CS545 – Machine Learning for Signal Processing

Classification:
The rest of the story

1 October 2023

University of Illinois at Urbana-Champaign

Today's lecture

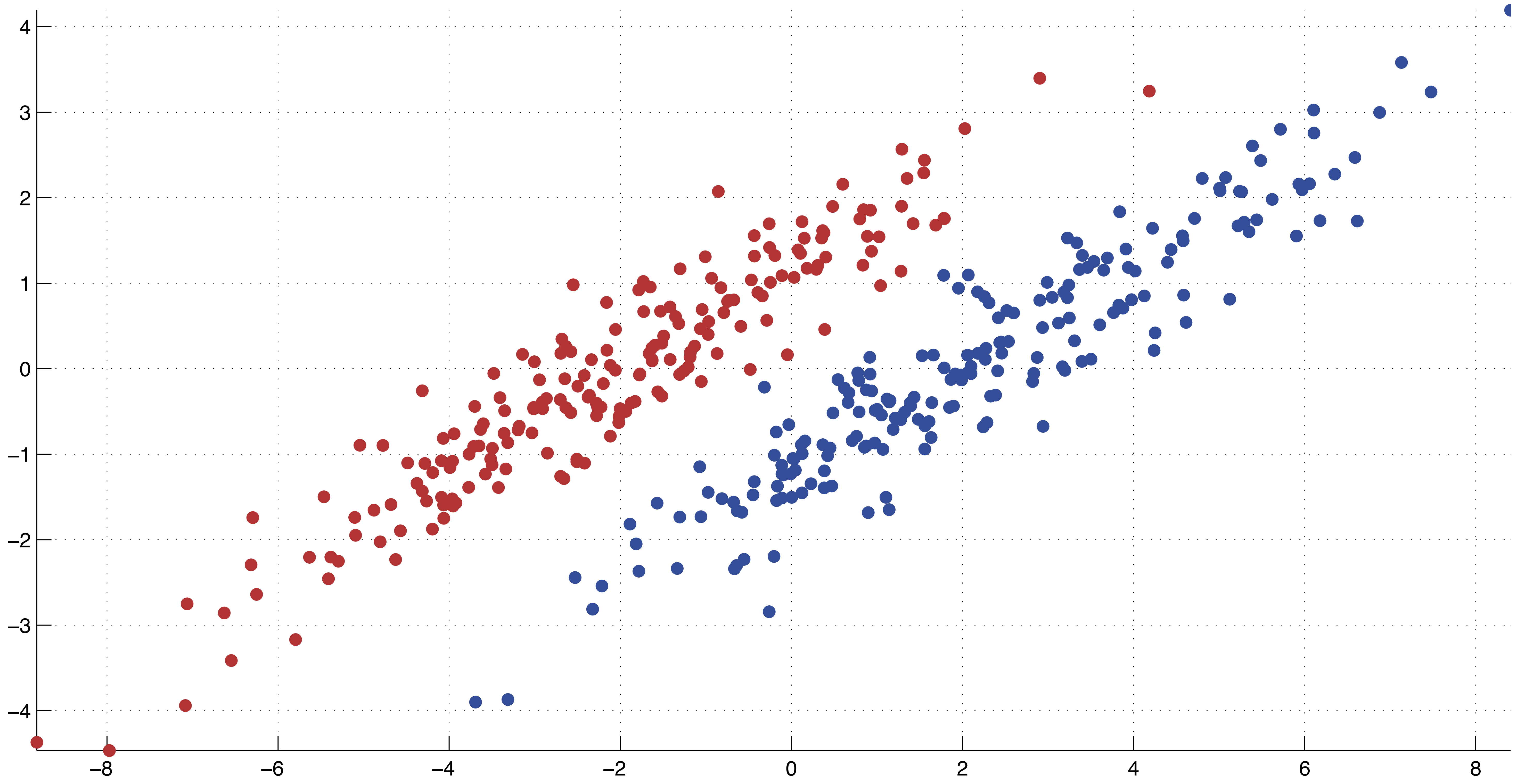
- Important things we haven't covered yet
  - Fisher Linear Discriminant Analysis
  - Nearest-neighbors classification
  - Combining multiple classifiers (boosting)
  - Multiple class classification
- Setting up experiments
  - Strategy, caveats, tradeoffs

Optimal feature discovery

- What does PCA do?
- Does that help us classify better?
- Is it doing the right thing?

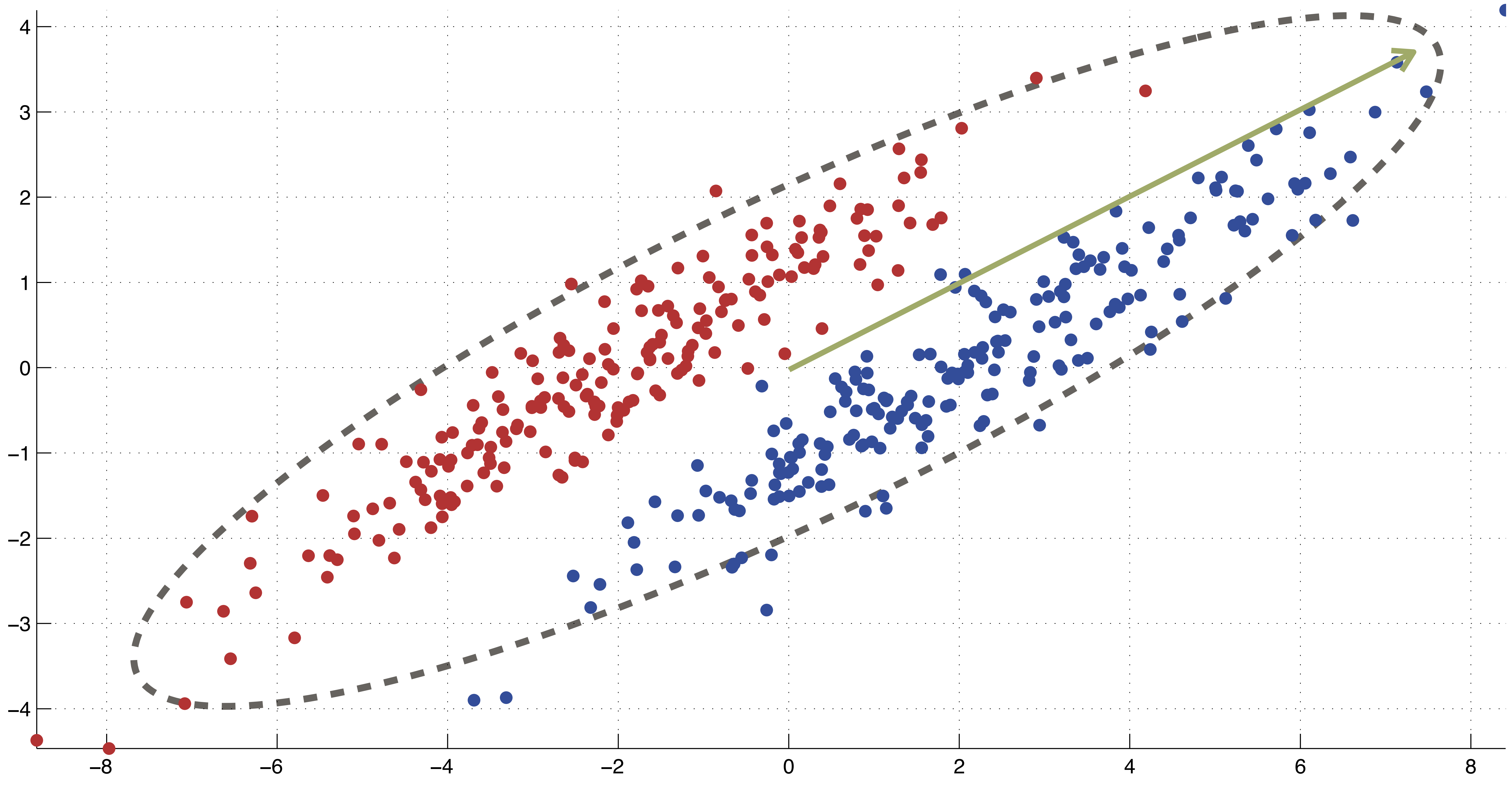
An example

- 2D, two classes
  - What would PCA do?



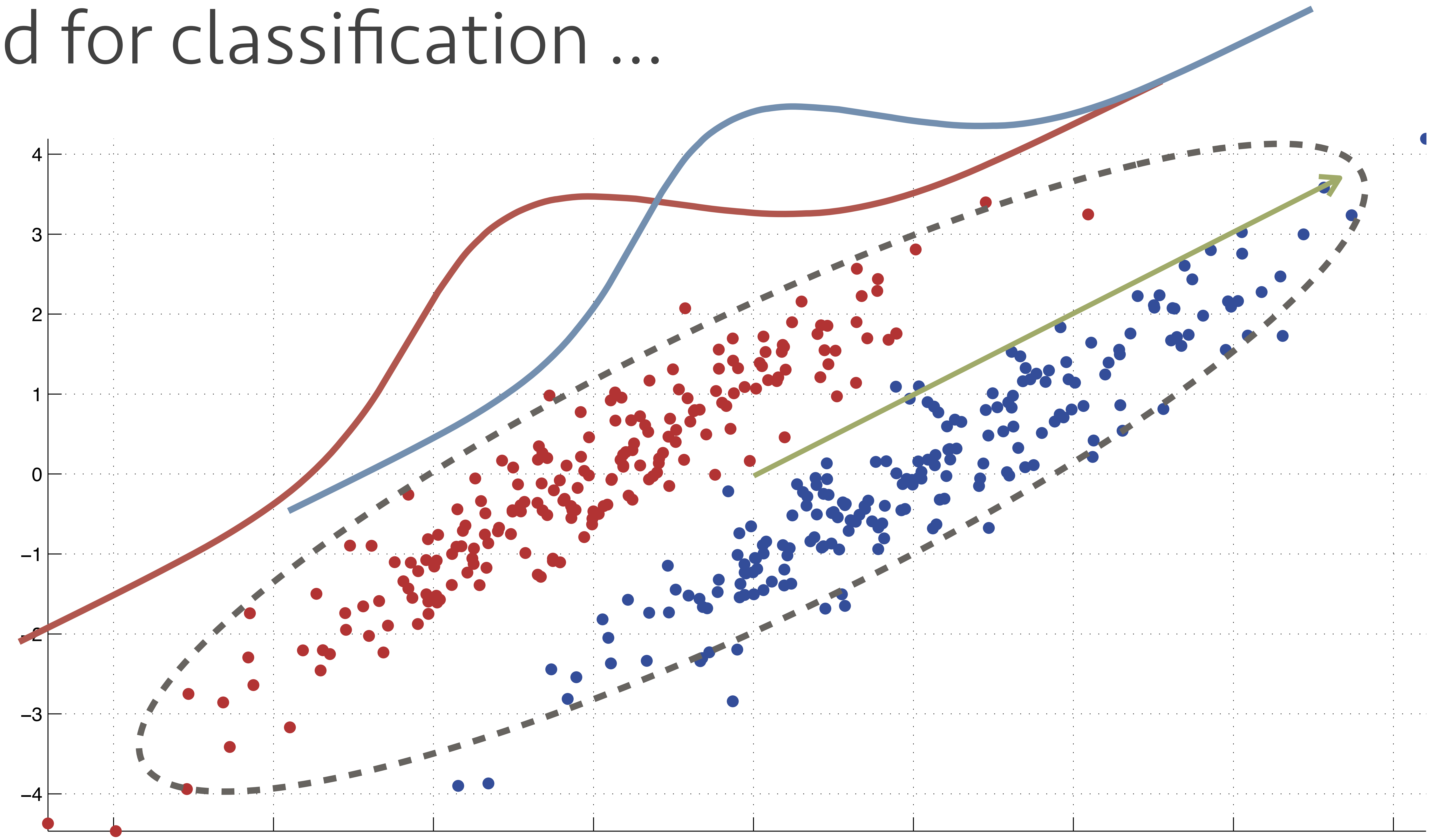
An example

- Get directions of maximal variance
  - Major axis maintains most information



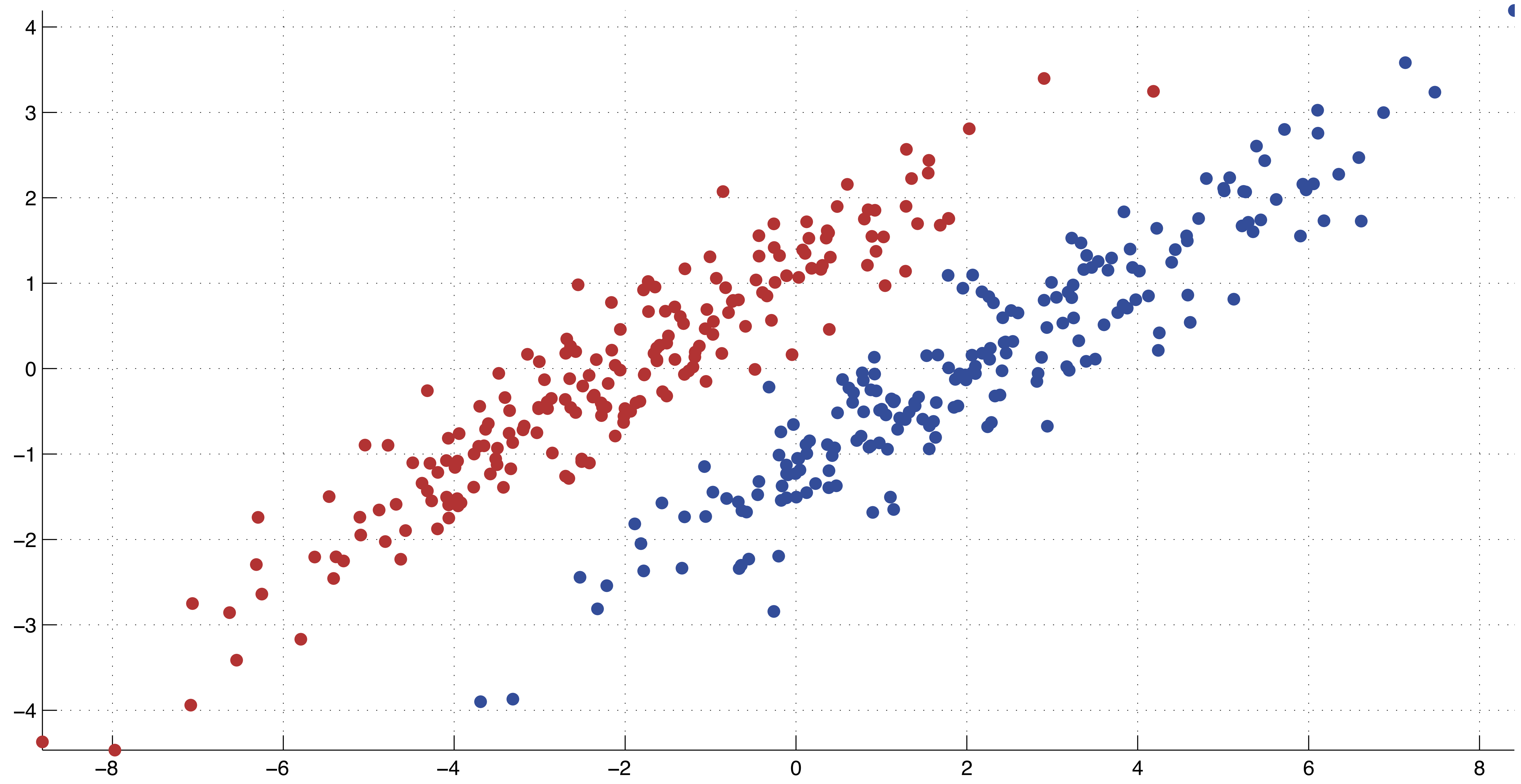
An example

- Reduce to 1D by discarding minor direction
  - No good for classification …



Informing dimensionality reduction

- What if all samples are class-labeled?
  - That should help us do the right thing
- What's the right thing to do?



