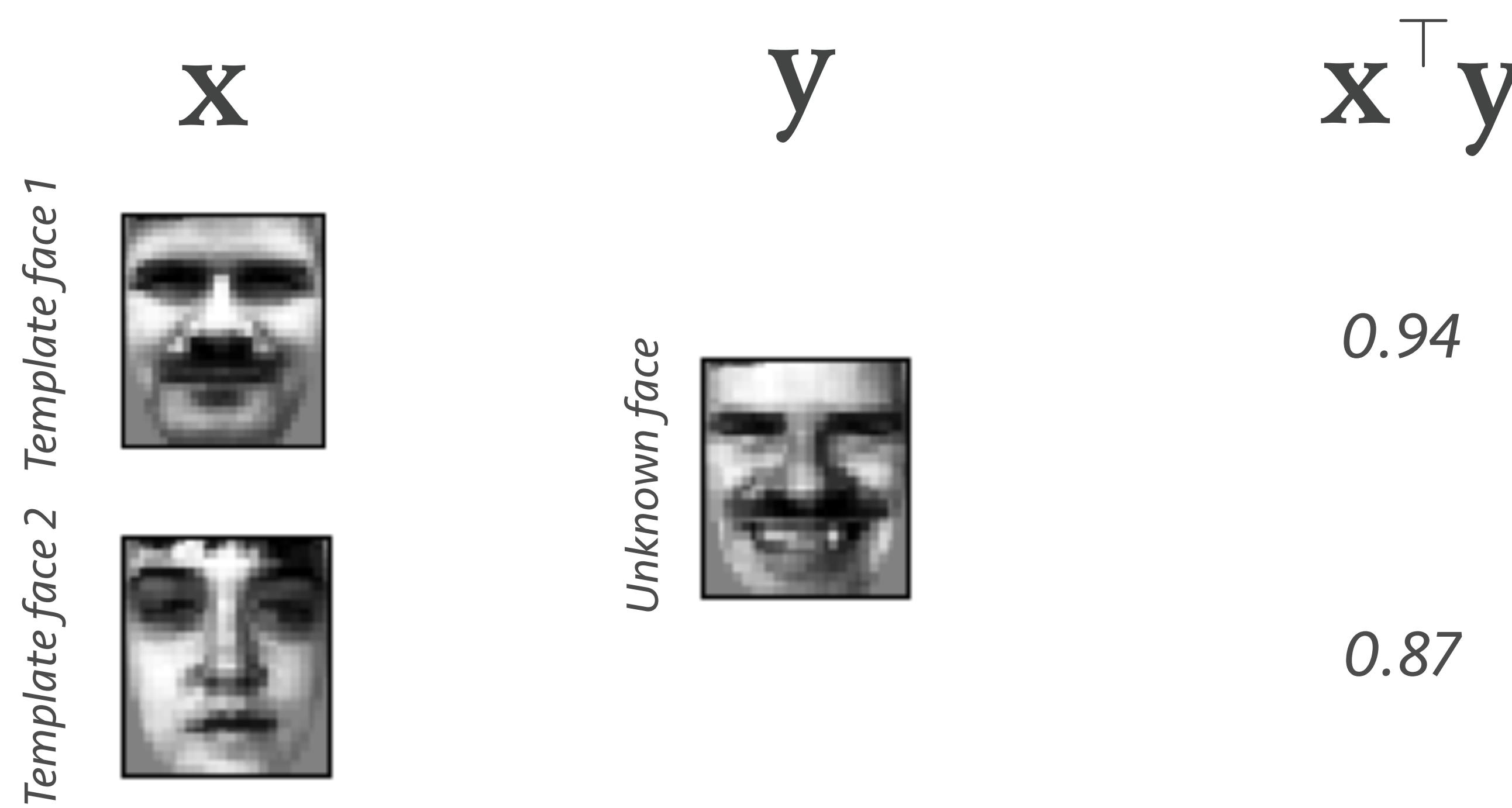
- We provide multiple examples of what we are looking for
  - Training data
- We make models of each class
  - Training process
- We match all inputs to a learned model
  - Classification

# Making a decision

- Face data example
  - Dot products relate to a likelihood of match
  - This is linked to a probability
- Having a class probability for each face,  
how do we make a decision?

# Motivating example

- Which do we pick?



# Motivating example

- Template 1 is more “likely”

$$\begin{array}{ccc} \mathbf{x} & \mathbf{y} & \mathbf{x}^\top \mathbf{y} \\ \text{Template face 2} & \text{Template face 1} \\ \text{Unknown face} & \end{array}$$

*Likelihood  $\propto 0.94$*

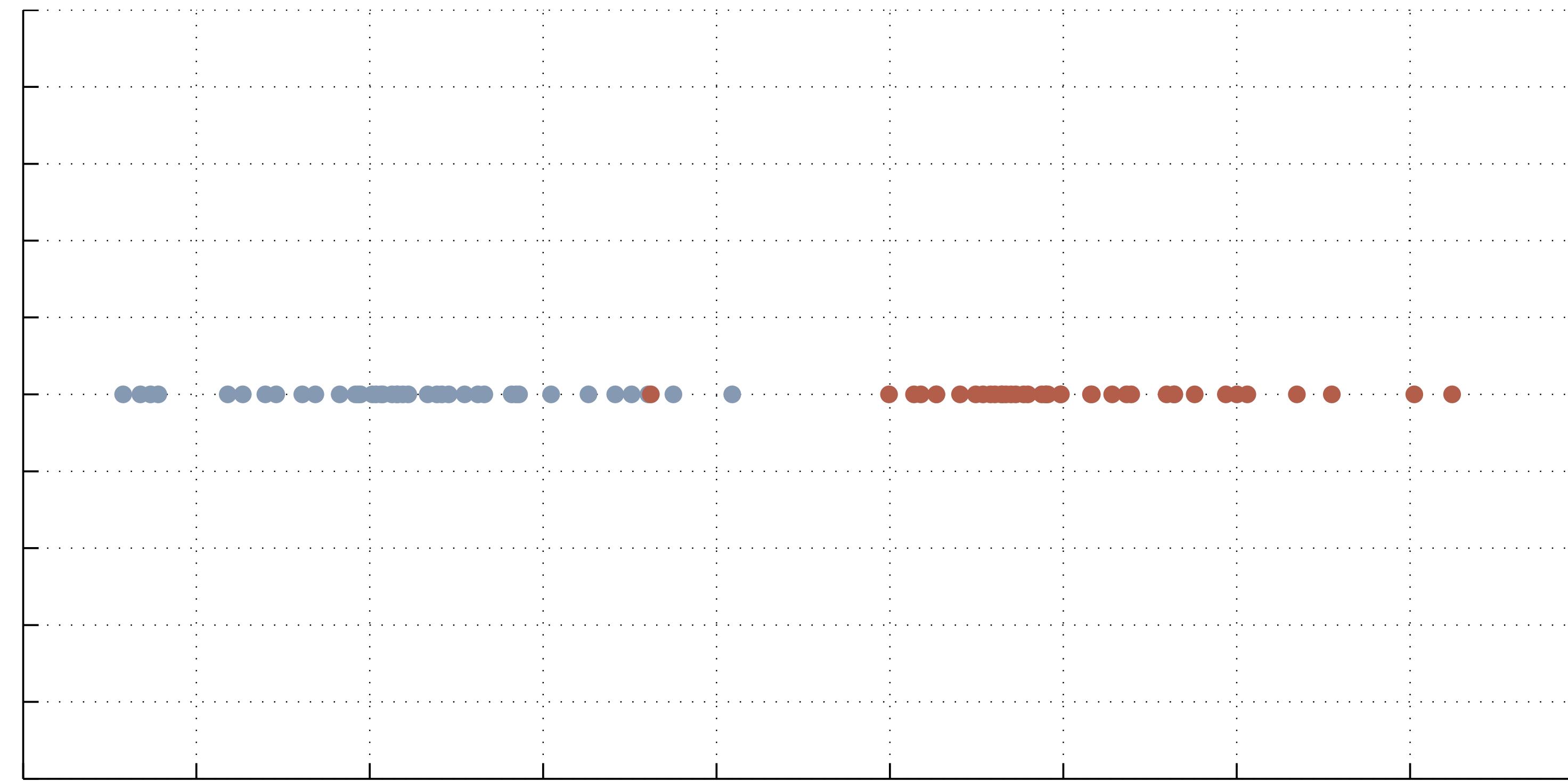
*Likelihood  $\propto 0.87$*

# How the decision is made

- In simple cases the answer is intuitive
- To get a complete picture we need to probe a bit deeper
- Bayesian decision theory

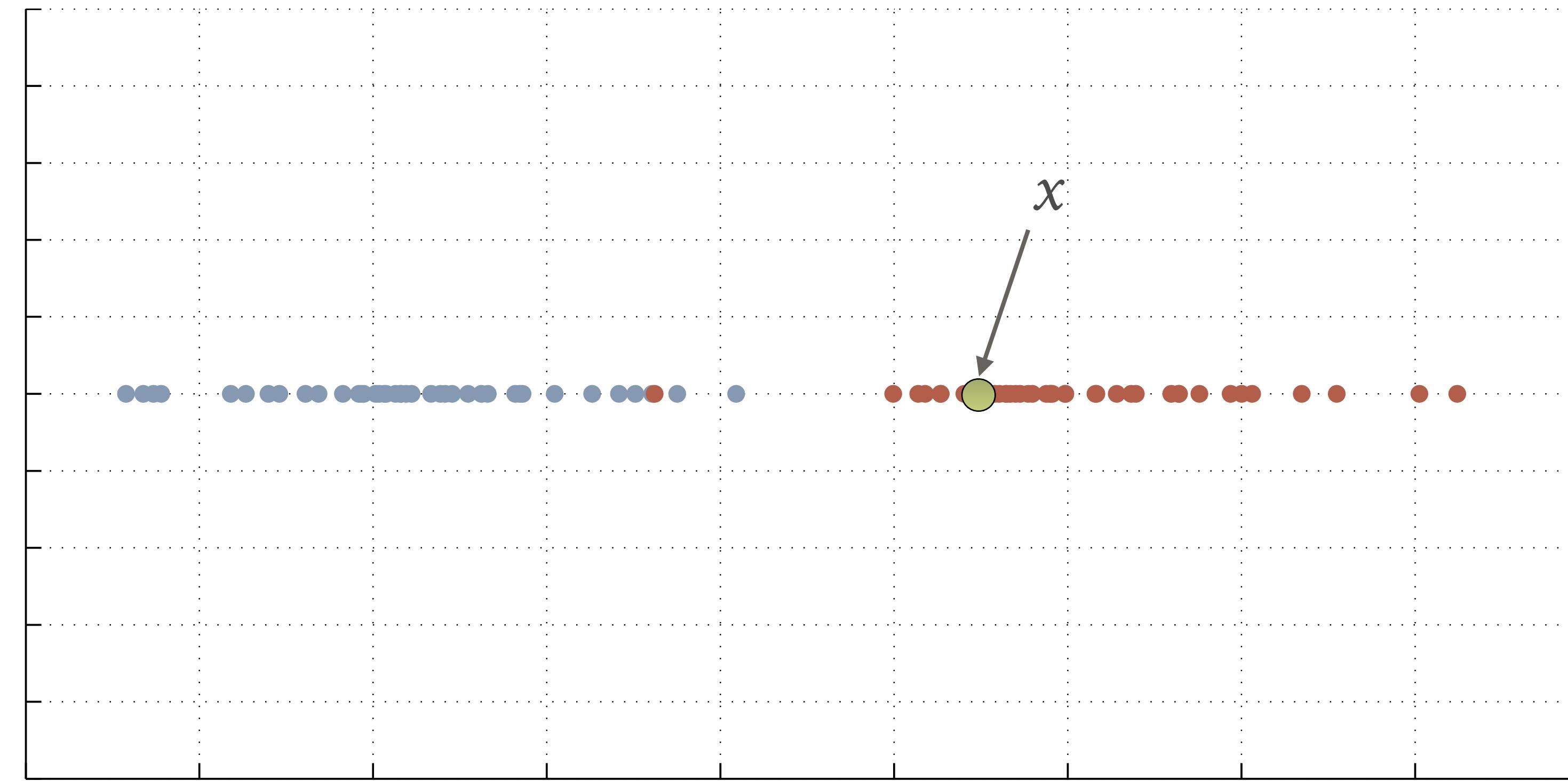
# Starting simple

- Two-class case,  $\omega_1$  and  $\omega_2$ 
  - 1-dimensional data



# Starting simple

- Given a sample  $x$ , is it  $\omega_1$  or  $\omega_2$ ?
  - i.e.  $P(\omega_i | x) = ?$



# Getting the answer

- The *class posterior probability* is:

$$P(\omega_i | x) = \frac{P(x | \omega_i) P(\omega_i)}{P(x)}$$

*Likelihood*      *Priors*  
*Evidence*

- To find the answer we need to fill in the terms in the right-hand-side

# Filling the unknowns

- Class priors

- How much of each class?

$$P(\omega_i) \approx N_i / N$$

- Class likelihood:  $P(x | \omega_i)$

- Requires that we have a model of each  $\omega_i$ 
    - E.g.  $\omega_i$  can be a Gaussian distributed so that:

$$P(x | \omega_i) = \mathcal{N}\left(x | \mu_{\omega_i}, \Sigma_{\omega_i}\right)$$

# Filling the unknowns

- Evidence:

$$P(x) = P(x | \omega_1)P(\omega_1) + P(x | \omega_2)P(\omega_2)$$

- We now can estimate the class posteriors:

$$P(\omega_1 | x), P(\omega_2 | x)$$

# Making the decision

- Bayes classification rule:

If  $P(\omega_1 | x) > P(\omega_2 | x)$  then  $x$  belongs to class  $\omega_1$

If  $P(\omega_1 | x) < P(\omega_2 | x)$  then  $x$  belongs to class  $\omega_2$

- Easier version:

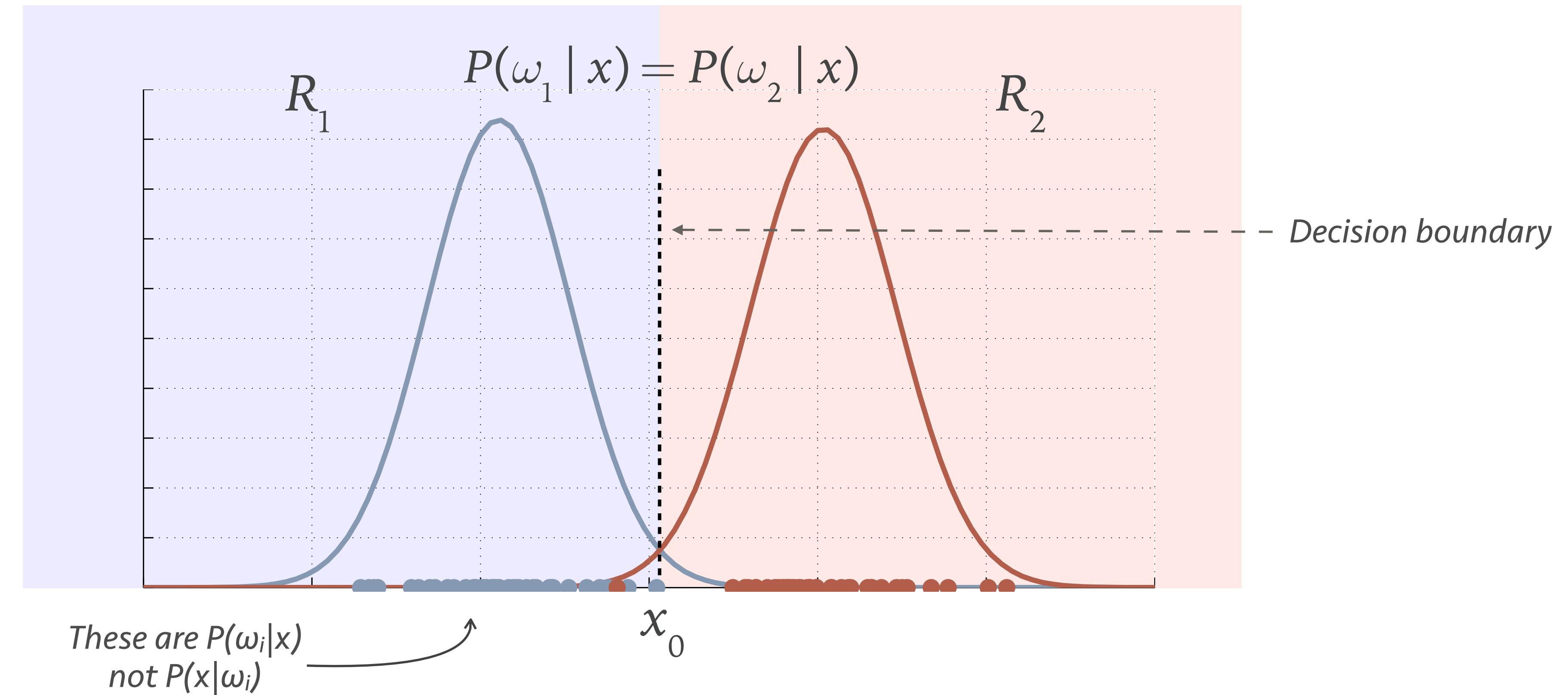
$$P(x | \omega_1)P(\omega_1) \geq P(x | \omega_2)P(\omega_2)$$

