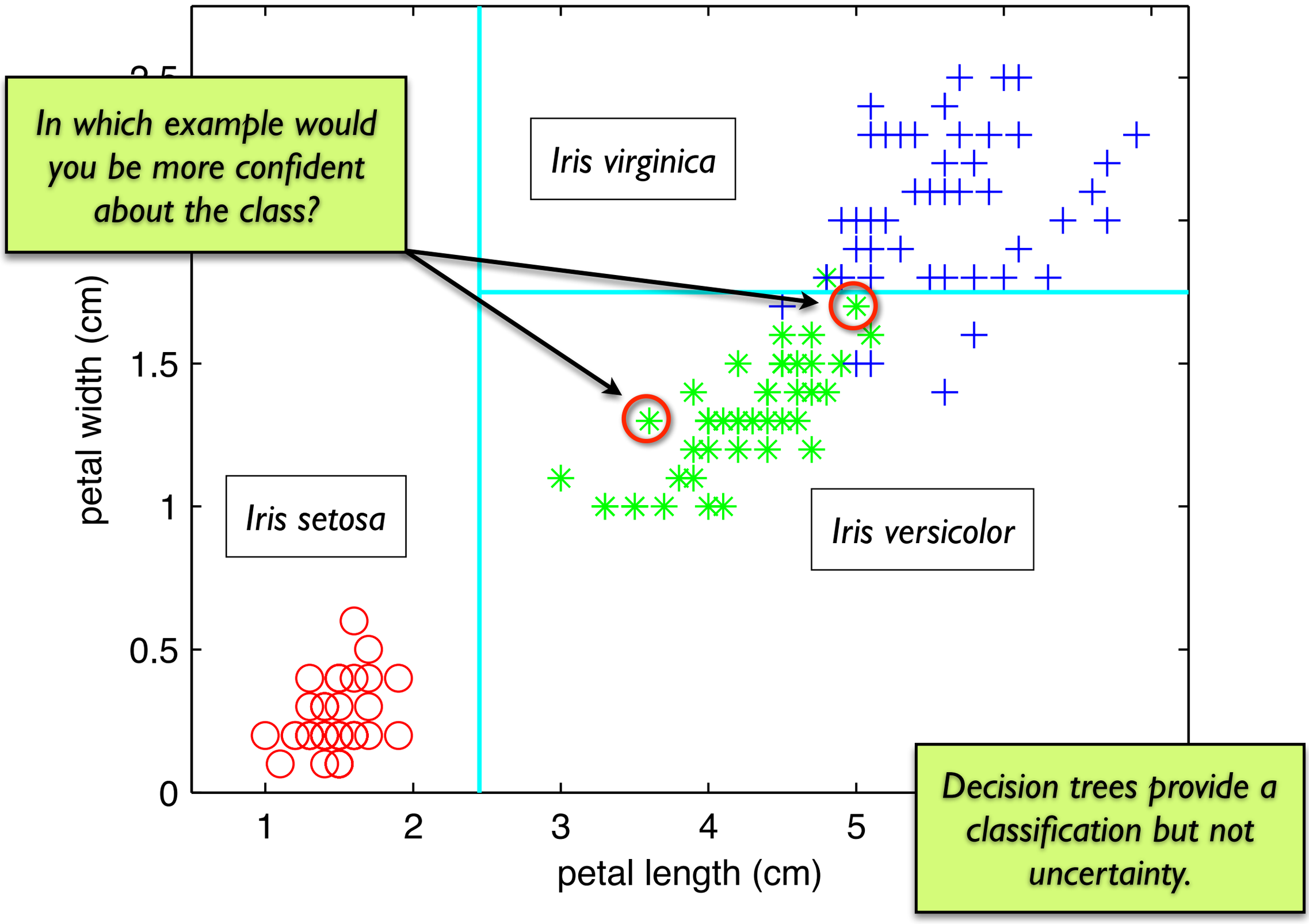
EECS 391
Intro to AI

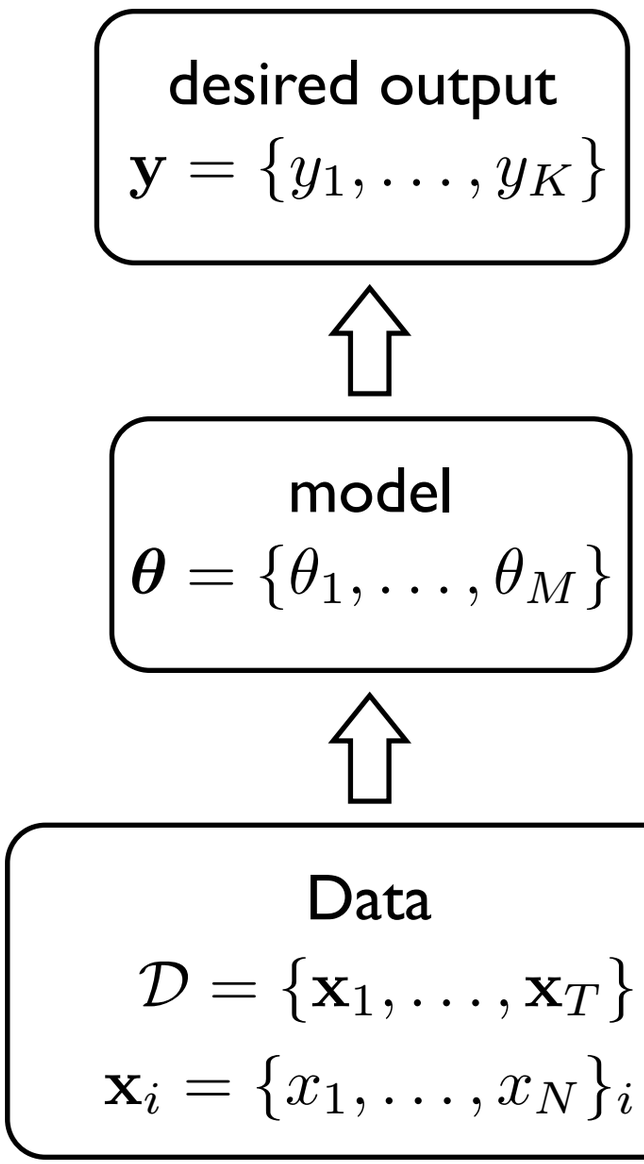
Unsupervised Learning and Clustering

L19 Tue Nov 14

Fisher's Iris data



The general classification problem



output is a binary classification vector:

$$y_i = \begin{cases} 1 & \text{if } \mathbf{x}_i \in C_i \equiv \text{class } i, \\ 0 & \text{otherwise} \end{cases}$$

model (e.g. a decision tree) is defined by M parameters.

input is a set of T observations, each an N-dimensional vector (binary, discrete, or continuous)

Given data, we want to learn a model that can correctly classify novel observations.

How do we approach this probabilistically?

The answer to all questions of uncertainty

- Let's apply Bayes' rule to infer the most probable class given the observation:

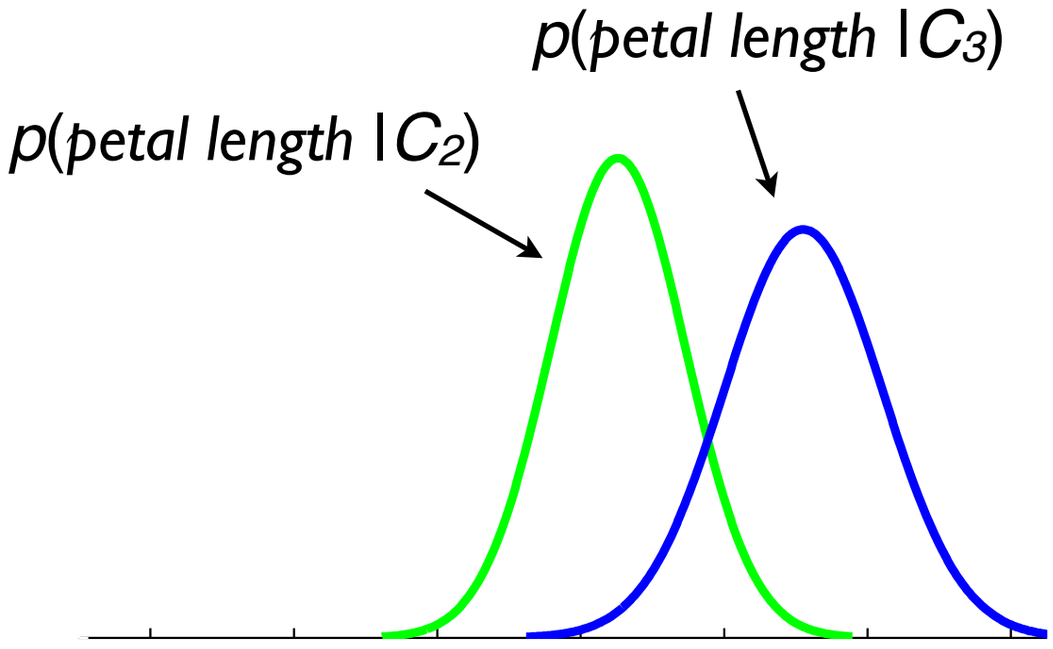
$$
\begin{aligned}
p(C_k|\mathbf{x}) &= \frac{p(\mathbf{x}|C_k)p(C_k)}{p(\mathbf{x})} \\
&= \frac{p(\mathbf{x}|C_k)p(C_k)}{\sum_k p(\mathbf{x}|C_k)p(C_k)}
\end{aligned}
$$

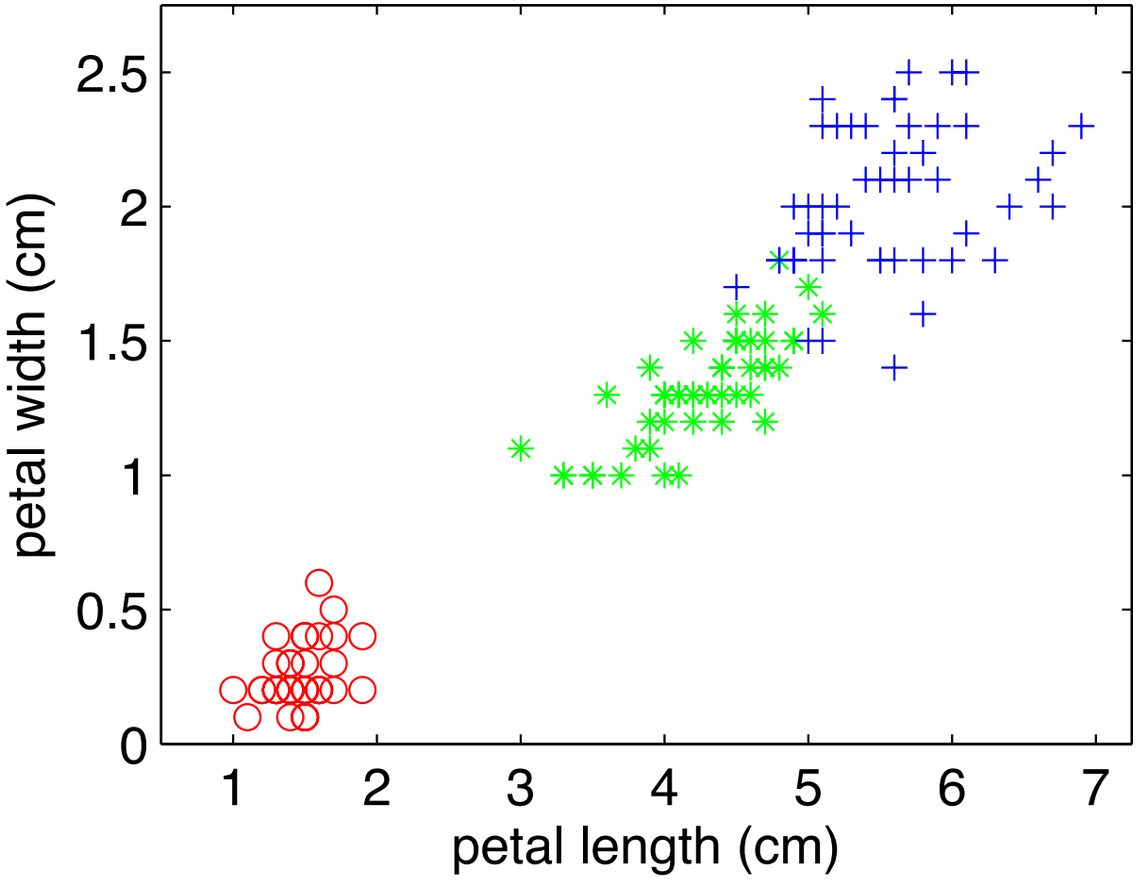
- This is the answer, but what does it mean?
- How do we specify the terms?
  - $p(C_k)$ is the prior probability on the different classes
  - $p(\mathbf{x}|C_k)$ is the data likelihood, ie probability of $\mathbf{x}$ given class C_k
- How should we define this?

What classifier would give "optimal" performance?

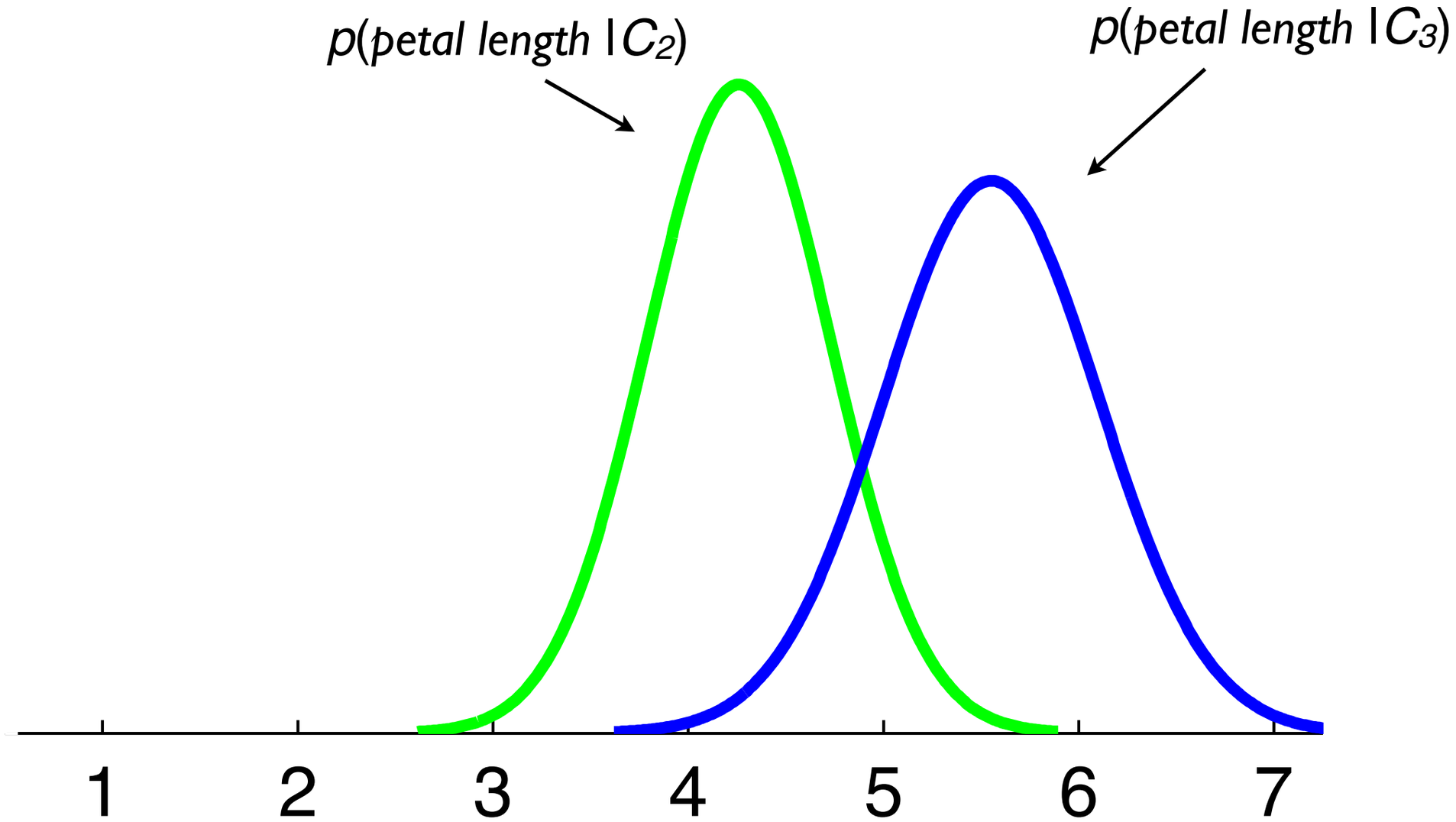
- Consider the iris data.
- How would we minimize the number of *future* mis-classifications?
- We would need to know the true *distribution* of the classes.
- Assume they follow a Gaussian distribution.
- The number of samples in each class is the same (50), so (assume) $p(C_k)$ is equal for all classes.
- Because $p(\mathbf{x})$ is the same for all classes we have:

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

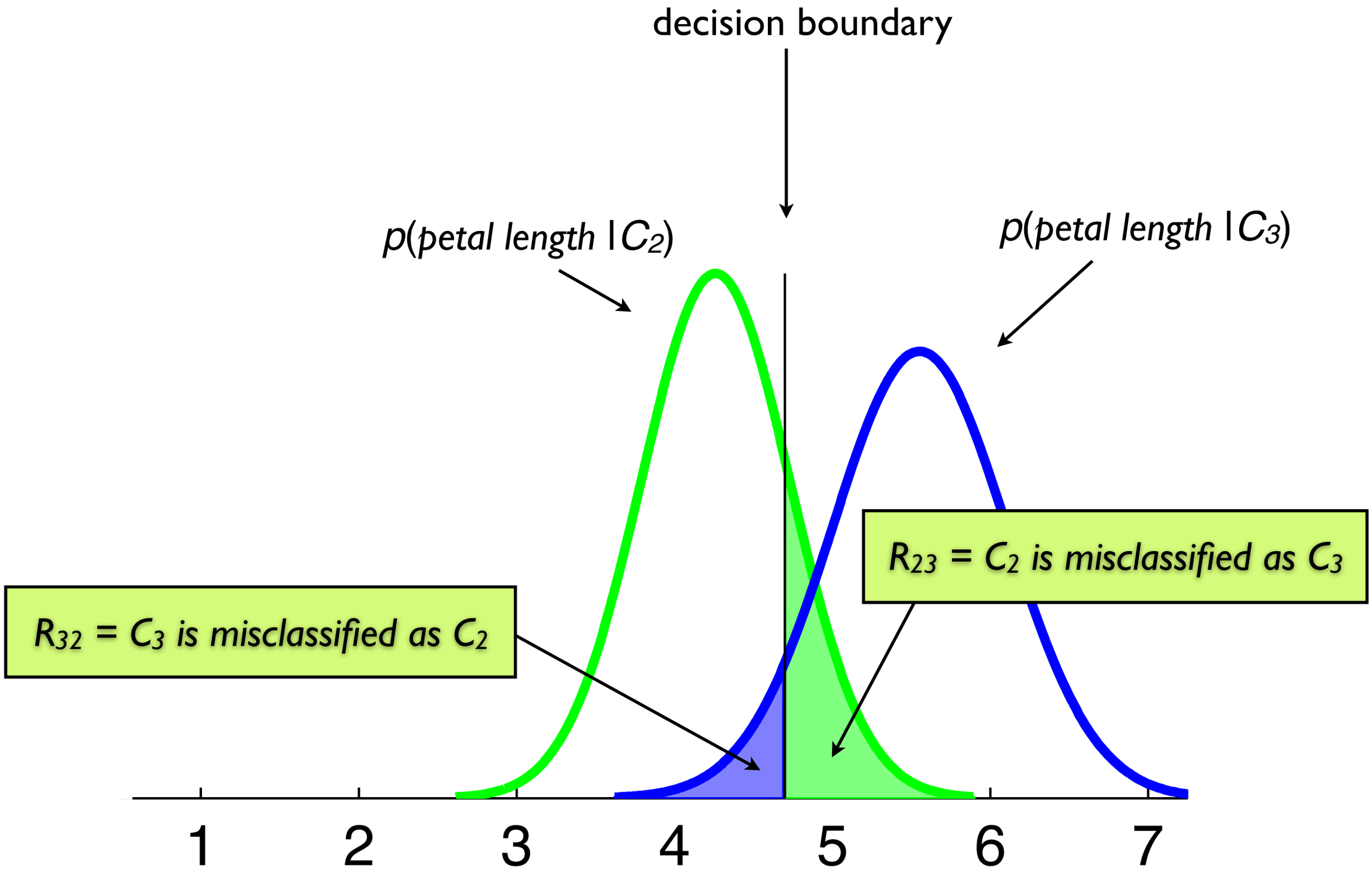




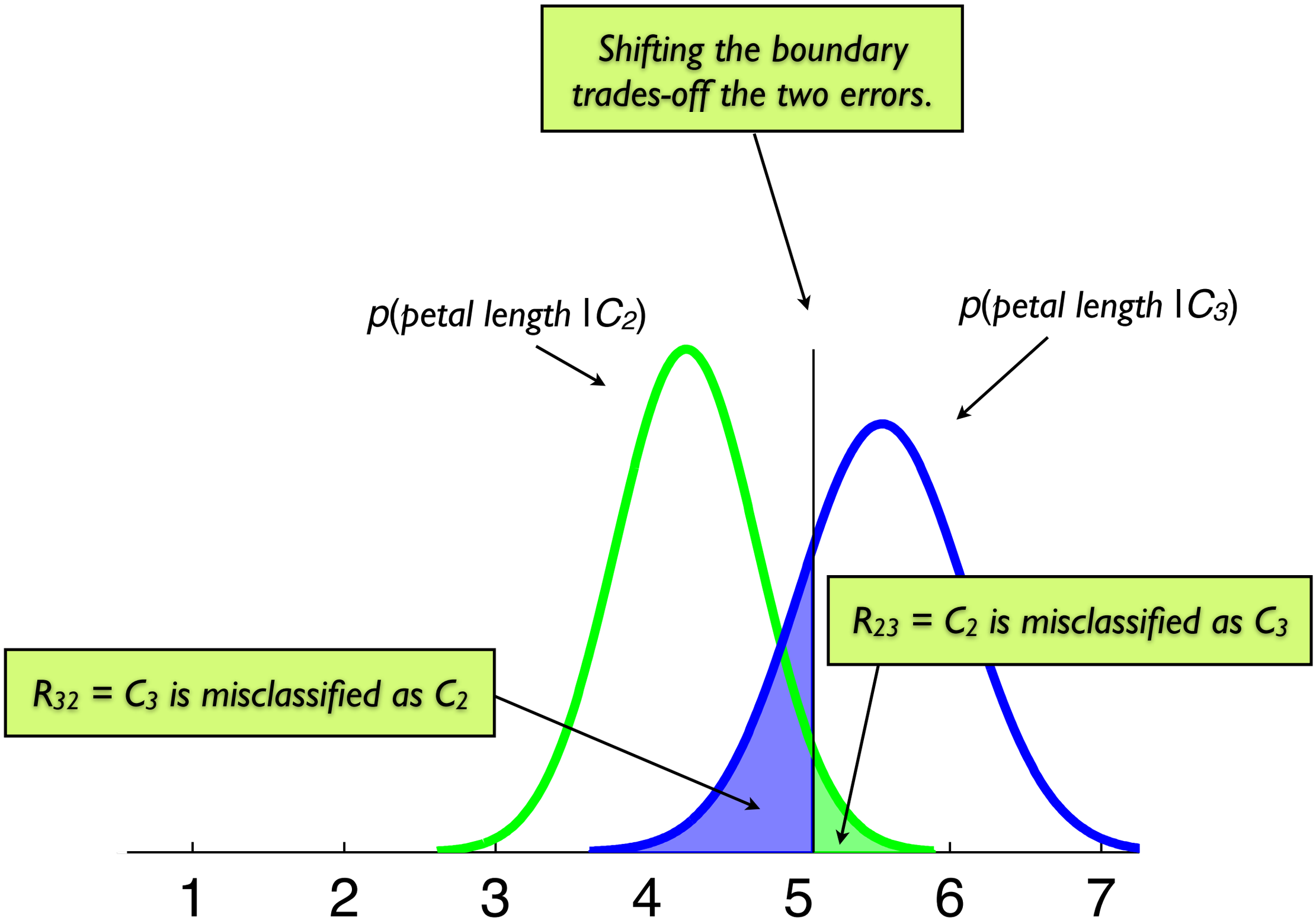
Where do we put the boundary?



Where do we put the boundary?



Where do we put the boundary?

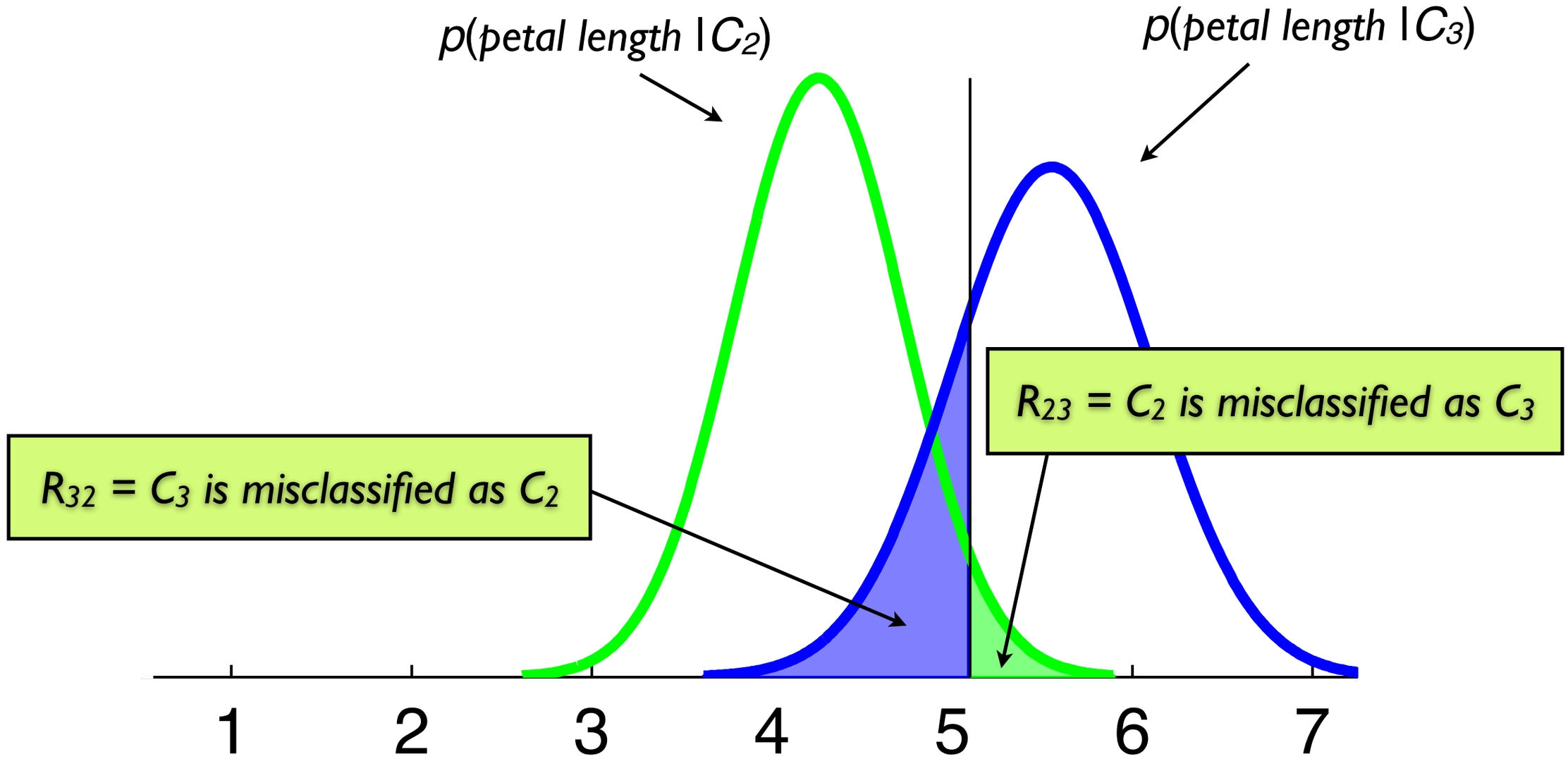


Where do we put the boundary?

- The misclassification error is defined by

$$p(\text{error}) = \int_{R_{32}} p(\mathbf{x}|C_3)P(C_3)d\mathbf{x} + \int_{R_{23}} p(\mathbf{x}|C_2)P(C_2)d\mathbf{x}$$

- which in our case is proportional to the data likelihood

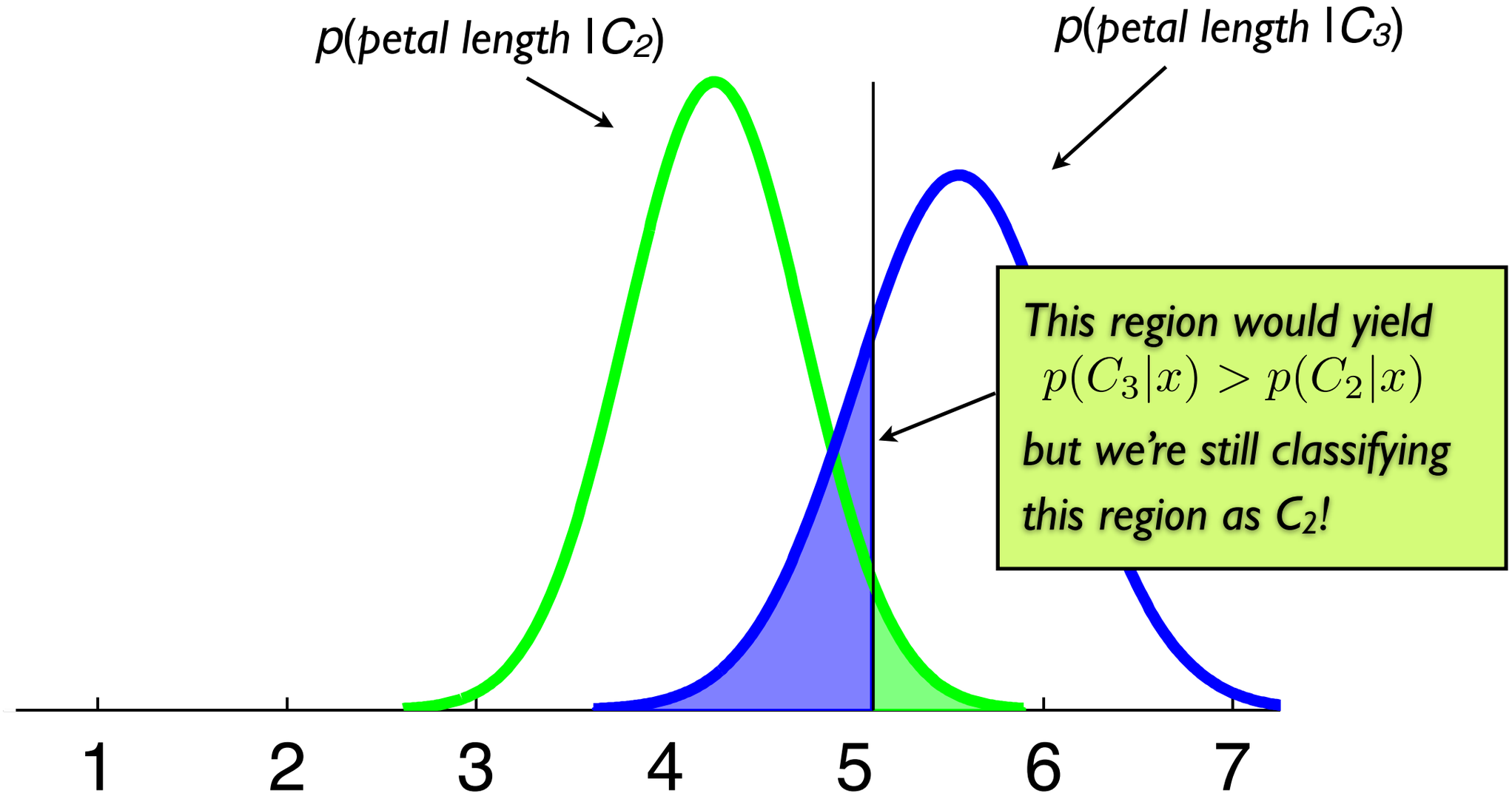


Where do we put the boundary?