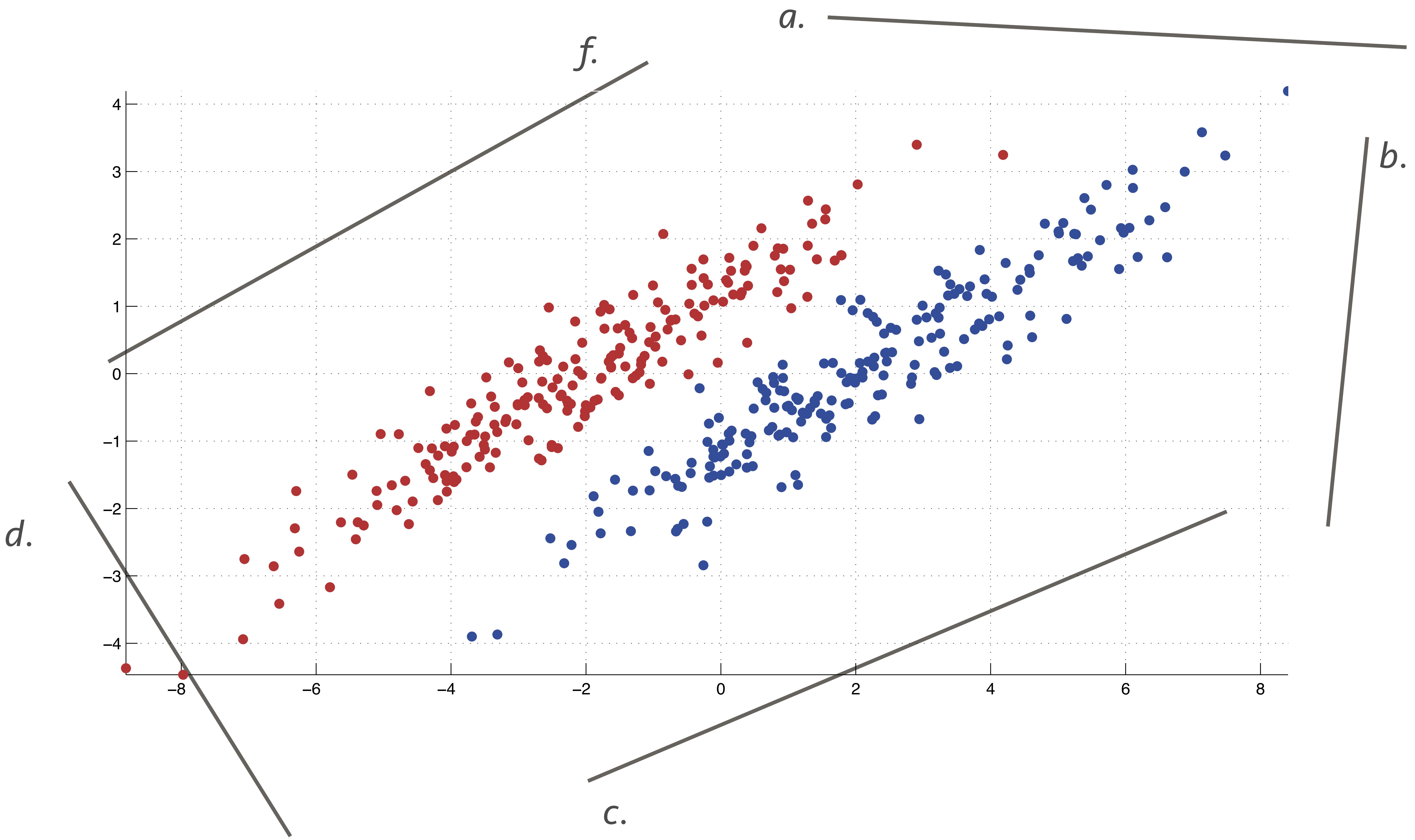
New plan

- Reduce from 2D to 1D using:

$$y = \mathbf{w}^\top \cdot \mathbf{x}$$

- Maximize projected class means distance
  - Makes classes easier to classify
- Minimize projected class scatter
  - Makes for compact class clusters

What does that mean?



Expressing this objective

- *Fisher's discriminant ratio*:

$$\text{Maximize} \longrightarrow F = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

Maximize (numerator)

Minimize (denominator)

- Means and variances of two classes of $y = \mathbf{w}^\top \cdot \mathbf{x}$ are:

$$\mu_i = \frac{1}{N}\sum y^{(i)} = \frac{1}{N}\sum \mathbf{w}^\top \cdot \mathbf{x}^{(i)} = \mathbf{w}^\top \cdot \frac{1}{N}\sum \mathbf{x}^{(i)} = \mathbf{w}^\top \cdot \mathbf{m}_i$$

$$\sigma_i^2 = \frac{1}{N}\sum (y^{(i)} - \mu_i)^2 = \frac{1}{N}\sum (\mathbf{w}^\top \cdot (\mathbf{x}^{(i)} - \mathbf{m}_i) \cdot (\mathbf{x}^{(i)} - \mathbf{m}_i)^\top \cdot \mathbf{w}) = \mathbf{w}^\top \cdot \mathbf{C}_i \cdot \mathbf{w}$$

Within-class scatter matrix

- The FDR denominator is:

$$\sigma_1^2 + \sigma_2^2 \propto \mathbf{w}^\top \cdot S_w \cdot \mathbf{w}$$

- Where:

$$S_w = \sum P_i \mathbf{C}_i$$

  - "Within-class scatter matrix"
  - P_i is i^{th} class prior
  - $\mathbf{C}_i$ is the covariance of the i-th class in high dimensions

Between-class scatter matrix

- And the FDR numerator is:

$$(\mu_1 - \mu_2)^2 = \mathbf{w}^\top \cdot (\mathbf{m}_1 - \mathbf{m}_2) \cdot (\mathbf{m}_1 - \mathbf{m}_2)^\top \cdot \mathbf{w} \propto \mathbf{w}^\top \cdot S_b \cdot \mathbf{w}$$

- Where:

$$S_b = \sum P_i (\mathbf{m}_i - \mathbf{m}_0) \cdot (\mathbf{m}_i - \mathbf{m}_0)^\top$$

  - “Between-class scatter matrix” (roughly speaking a covariance of the means)
  - P_i is i^{th} class prior
  - $\mathbf{m}_0$ is overall data mean in high dimensions

Overall objective

- Find the $\mathbf{w}$ that maximizes:

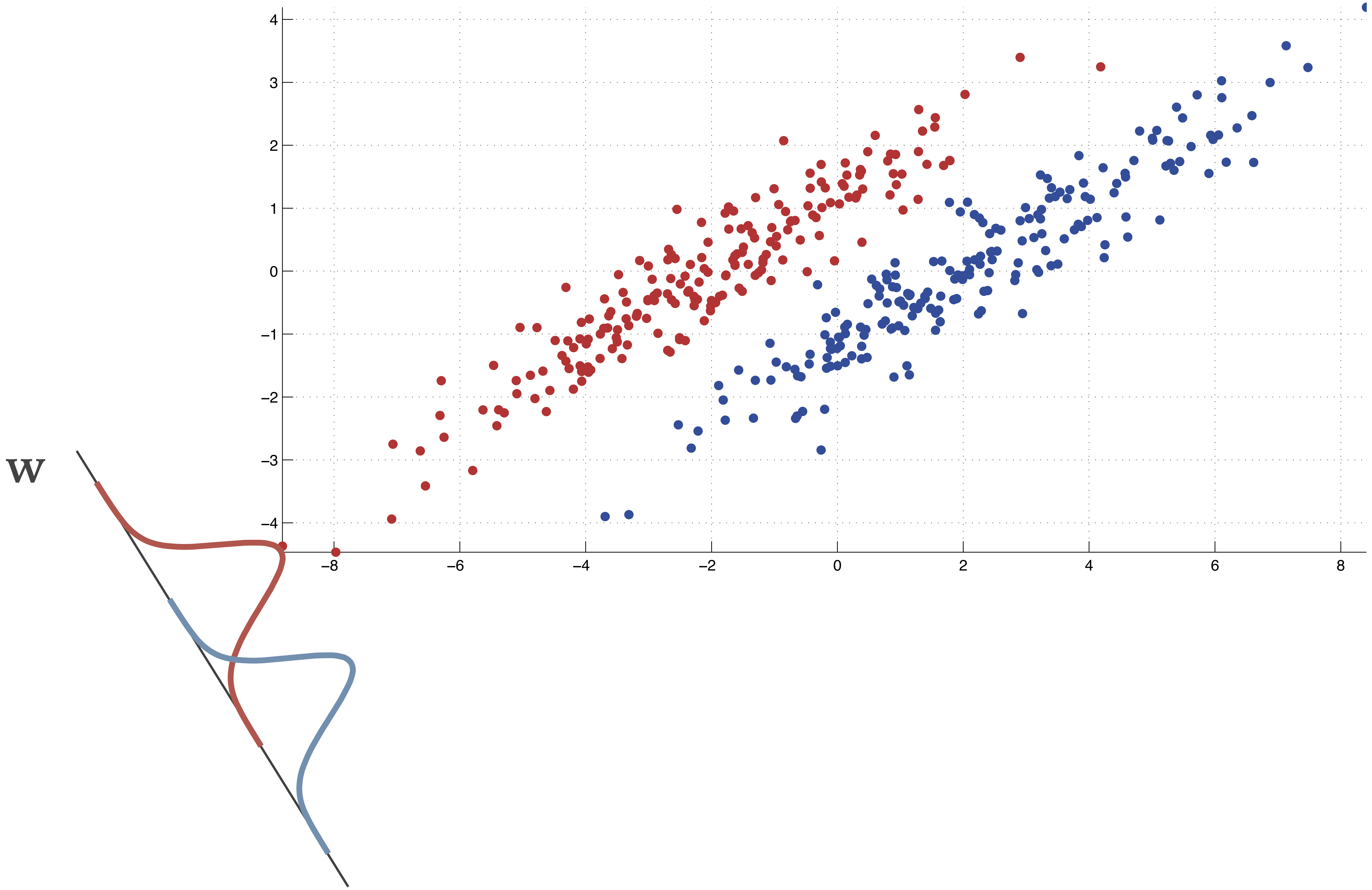
$$F(\mathbf{w}) = \frac{\mathbf{w}^\top \cdot S_b \cdot \mathbf{w}}{\mathbf{w}^\top \cdot S_w \cdot \mathbf{w}}$$

- This is solved by:

$$\mathbf{w} \propto S_w^{-1} \cdot (\mathbf{m}_1 - \mathbf{m}_2)$$

- And is called *Fisher's Linear Discriminant*

End result



Higher dimensional formulation

- To reduce to > 1D, transform will be a matrix:

$$\mathbf{y} = \mathbf{W}^\top \cdot \mathbf{x}$$

- New objective is defined as:

$$J(\mathbf{W}) = \frac{\left|\mathbf{W}^\top \cdot S_b \cdot \mathbf{W}\right|}{\left|\mathbf{W}^\top \cdot S_w \cdot \mathbf{W}\right|}$$

determinants, not norm!

Multiple Discriminant Analysis

- Columns of $\mathbf{W}$ are the largest eigenvectors of:

$$S_b \cdot \mathbf{w}_i = \lambda S_w \cdot \mathbf{w}_i$$

- How many?
  - No more than $(m - 1)$ for m classes
  - More than that offers no advantage (zero eigenvalues)
  - Fewer produces a loss in discrimination

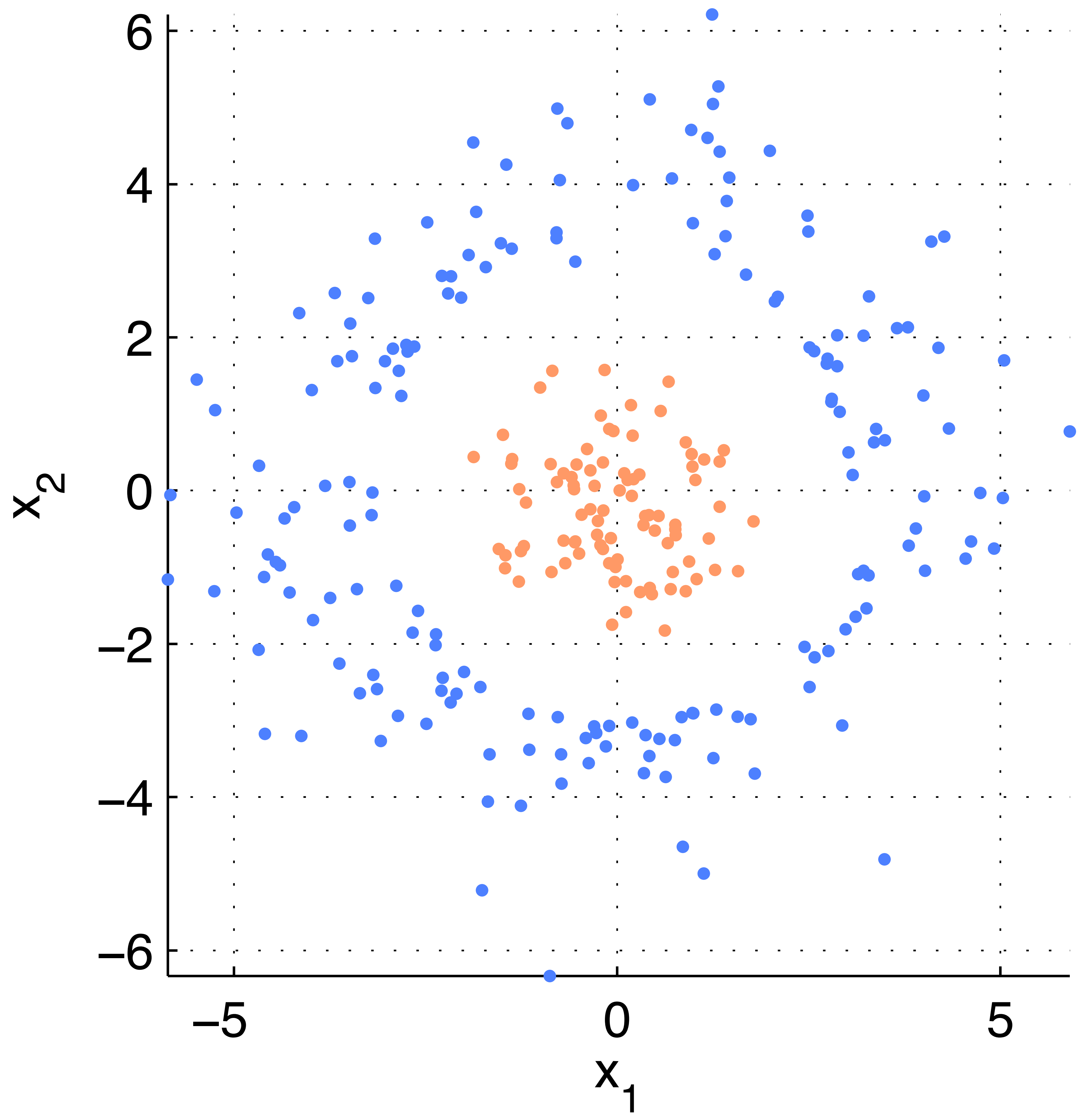
What about non-linear projections?