- Equiprobable class version:

$$P(x | \omega_1) \geq P(x | \omega_2)$$

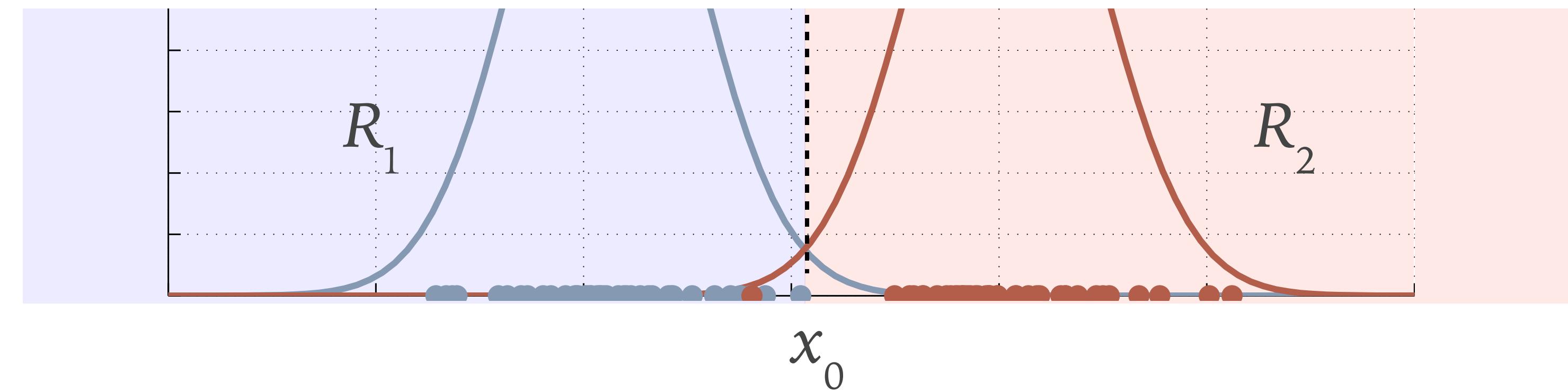
# Visualizing the decision

- Assume a Gaussian model for the classes
  - Likelihood:  $P(x | \omega_i) = \mathcal{N}(x | \mu_i, \sigma_i)$



# Errors in classification

- We can't win all the time though
  - Some inputs will be misclassified

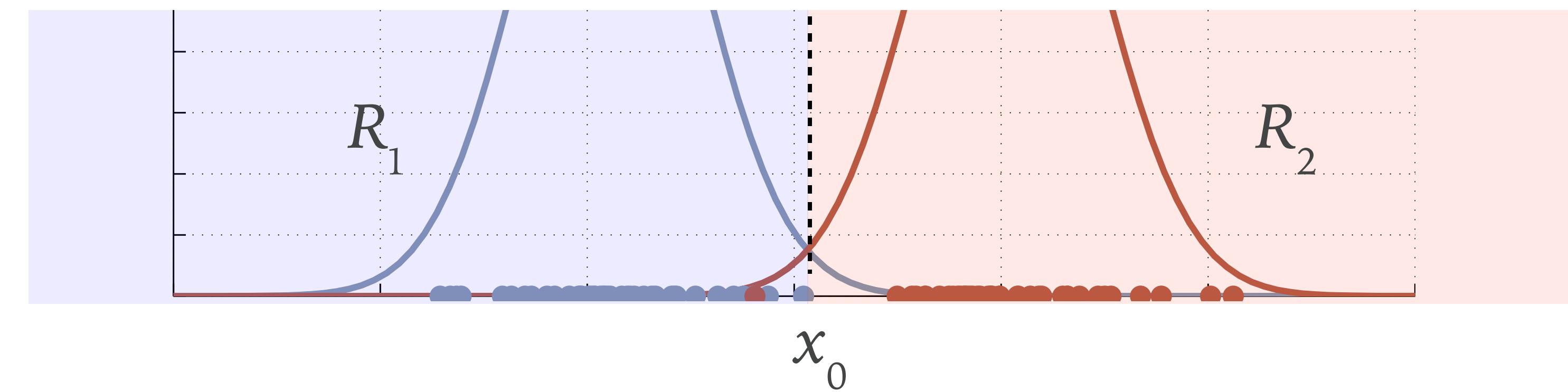


- What are the errors?

$$\varepsilon_2 = \int_{-\infty}^{x_0} P(x | \omega_2) P(\omega_2) dx, \quad \varepsilon_1 = \int_{x_0}^{\infty} P(x | \omega_1) P(\omega_1) dx$$

# Minimizing misclassifications

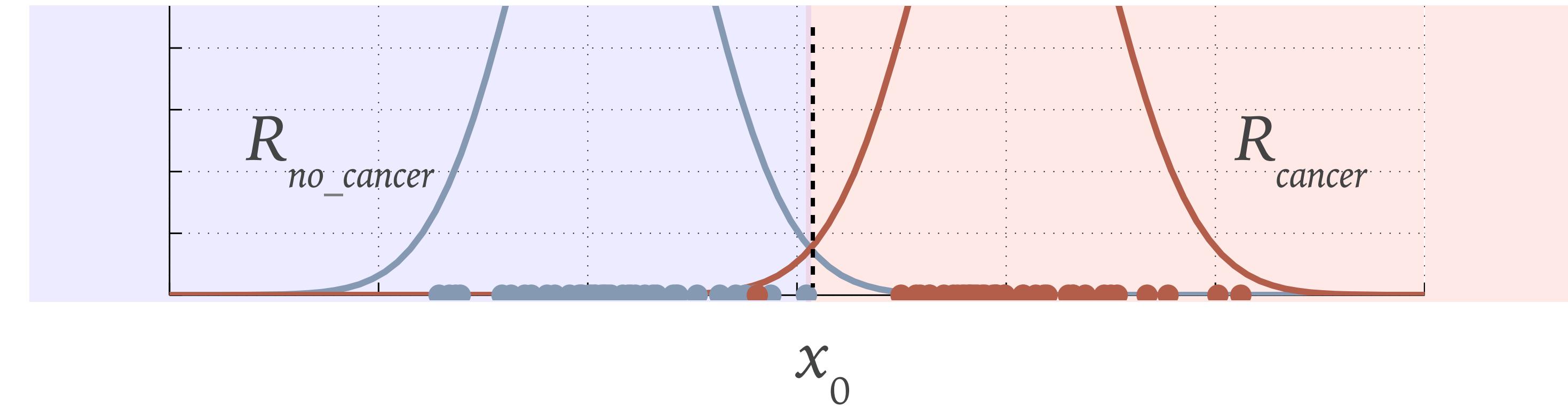
- The Bayes classification rule minimizes these potential misclassifications



- Can you do any better by moving the line?

# Minimizing risk

- Not all errors are equal!
  - e.g. medical diagnoses



- Misclassification can be tolerable, or not, depending on the assumed risks

# Adding risks

- Implement a *loss* factor for each decision to compute *risk* for each class

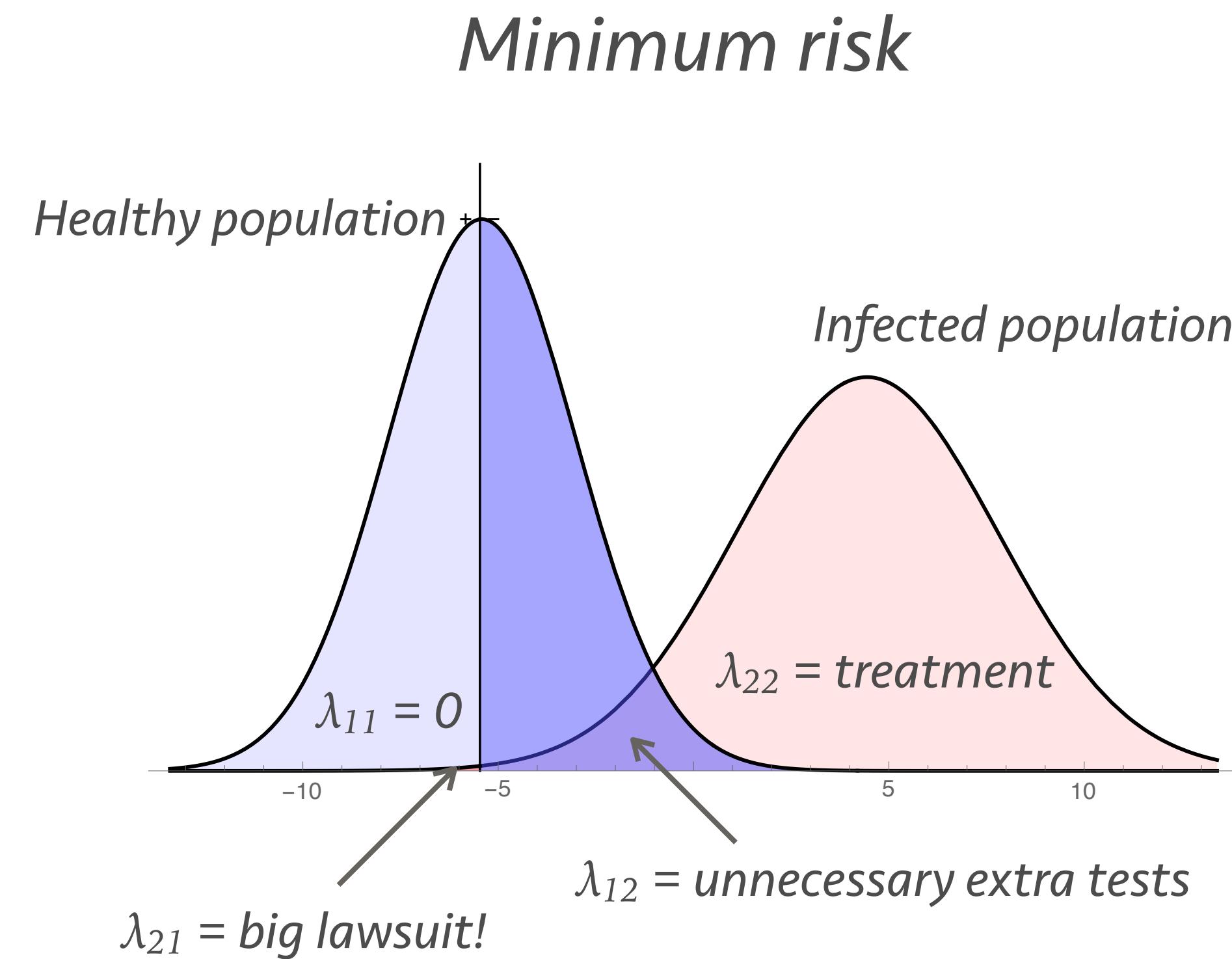
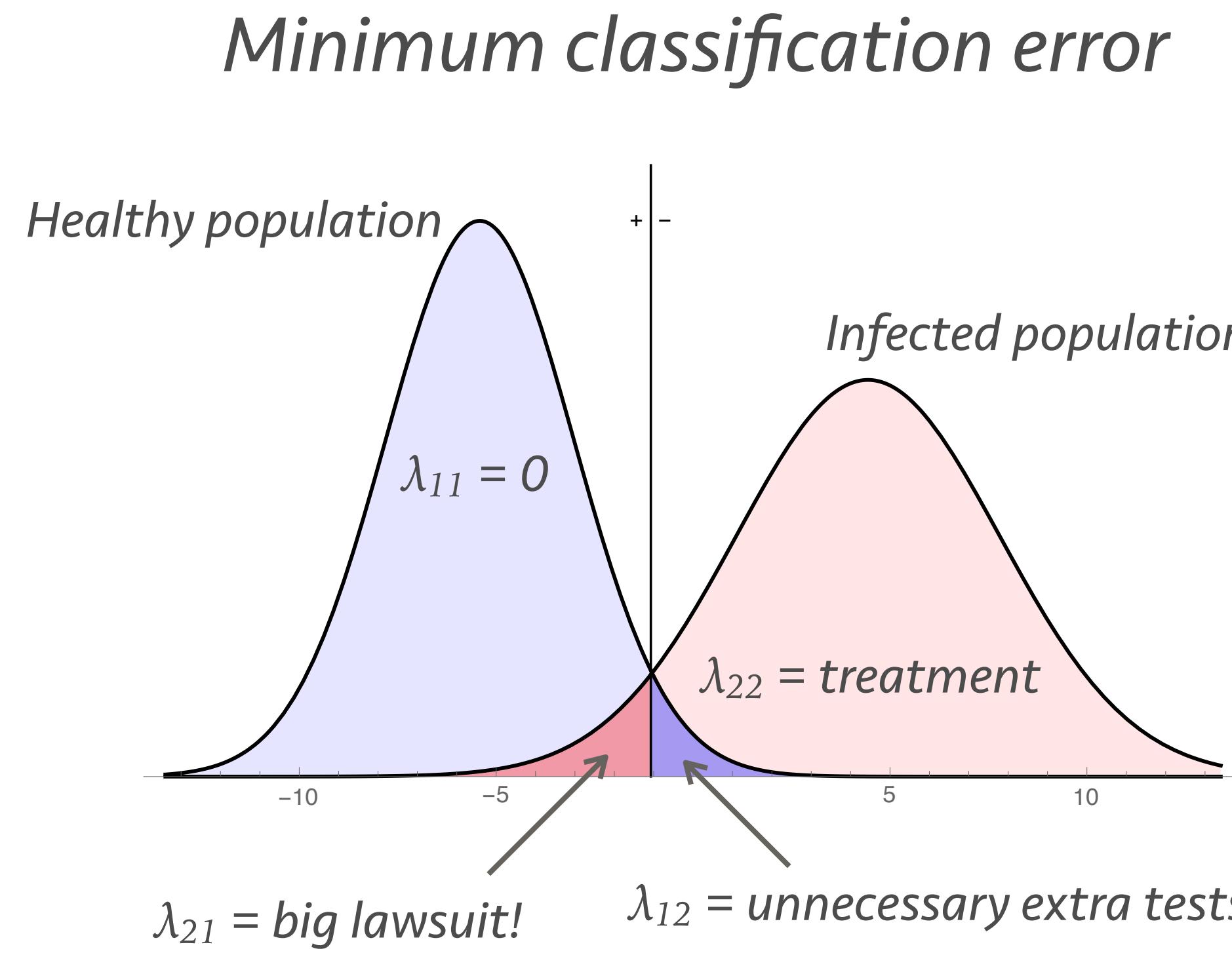
$$r_j = \sum_i \lambda_{j,i} \int_{R_i} P(x|\omega_j) dx$$

- The  $\lambda$ 's specify how costly each decision is
  - $\lambda_{j,i}$  is cost for samples in region  $i$  while being of class  $j$
- Choose regions to minimize overall risk

$$r = \sum_j r_j P(\omega_j) = \sum_i \int_{R_i} \sum_j \lambda_{j,i} P(x|\omega_j) P(\omega_j) dx$$

# Example case

- A deadly disease detector
  - Optimizing the cost of each decision; note that  $\lambda_{2,1} \gg \lambda_{1,2}$



# New decision process

- Assign  $x$  to  $\omega_1$  if

$$(\lambda_{2,1} - \lambda_{2,2})P(x|\omega_2)P(\omega_2) < (\lambda_{1,2} - \lambda_{1,1})P(x|\omega_1)P(\omega_1)$$