- The misclassification error is defined by

$$p(\text{error}) = \int_{R_{32}} p(\mathbf{x}|C_3)P(C_3)d\mathbf{x} + \int_{R_{23}} p(\mathbf{x}|C_2)P(C_2)d\mathbf{x}$$

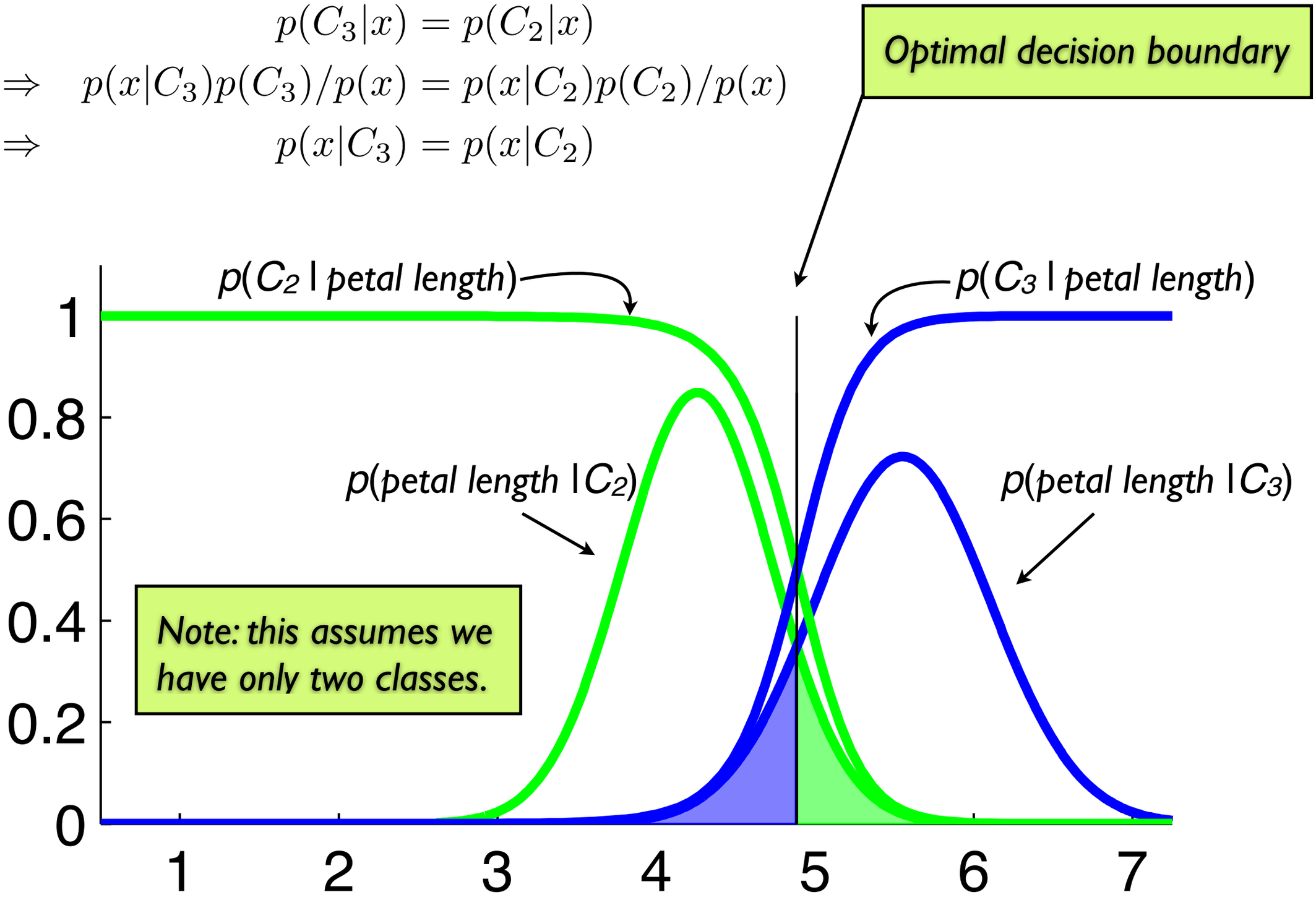
- which in our case is proportional to the data likelihood



The optimal decision boundary

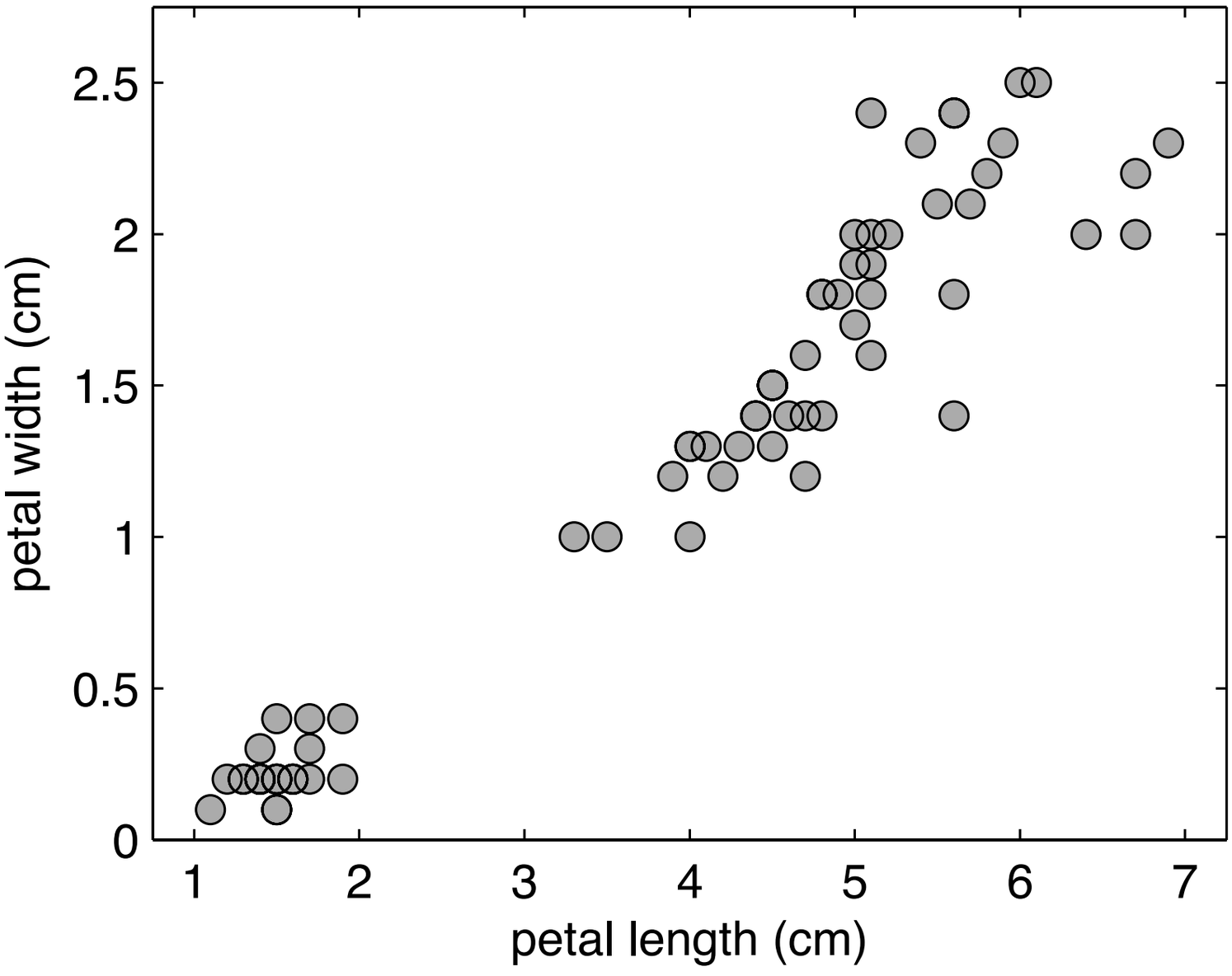
- The minimal misclassification error at the point where

$$
\begin{aligned}
& p(C_3|x) = p(C_2|x) \\
\Rightarrow \quad & p(x|C_3)p(C_3)/p(x) = p(x|C_2)p(C_2)/p(x) \\
\Rightarrow \quad & p(x|C_3) = p(x|C_2)
\end{aligned}
$$



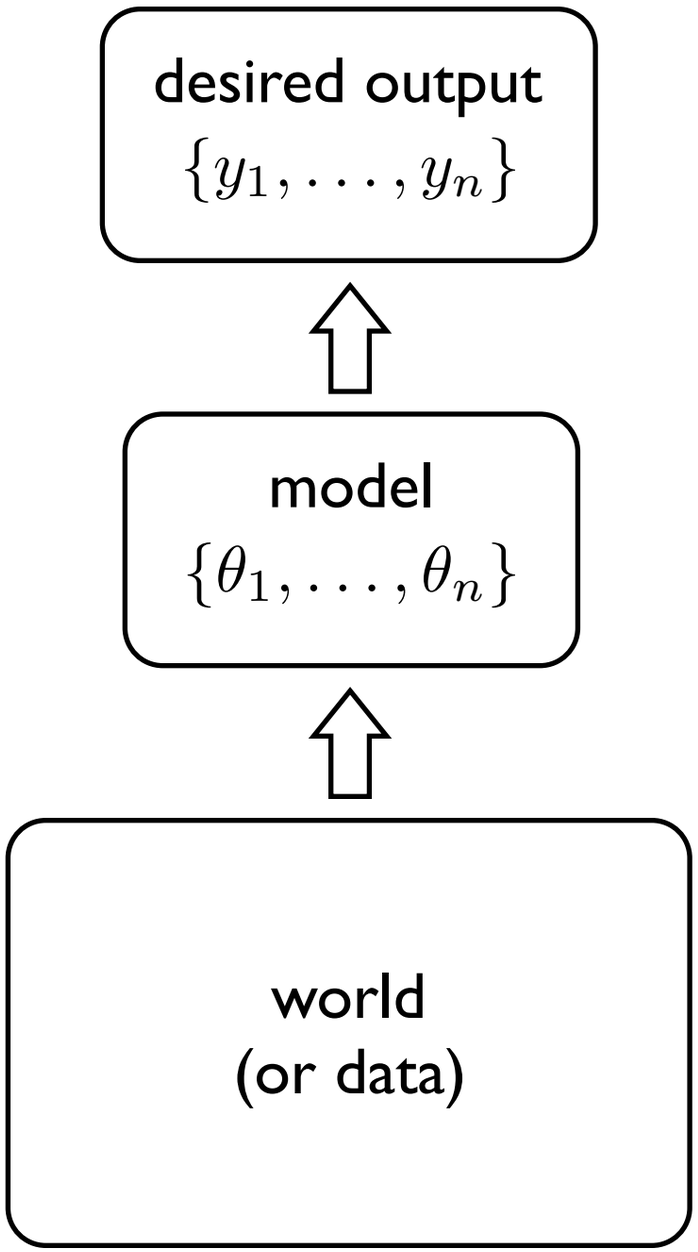
Clustering: Classification without labels

- In many situations we don't have labeled training data, only unlabeled data.
- Eg, in the iris data set, what if we were just starting and didn't know any classes?

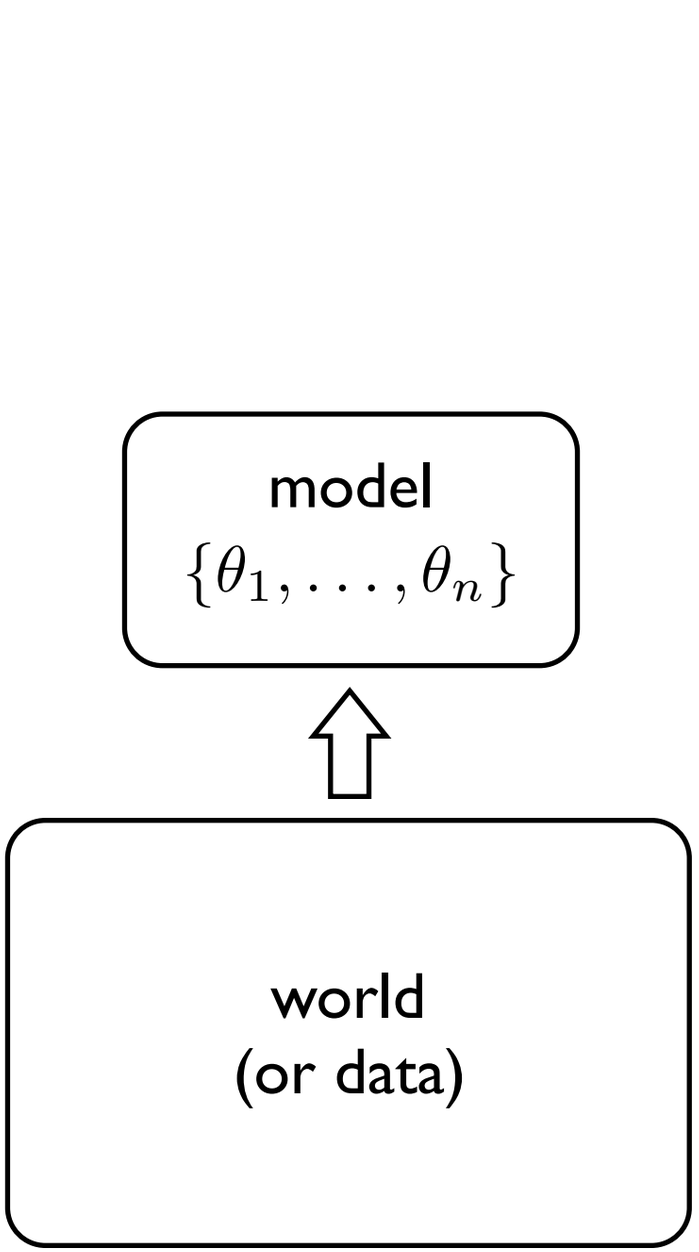


Types of learning

supervised



unsupervised



reinforcement

A different approach to classification

- Nearby points are likely to be members of the same class.
- What if we used the points themselves to classify?