- MDA uses a linear projection
  - What if the data isn't separable that way?

- Use a kernel version:

$$J(\mathbf{W}) = \frac{\left|\mathbf{W}^\top \cdot S_b^\phi \cdot \mathbf{W}\right|}{\left|\mathbf{W}^\top \cdot S_w^\phi \cdot \mathbf{W}\right|}$$

  - Map to more dimensions, use the kernel trick, etc ...

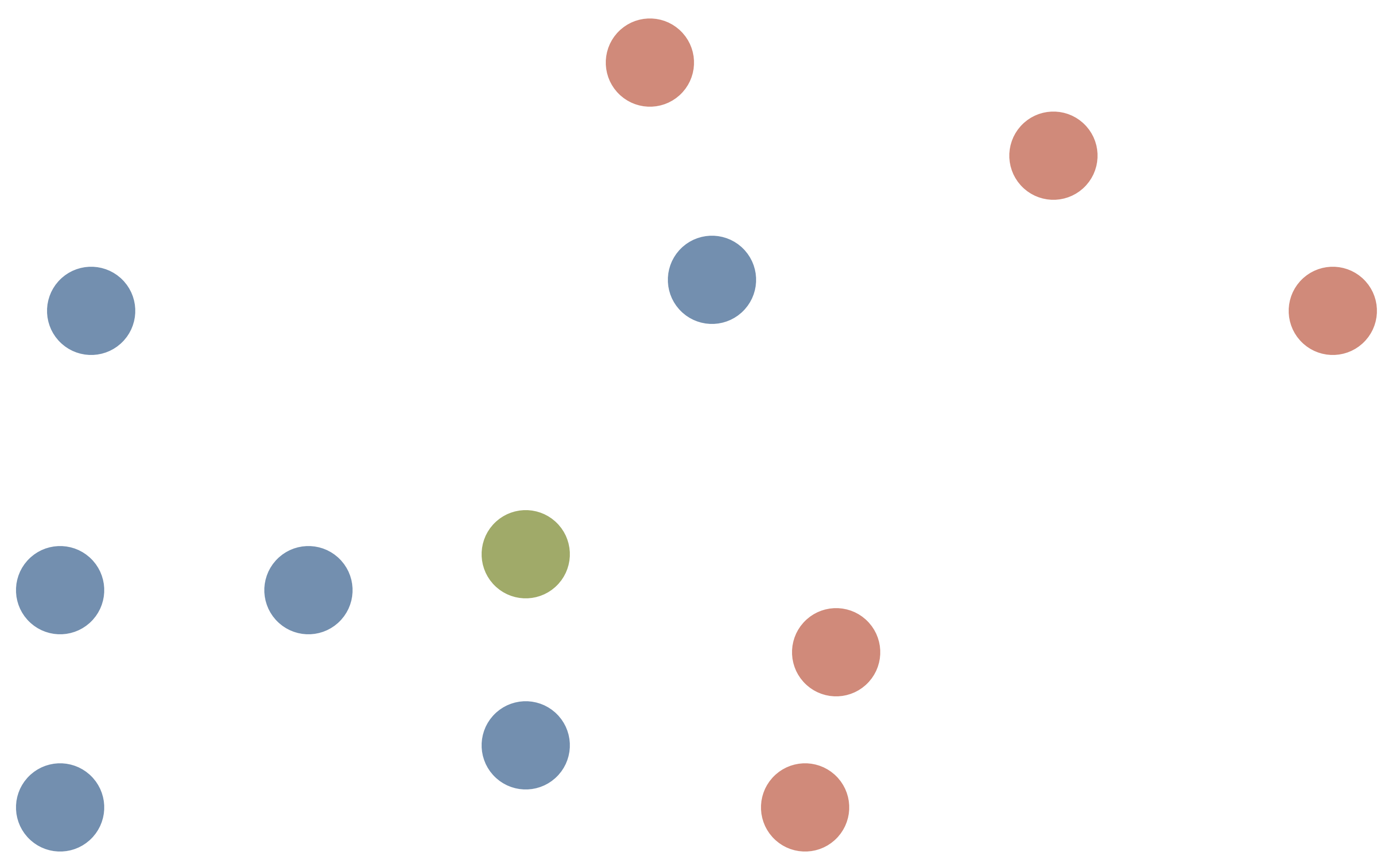


Non-parametric approaches

- So far we made a lot of models for our data
  - E.g., Gaussian classes
- This is not always necessary
  - Non-parametric approaches
    - Use the data, not the models

A simple case

- What class should the green sample be?
  - Why?



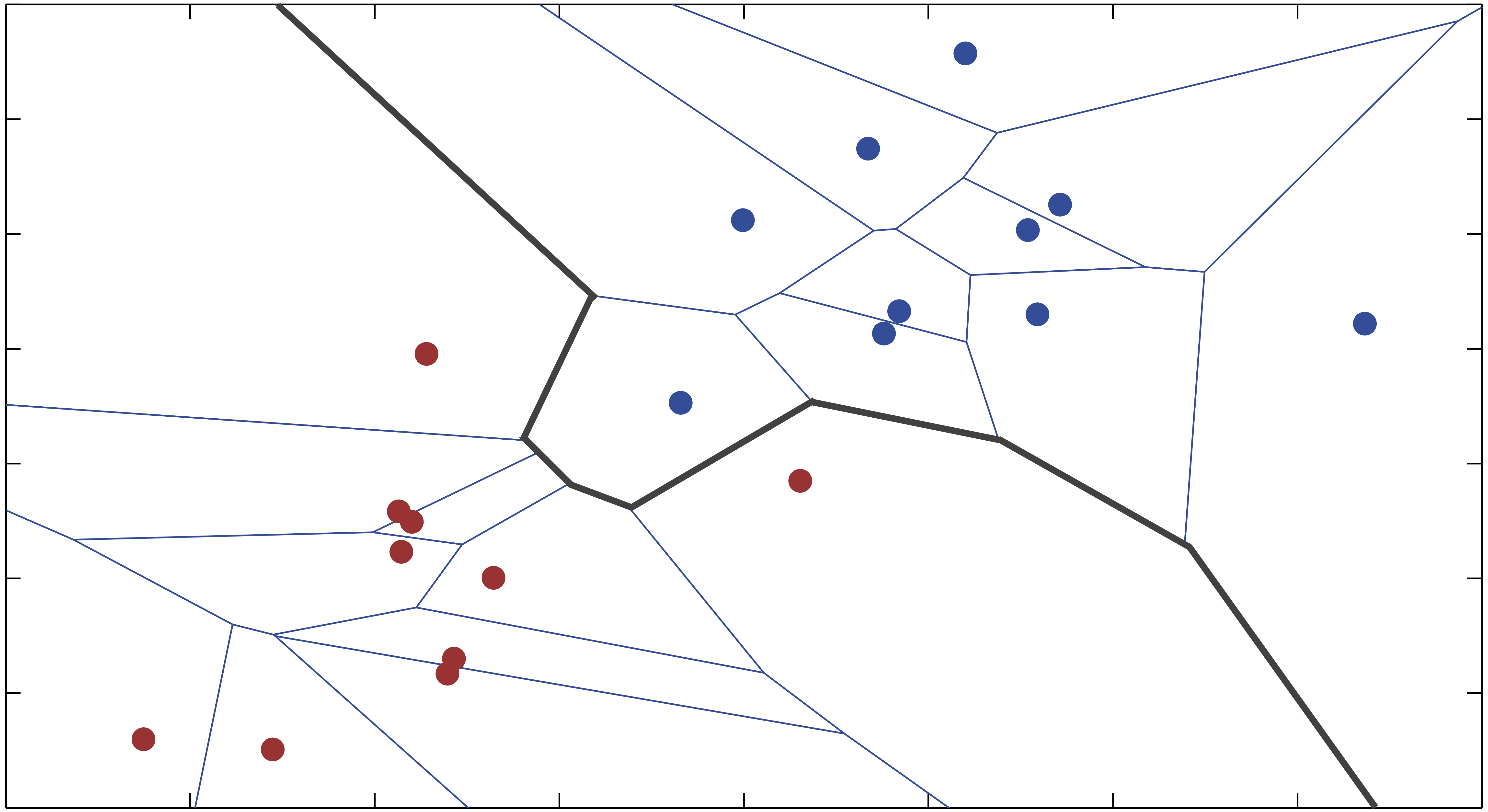
Nearest neighbor rule

- *Nearest neighbors* process
  - Compare input with all known data
  - Find closest training point
  - Assign class to match that point

- Very robust, very useful!

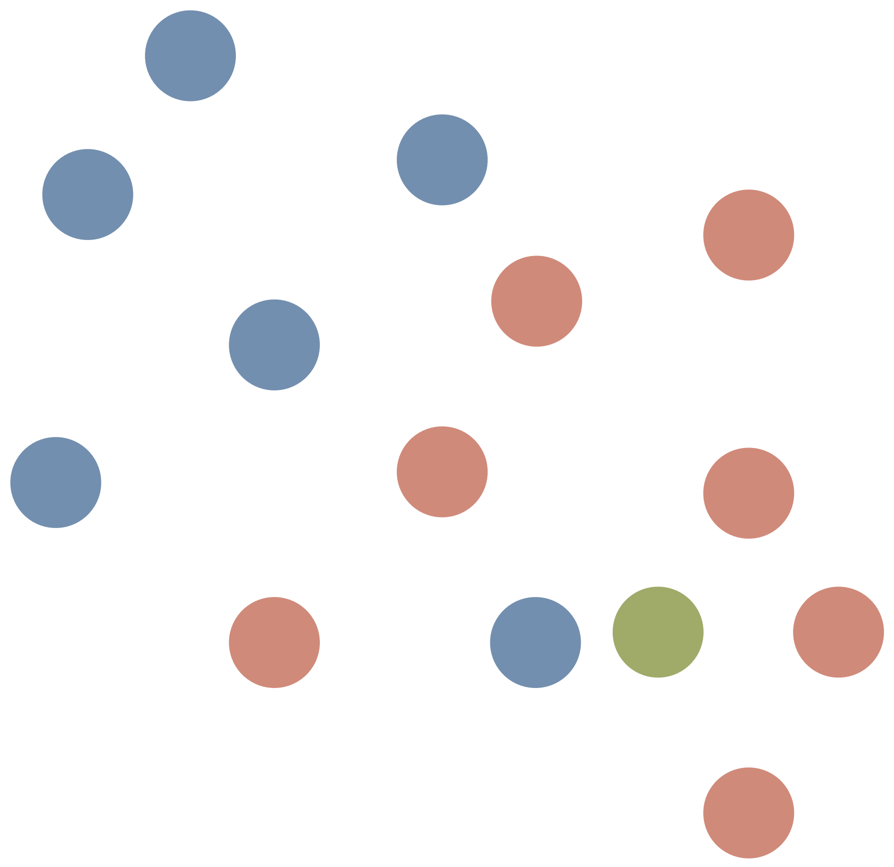
Decision boundaries

- This rule results in a *Voronoi tesselation*



Generalizing

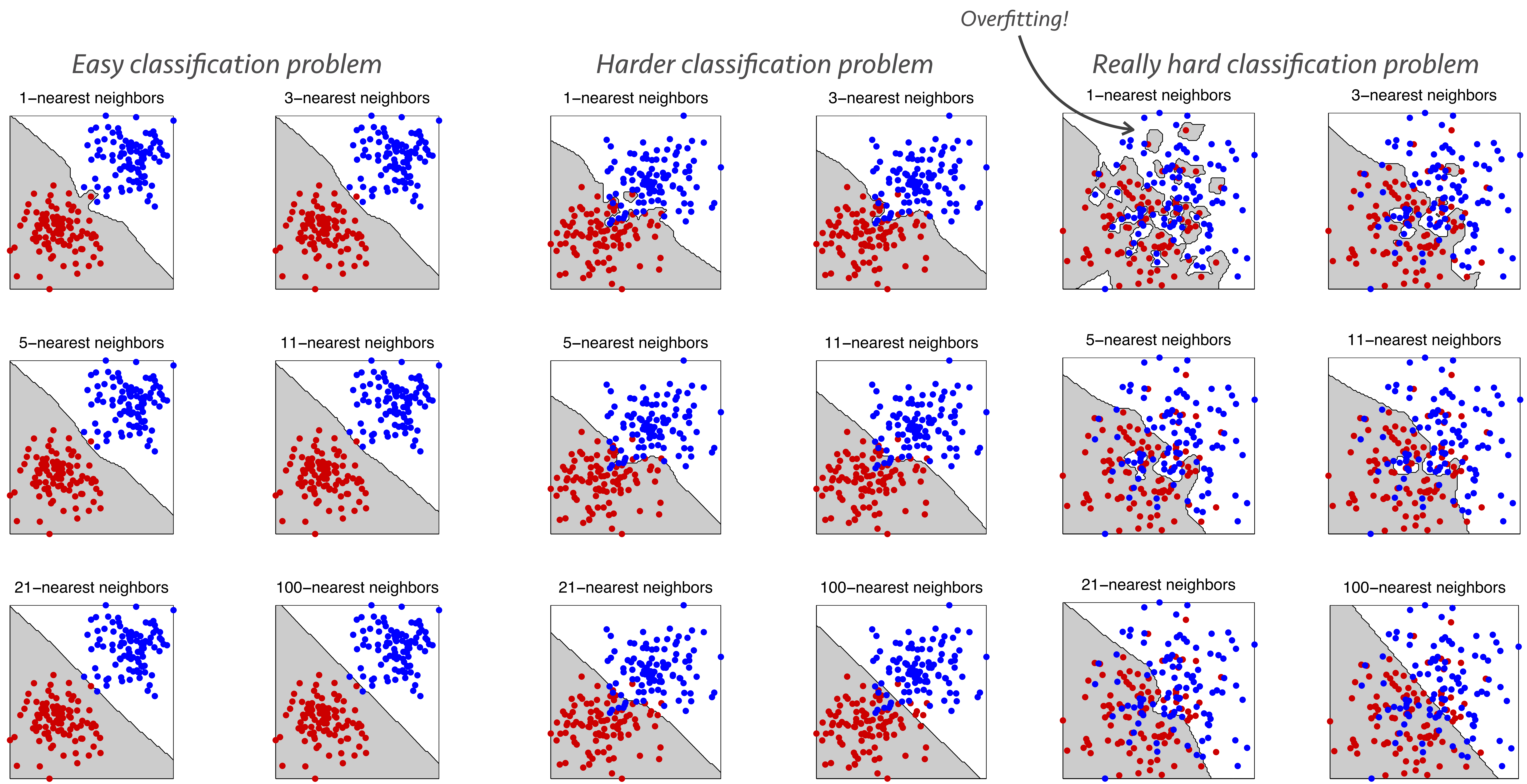
- What class is the green sample now?



k-nearest neighbors

- Same as before, but count K neighbors
  - Compare input to all training data
  - Find K closest neighbors
  - Assign class to the majority of neighbors
- Pick an odd K to avoid ties!
- Results in varying decision boundaries
  - A bigger K generalizes more

Example cases



kNN pros and cons

- Good news:
  - Easy to implement
  - Quite reliable
- Bad news:
  - On the slow side with large data sets …
    - Although there’s nice research to help here
  - Makes *hard* decisions

A softer approach

- kNN makes "hard" decisions
  - In general, this is something to avoid
    - Remember, we like probabilities!