*Cost of being  $\omega_1$  at either region*

- and vice-versa
- Or using the *likelihood ratio test*:

$$x \in \omega_1 \text{ if } \frac{P(x|\omega_1)}{P(x|\omega_2)} > \frac{P(\omega_2)(\lambda_{2,1} - \lambda_{2,2})}{P(\omega_1)(\lambda_{1,2} - \lambda_{1,1})}$$

- and vice-versa

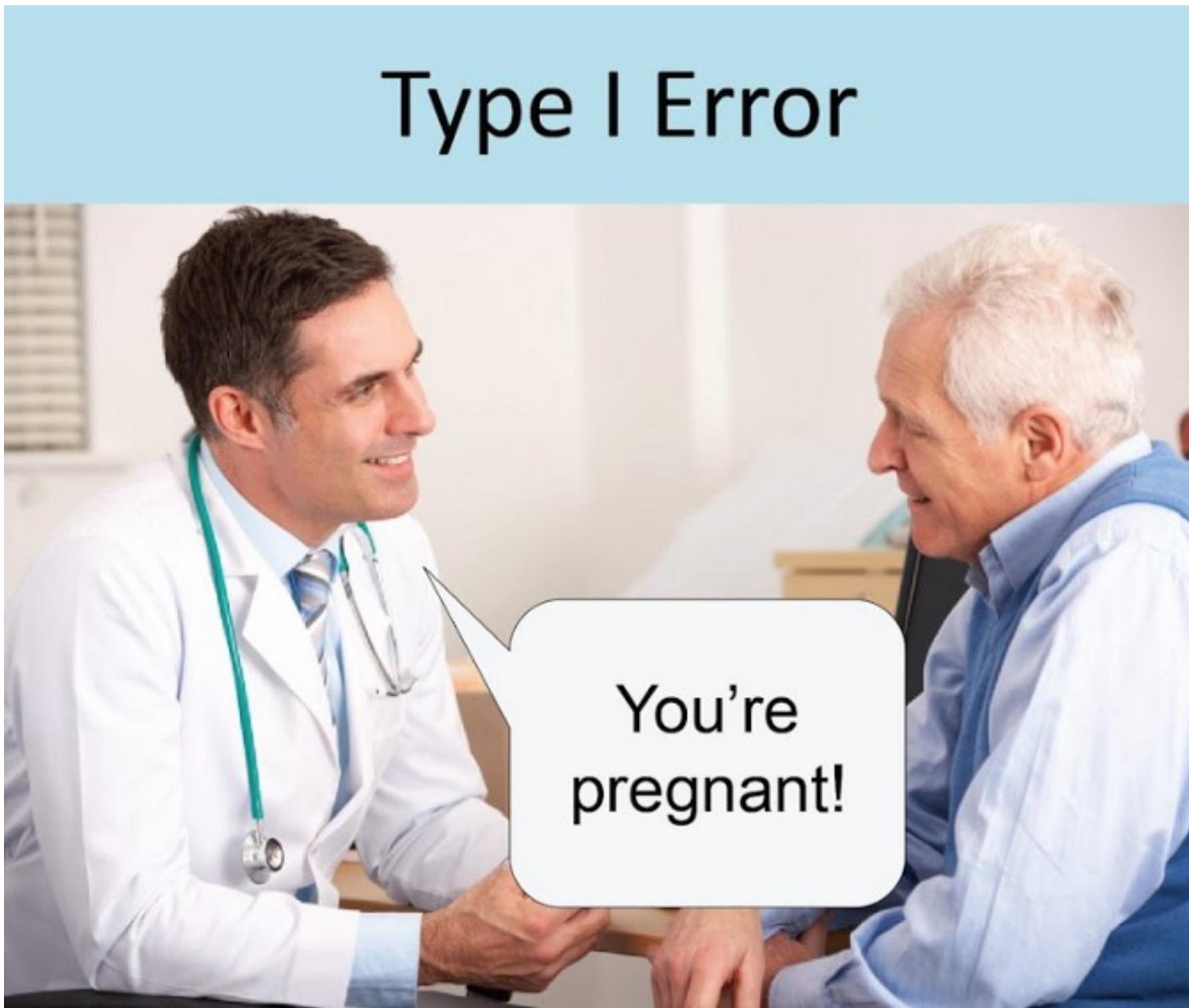
# True/False - Positives/Negatives

- Naming the outcomes

Looking for $\omega_1$	$x$ is $\omega_1$	$x$ is $\omega_2$
$x$ classified as $\omega_1$	<b>True positive</b>	<b>False positive</b>
$x$ classified as $\omega_2$	<b>False negative</b>	<b>True negative</b>

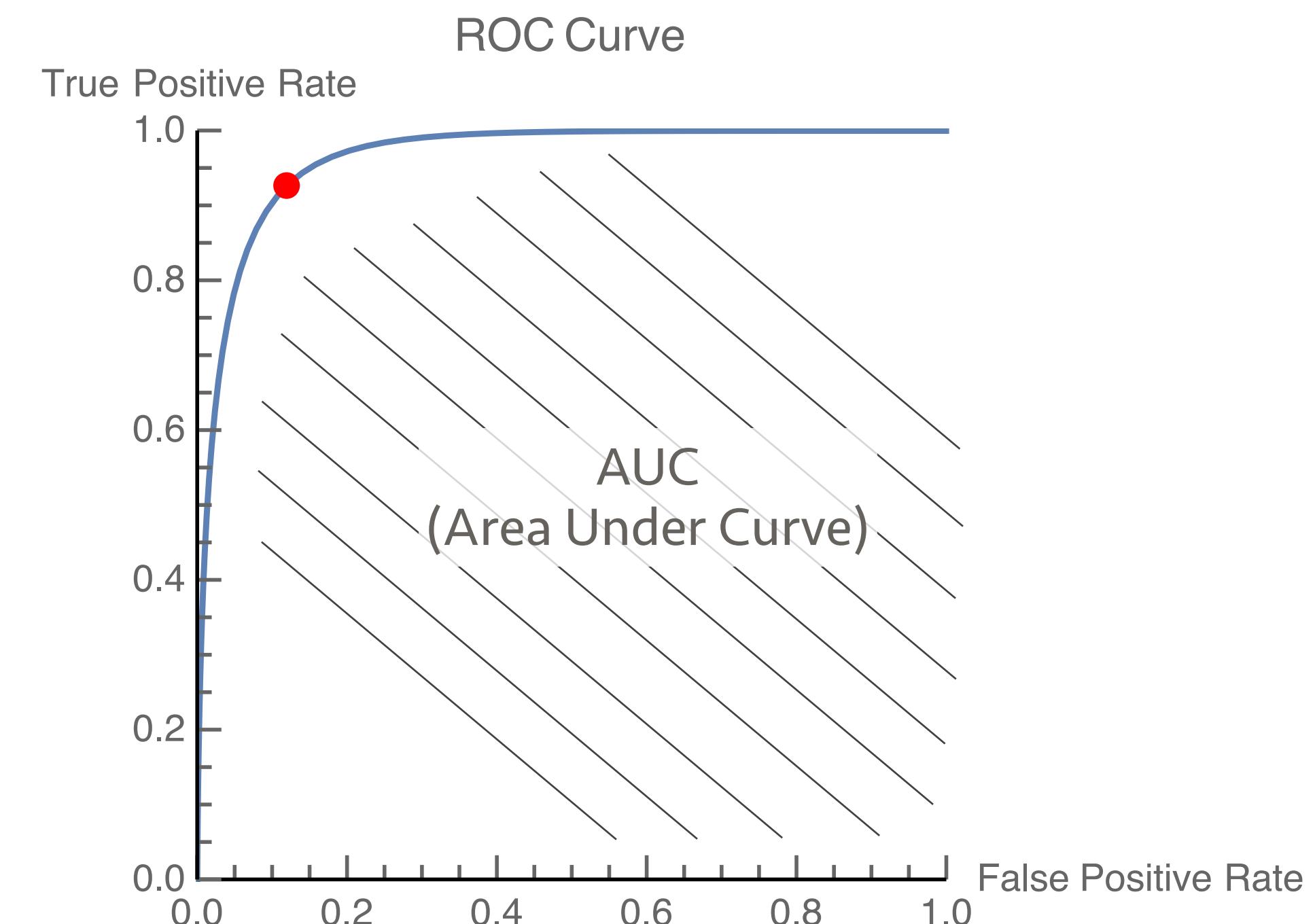
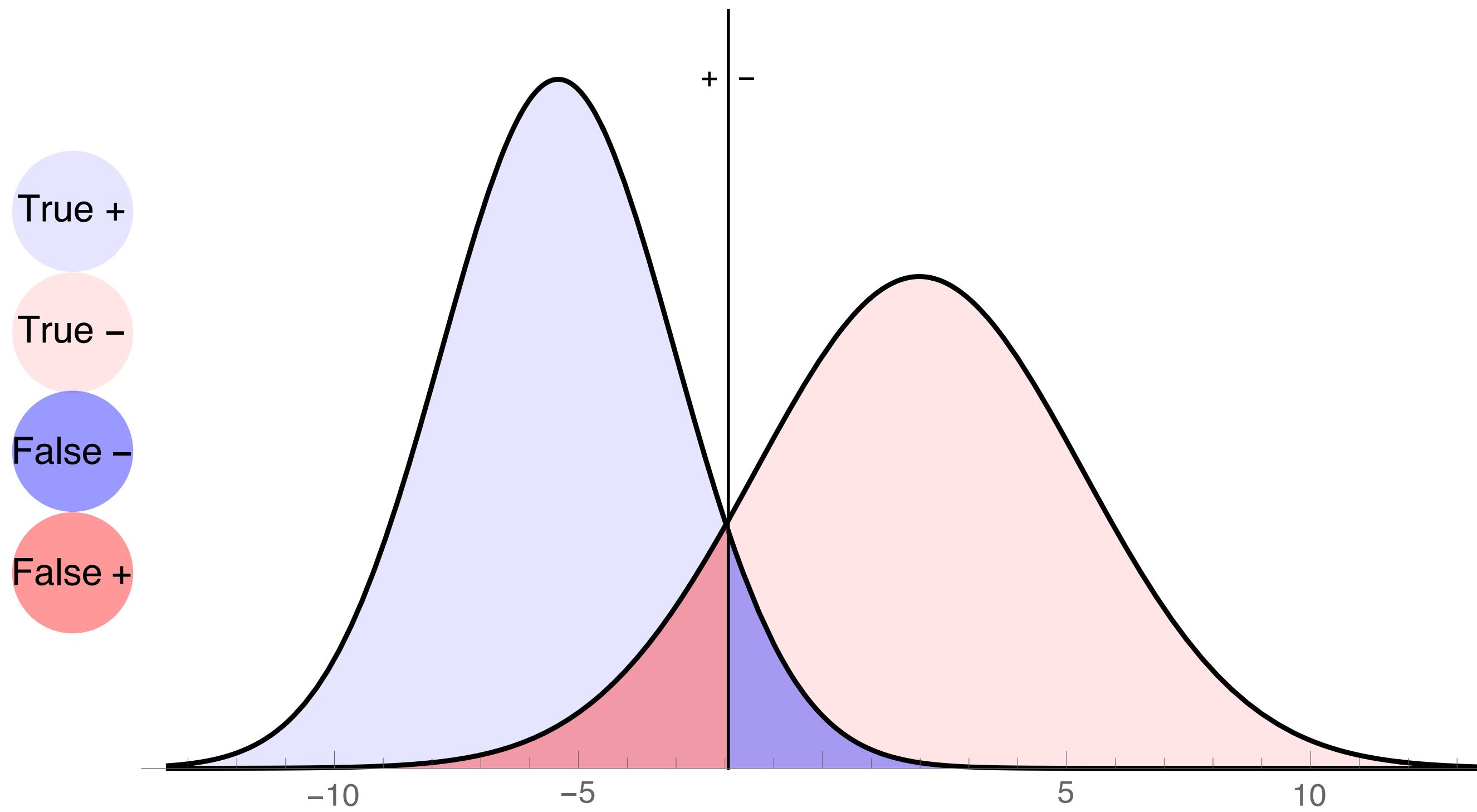
- False positive / False alarm / Type I error
- False negative / Miss / Type II error

# Simple way to remember



# Receiver Operating Characteristic

- Visualizing how well we can expect to do



# Classifying Gaussian data

- Remember that we need the class likelihood to make a decision
  - For now let's assume that:

$$P(x | \omega_i) = \mathcal{N}(x | \mu_i, \sigma_i)$$

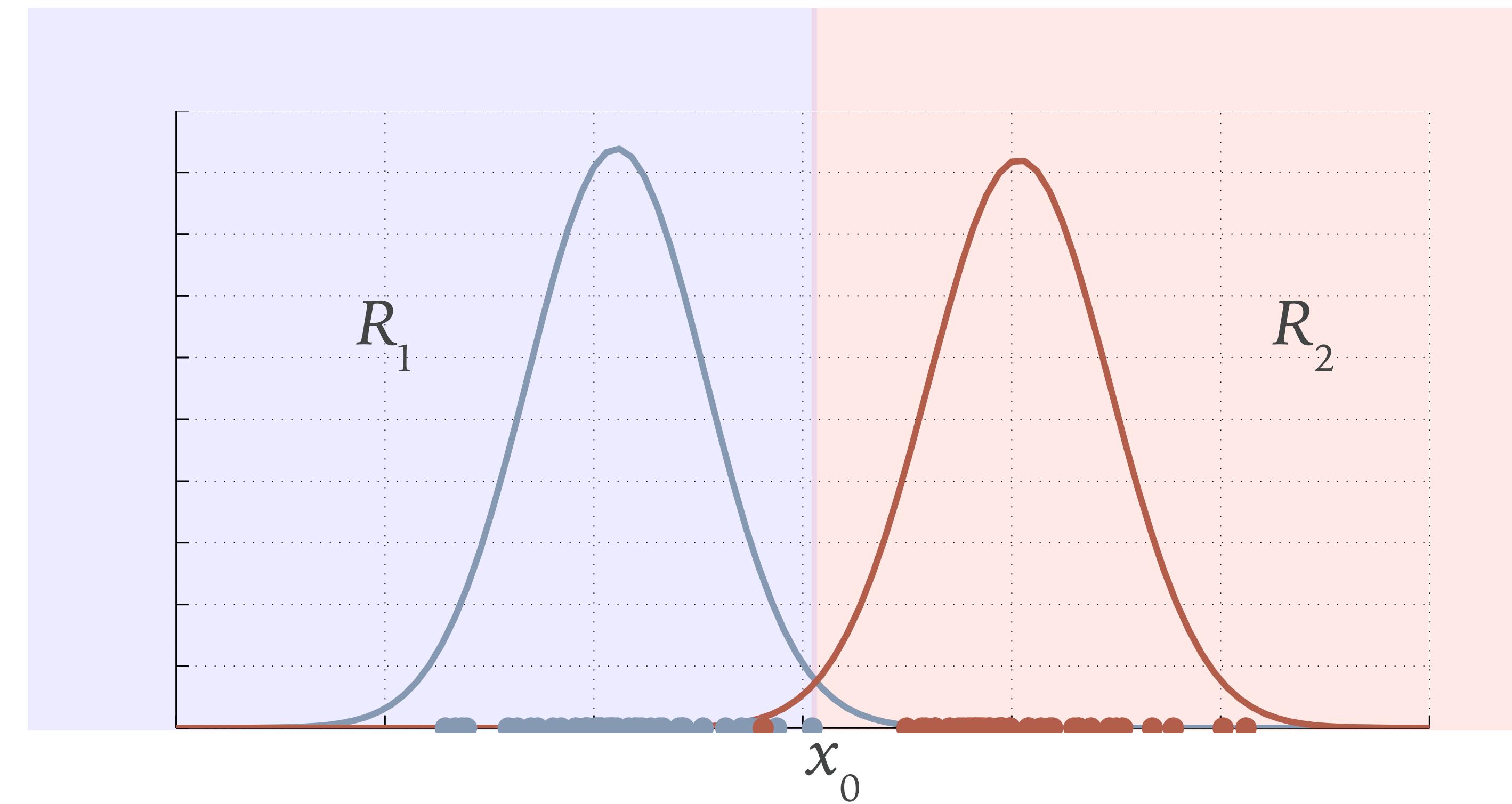
- i.e. that the input data is Gaussian distributed