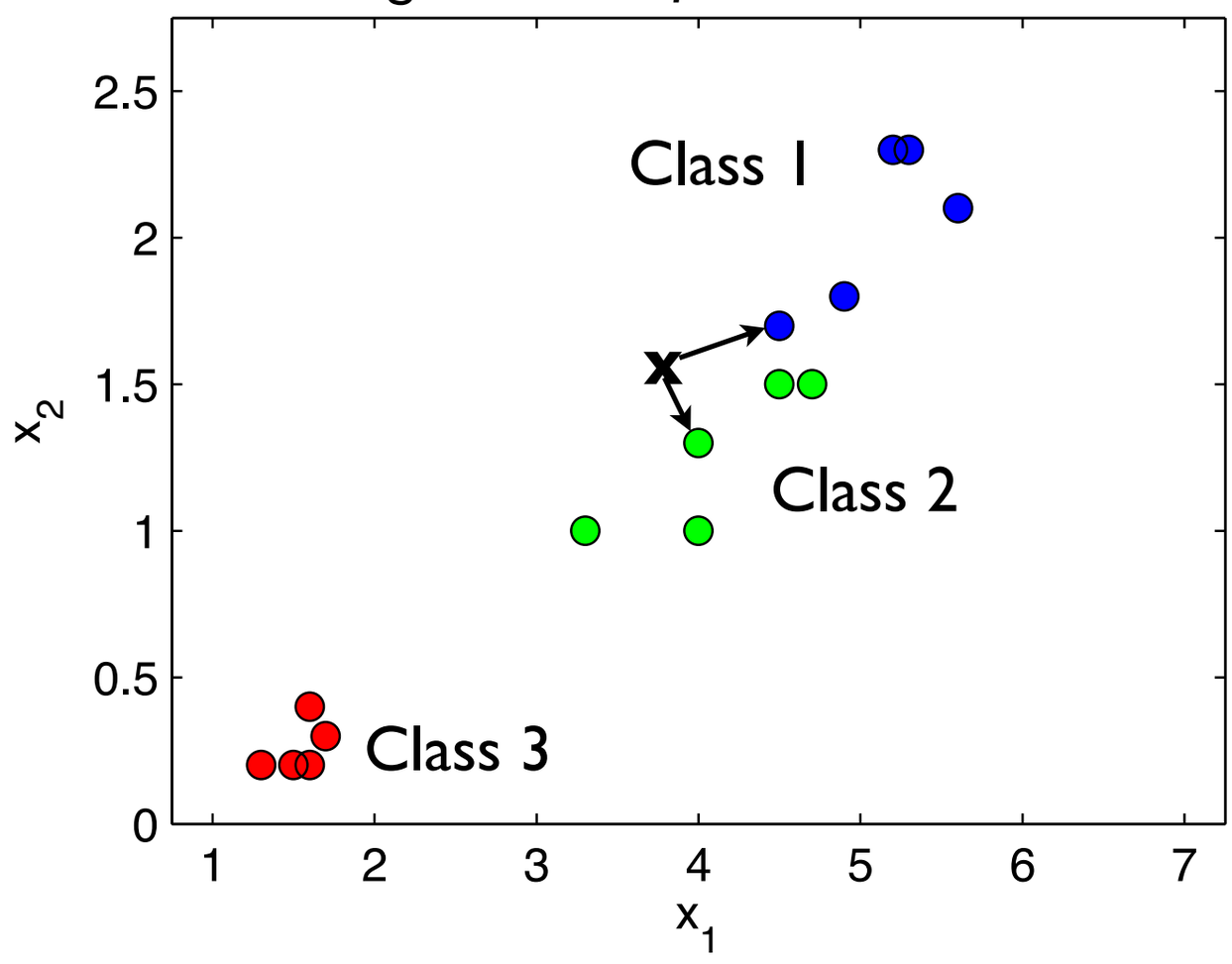
  classify $\mathbf{x}$ *in* C_k *if* $\mathbf{x}$ *is "similar" to a point we already know is in* C_k.

- Eg: unclassified point $\mathbf{x}$ is more similar Class 2 than Class 1.
- Issue: How to define "similar" ? Simplest is Euclidean distance:

$$d(\mathbf{x}, \mathbf{y}) = \sum_i (x_i - y_i)^2$$

- Could define other metrics depending on application, e.g. text documents, images, etc.

Nearest neighbor classification on the iris dataset



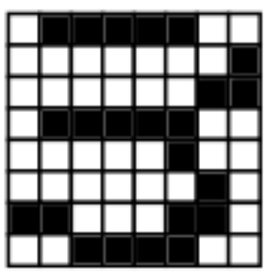
Potential advantages:

- don't need an explicit model
- the more examples the better
- might handle more complex classes
- easy to implement
- "no brain on part of the designer"

Example: Handwritten digits

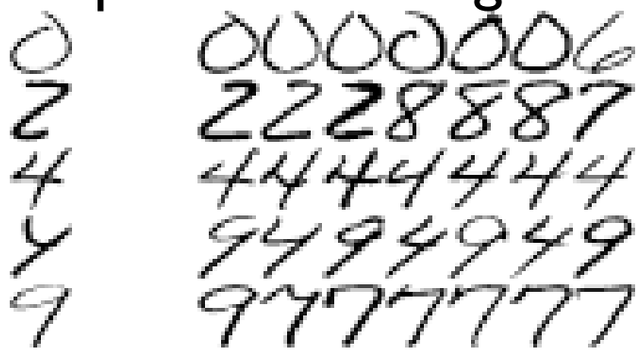
from LeCun etal, 1998
digit data available at:
http://yann.lecun.com/exdb/mnist/

Digits are just represented as a vector.



- Use Euclidean distance to see which known digit is closest to each class.
- But not all neighbors are the same:

example nearest neighbors



example from Sam Roweis

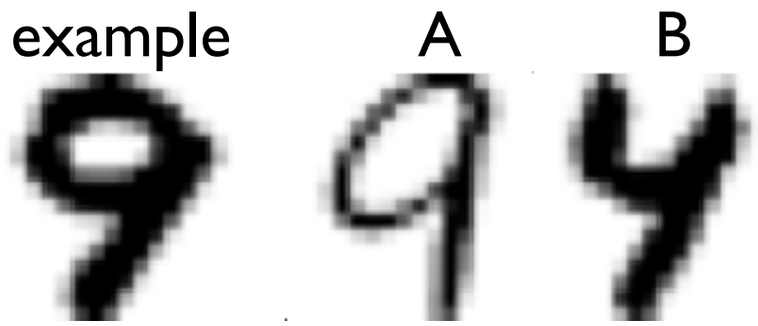
- "k-nearest neighbors":

  look at k-nearest neighbors and choose most frequent.

- Cautions: can get expensive to find neighbors

The problem of using templates (ie Euclidean distance)

- Which of these is more like the example? A or B?



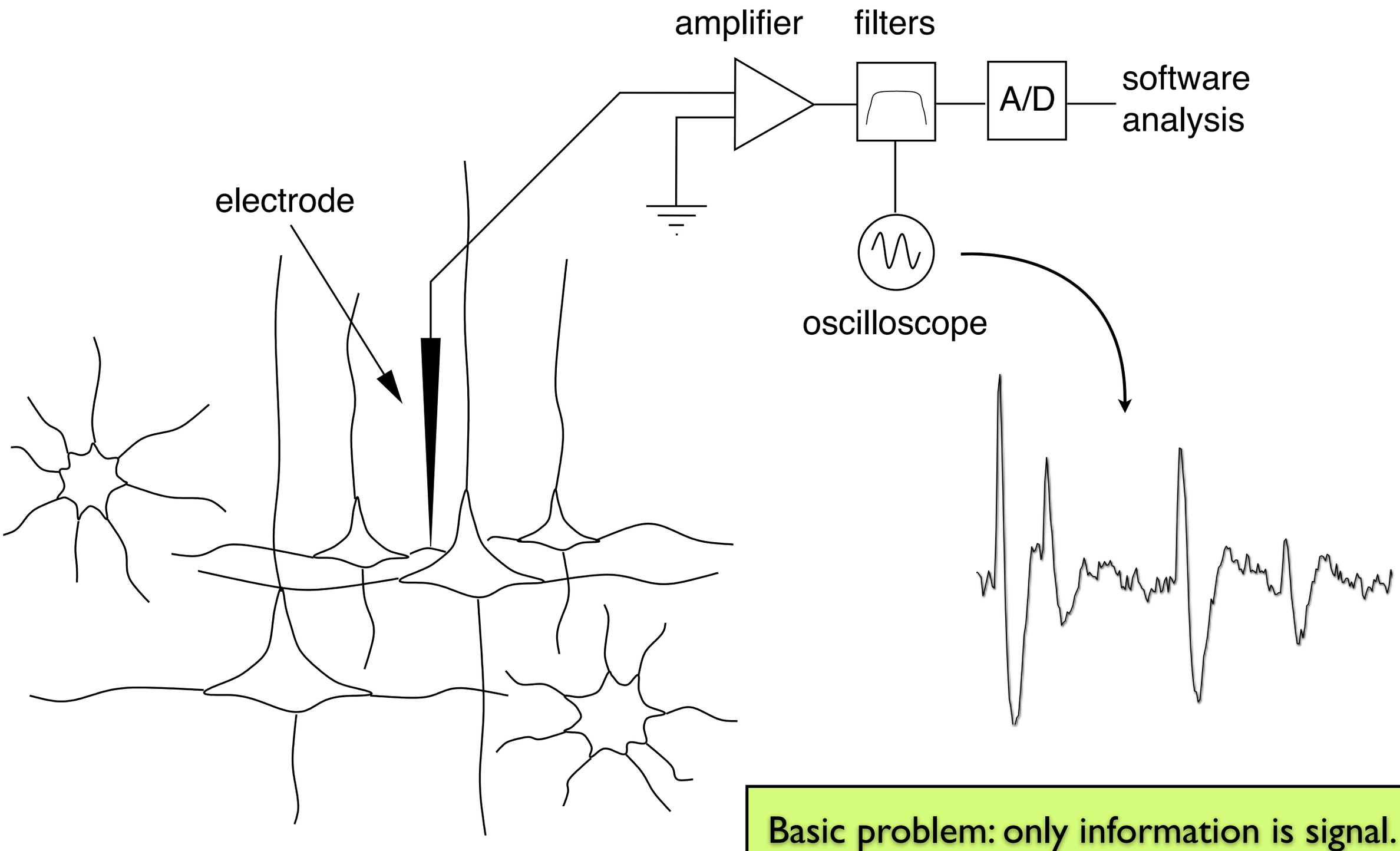
from Simard etal, 1998

- Euclidean distance only cares about how many pixels overlap.
- Could try to define a distance metric that is insensitive to small deviations in position, scale, rotation, etc.
- Digit example:
  - 60,000 training images,
  - 10,000 test images
  - no "preprocessing"

performance results of various classifiers (from http://yann.lecun.com/exdb/mnist/)

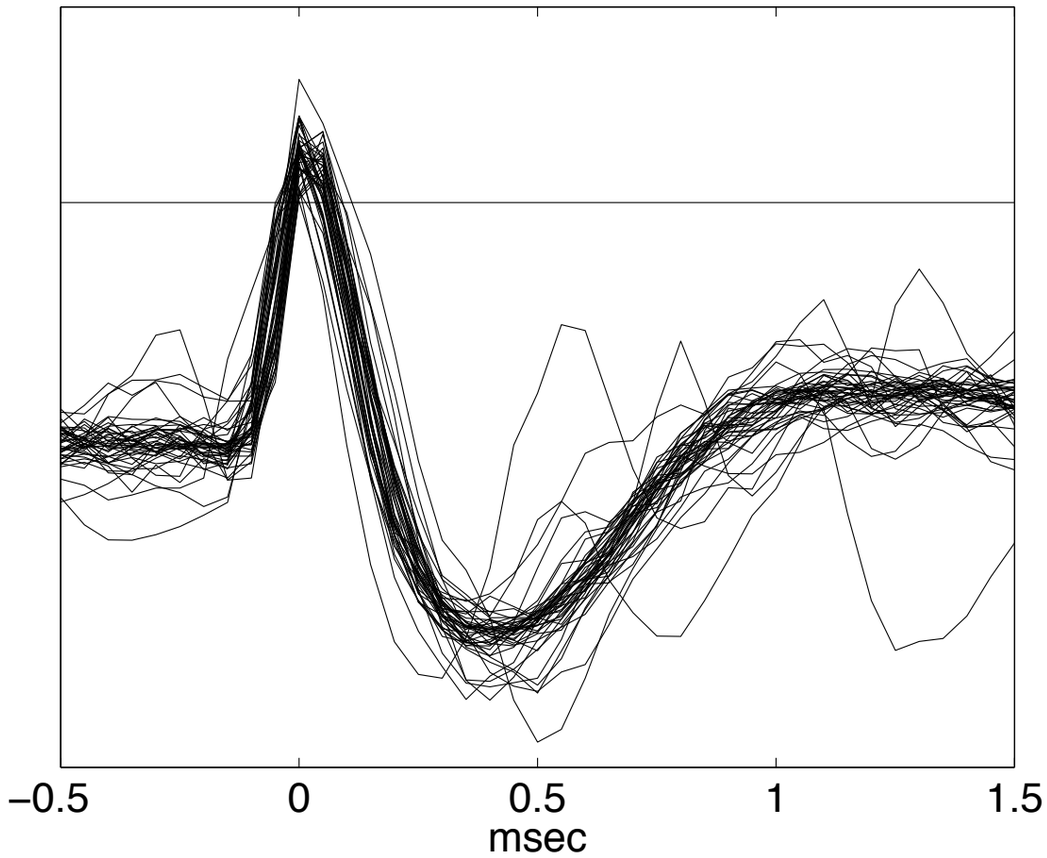
Classifier	**error rate on test data (%)**
linear	12
k=3 nearest neighbor (Euclidean distance)	5
2-layer neural network (300 hidden units)	4.7
nearest neighbor (Euclidean distance)	3.1
k-nearest neighbor (improved distance metric)	1.1
convolutional neural net	0.95
best (the conv. net with elastic distortions)	0.4
humans	0.2 - 2.5

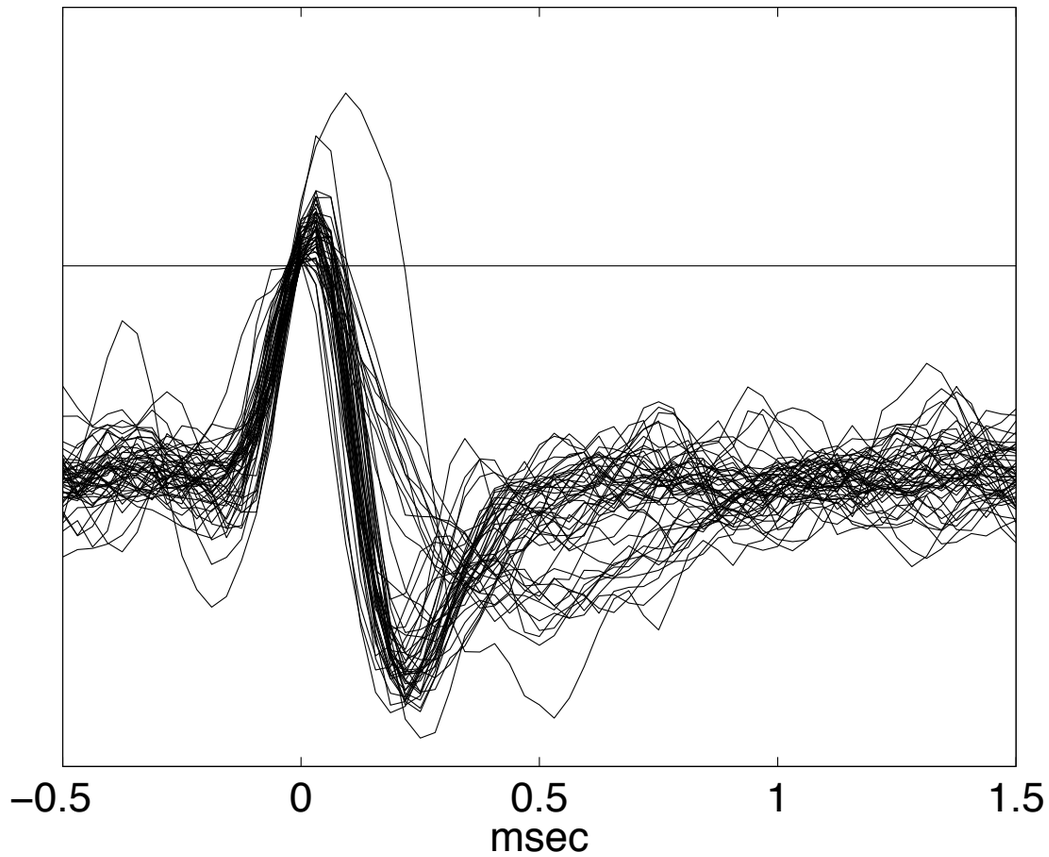
A real example: clustering electrical signals from neurons



Basic problem: only information is signal.
The true classes are *always* unknown.