- Can we reformulate kNN?

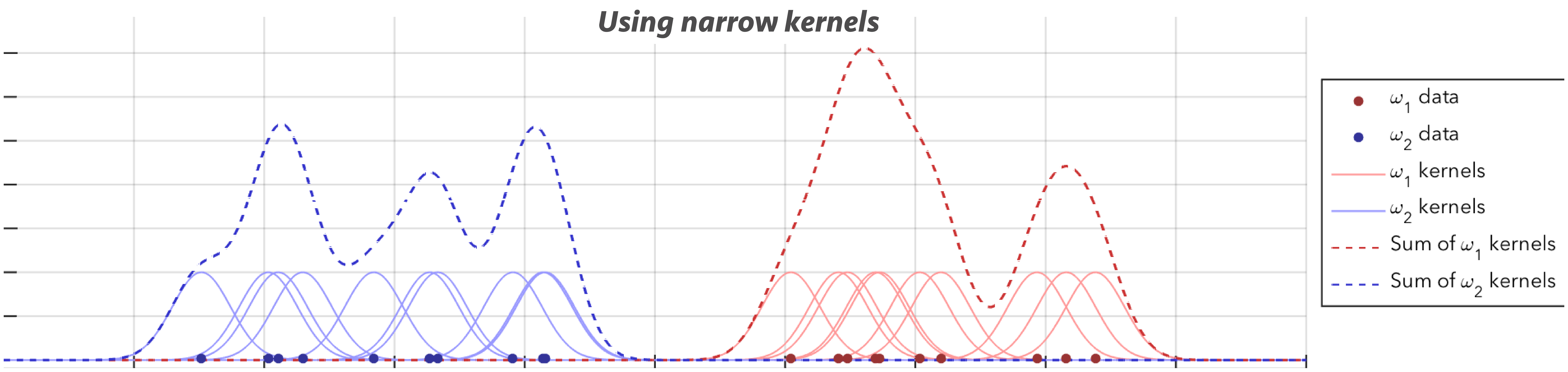
Parzen estimation

- Place a “kernel” on each data point
  - e.g. make each data point a Gaussian

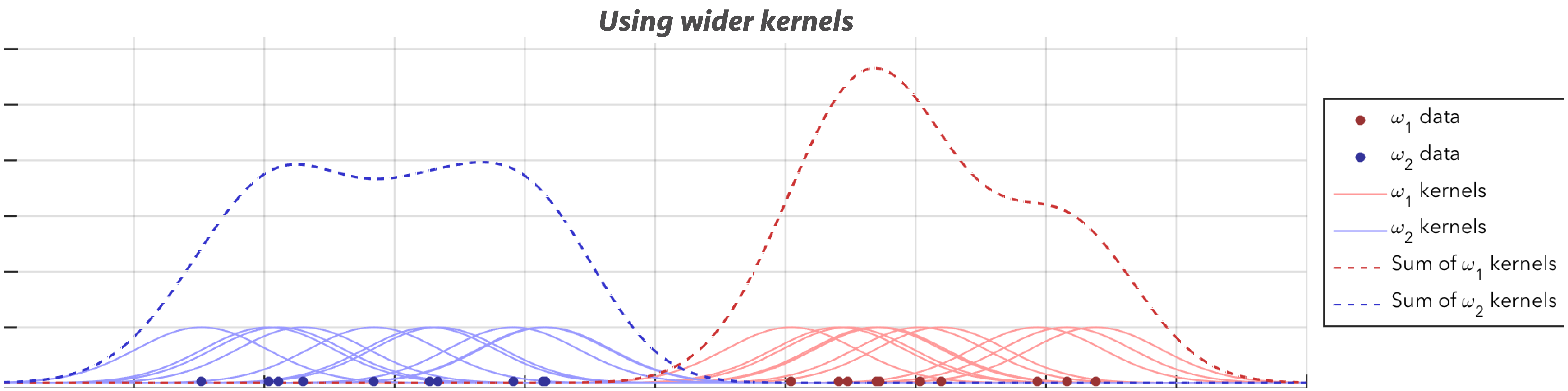
$$P(\mathbf{x}) = \frac{1}{N}\sum \frac{1}{(2\pi)^{l/2} b^l} e^{-\frac{(x-x_i)^\top \cdot (x-x_i)}{2b^2}}$$

- These kernels get superimposed to approximate the distribution of the data

1-D example

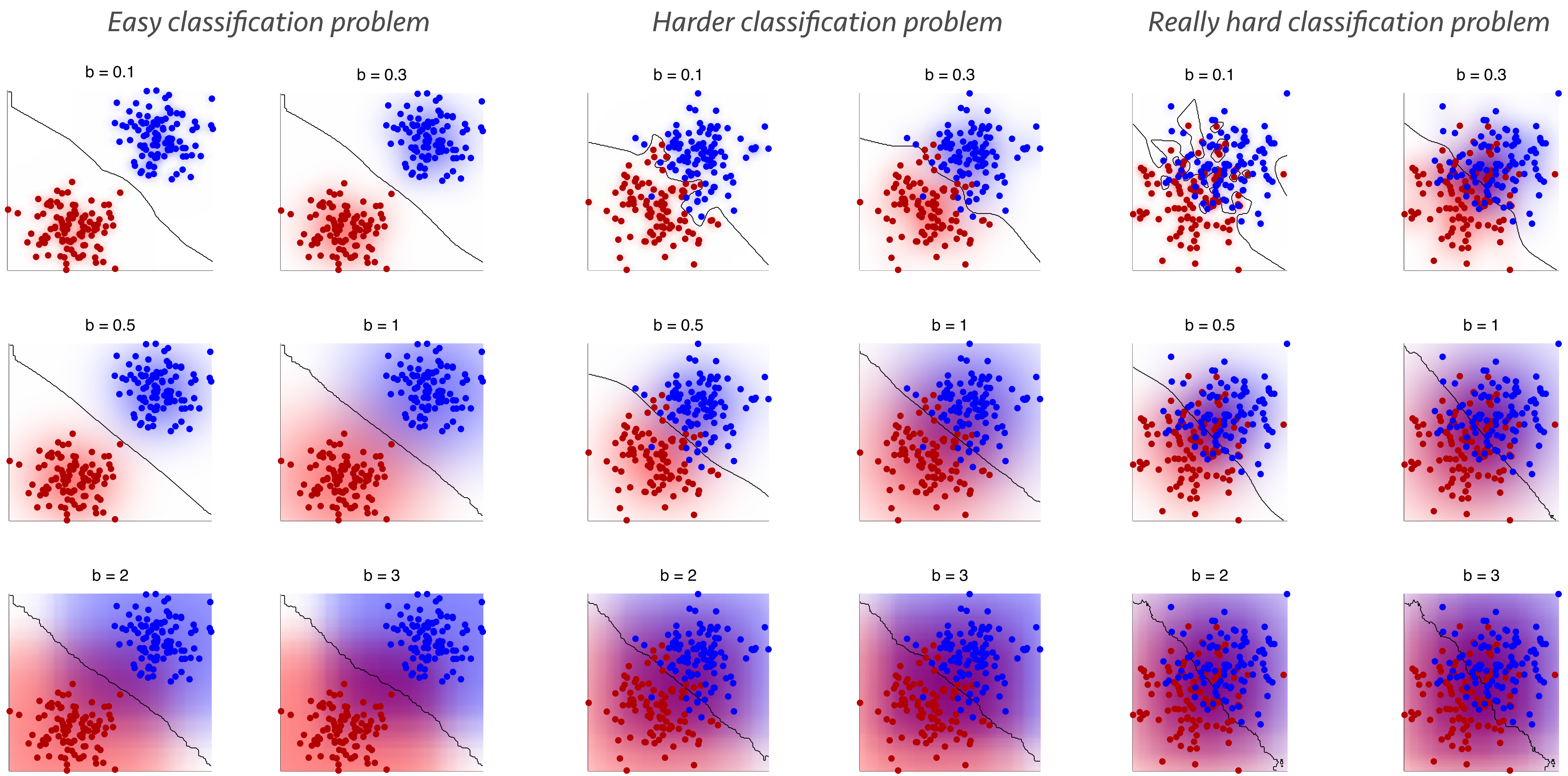


Using narrow kernels
ω_1 data
ω_2 data
ω_1 kernels
ω_2 kernels
Sum of ω_1 kernels
Sum of ω_2 kernels



Using wider kernels
ω_1 data
ω_2 data
ω_1 kernels
ω_2 kernels
Sum of ω_1 kernels
Sum of ω_2 kernels

Example cases



Parzen vs. kNN

- Parzen
  - Smooth boundaries
  - Probabilistic interpretation
  - Costly in high dimensions
  - Computation heavy
- kNN
  - Unorthodox boundaries
  - Hard assignments
  - Slow, but can be sped up with approximate methods

Democracy in classification

- Different approaches have different advantages
- Combining multiple classifiers should give us some performance advantage
- How do we do that?
  - majority rule, averaging, Bayesian approaches ...
  - “Meta-learning”

Boosting

- Combining classifiers, with a twist
- Use many weak learners
- Boost performance by combining them

AdaBoost formulation

- Classify using: $f(\mathbf{x}) = \operatorname{sgn}\left(F(\mathbf{x})\right)$

$$F(\mathbf{x}) = \sum a_k \phi(\mathbf{x};\theta_k)$$

- Each ϕ being a classifier with parameter θ_k
- We then optimize:

$$\underset{a_k,\theta_k}{\arg\min} \sum e^{-y_i F(\mathbf{x}_i)}$$

Negative exponent == *correct classification, small value*
Positive exponent == *incorrect classification, large value*

  - i.e., errors are being penalized more

An incremental approach

- Split classifier output as current and past contributions:

$$\begin{aligned} F_m(\mathbf{x}) &= \sum^{m} a_k \phi(\mathbf{x}; \theta_k) \\ &= F_{m-1}(\mathbf{x}) + a_m \phi(\mathbf{x}; \theta_m) \end{aligned}$$

- Each time we add one classifier, and we worry about:

$$J(a, \theta) = \sum e^{-y_i\left(F_{m-1}(\mathbf{x}_i) + \underline{a\phi(\mathbf{x}_i; \theta)}\right)}$$

An incremental approach

- Which separates to:

$$\theta = \arg\min_{\theta} \sum w_i^{(m)} e^{-y_i a\phi(\mathbf{x}_i;\theta)}$$

$$w_i^{(m)} \propto e^{-y_i F_{m-1}(\mathbf{x}_i)} \quad \longleftarrow \textit{Big for misclassified } \mathbf{x}_i$$

  - i.e. learn a classifier that focuses on addressing past failures

- The new weight will be:

$$a_m = \frac{1}{2}\log\frac{1-\varepsilon_m}{\varepsilon_m}$$

$$\varepsilon_m = \sum_i w_i^{(m)} \left(y_i \neq \operatorname{sgn}\left(\phi(\mathbf{x}_i;\theta)\right)\right) \quad \longleftarrow \textit{New classifier's error on past failures}$$

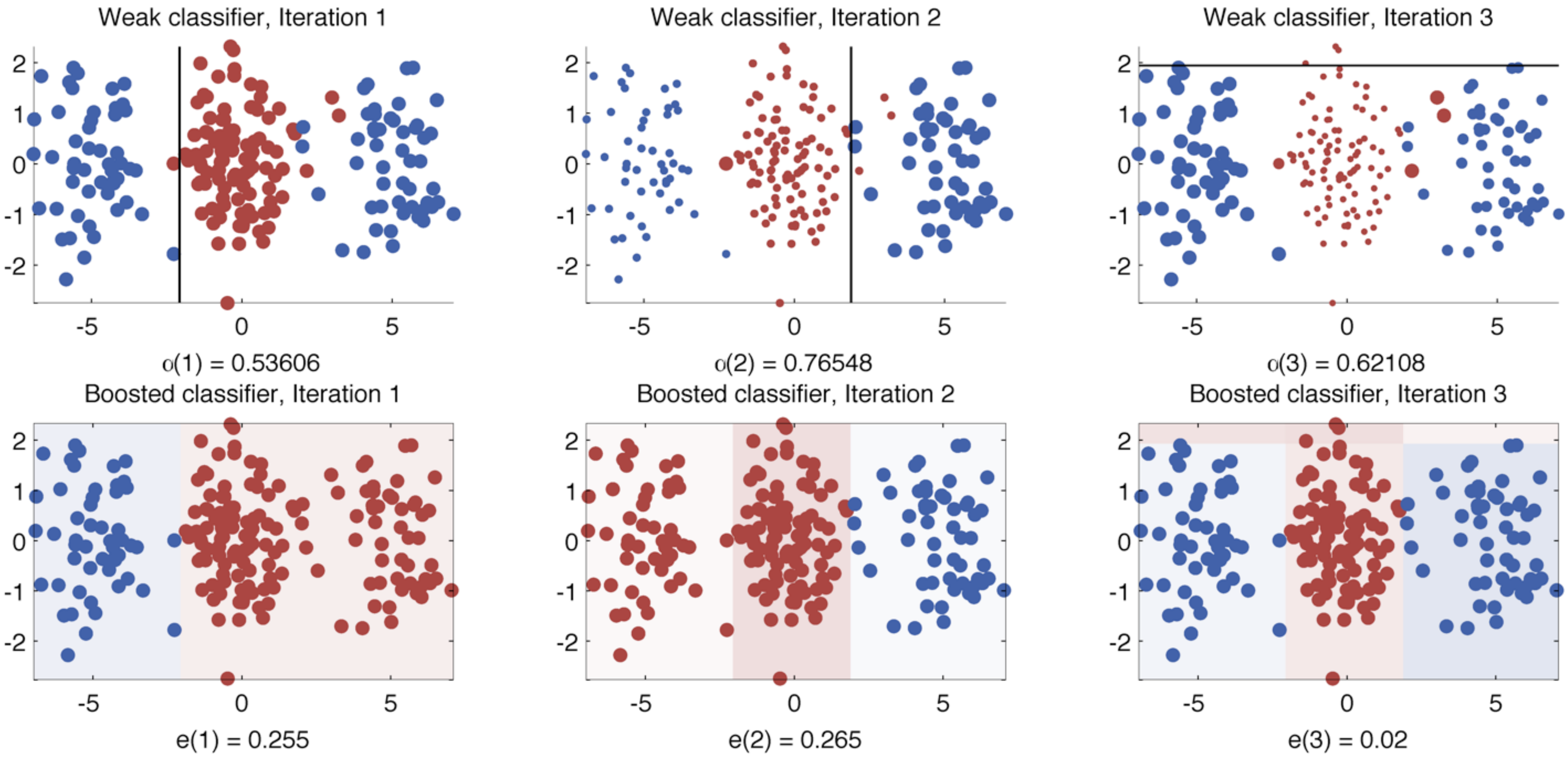
Simply stated

- Learn a new classifier each time
  - Don't refine the previous ones
- Bias its learning to make up for any mistakes by the existing models
- Estimate an optimum weight to combine it with the rest of the models