# Overall methodology

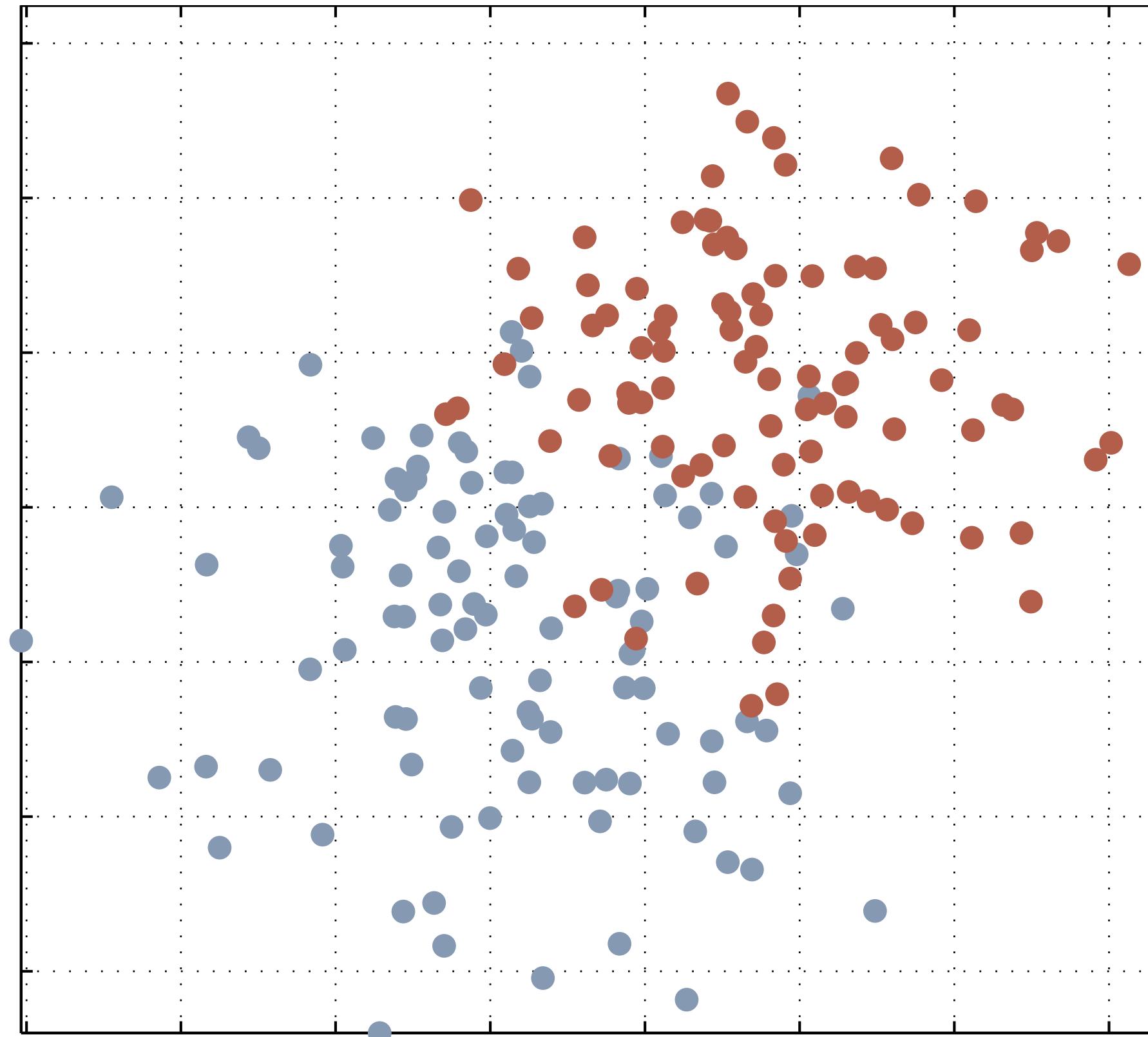
- Obtain training data
- Fit a Gaussian model to each class
  - Perform parameter estimation for mean, variance and class priors
- Define decision regions based on models and any given constraints

# 1D example

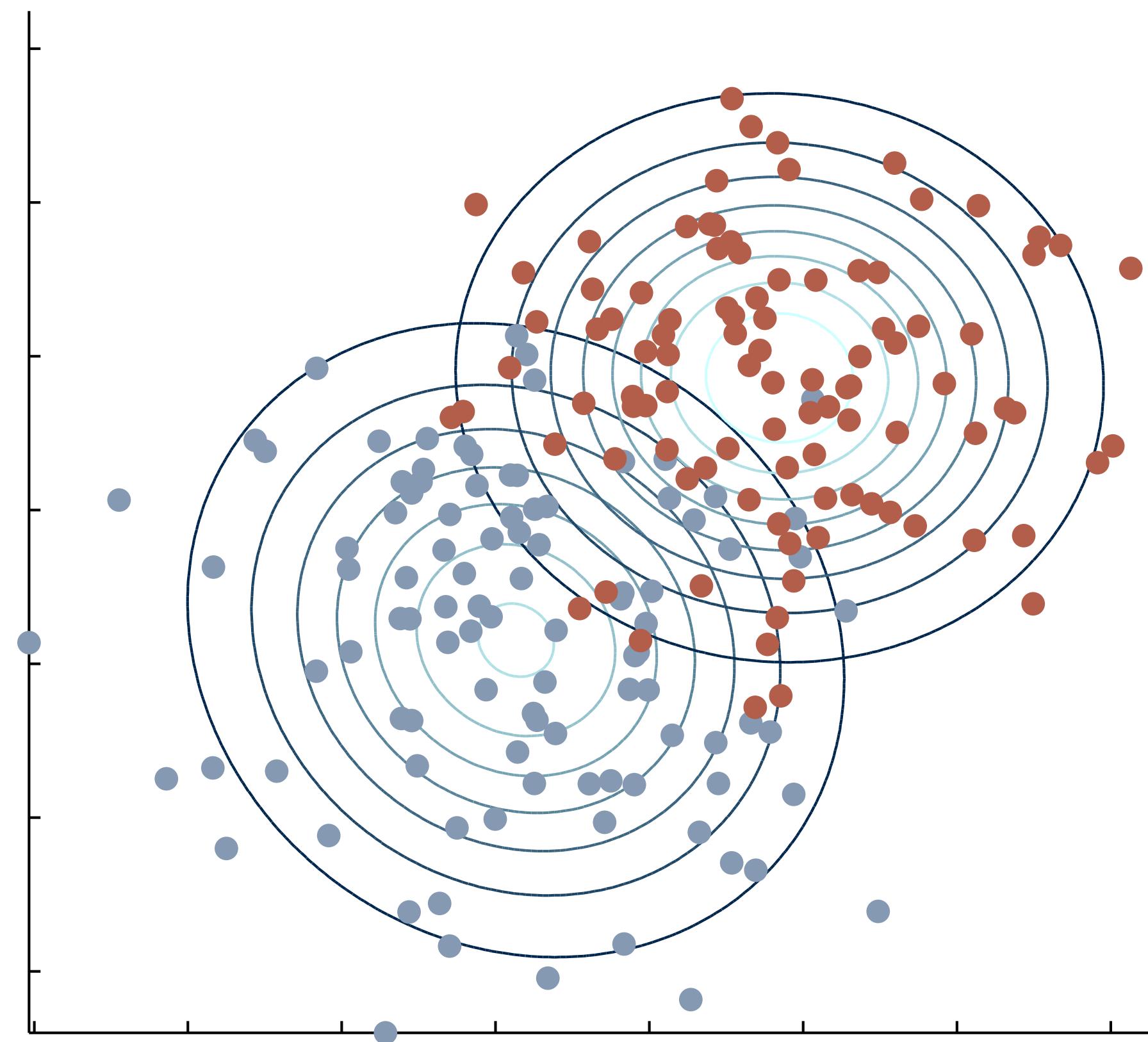
- The decision boundary will always be a point separating the two class regions



# 2D example



# 2D example fitted Gaussians



# Gaussian decision boundaries

- The decision boundary is defined as:

$$P(\mathbf{x} | \omega_1)P(\omega_1) = P(\mathbf{x} | \omega_2)P(\omega_2)$$

- Replace likelihoods with Gaussians and solve to find what the boundary looks like

# Discriminant functions

- Define a set of functions  $g_i(\mathbf{x})$  so that:

classify  $\mathbf{x}$  in  $\omega_i$  if  $g_i(\mathbf{x}) > g_j(\mathbf{x}), \forall i \neq j$

- Decision boundaries are now defined as parts where:

$$G_{ij}(\mathbf{x}) \equiv \left( g_i(\mathbf{x}) = g_j(\mathbf{x}) \right)$$

*Decision boundary*      *Discriminant functions*

# Discriminant functions for Gaussians

- We remove the exponentiation:

$$\begin{aligned}g_i(\mathbf{x}) &= \log(P(\mathbf{x} | \omega_i)P(\omega_i)) = \log P(\mathbf{x} | \omega_i) + \log P(\omega_i) \\&= -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_i)^\top \cdot \boldsymbol{\Sigma}_i^{-1} \cdot (\mathbf{x} - \boldsymbol{\mu}_i) + \log P(\omega_i) + C_i \\&= \frac{1}{2} \left[ -\mathbf{x}^\top \cdot \boldsymbol{\Sigma}_i^{-1} \cdot \mathbf{x} + \mathbf{x}^\top \cdot \boldsymbol{\Sigma}_i^{-1} \cdot \boldsymbol{\mu}_i - \boldsymbol{\mu}_i^\top \cdot \boldsymbol{\Sigma}_i^{-1} \cdot \boldsymbol{\mu}_i + \boldsymbol{\mu}_i^\top \cdot \boldsymbol{\Sigma}_i^{-1} \cdot \mathbf{x} \right] \\&\quad + \log P(\omega_i) + C_i\end{aligned}$$

- The decision boundaries  $g_i(\mathbf{x}) = g_j(\mathbf{x})$  will be *quadratics*
  - Since  $g_i(\mathbf{x})$  has a quadratic form

# Back to the data

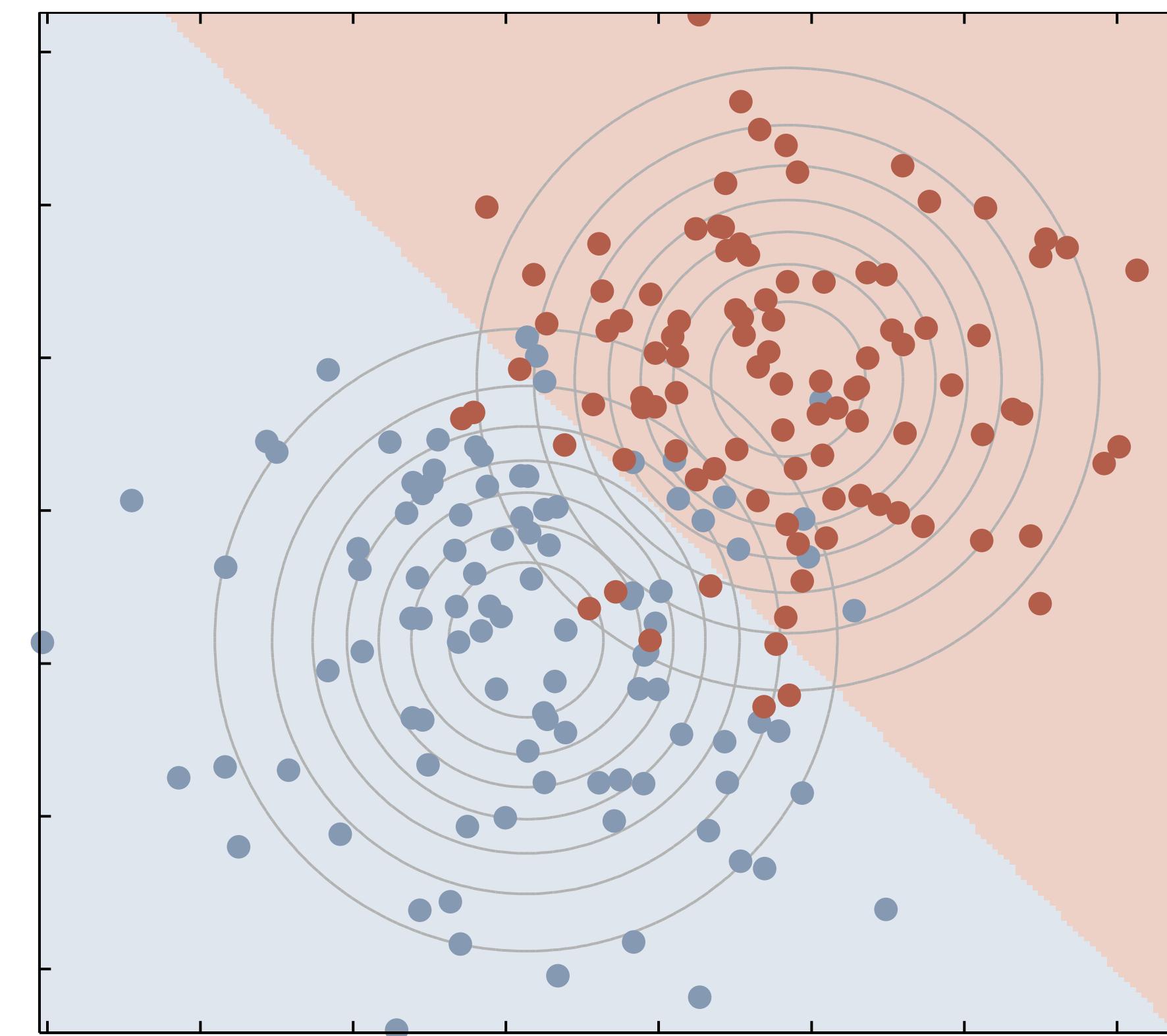
- $\Sigma_i = \sigma^2 \mathbf{I}$  produces line boundaries

- Discriminant:

$$g_i(\mathbf{x}) = \mathbf{w}_i^\top \cdot \mathbf{x} + b$$

$$\mathbf{w}_i = \boldsymbol{\mu}_i / \sigma^2$$

$$b = -\frac{\boldsymbol{\mu}_i^\top \cdot \boldsymbol{\mu}_i}{2\sigma^2} + \log P(\omega_i)$$