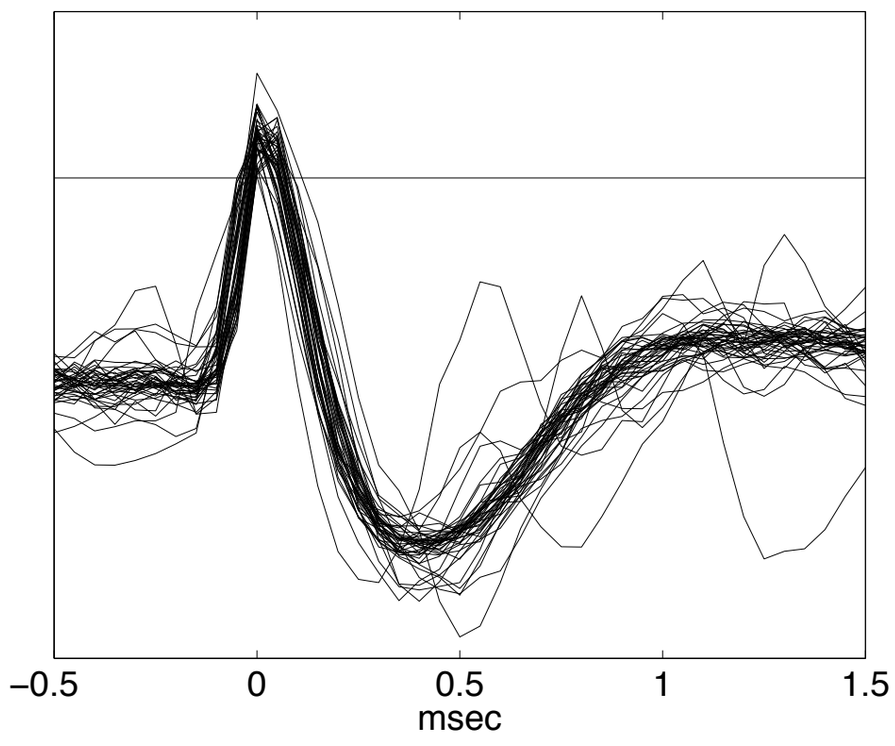
Sorting with level detection on an oscilloscope

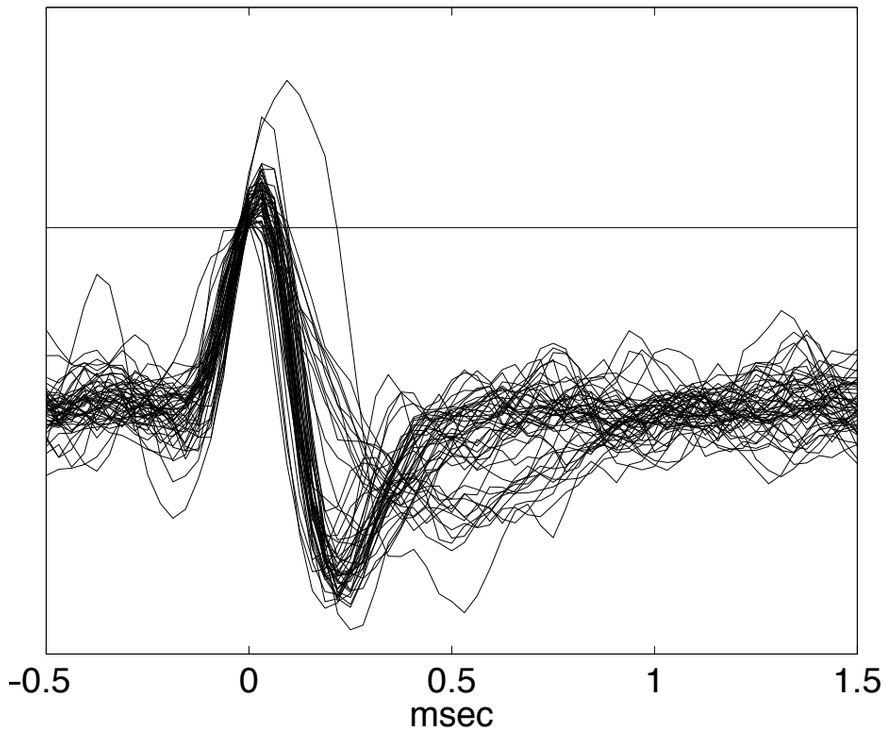


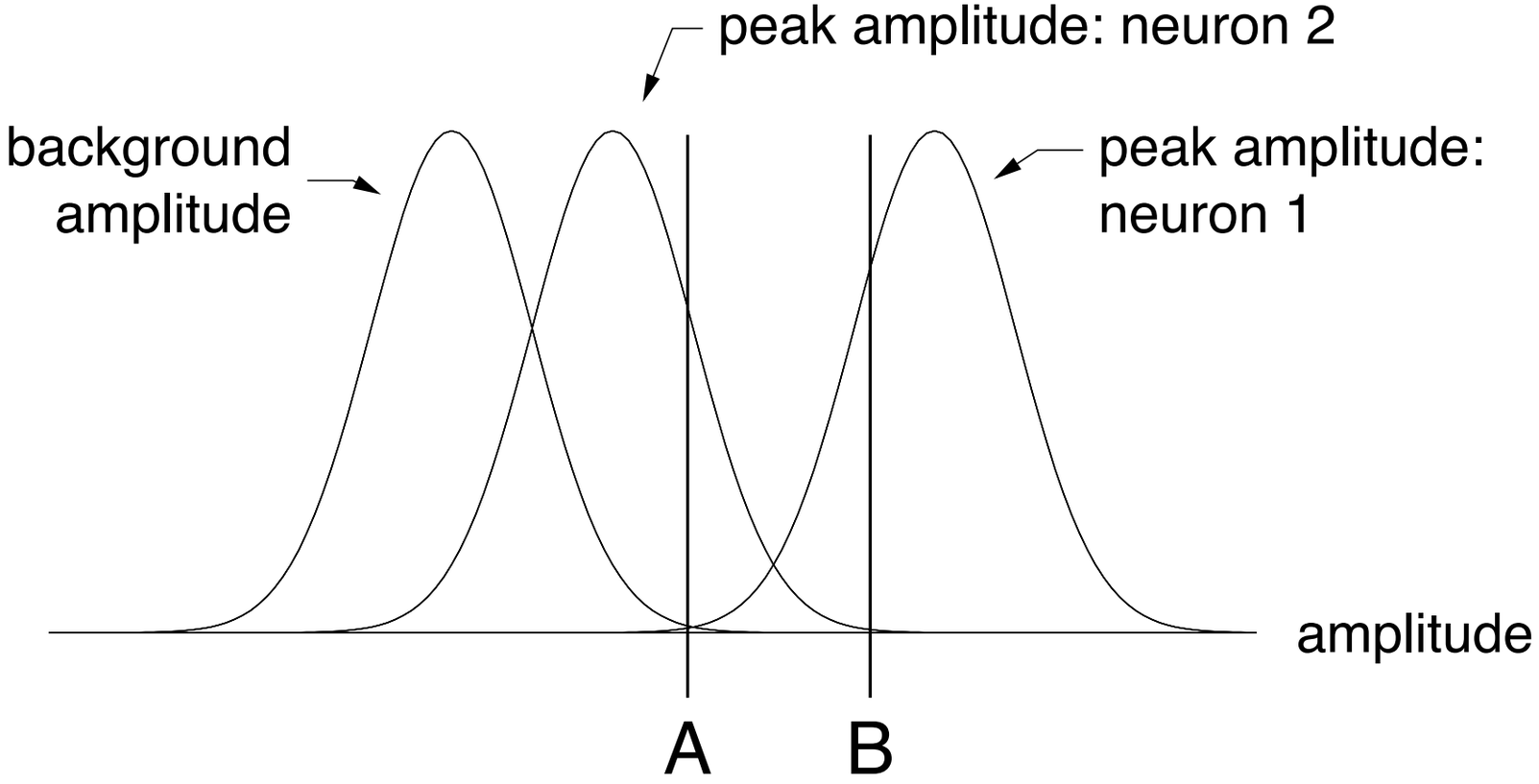


Level detection doesn't always work.