AdaBoost benefits

- Simple core models
  - Can be simple stump classifiers (threshold in one dimension)
  - No need for fancy learning and complex code
- Robust to overfitting
  - Adding more classifiers tends to push the error down
- Works fine for huge datasets

Example

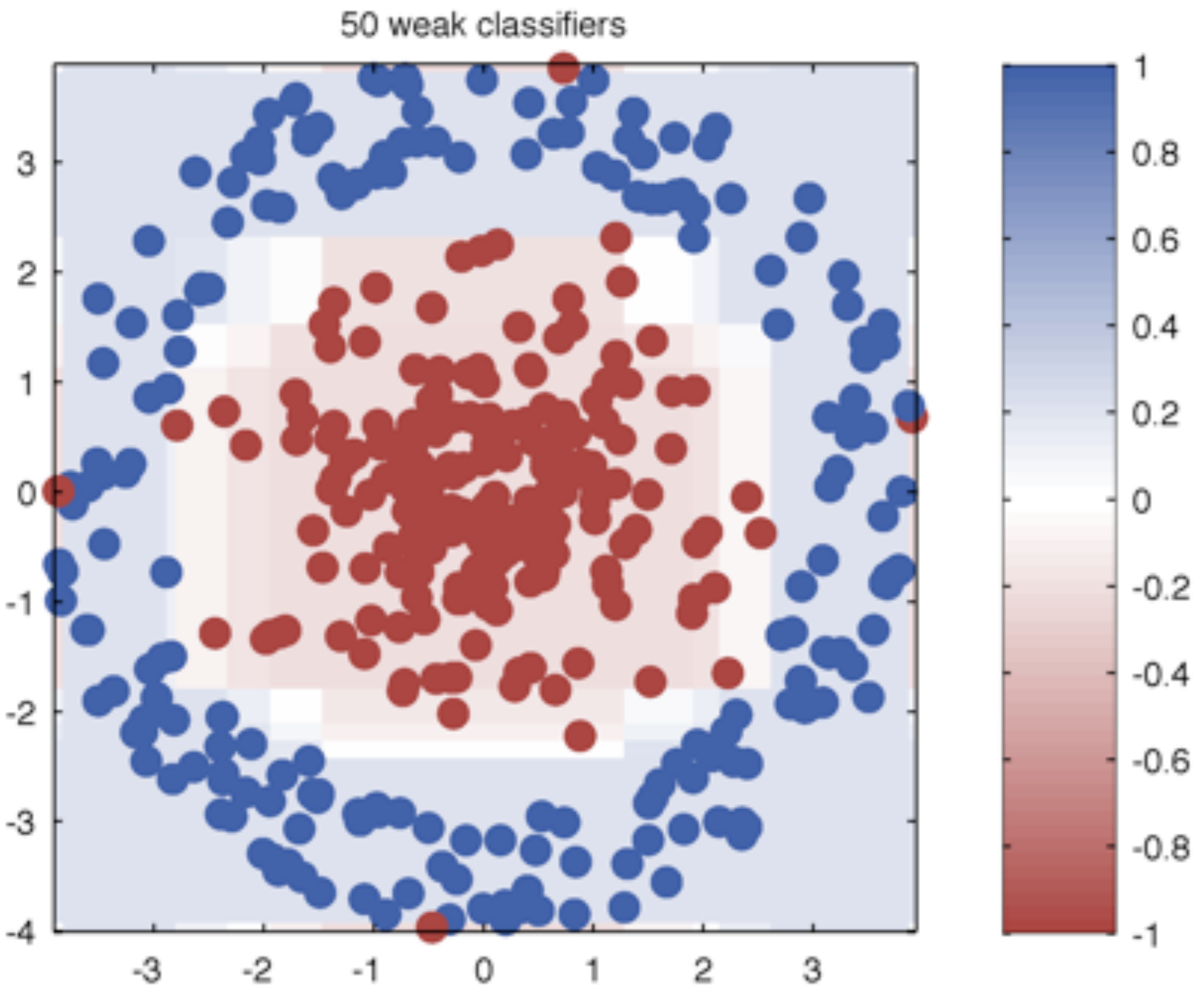


Weak classifier, Iteration 1
Weak classifier, Iteration 2
Weak classifier, Iteration 3
α(1) = 0.53606
α(2) = 0.76548
α(3) = 0.62108
Boosted classifier, Iteration 1
Boosted classifier, Iteration 2
Boosted classifier, Iteration 3
e(1) = 0.255
e(2) = 0.265
e(3) = 0.02

Example

- Weak learners combine to form very complex decision boundaries

- Can be highly powerful, yet fairly simple to learn

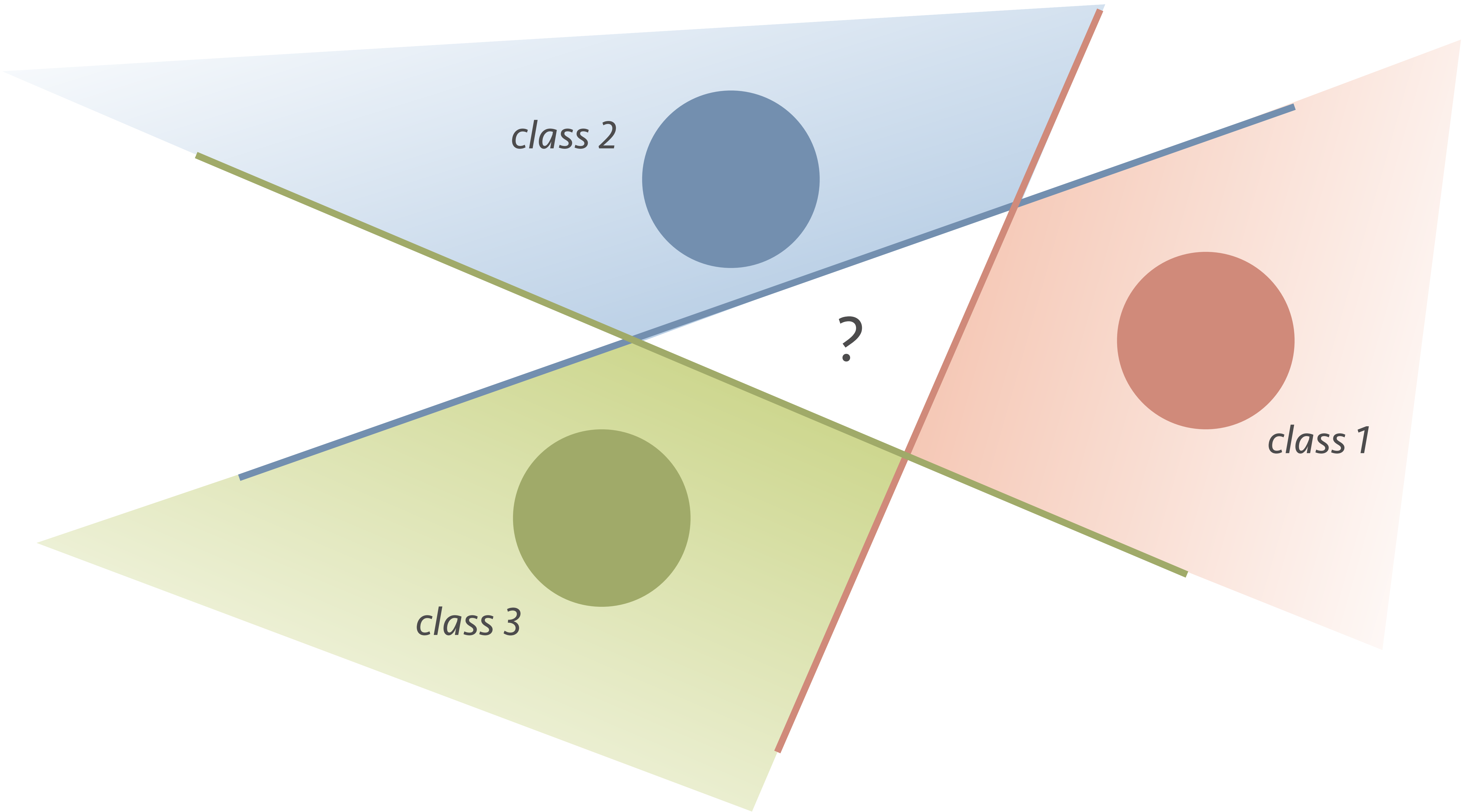


Multiple classes

- So far we covered binary classification
  - Good for detection problems (e.g. find faces in a picture)
- What if we have more than two classes?
  - e.g. face or handwriting recognition

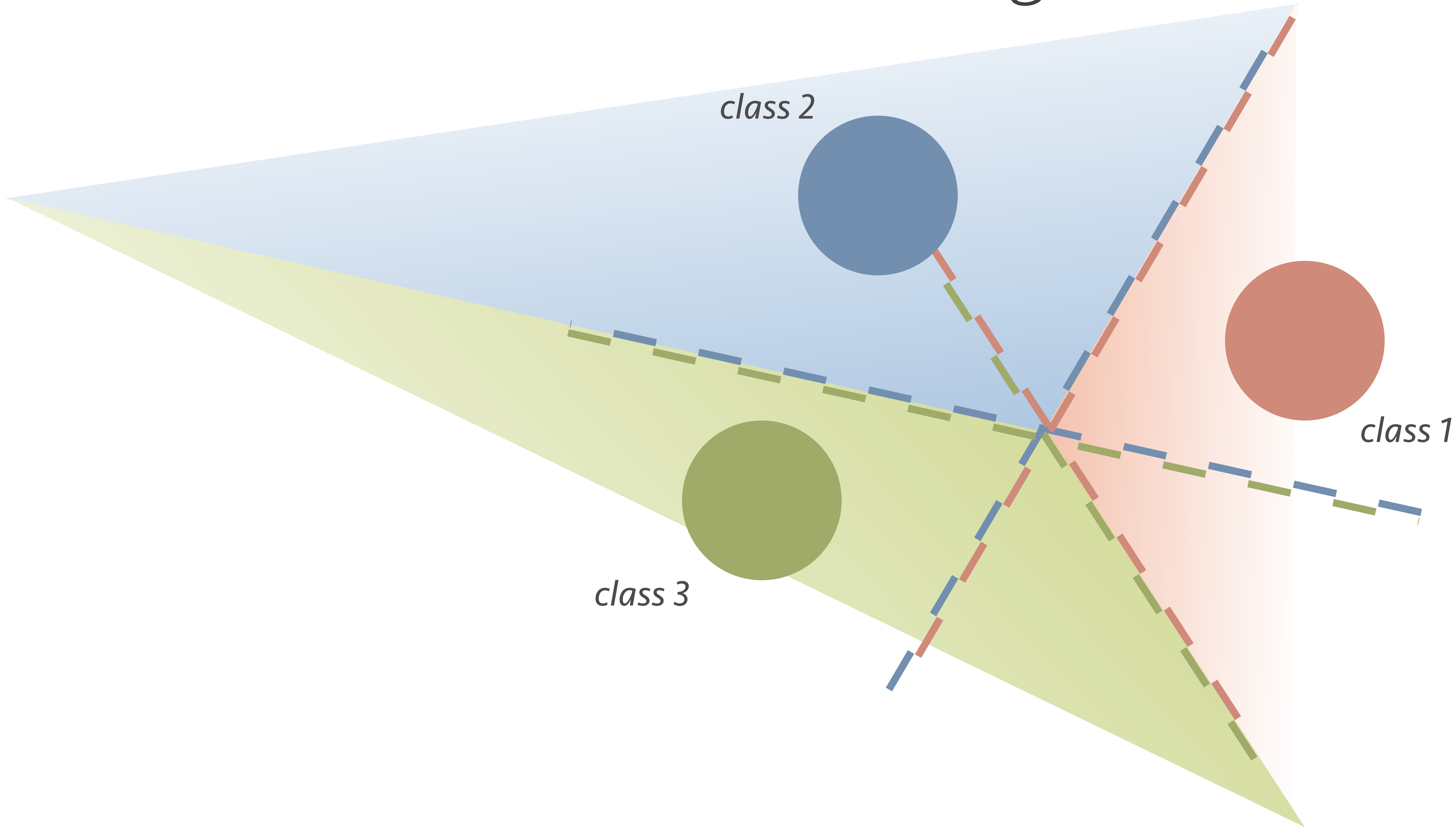
One against all

- M classifiers for M classes
  - Each classifier is trained on one class against all others



One against one

- $M(M\text{-}1)/2$ classifiers for M classes
  - Each classifier does one class against one other only



Binary coding approach

- For M classes and L classifiers
  - Assign a code for each class
  - Each bit in the code is a classifier
    - e.g., three classes, two classifiers

2 classifier bank

	C_1	C_2
Class 1	−1	−1
Class 2	+1	+1
Class 3	−1	+1

- Can use robust codes to help performance

Important concepts to keep in mind

- Low-level
  - Curse of Dimensionality
  - Bias/Variance Tradeoff
  - Cross-validation and overfitting
- High-level
  - GIGO
  - No Free Lunch Theorem
  - Occam's Razor