# Quadratic boundaries

- Arbitrary covariance matrices can produce more elaborate boundaries

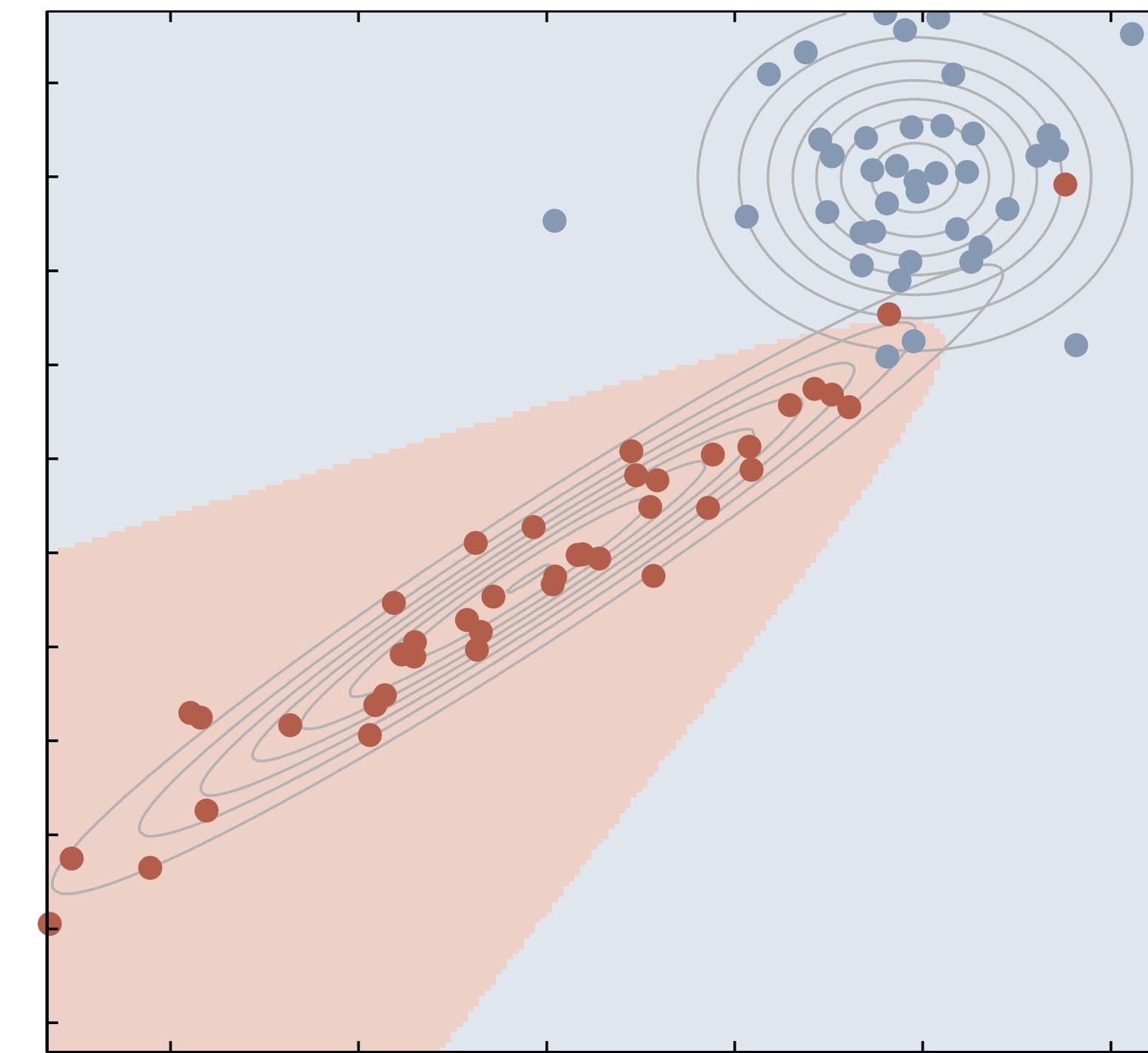
$$g_i(\mathbf{x}) = \mathbf{x}^\top \cdot \mathbf{W}_i \cdot \mathbf{x} + \mathbf{w}_i^\top \cdot \mathbf{x} + w_i$$

$$\mathbf{W}_i = -\frac{1}{2} \Sigma_i^{-1}$$

$$\mathbf{w}_i = \Sigma_i^{-1} \cdot \boldsymbol{\mu}_i$$

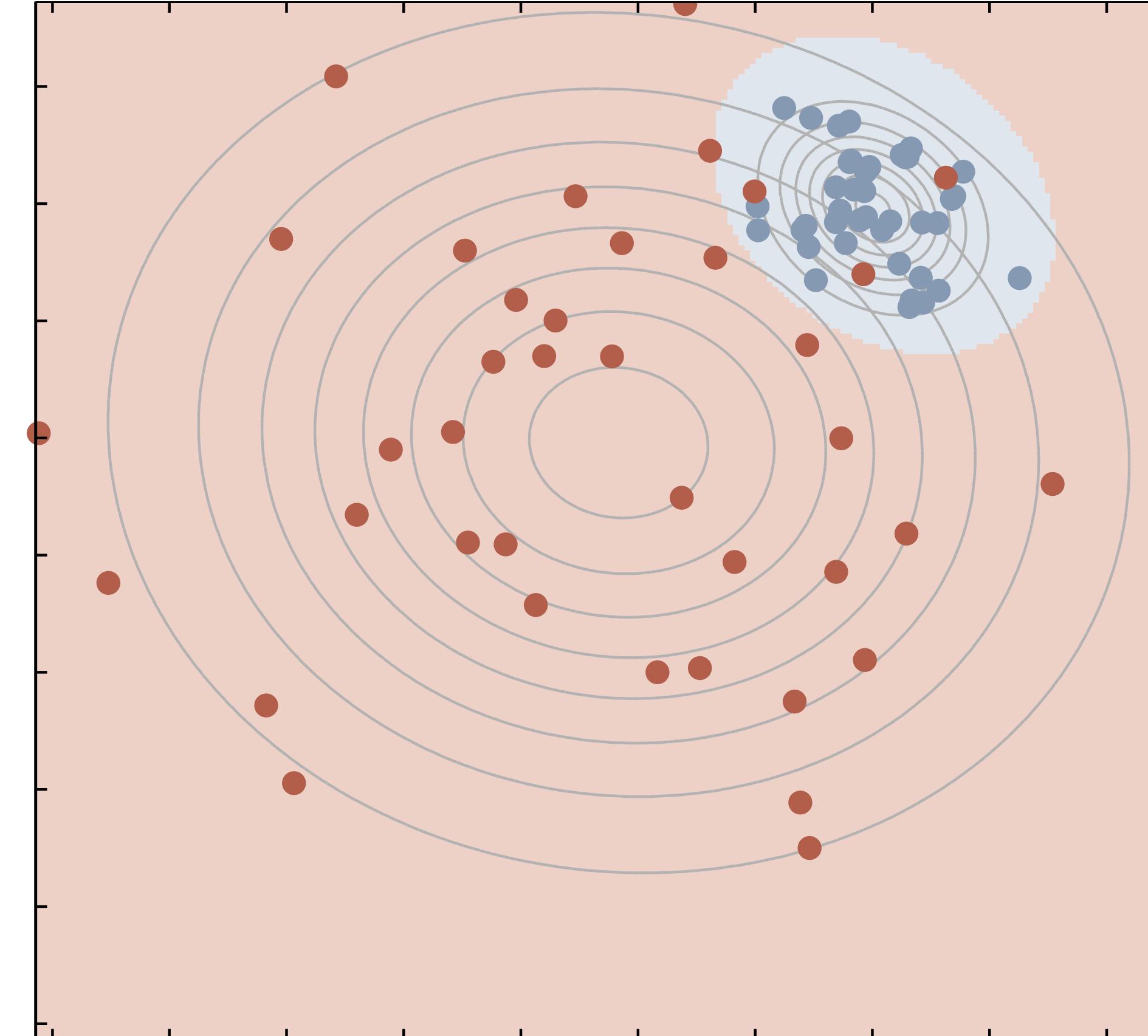
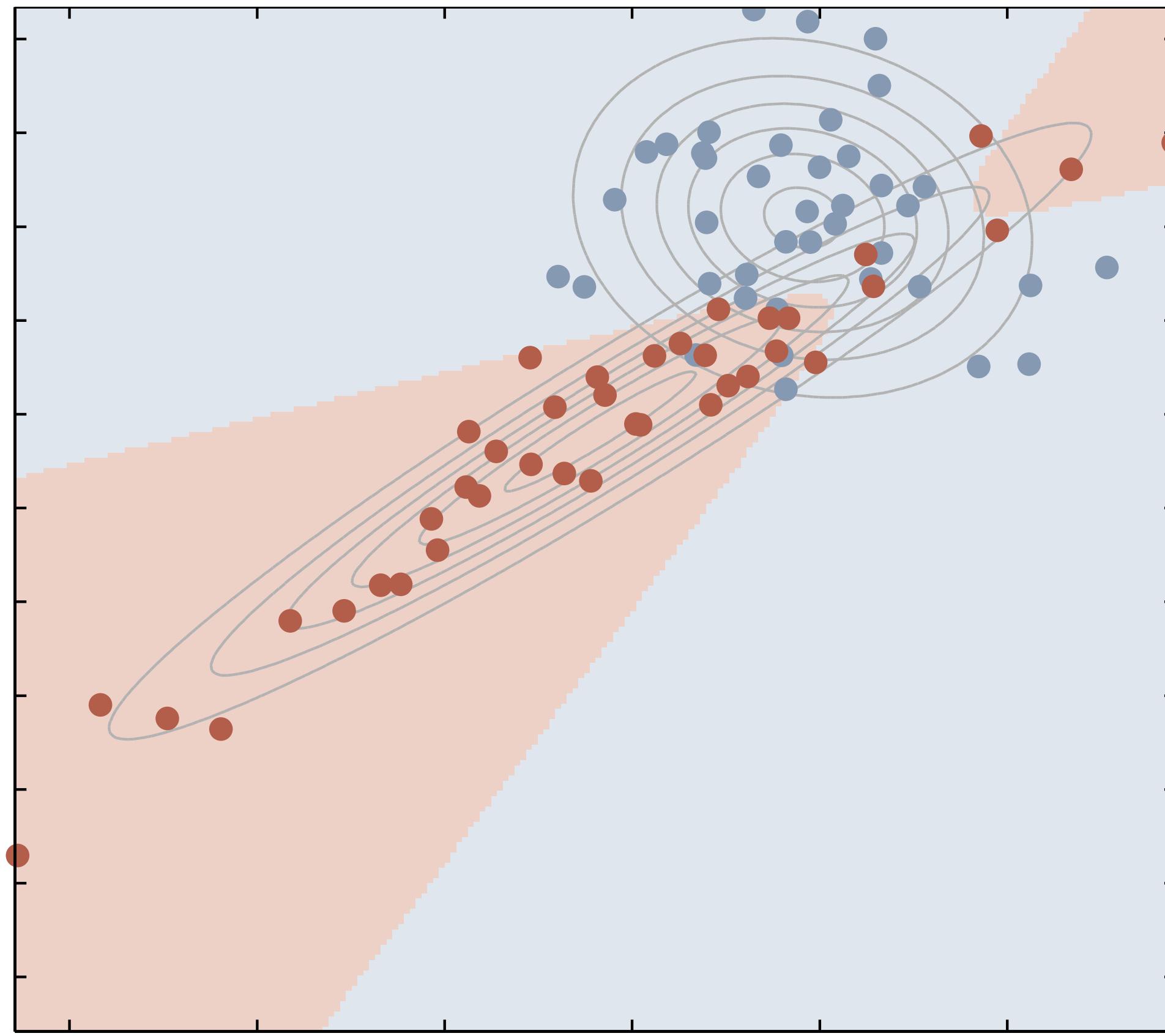
$$w = -\frac{1}{2} \boldsymbol{\mu}_i^\top \cdot \Sigma_i^{-1} \cdot \boldsymbol{\mu}_i - \frac{1}{2} \log |\Sigma_i|$$

$$+ \log P(\omega_i)$$



# Quadratic boundaries

- Arbitrary covariance matrices can produce more elaborate boundaries



# Naïve Bayes classifier

- Dimensionality issues
  - For large dimensions the Gaussian estimate will require a lot of data! In the order of  $N$  dimensions
- Naïve Bayes classifier assumes independence across dimensions

# Naïve Bayes classifier

- Each dimension is sampled independently
  - Thus we don't require many training samples

$$P(\mathbf{x} \mid \omega_i) = \prod_j P(x_j \mid \omega_i)$$

- Overall classification is:

$$\omega = \arg \max_{\omega_i} \prod_j P(x_j \mid \omega_i)$$

- Not the best model, but reasonably reliable and much faster

# A different perspective

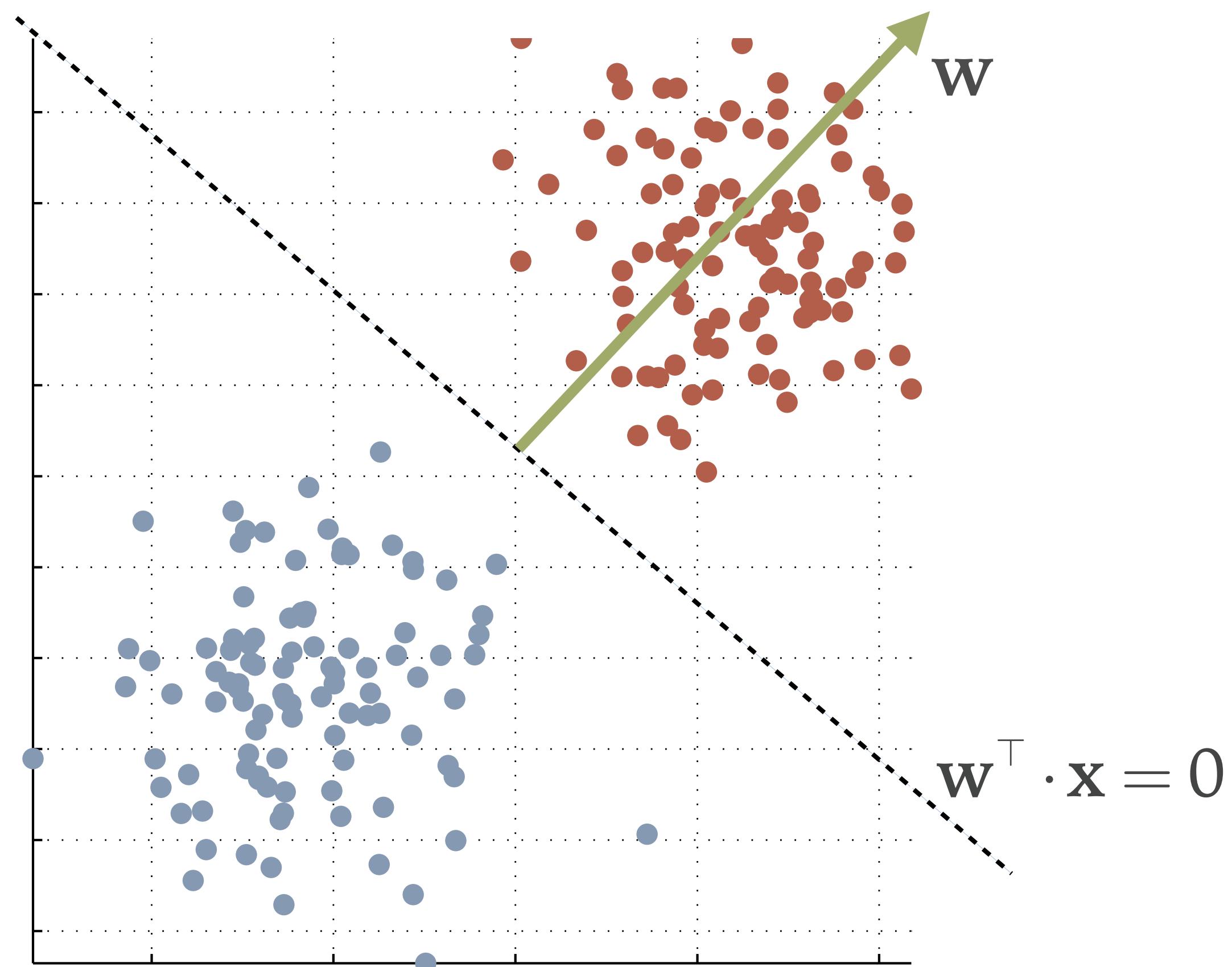
- Obtaining the discriminant function directly
  - Use training data  $\mathbf{x}_i$ , and corresponding labels  $y_i \in \{-1, 1\}$
  - Linear discriminant  $\mathbf{w}$  and bias  $b$

$$y_i = \mathbf{w}^\top \cdot \mathbf{x}_i + b$$

- Or we can skip the  $b$  and set  $\mathbf{x} = \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$  in which case  $\mathbf{w} = [\mathbf{w}, b]$
- This is the same as the discriminant function for unit covariance Gaussians