Why level detection doesn't work

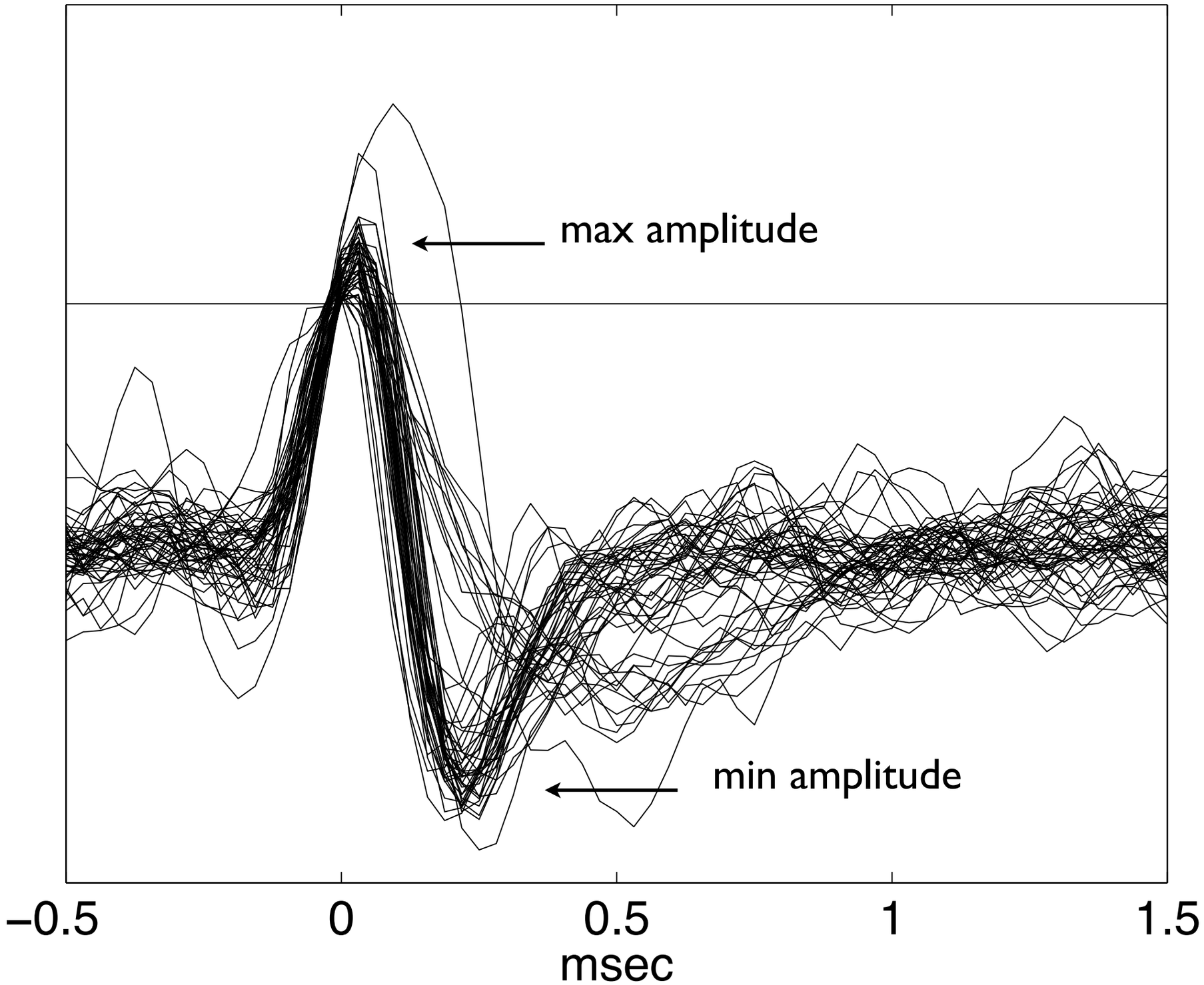




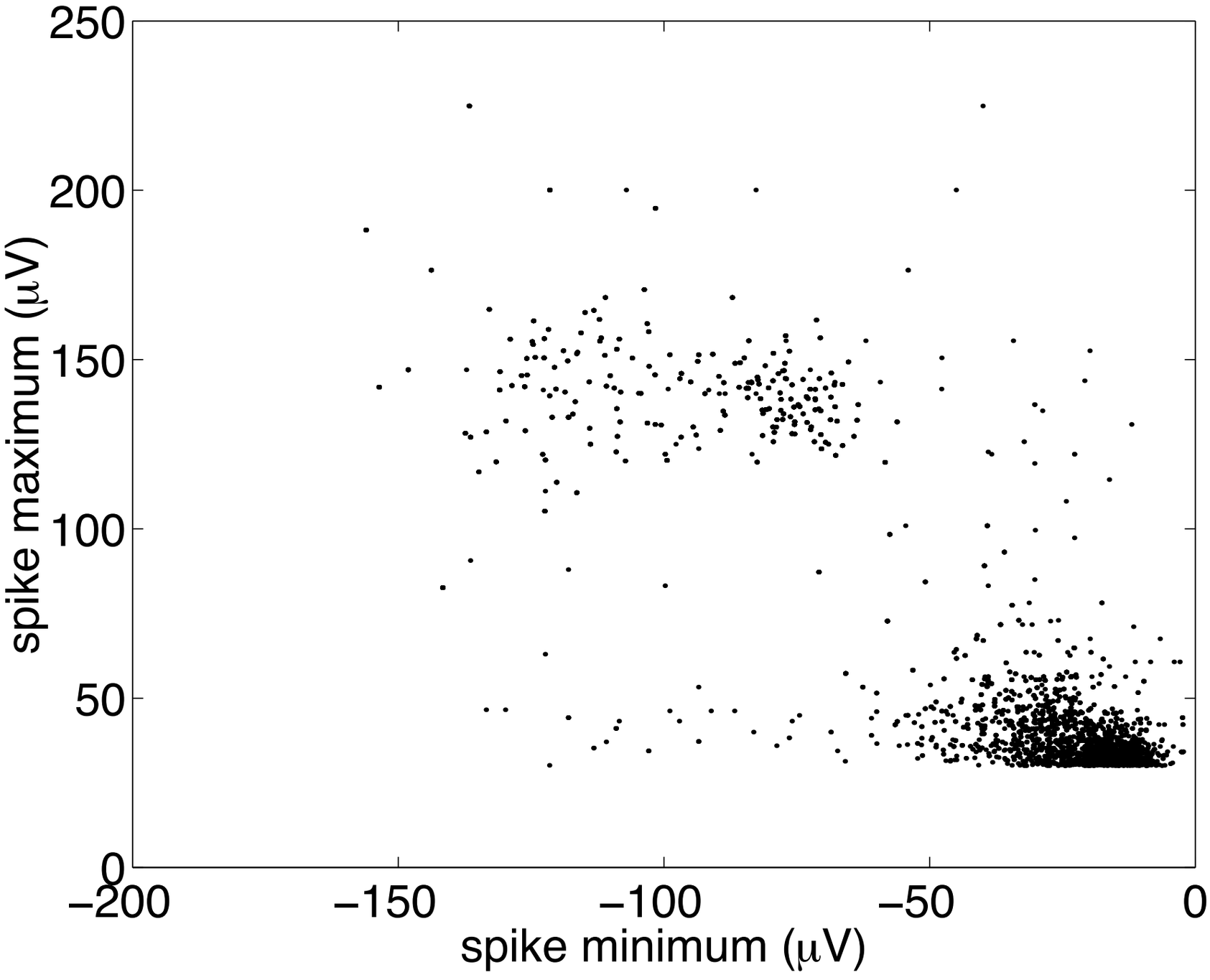


One dimension is not sufficient to separate the spikes.

Idea: try more features

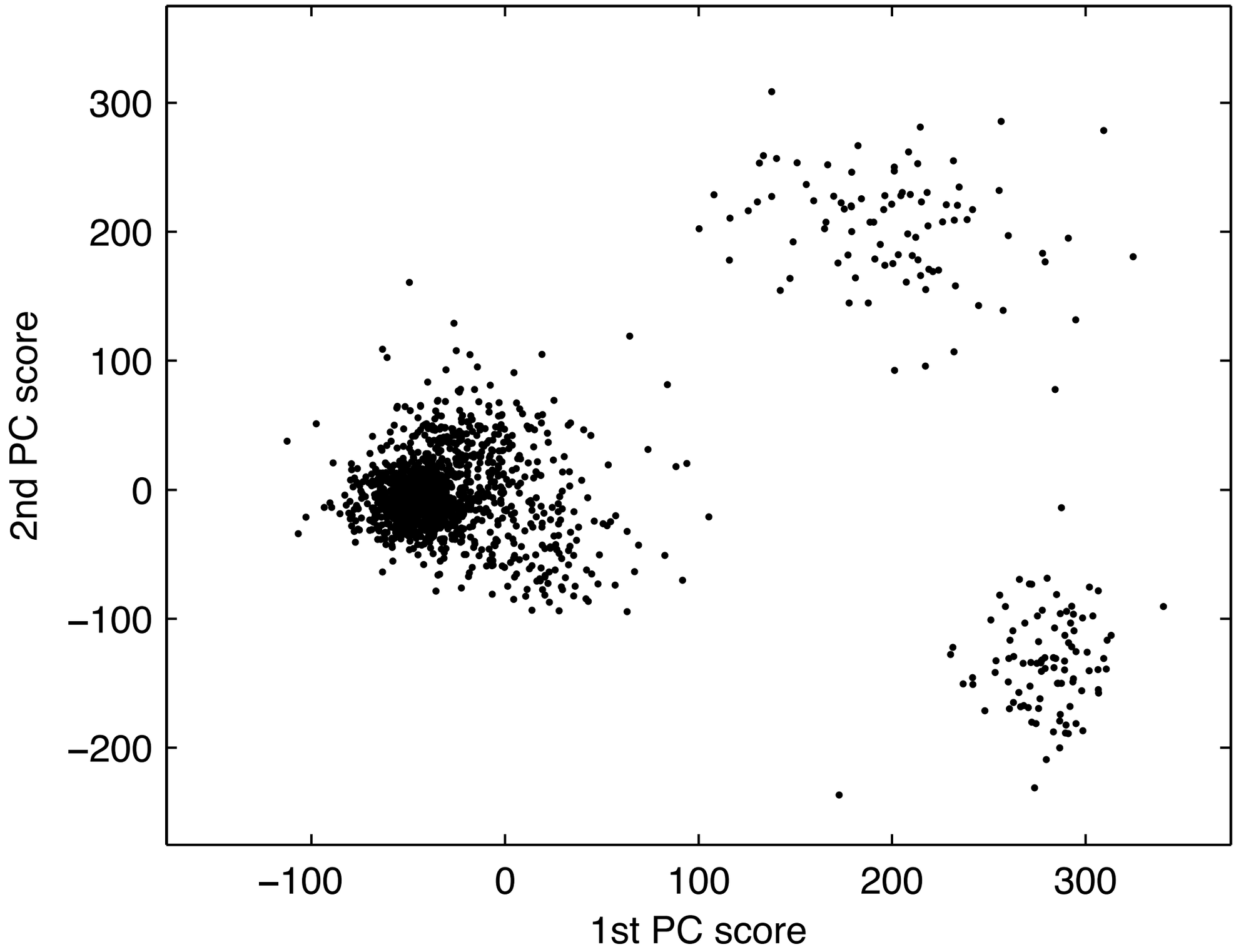


Maximum vs minimum



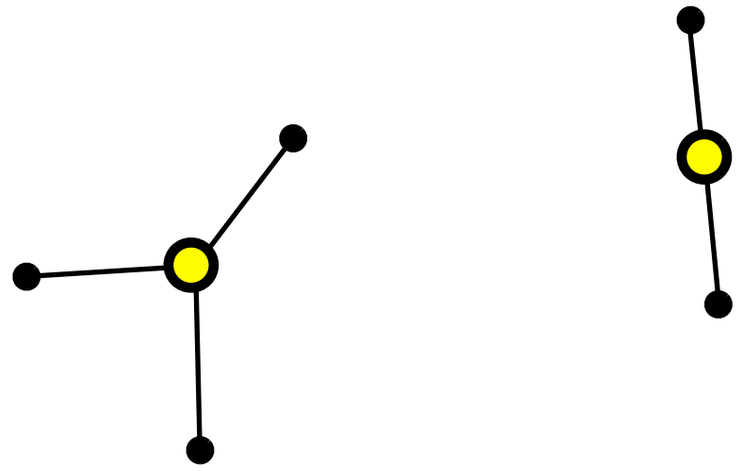
This allows better discrimination than max alone, but is it optimal?

Features using Principal Components (not covered)



k-means clustering

- Idea: try to estimate k cluster centers by minimizing "distortion"
- Define distortion as:



$$D = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk} \parallel \mathbf{x}_n - \boldsymbol{\mu}_k \parallel^2$$

$$r_{nk} = 1 \text{ if } \mathbf{x}_n \in \text{cluster } k, \text{ 0 otherwise.}$$

- r_{nk} is 1 for the closest cluster mean to $\mathbf{x}_n$.
- Each point $\mathbf{x}_n$ is the minimum distance from its closet center.
- How do we learn the cluster means?
- Need to derive a ***learning rule***.

Deriving a learning rule for the cluster means

- Our objective function is:

$$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \parallel \mathbf{x}_n - \boldsymbol{\mu}_k \parallel^2$$

- Differentiate w.r.t. to the mean (the parameter we want to estimate):

$$\frac{\partial D}{\partial \boldsymbol{\mu}_k} = 2 \sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k)$$

- We know the optimum is when

$$\frac{\partial D}{\partial \boldsymbol{\mu}_k} = 2 \sum_{n=1}^{N} r_{nk}(\mathbf{x}_n - \boldsymbol{\mu}_k) = 0$$

- Here, we can solve for the mean:

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}}$$

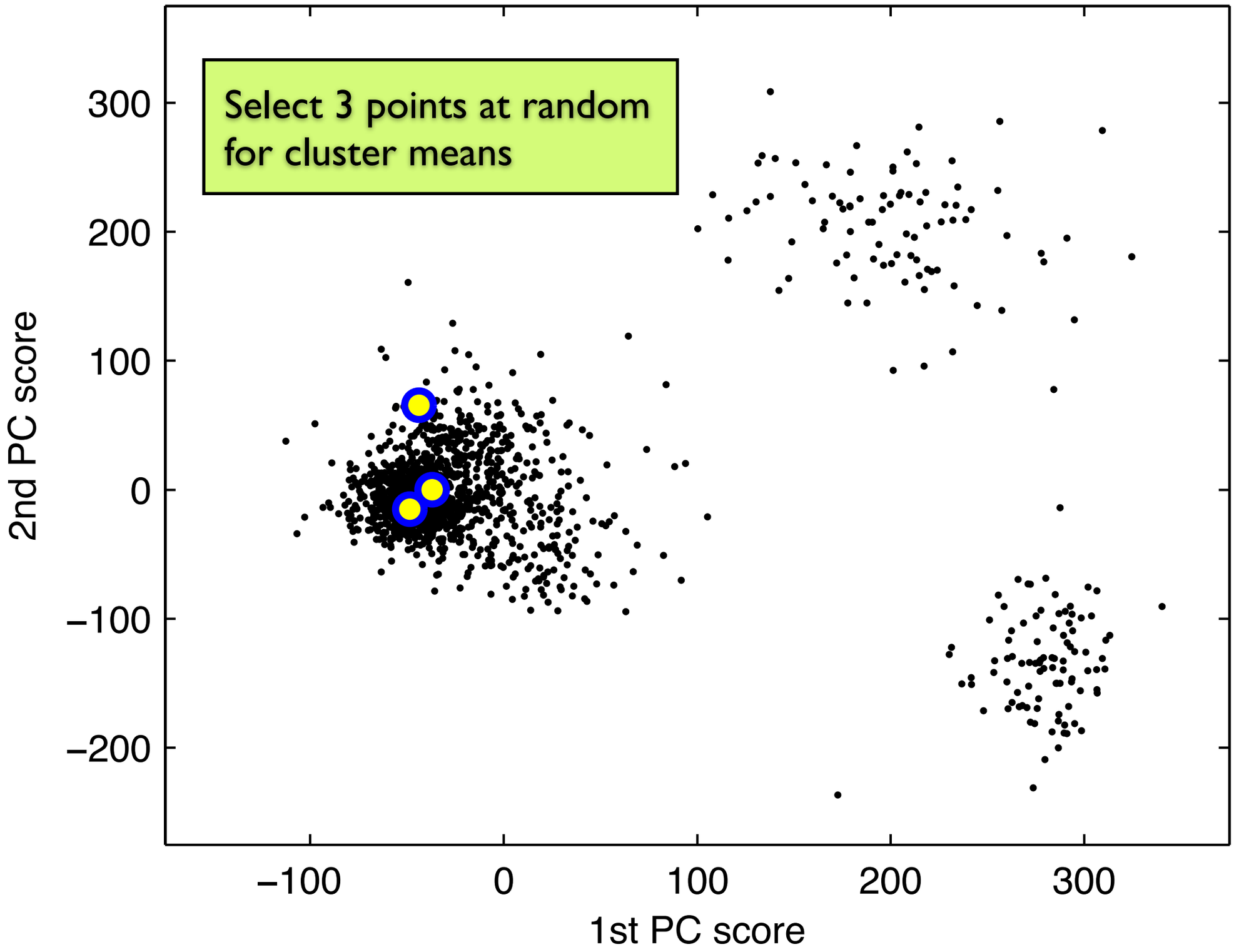
- This is simply a weighted mean for each cluster.

- Thus we have a simple estimation algorithm (*k-means clustering*)

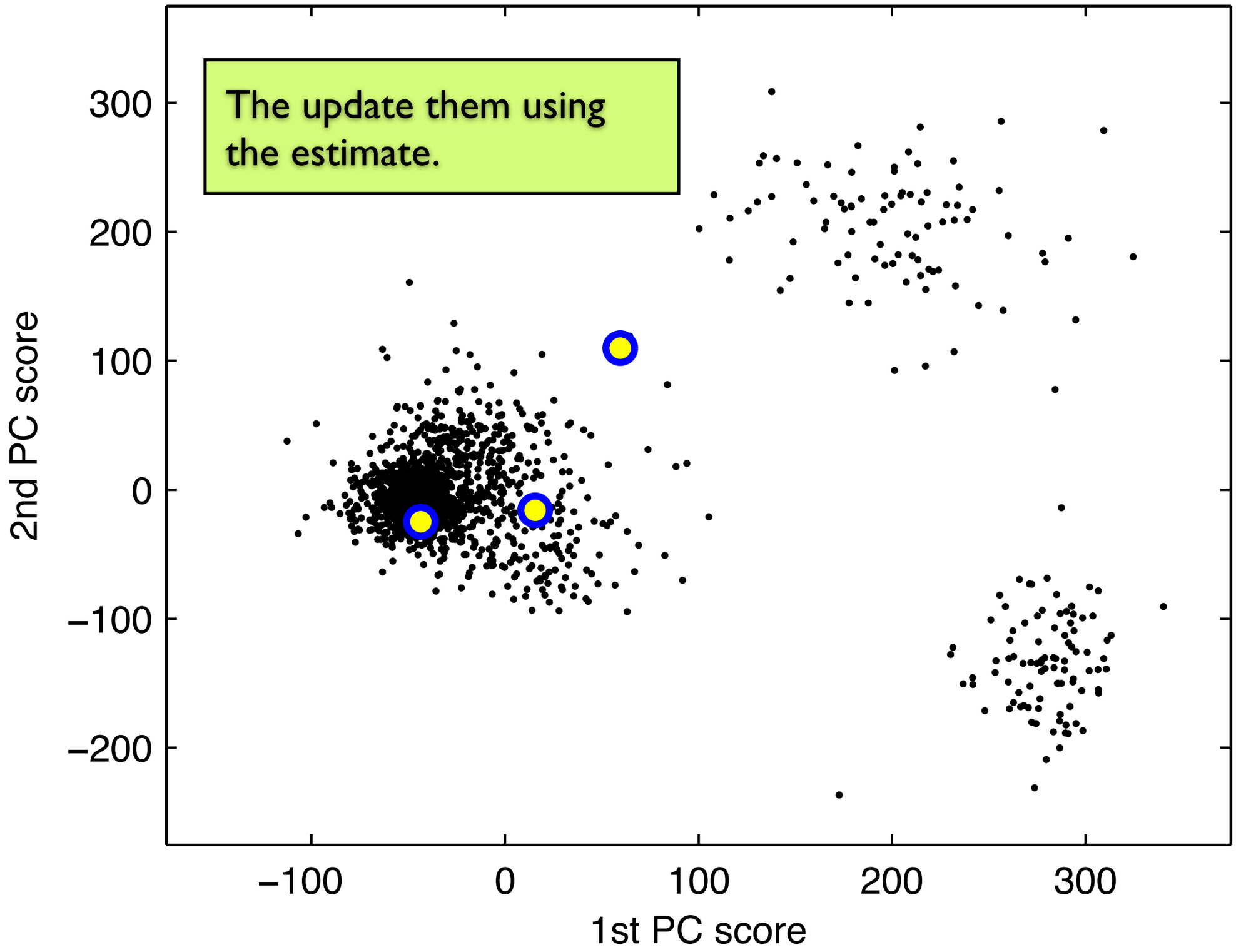
  1. select k points at random
  2. estimate (update) means
  3. repeat until converged