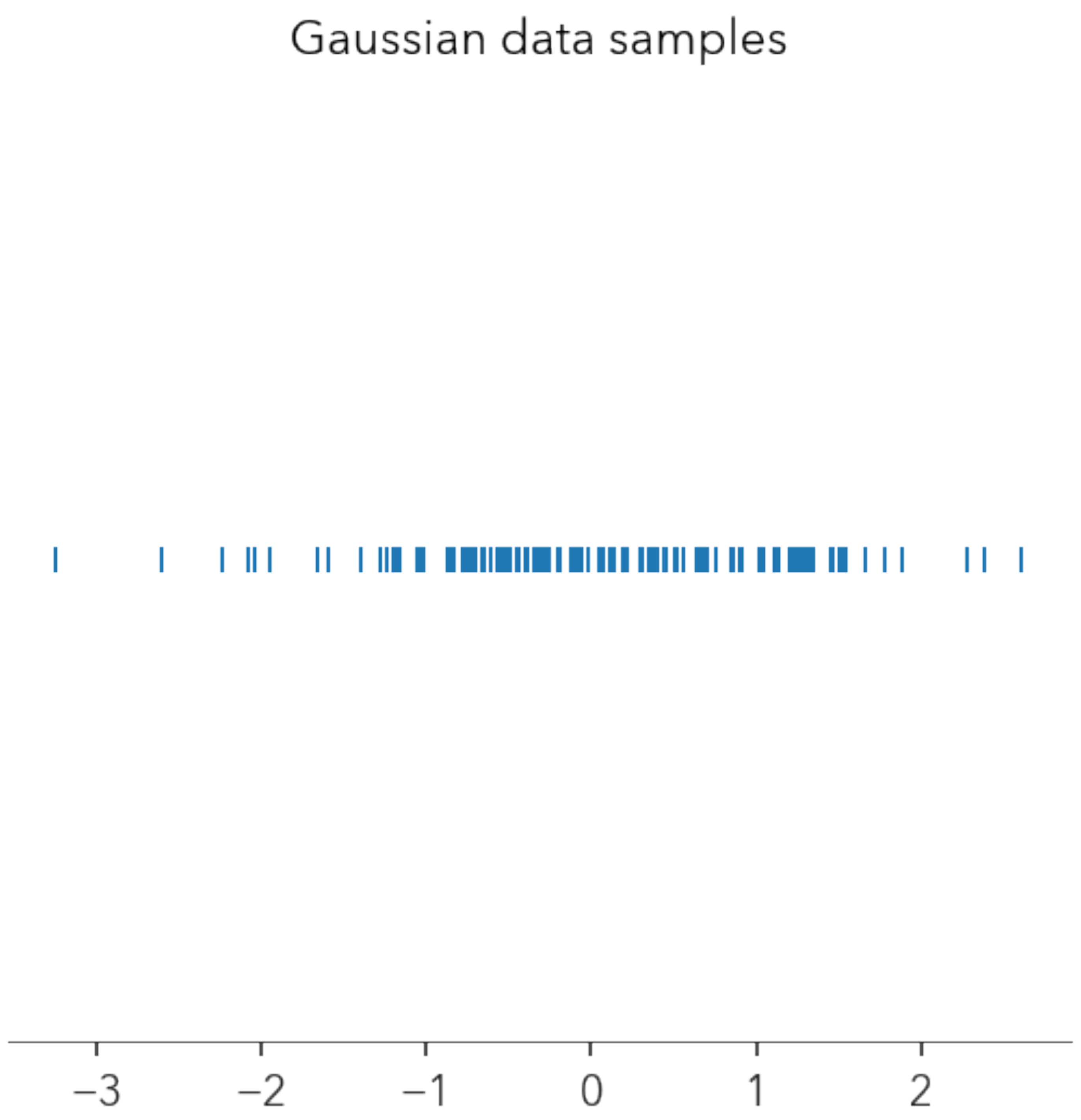
Curse of Dimensionality

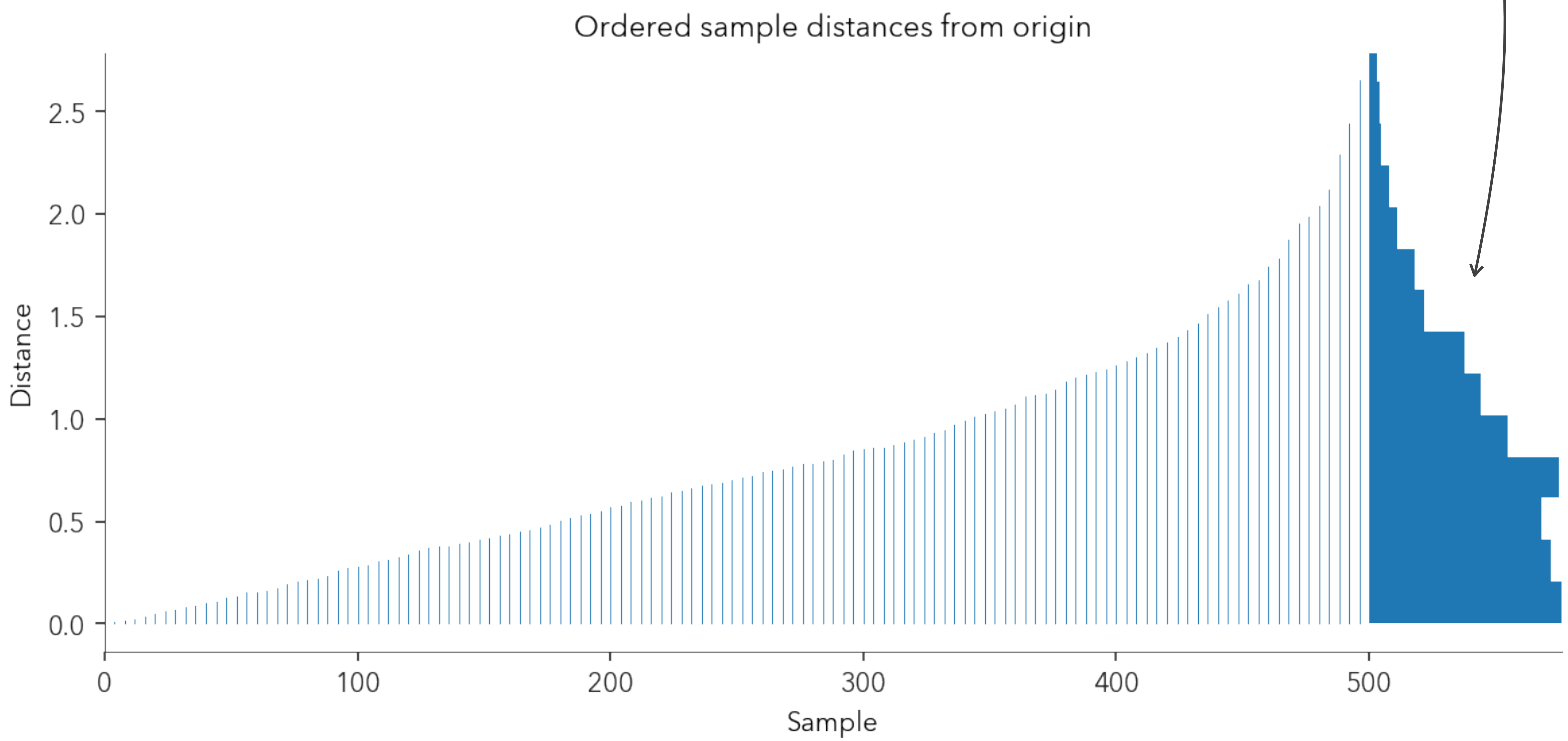
- High dimensions are difficult
  - High-D estimators need lots of data!!
  - Geometry at high dimensions is very counterintuitive!

- Sometimes high-D's are a blessing
  - Surely more data is good
  - Remember the kernel trick …

Distances in 1D

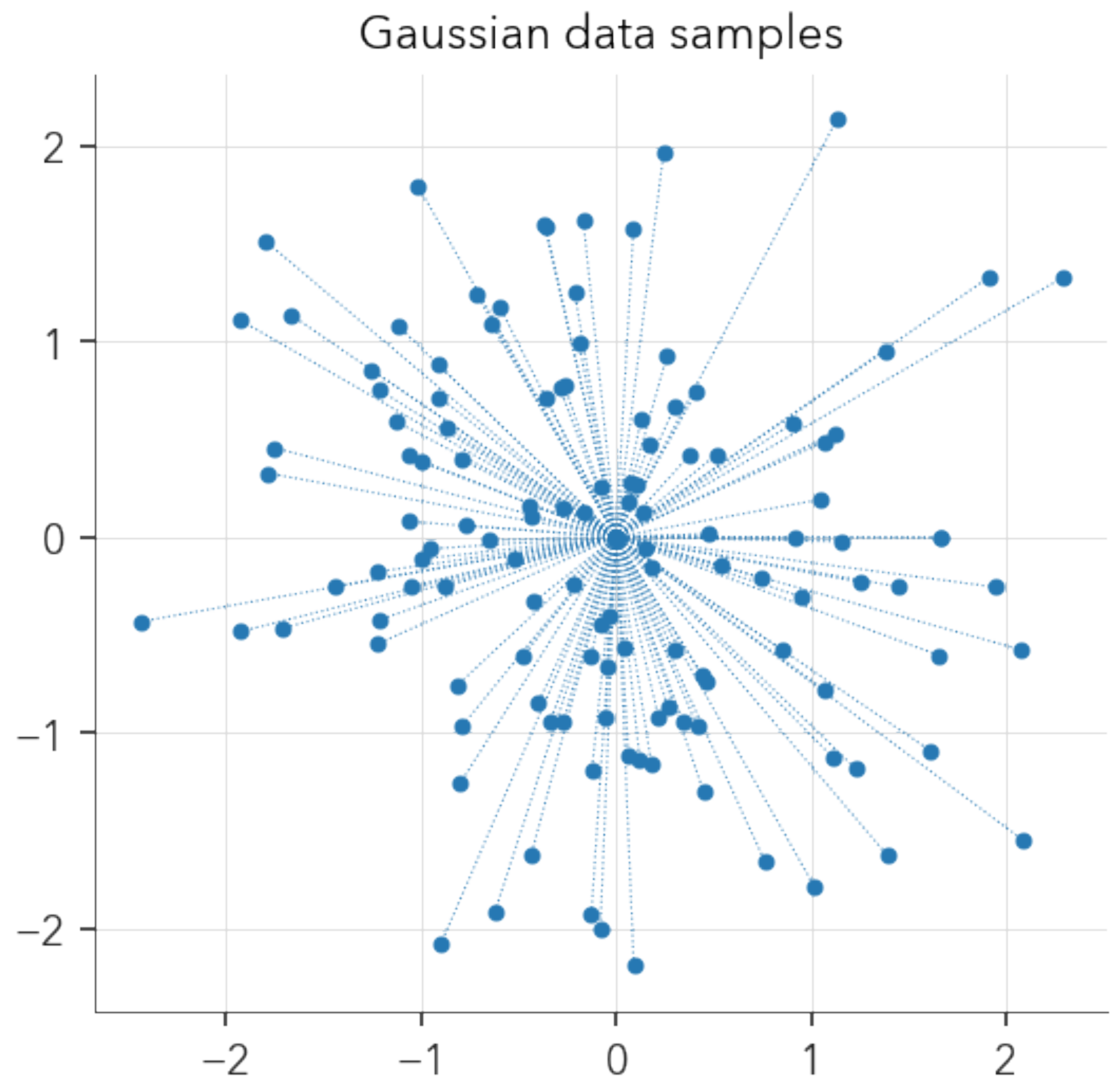
- Take some Gaussian samples and histogram their distance from the mean of the distribution
  - Note that most samples are close to the mean