# Linear classifiers

- Directly defines the class boundary line



# Initial take: The Perceptron

- Assume there is a solution
- Then find  $w$  such that:

$$w^\top \cdot x_i > 0 \text{ if } x_i \in \omega_1$$

$$w^\top \cdot x_i < 0 \text{ if } x_i \in \omega_2$$

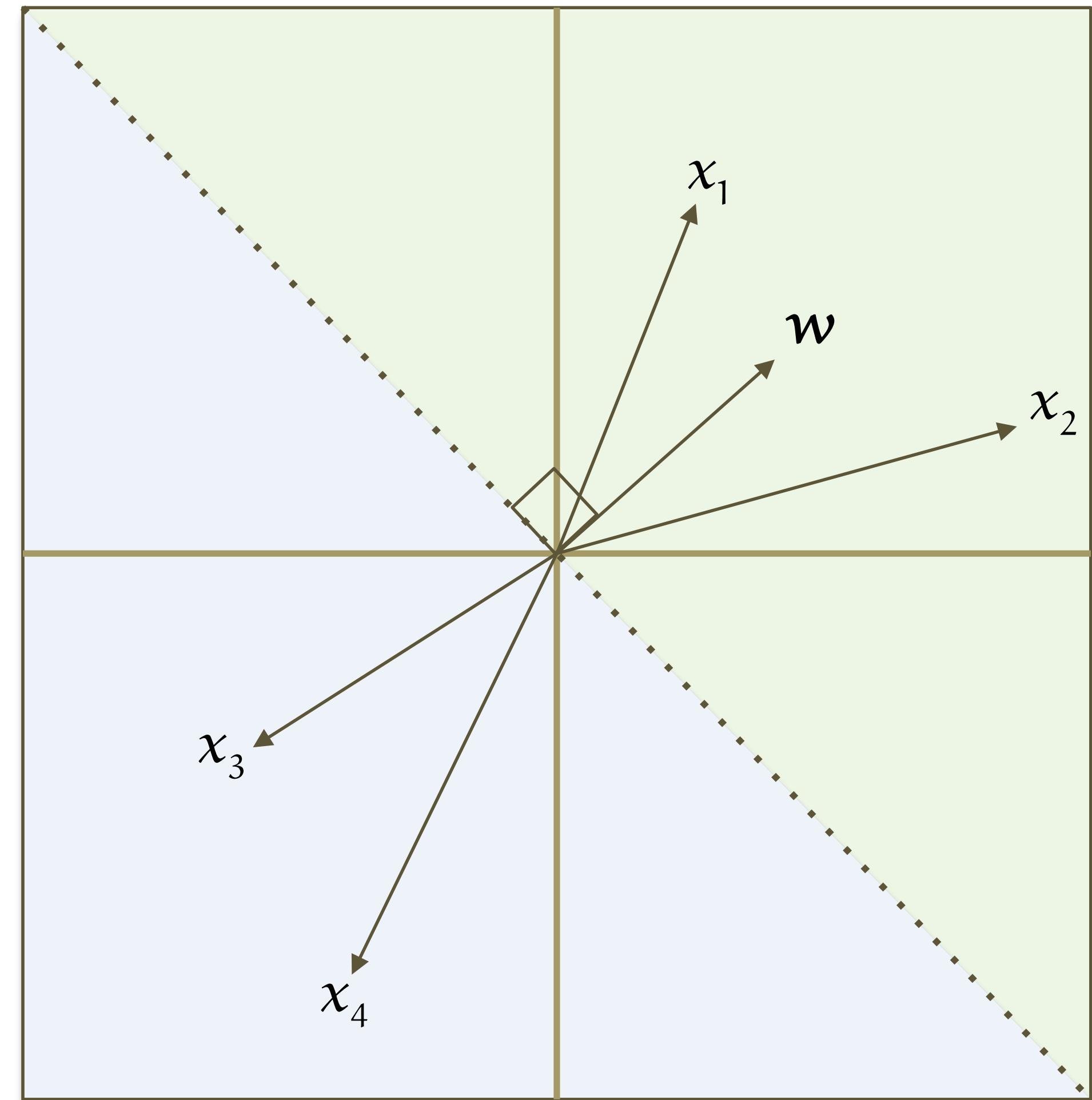
- How do we solve this?

# A simple update algorithm

- Using corrections
- For all training samples  $\mathbf{x}$ :
  - If  $\text{sgn}(\mathbf{w}^\top \cdot \mathbf{x}_i) = y_i$ 
    - Do nothing
  - If  $\text{sgn}(\mathbf{w}^\top \cdot \mathbf{x}_i) = -y_i$ 
    - Then  $\mathbf{w} = \mathbf{w} + \eta y_i \mathbf{x}_i$ ,  $0 < \eta < 1$
  - Repeat until no error (or progress) is made

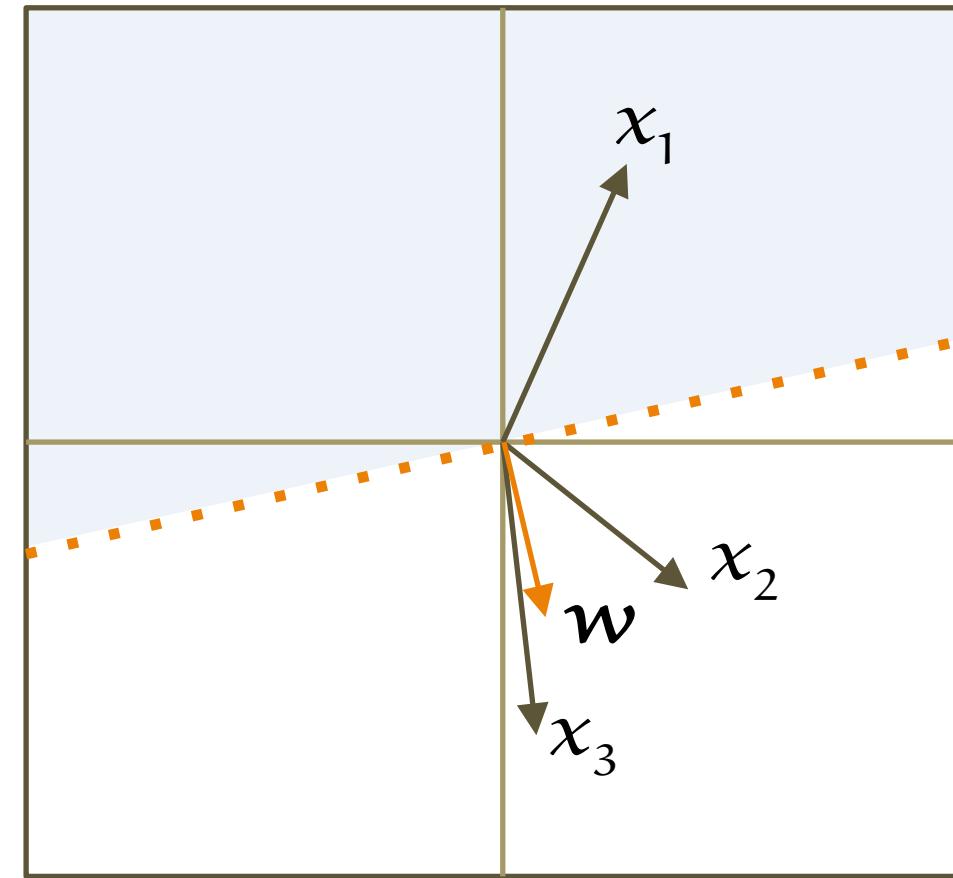
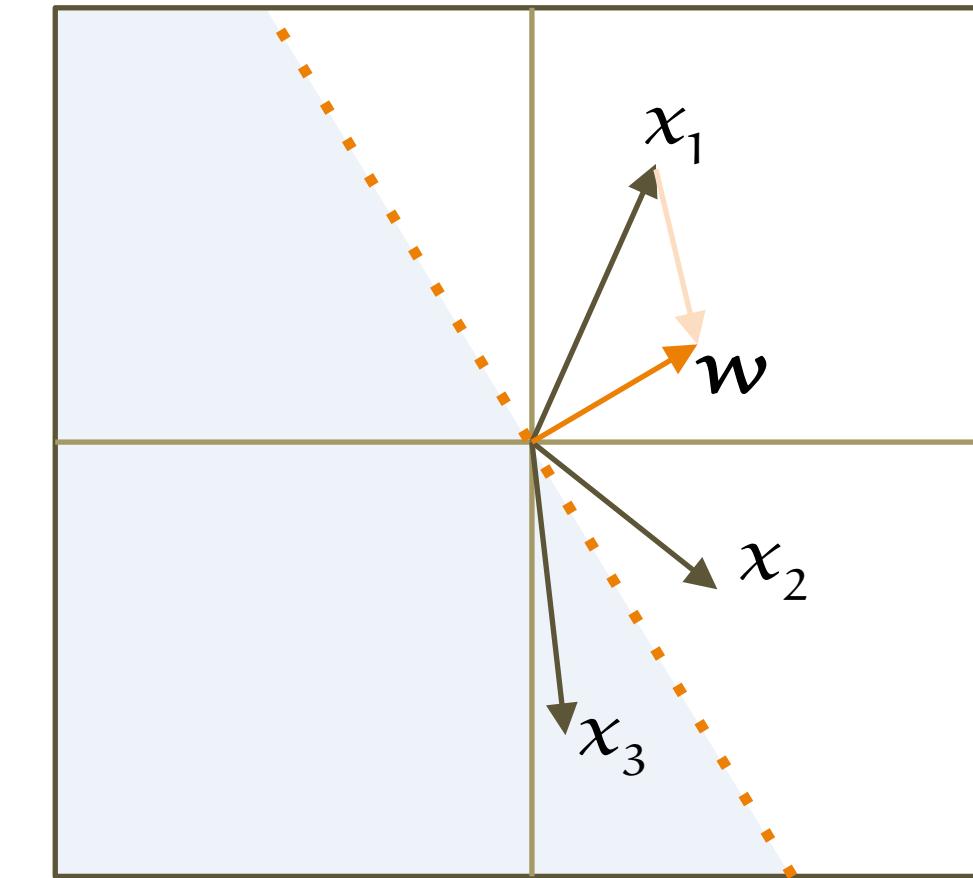
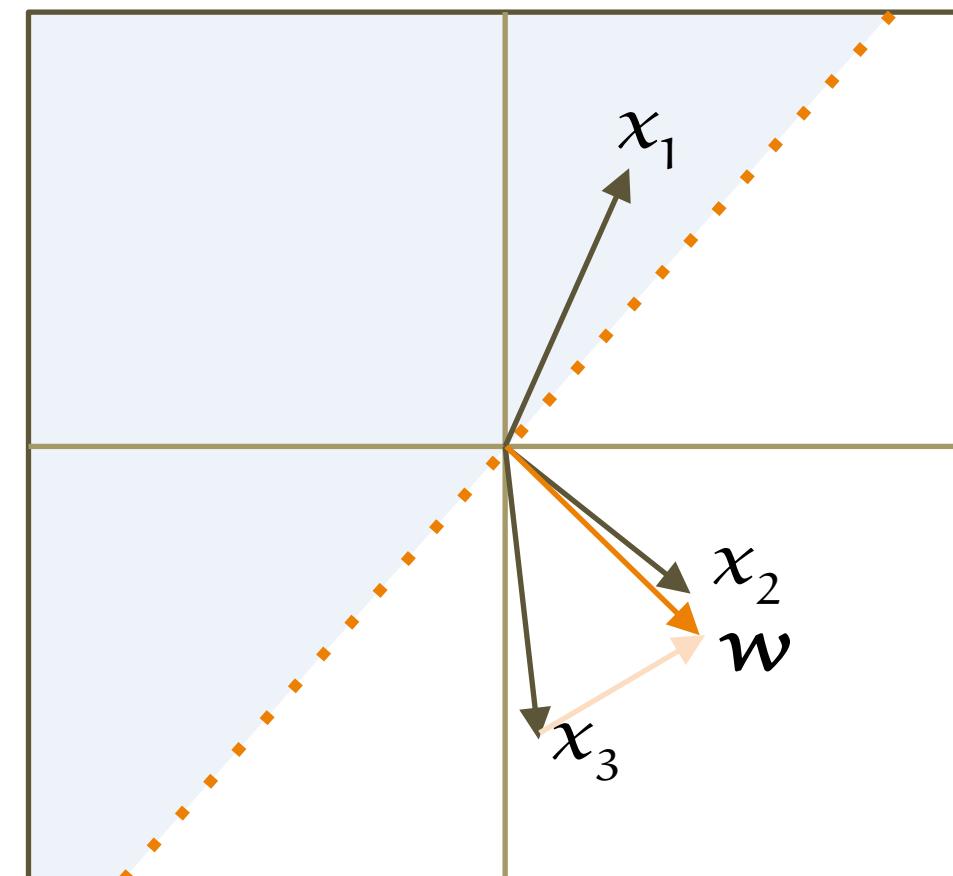
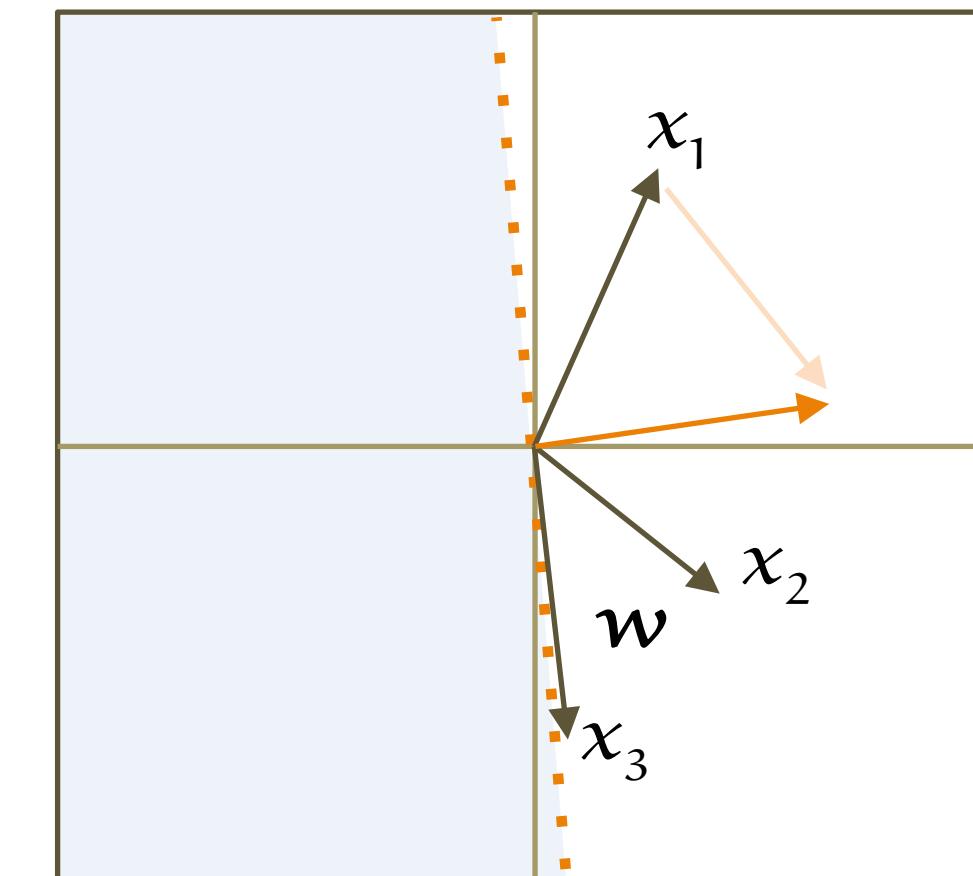
# What does this mean?