- convergence (to a local minimum) is guaranteed

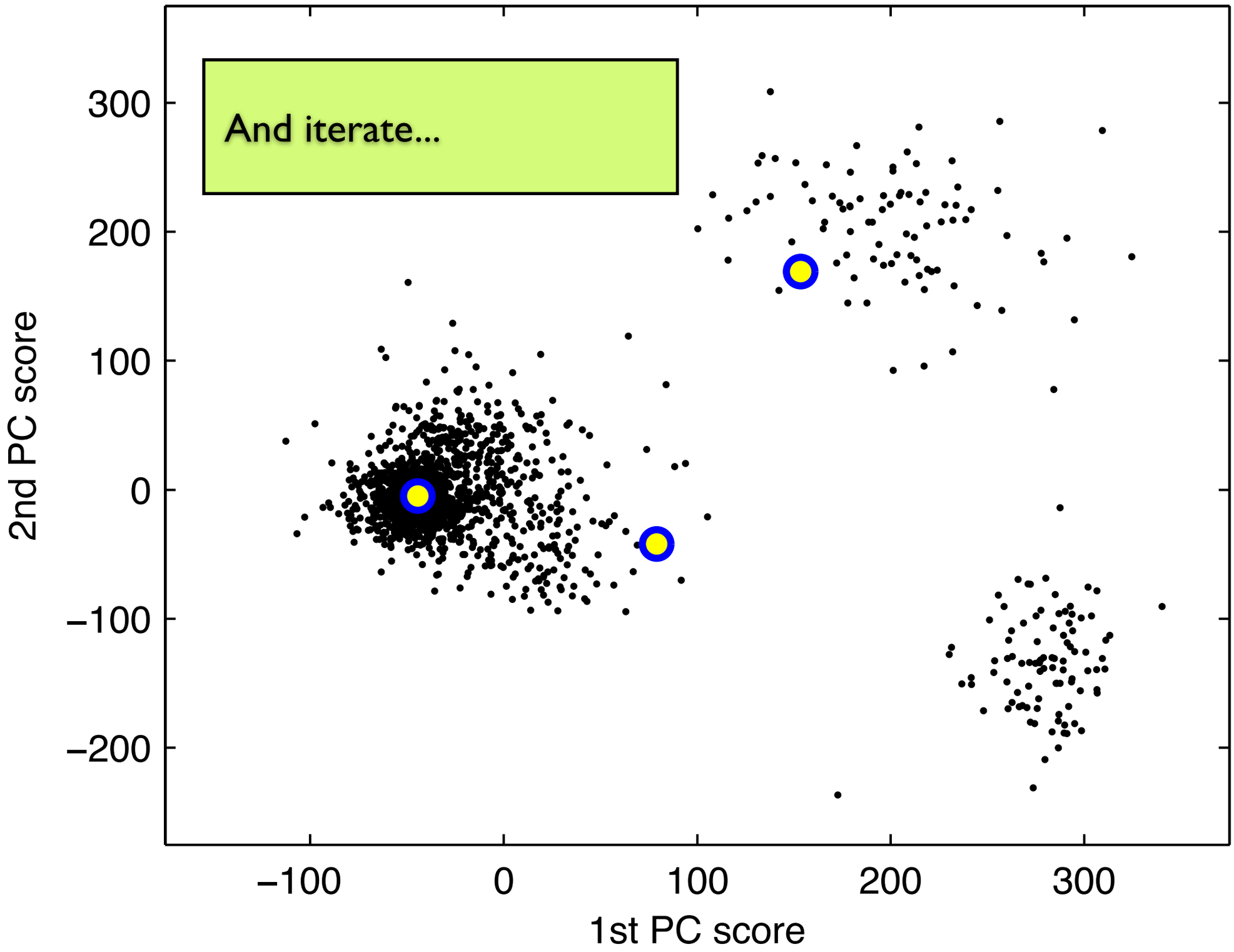
k-means clustering example



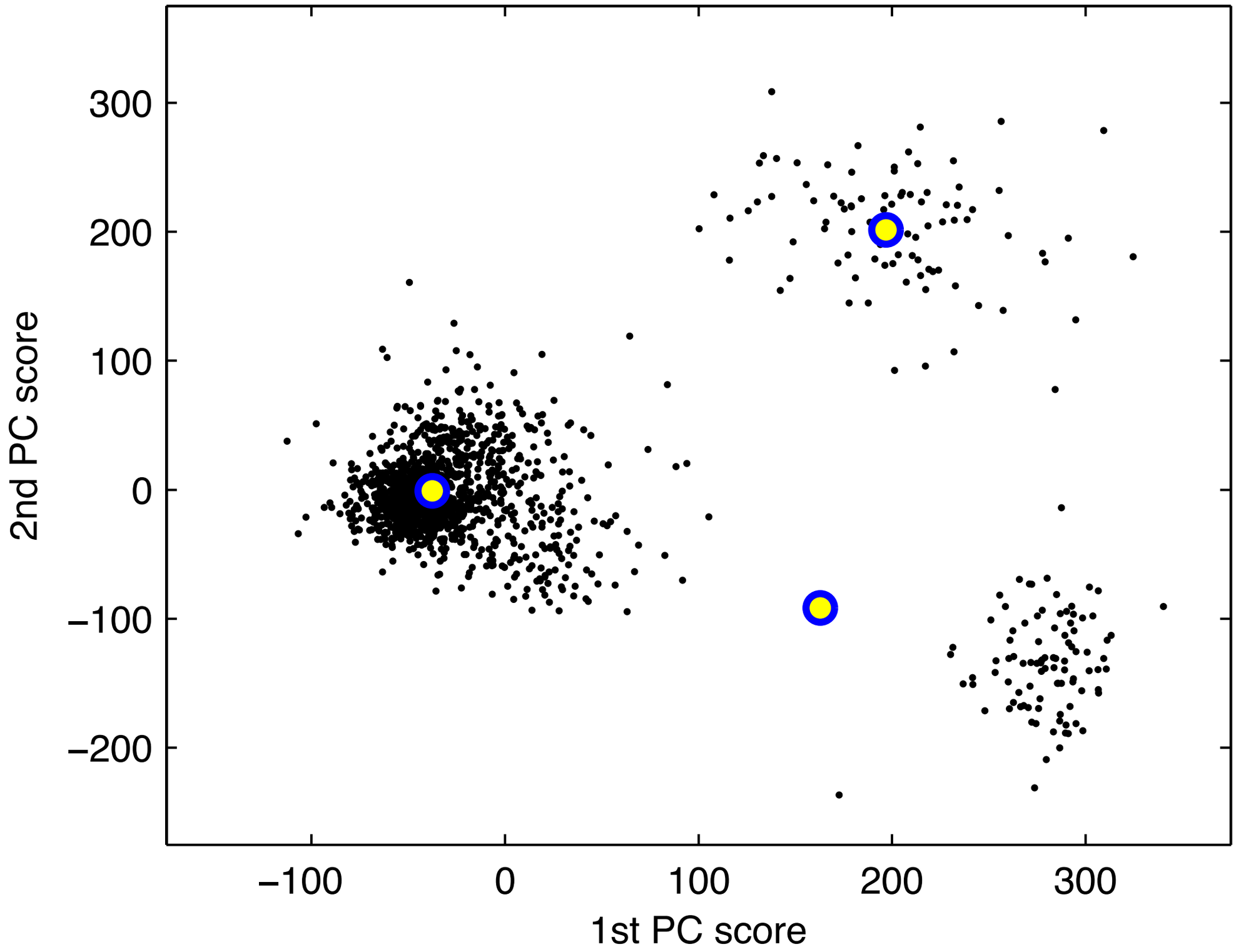
k-means clustering example



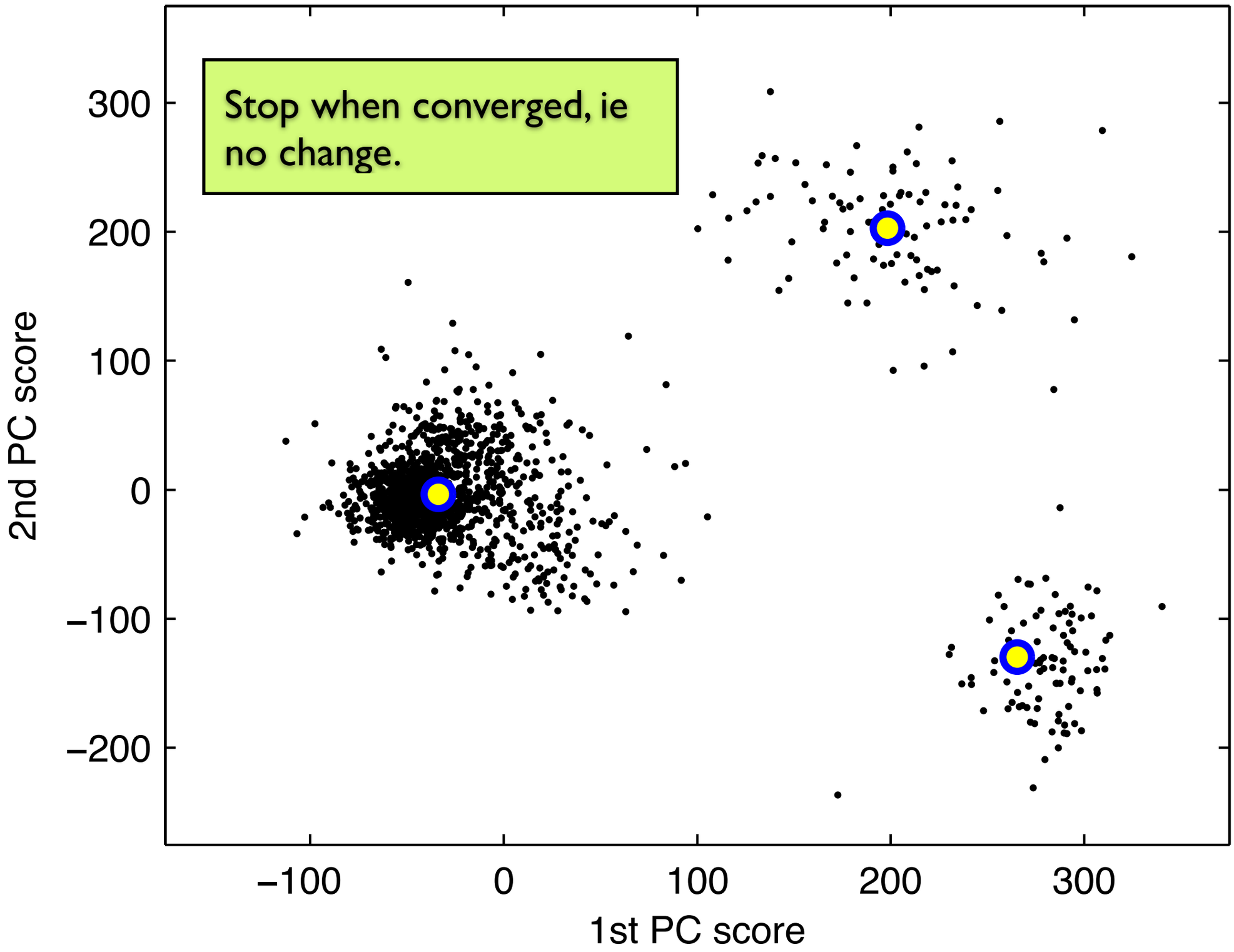
k-means clustering example



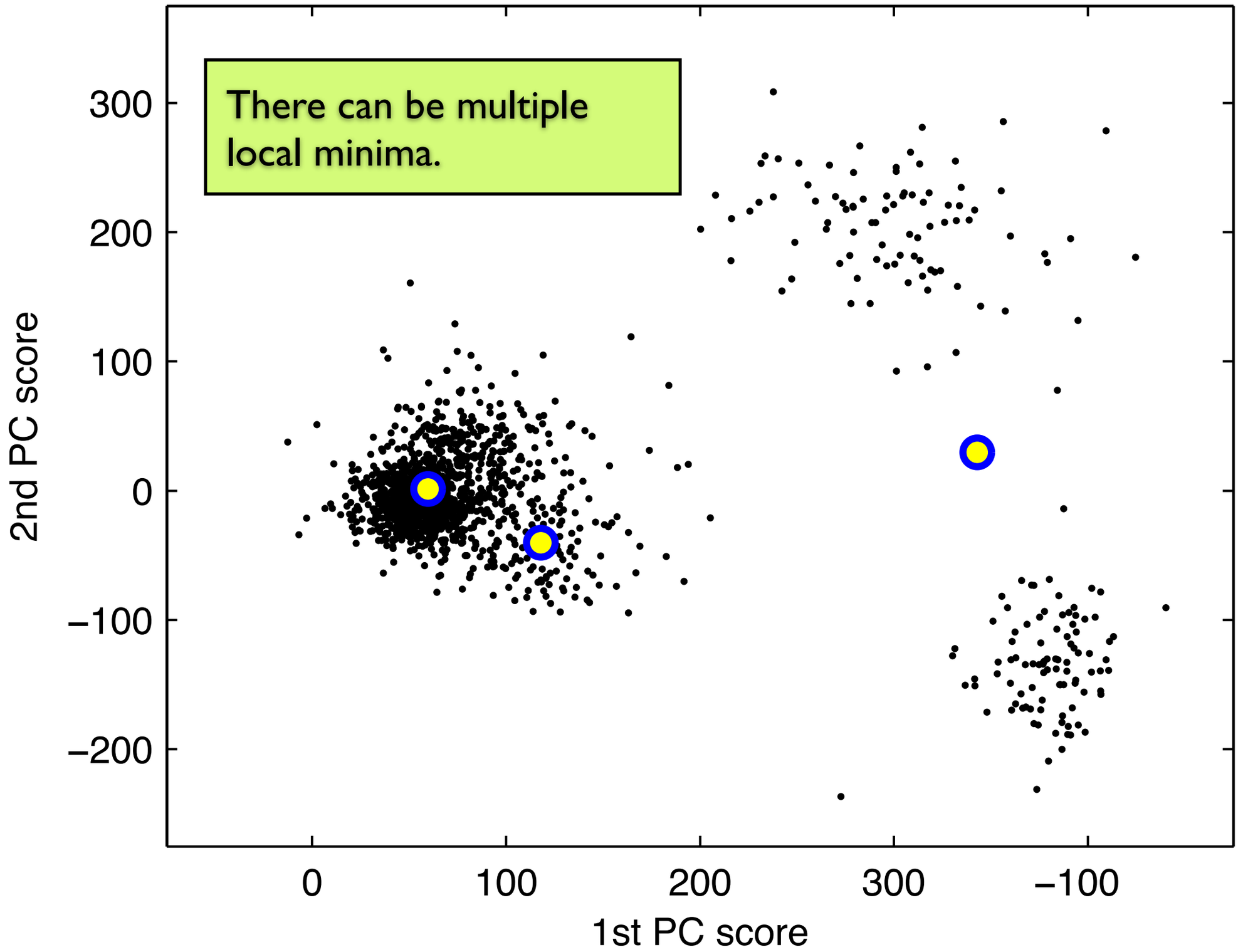
k-means clustering example



k-means clustering example

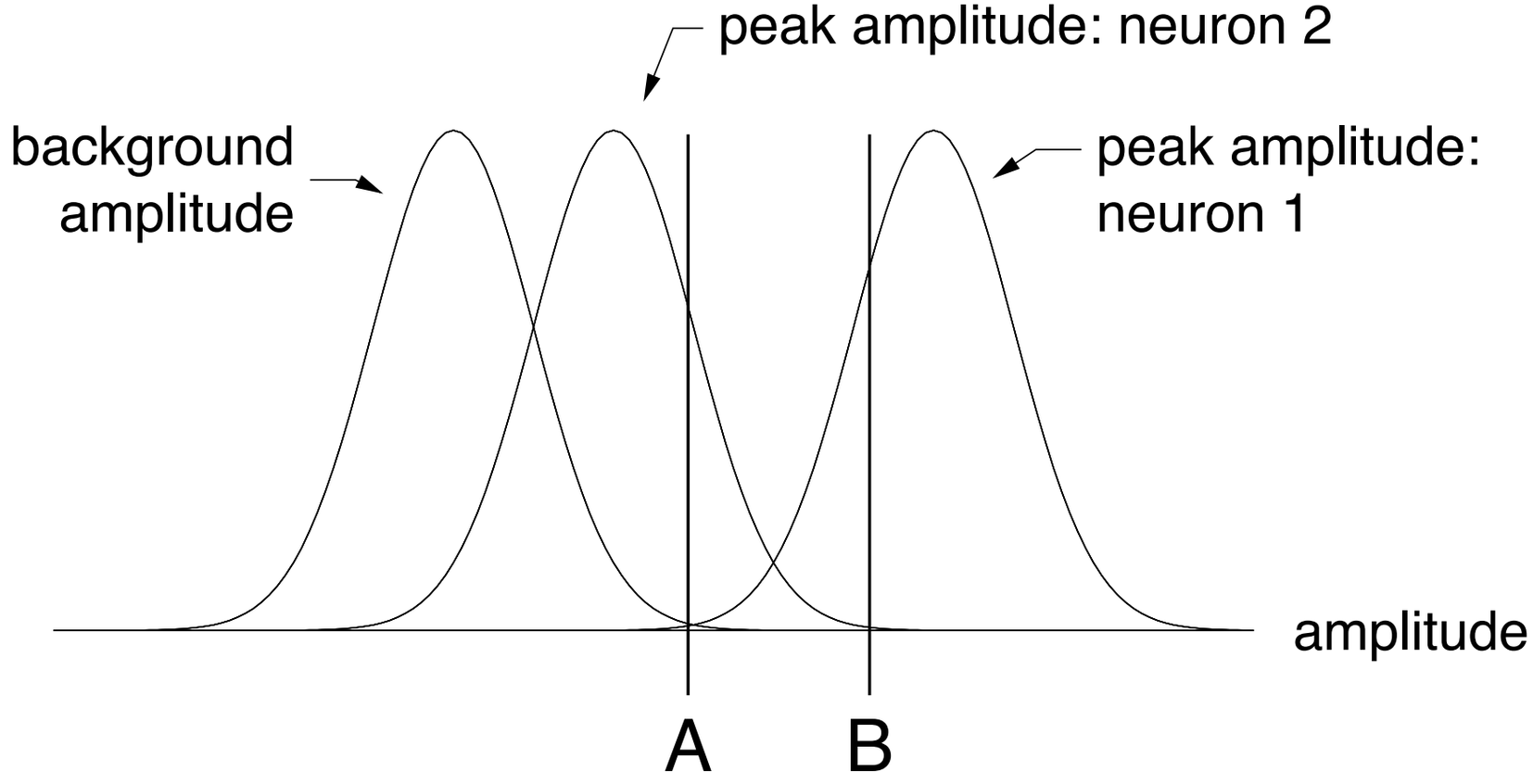


An example of a local minimum



A probabilistic interpretation: Gaussian mixture models

- We've already seen a one-dimensional version
- This example has three classes: neuron 1, neuron 2, and background noise.
- Each can be modeled as a Gaussian
- Any given data point comes from just *one* Gaussian
- The whole set of data is modeled by a *mixture* of three Gaussians
- How do we model this?



The Gaussian mixture model density

- The likelihood of the data given a particular class c_k is given by

$$p(x|c_k, \mu_k, \Sigma_k)$$

- x is the spike waveform, μ_k and Σ_k are the mean and covariance for class c_k.

- The marginal likelihood is computed by summing over the likelihood of the K classes

$$p(x|\theta_{1:\mathrm{K}}) = \sum_{k=1}^{K} p(x|c_k, \theta_k)p(c_k)$$

- $\theta_{1:\mathrm{K}}$ defines the parameters for all of the classes, $\theta_{1:\mathrm{K}} = \{\mu_1, \Sigma_1, \ldots, \mu_K, \Sigma_K\}$.

- $p(c_k)$ is the probability of the kth class, with $\sum_k p(c_k) = 1$.

- What does this mean in this example?

Bayesian classification with multivariate Gaussian mixtures

- How do we determine the class c_k from the data x ?

- Again use Bayes' rule

$$p(c_k|x^{(n)}, \theta_{1:\mathrm{K}}) = p_{k,n} = \frac{p(x^{(n)}|c_k, \theta_k)p(c_k)}{\sum_k p(x^{(n)}|c_k, \theta_k)p(c_k)}$$

- This tells is the probability that waveform $x^{(n)}$ came from class c_k.

Estimating the parameters: fitting the model density to the data