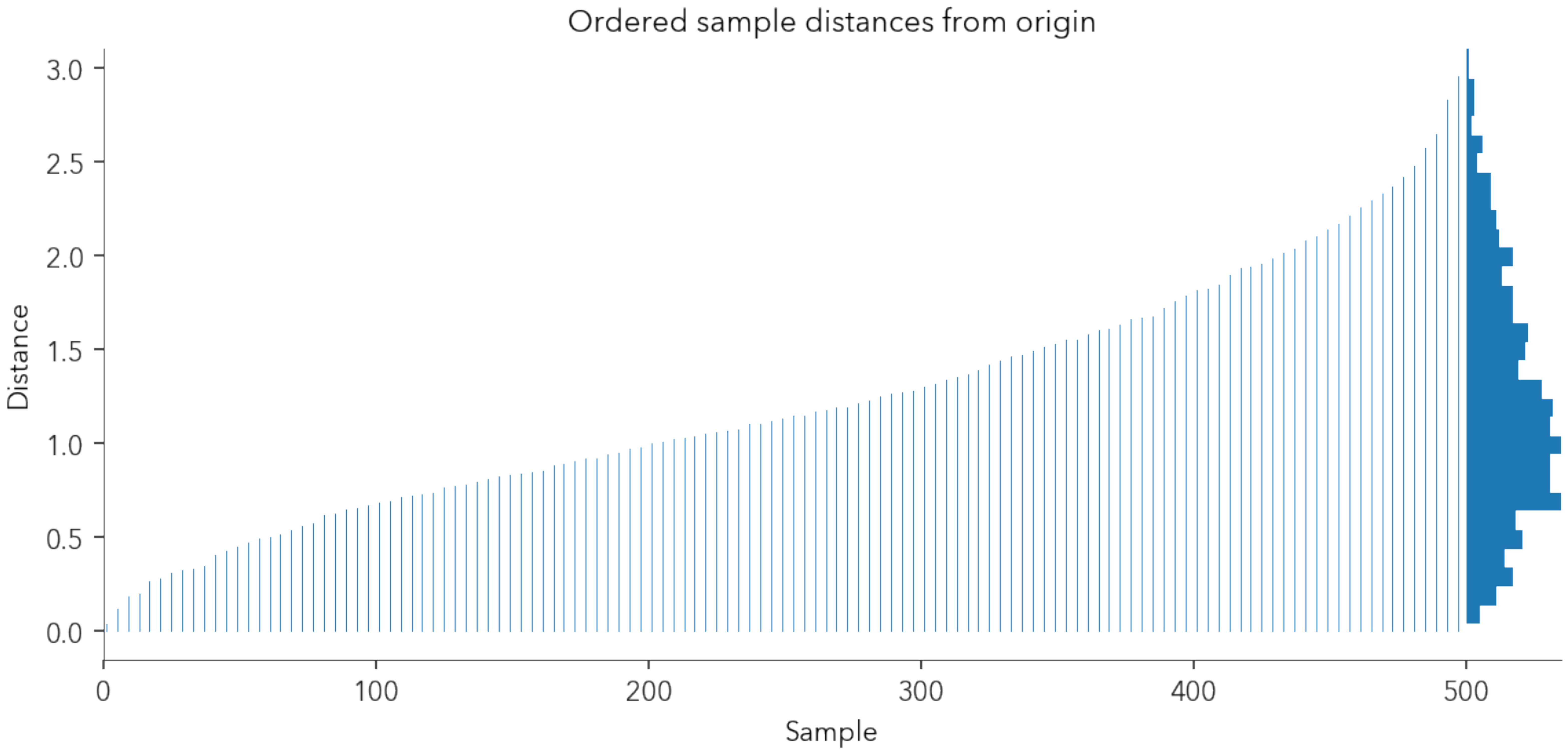




Redo in 2D

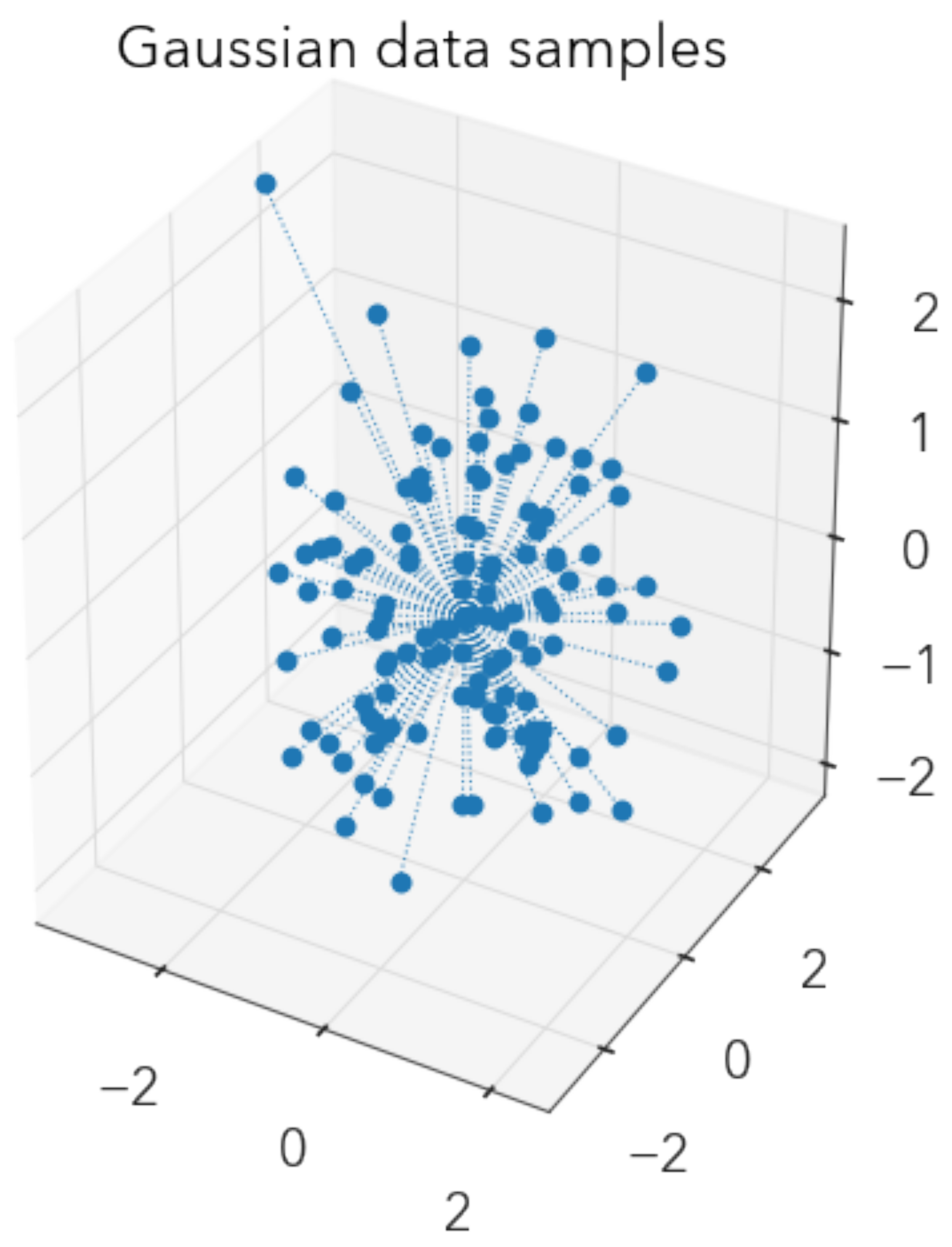
- Roughly the same in 2D
  - Hmm, the histogram mode has moved a bit

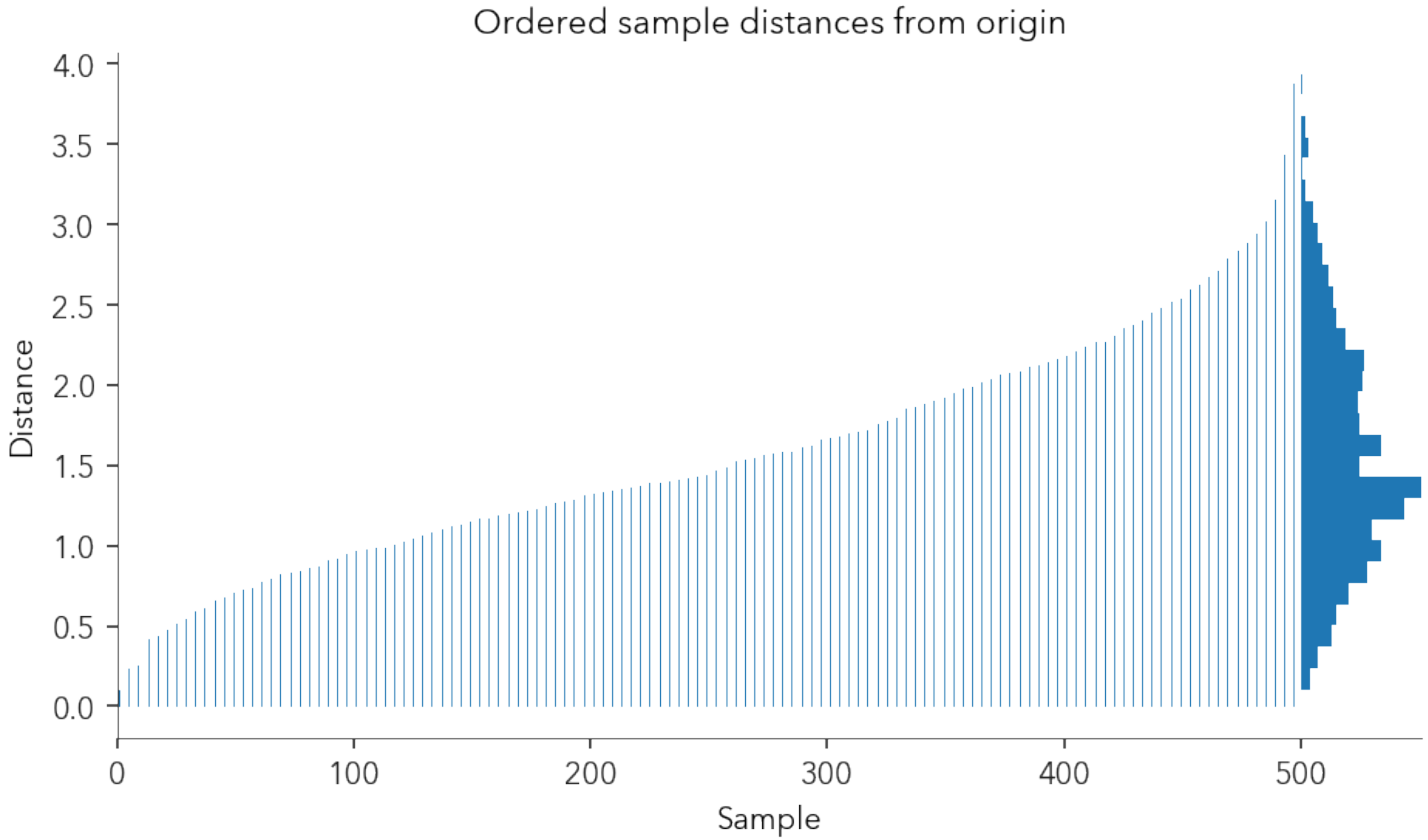




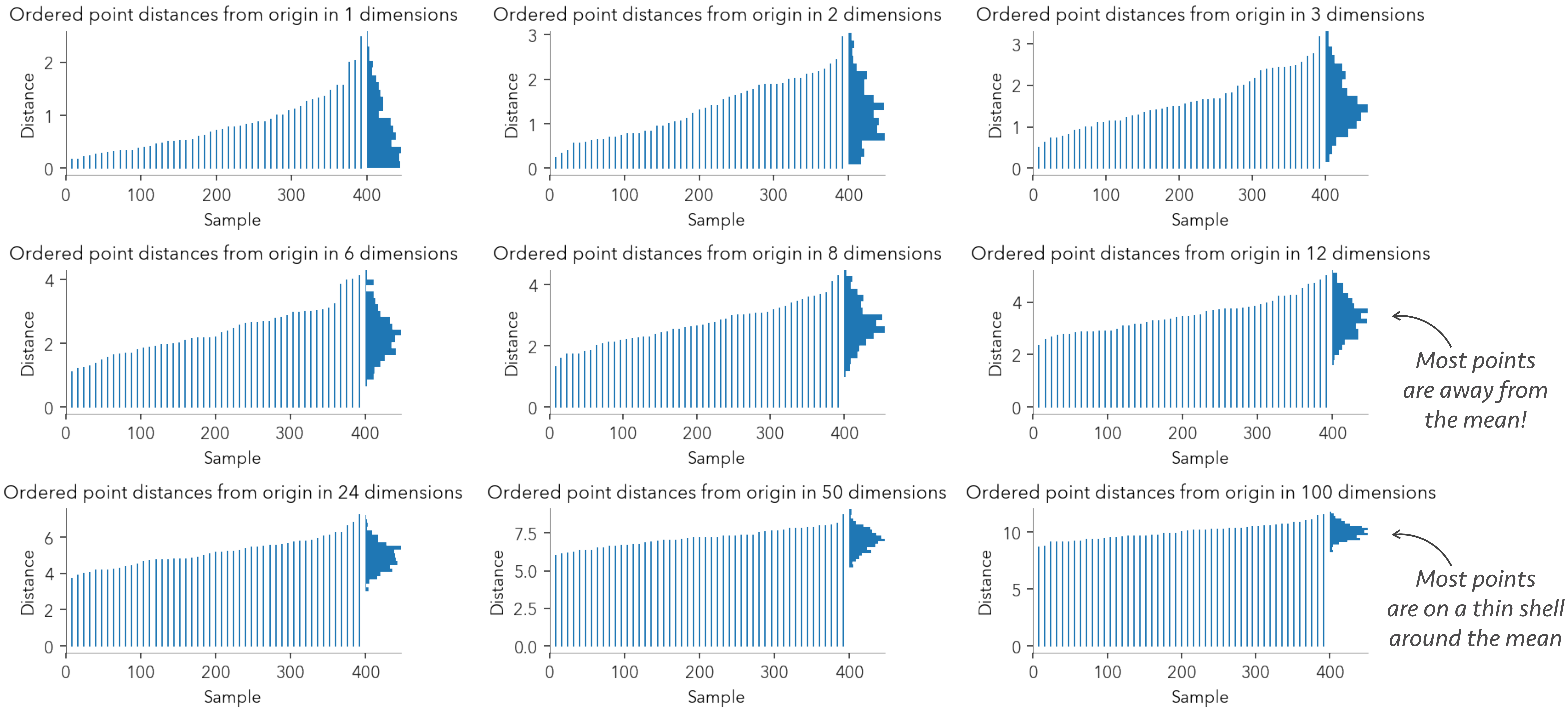
In 3D

- This is starting to look strange
  - Most points are not close to the mean!



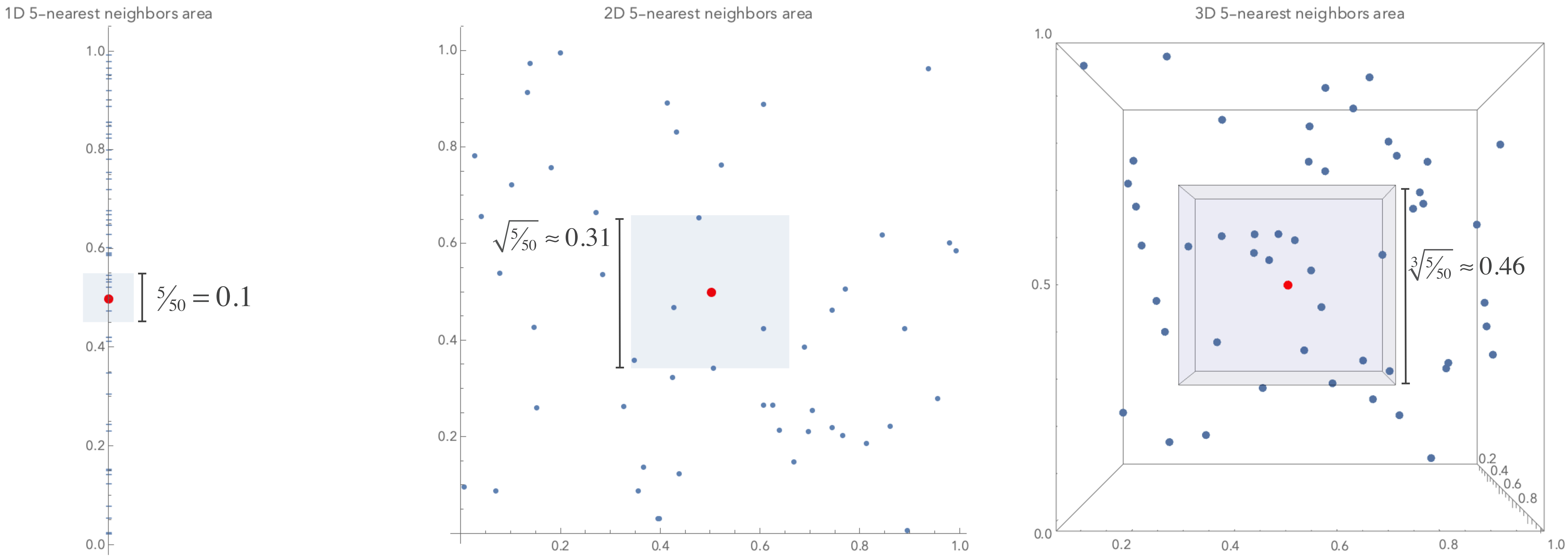


More dimensions, more weirdness



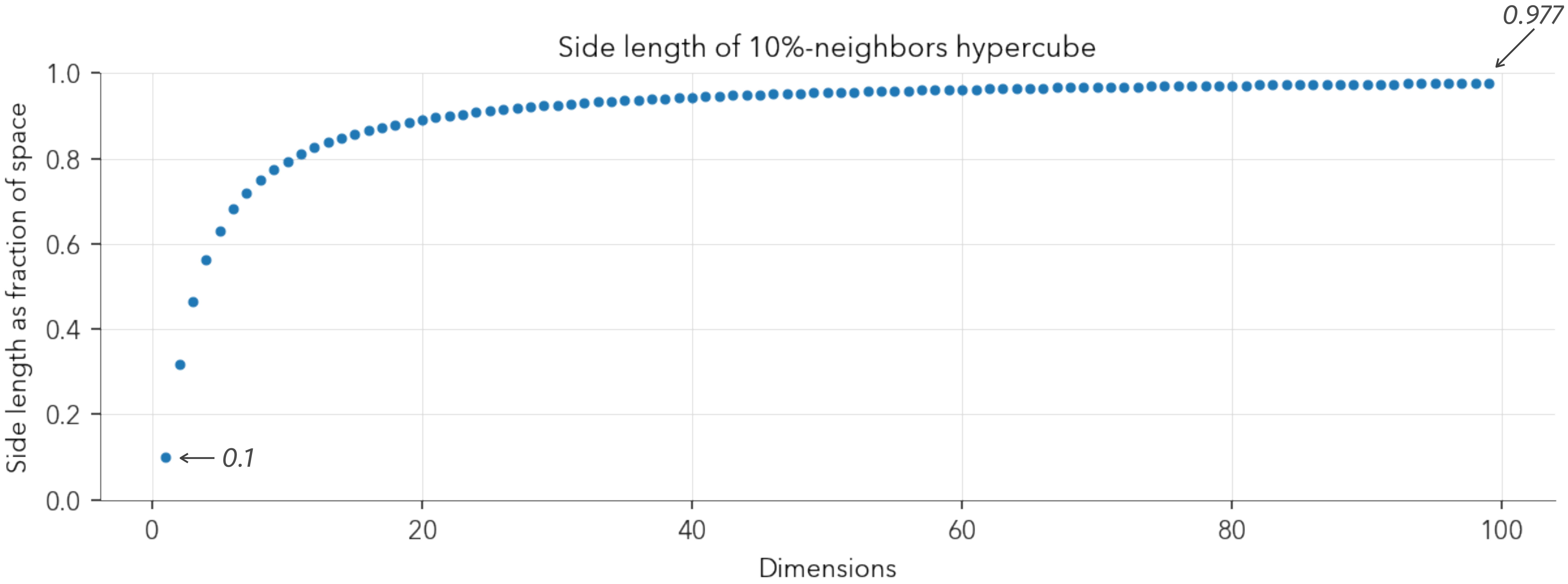
Nearest Neighbor Example

- Get n=5 nearest neighbors from a set of N=50 points
  - In d-dims should be in a hypercube with a side length $(n/N)^{1/d}$