- $w$  is normal to the boundary line
- To produce  $w^\top \cdot x_i > 0$ 
  - $x_i$  has to be within  $\pm 90^\circ$  of  $w$
- To produce  $w^\top \cdot x_i < 0$ 
  - $x_i$  has to be past  $\pm 90^\circ$  of  $w$

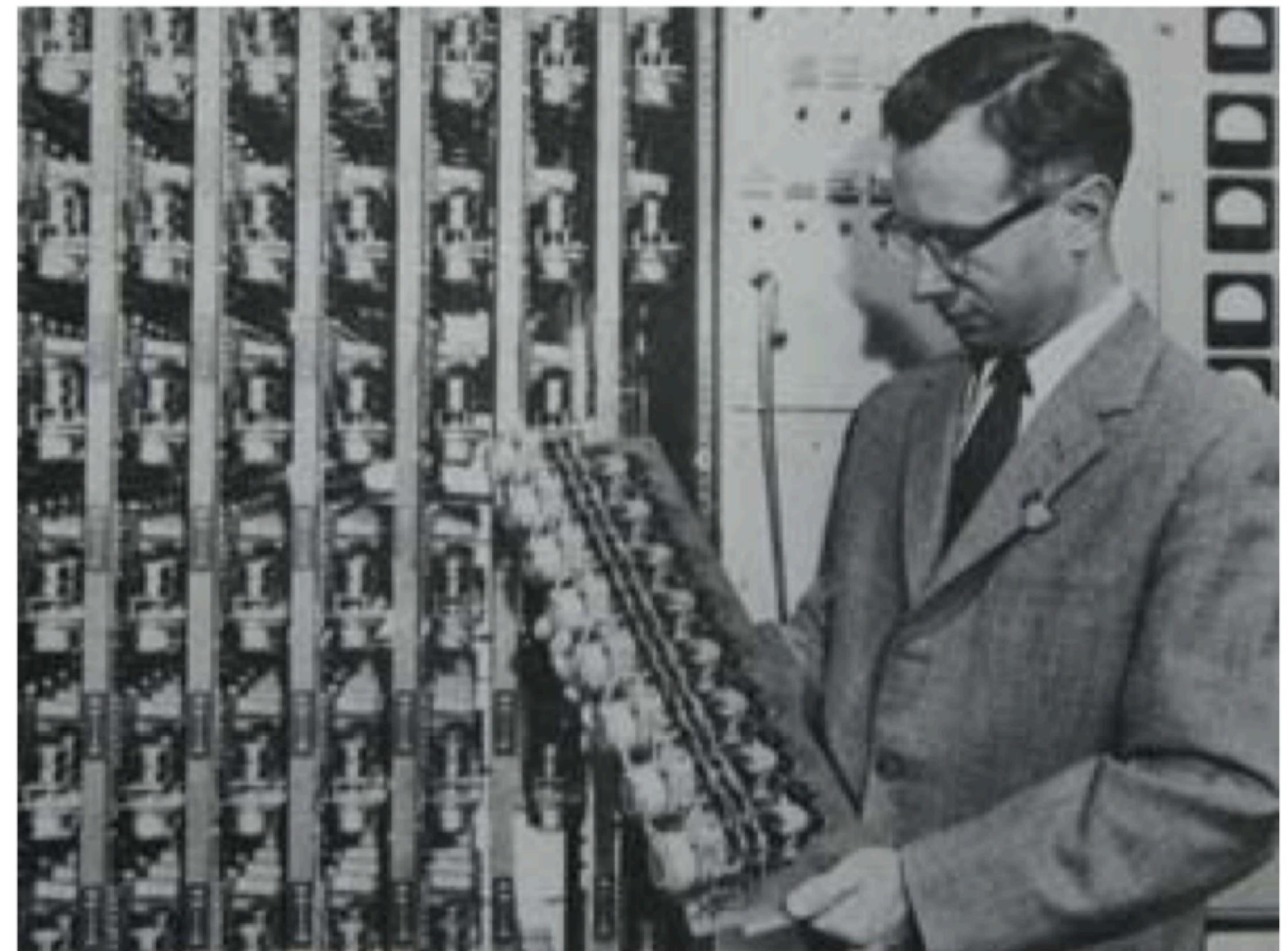


# Looking at one class

1. Initial conditions

2. Correction with  $x_1$ 3. Correction with  $x_3$ 4. Correction with  $x_1$ 

# Relevant historical pictures



# Refining: Using a cost function

- Minimize the cost function:

$$J(\mathbf{w}) = \sum_{\forall \text{ sgn}(\mathbf{w}^\top \cdot \mathbf{x}_i) \neq y_i} \delta_i \mathbf{w}^\top \cdot \mathbf{x}_i$$

$$\delta_i = -y_i = \begin{cases} -1, & \text{if } \mathbf{x}_i \in \omega_1 \\ +1, & \text{if } \mathbf{x}_i \in \omega_2 \end{cases}$$

		<i>Contribution to cost function</i>	
$\mathbf{x}_i \in \omega_1$	$\mathbf{x}_i \in \omega_2$	$\delta_i = -1$	$\delta_i = +1$
$\mathbf{w}^\top \cdot \mathbf{x}_i > 0$		0	$\delta_i \mathbf{w}^\top \cdot \mathbf{x}_i > 0$
$\mathbf{w}^\top \cdot \mathbf{x}_i < 0$		$\delta_i \mathbf{w}^\top \cdot \mathbf{x}_i > 0$	0

- Cost function will be zero if all classifications are correct

# Gradient descent approach

- Use a gradient descend algorithm:

$$\mathbf{w} = \mathbf{w} - \eta \sum_{\forall \text{sgn}(\mathbf{w}^\top \cdot \mathbf{x}_i) \neq y_i} \delta_i \mathbf{x}_i$$

- Same as the perceptron!
- Proven convergence, fast and small!
  - Many variants exist
  - A core idea behind neural nets

# Linear classifiers

- As long as we use a Euclidean distance metric, there is a Gaussian assumption
- Linear classifiers are easier to understand
  - But remember what they actually imply!

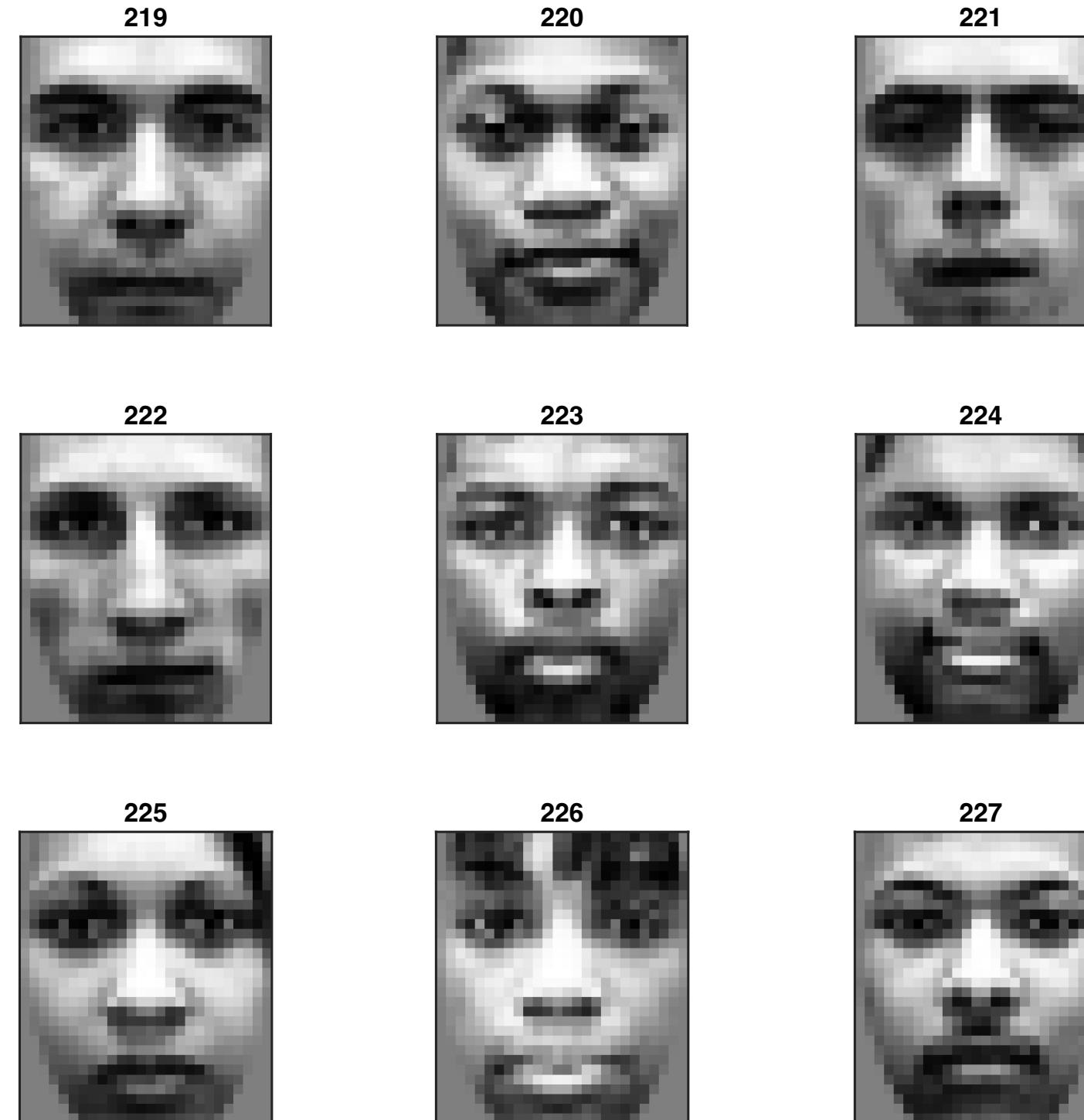
# Looking back on detection

- We can now make detection more elegant
  - No need to look at correlation peaks
- Learn class models from examples
  - Doesn't have to be a single template
- Translate the dot products to likelihoods
  - Apply decision theory to classify in either of the two classes

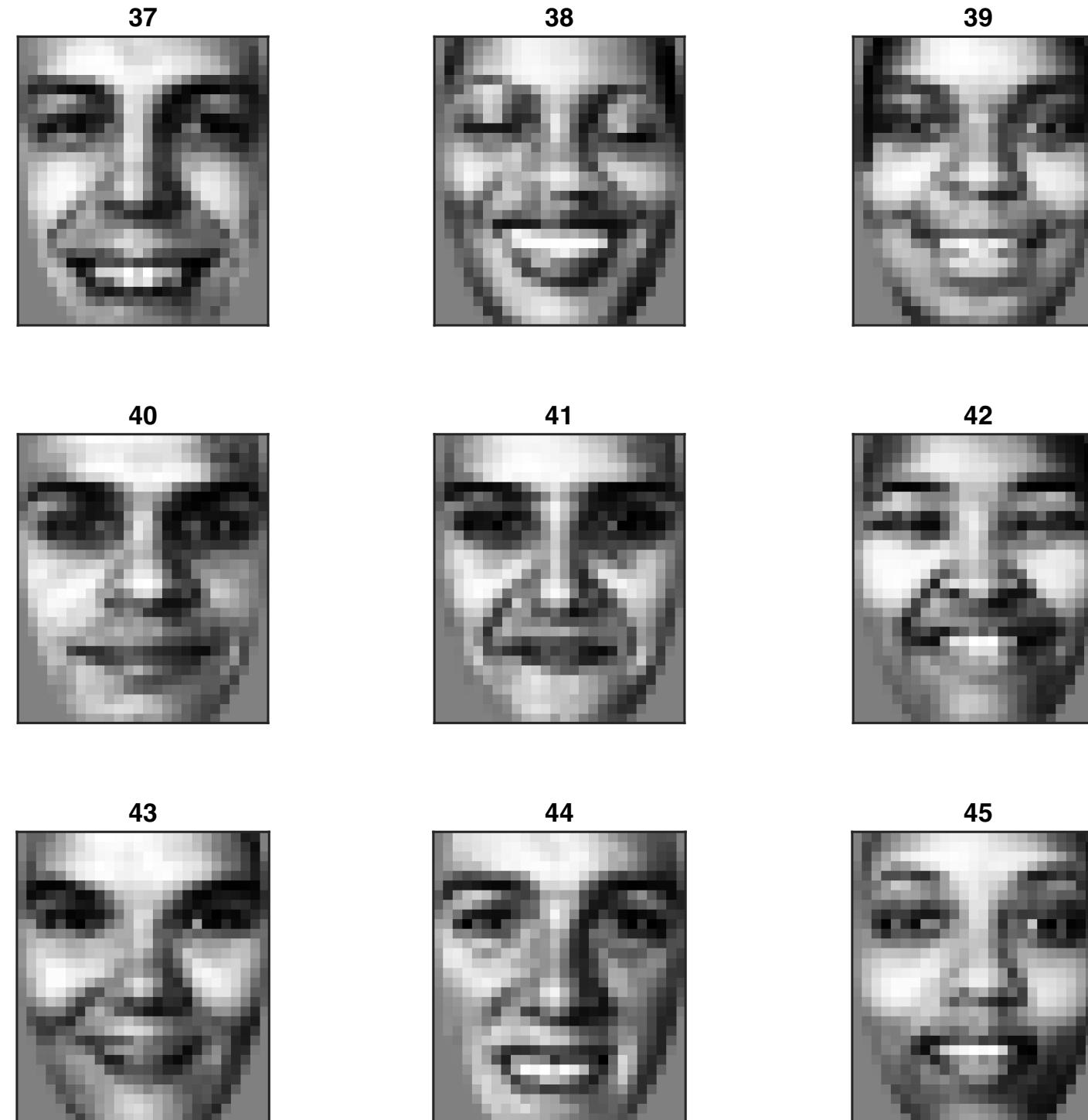
# Example: A smile detector

- Make a classifier that find smiling faces

*Some non-smiling samples*

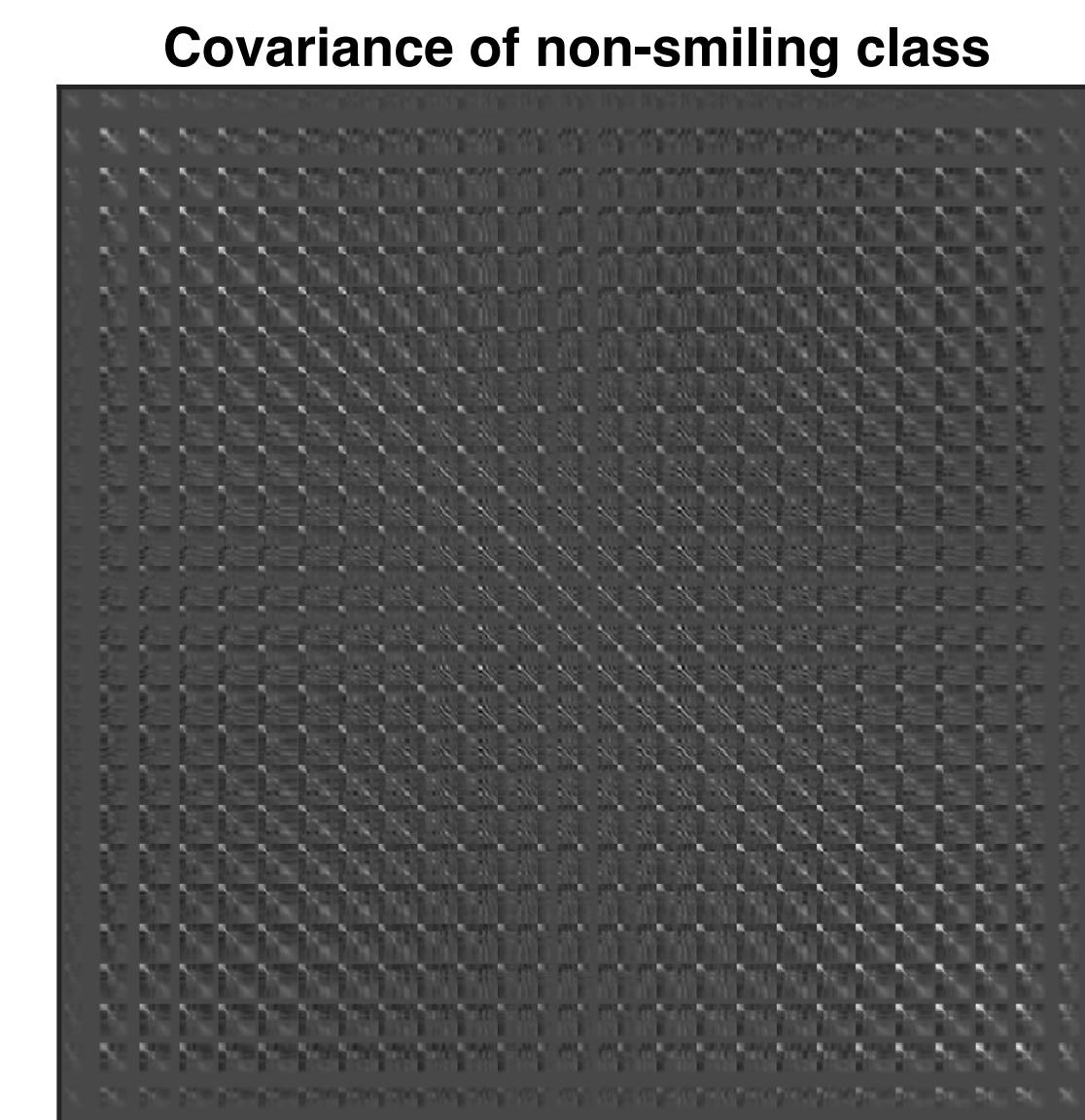
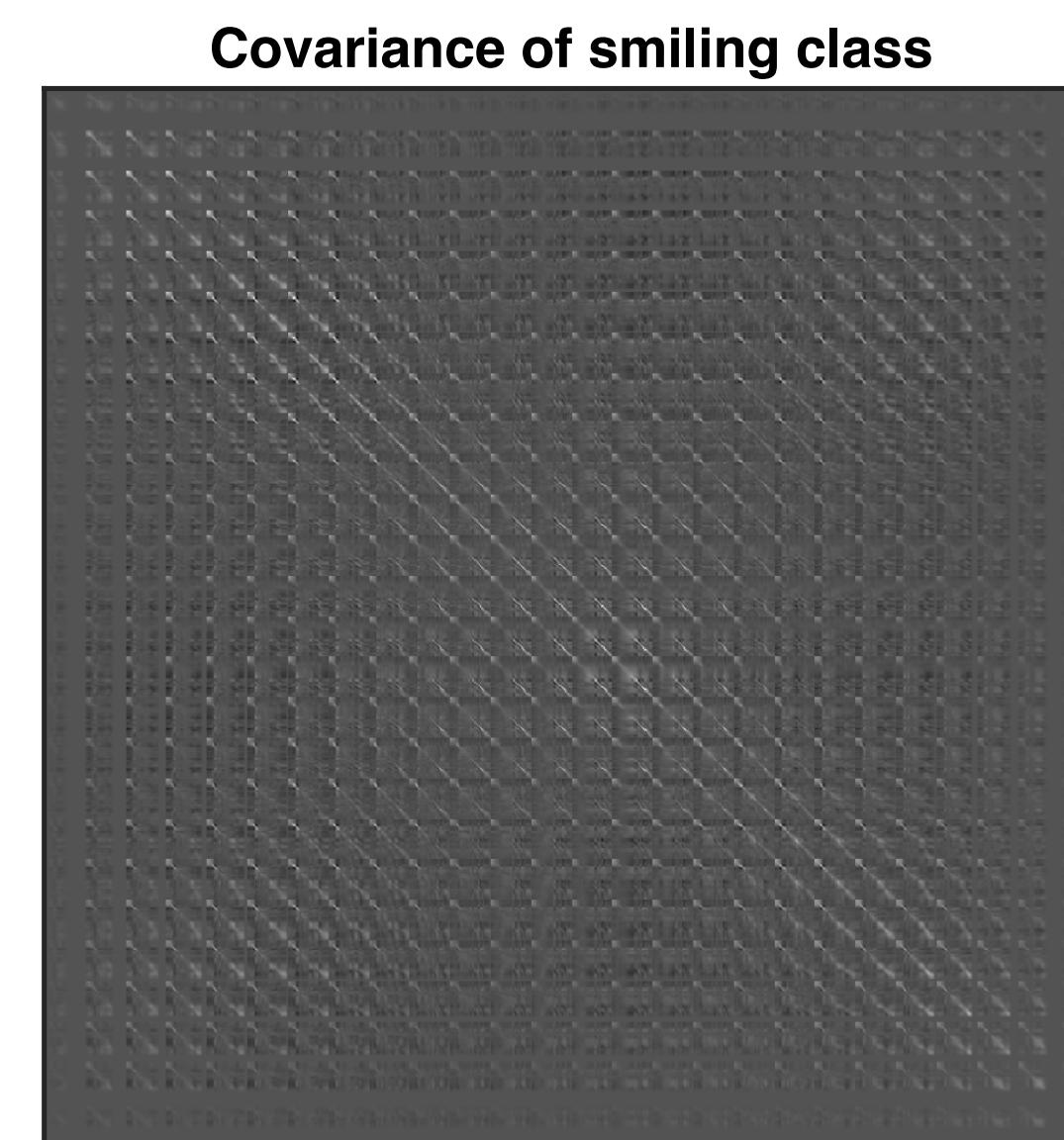