- The objective of density estimation is to maximize the likelihood of the data
- If we assume the samples are independent, the data likelihood is just the product of the *marginal* likelihoods

$$p(x_{1:\mathrm{N}}|\theta_{1:\mathrm{K}}) = \prod_{n=1}^{N} p(x_n|\theta_{1:\mathrm{K}})$$

- The class parameters are determined by optimization.
- Is far more practical to optimize the log-likelihood.
- One elegant approach to this is the EM algorithm.

The Gaussian mixture

- EM stands for Expectation-Maximization, and involves two steps that are iterated. For the case of a Gaussian mixture model:

  1. E-step: Compute $p_{n,k} = p(c_k|x^{(n)}, \theta_{1:\mathrm{K}})$. Let $p_k = \sum_n p_{i,n}$
  2. M-step: Compute new mean, covariance, and class prior for each class:

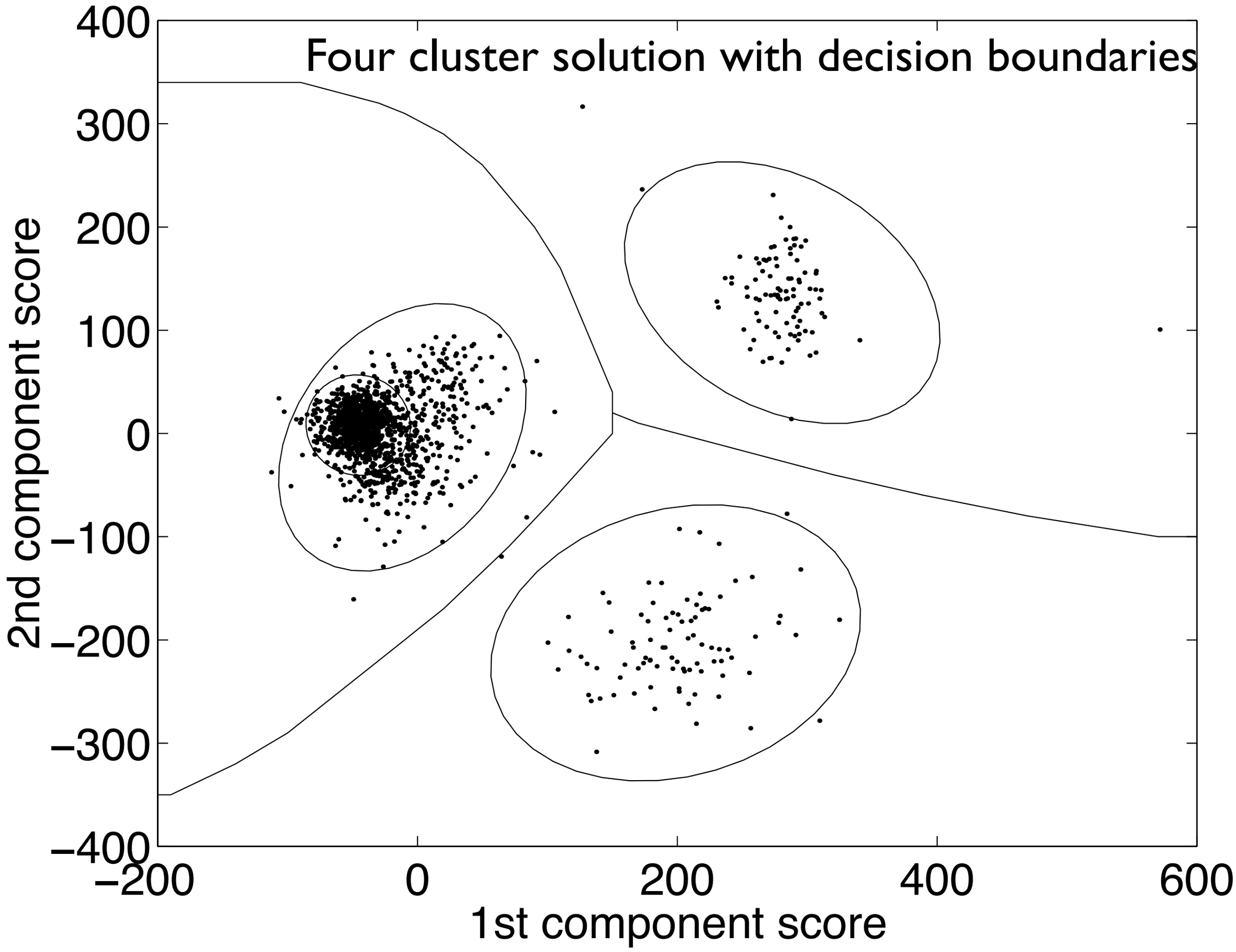
$$\mu_k \leftarrow \sum_n p_{n,k} x^{(n)} / p_k$$

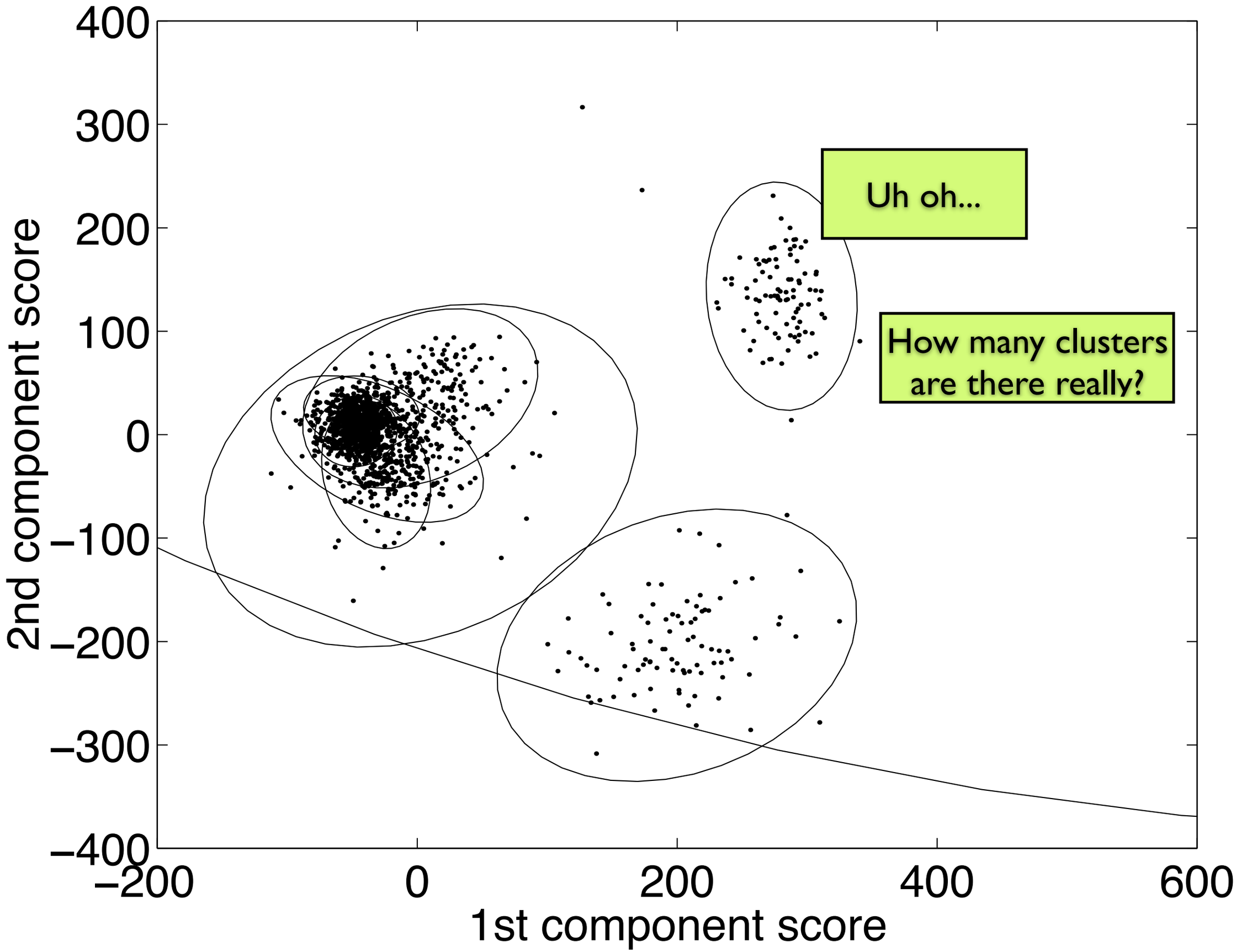
$$\Sigma_k \leftarrow \sum_n p_{n,k} (x^{(n)} - \mu_k)(x^{(n)} - \mu_k)^T / p_k$$

$$p(c_k) \leftarrow p_k$$

- This is just the sample mean and covariance, weighted by the class conditional probabilities $p_{n,k}$.

- Derived by solving setting log-likelihood gradient to zero (i.e. the maximum).


Four cluster solution with decision boundaries
2nd component score
1st component score
400
300
200
100
0
−100
−200
−300
−400
−200
0
200
400
600


Uh oh...
How many clusters are there really?
2nd component score
1st component score
400
300
200
100
0
−100
−200
−300
−400
−200
0
200
400
600

How do we choose k?

- *Increasing* k, will always *decrease* our distortion. This will ***overfit*** the data.
  - How can we avoid this?
  - Or how do we choose the best k?
- One way: ***cross validation***
- Use our distortion metric:

$$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \parallel \mathbf{x}_n - \boldsymbol{\mu}_k \parallel^2$$

- Then just measure the distortion on a *test data set*, and stop when we reach a minimum.