*Some smiling samples*



# Simple way

- Assume classes are Gaussian distributed (are they?)
  - Get means and covariances for each class



# Standard ways to classify

- Assume a linear classifier (unit covariance Gaussian):

$$P(\mathbf{x} | \omega_i) = \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_i, \mathbf{I}) \propto e^{-(\mathbf{x} - \boldsymbol{\mu}_i)^\top \cdot (\mathbf{x} - \boldsymbol{\mu}_i)}$$

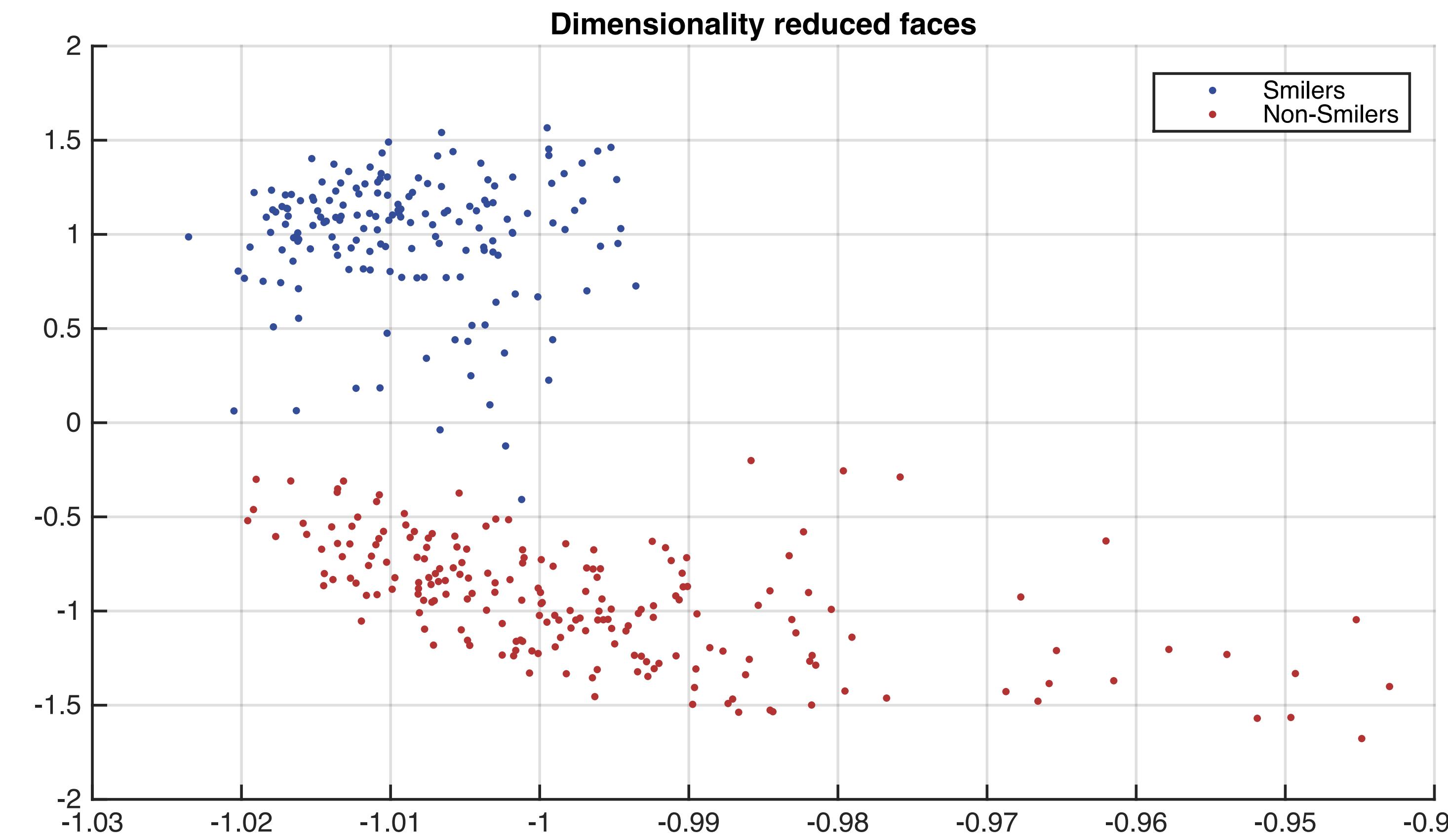
- Assume equal priors and assign data to maximum likelihood class
  - Get's us okay results most of the time
- Or assume a quadratic classifier (full or diagonal covariance)

$$P(\mathbf{x} | \omega_i) = \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \propto e^{-(\mathbf{x} - \boldsymbol{\mu}_i)^\top \cdot \boldsymbol{\Sigma}_i^{-1} \cdot (\mathbf{x} - \boldsymbol{\mu}_i)}$$

- Careful, the covariance might be non-invertible

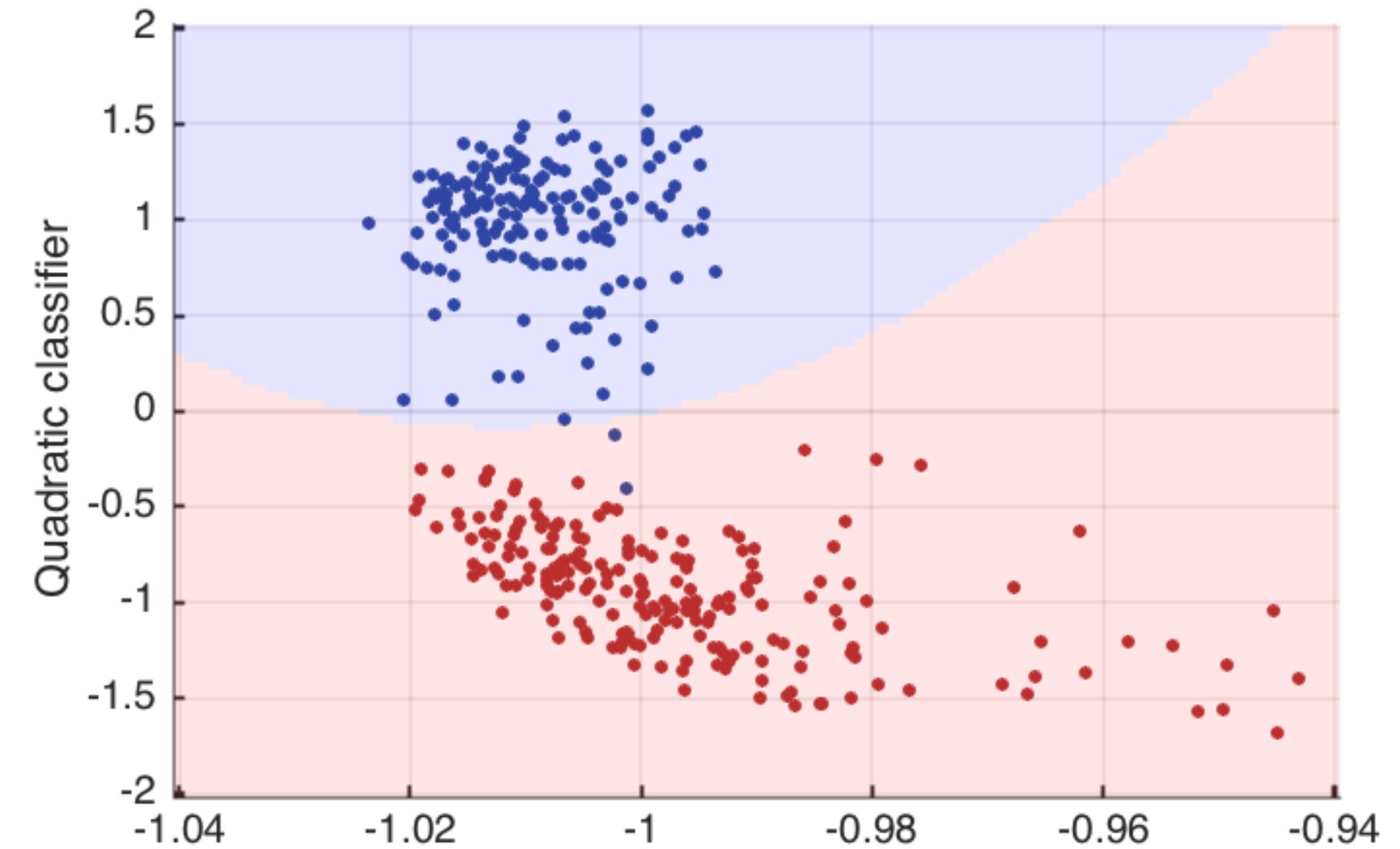
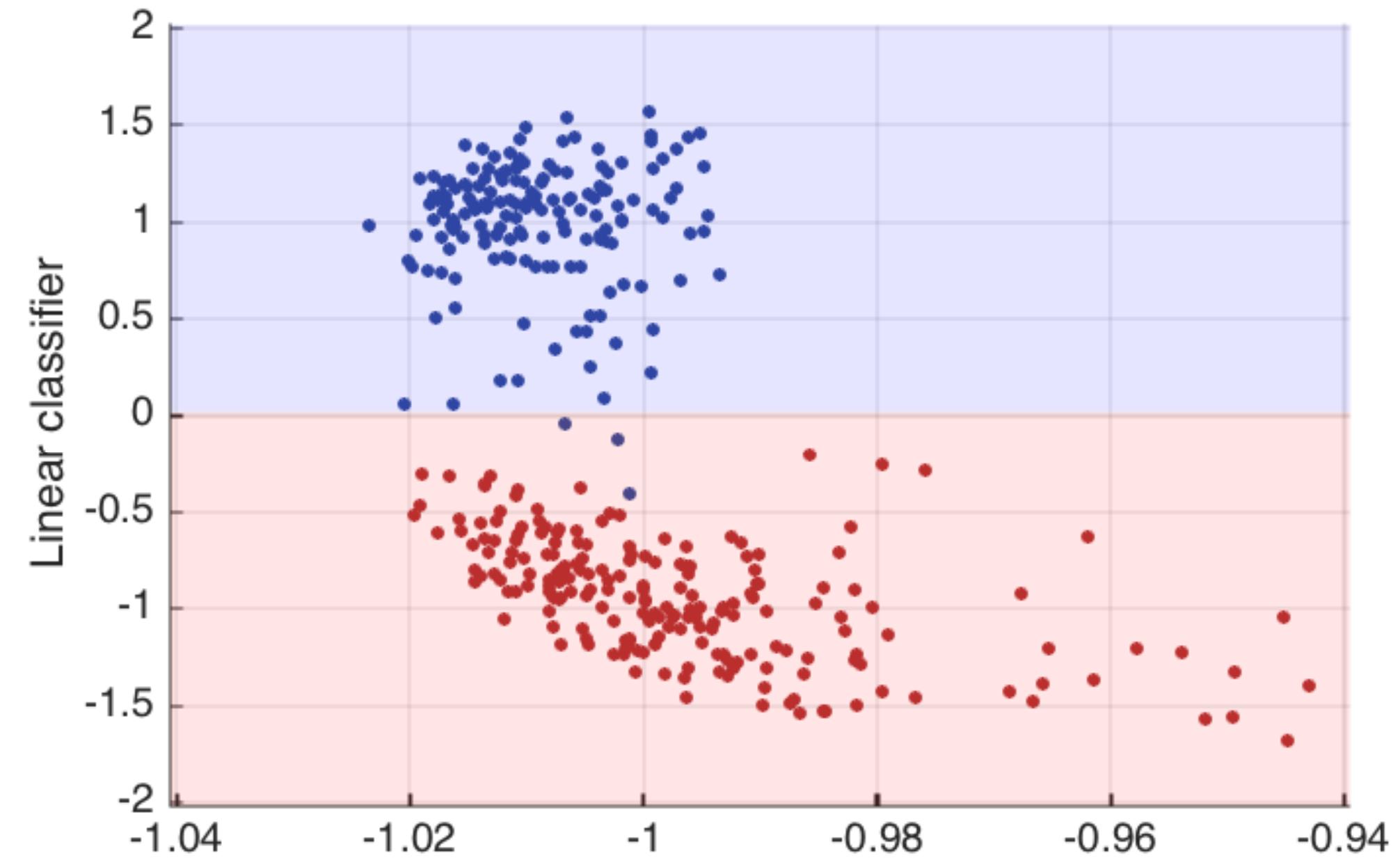
# Or we can use features

- PCA the faces to lower dimensionality
  - Now the covariance estimate is better behaved



# Redo the classification on the low dims

- Faster and easier, and just as good
  - Not crucial in this case, but the quadratic classifier is a bit better



- How can we improve this? Is it worth it?

# Recap

- The Bayesian view
  - Risk, decision regions
  - Gaussian classifiers, Naïve Bayes
- Linear classifiers
  - The perceptron, separating hyperplanes

# Next lecture

- Support Vector Machines
- Non-linear classifiers
  - Neural nets and Kernels

# Reading

- Textbook chapters 2-2.4, 2.5.7 and 3-3.6