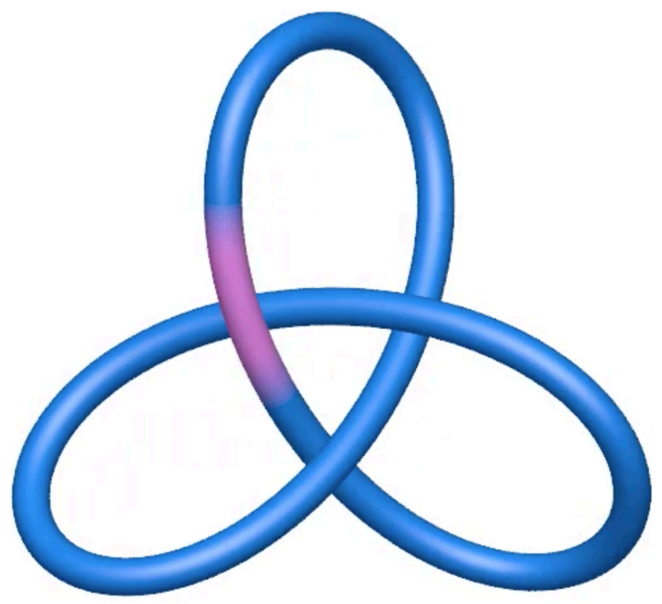
What happens with more dimensions?

- To get a few neighbors in high dimensions we need to almost scan along the full length of all axes!



More dimensional weirdness

- In 4D+ you cannot make string knots
  - No high-dimensional shoelaces!
  - They would always unravel



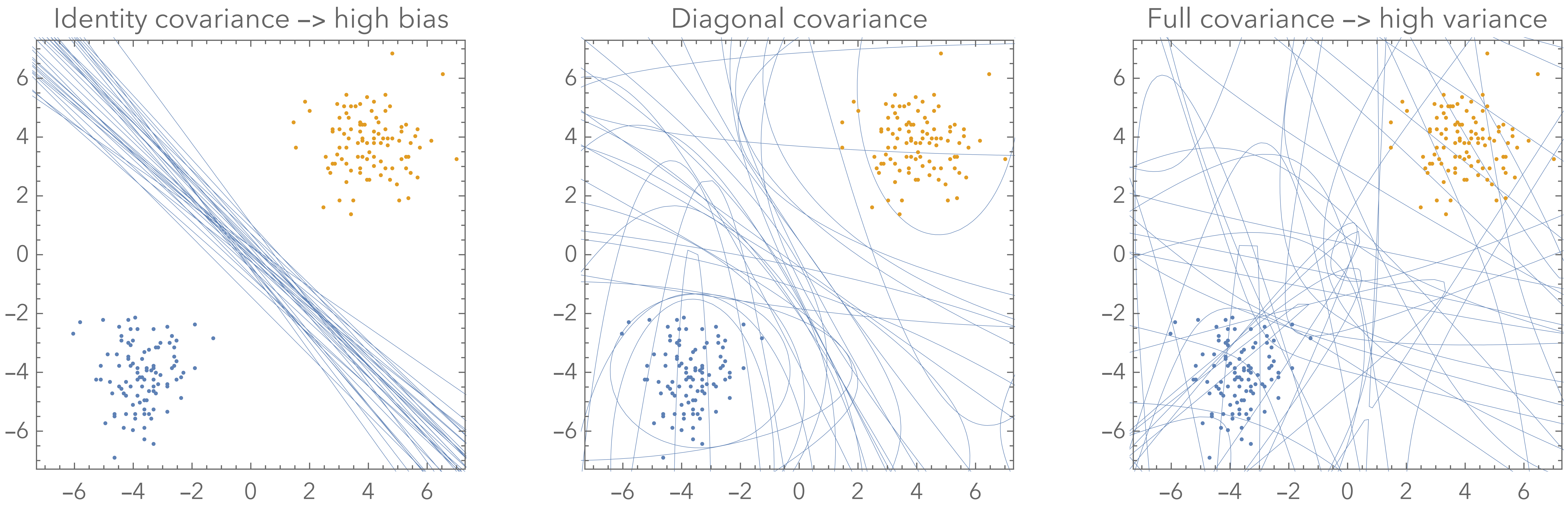
- Planet orbits would be unstable after more than 3D
  - Gravity is proportional to $1/r^{d-1}$
  - Centrifugal will overpower it easily for large d
- The bottom line: Leave your intuition at the door!

Bias/Variance Tradeoff

- Stubborn or flexible?
  - More parameters result in more flexibility, but also a larger margin for misinterpretation of the data
- Bias/Variance tradeoff
  - Few resources == large bias
    - Tends to ignore subtleties due to less learning capacity
  - Many resources == large variance
    - Potentially learns irrelevant details since it can

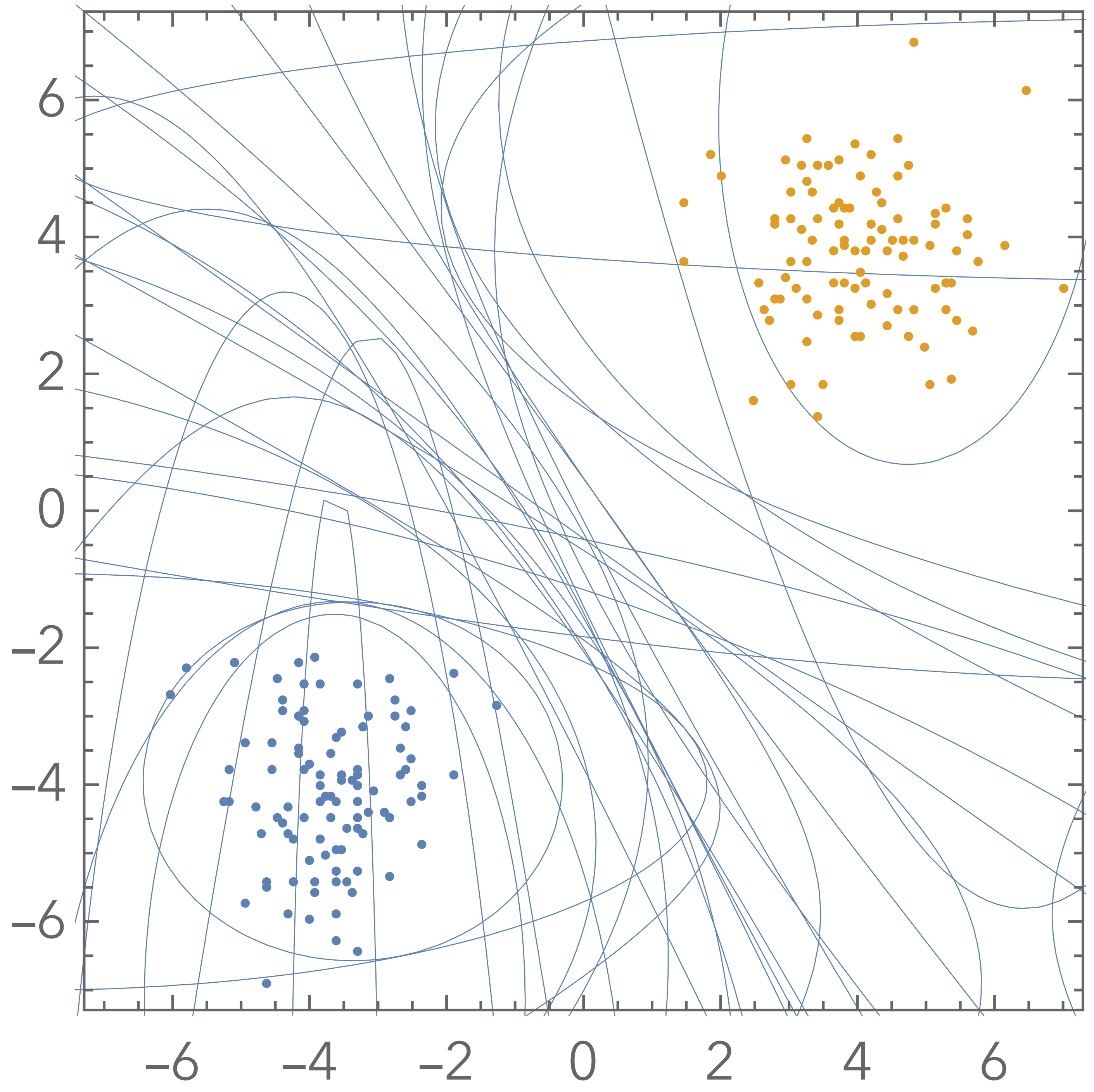
Learning classifiers on 3-sample subsets

- Simpler covariances have more bias and less variance
  - i.e. consistent decision boundaries

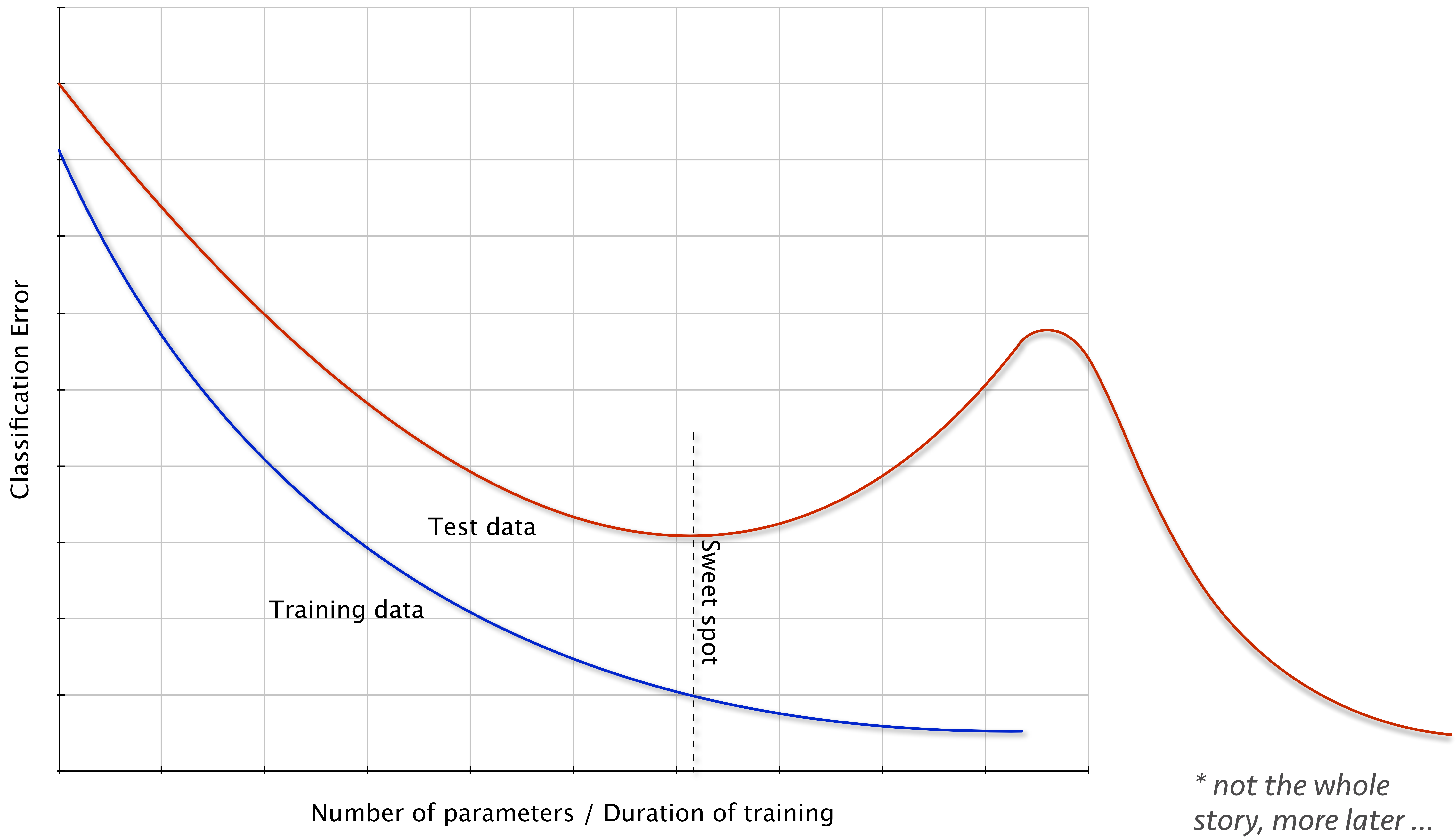


Cross-validation and overfitting

- Test on unseen data
  - Leave a subset of the training data on the side
    - Multiple times if you have to
  - Jackknife, K-fold, etc ...
- Avoid overfitting by comparing training and testing data performance
  - Find the sweet spot



What simple overfitting looks like



** not the whole story, more later …*

G.I.G.O.

- Garbage in, Garbage out
  - Pick your data carefully!
  - Make sure they are plenty, representative, and relatively clean to facilitate training

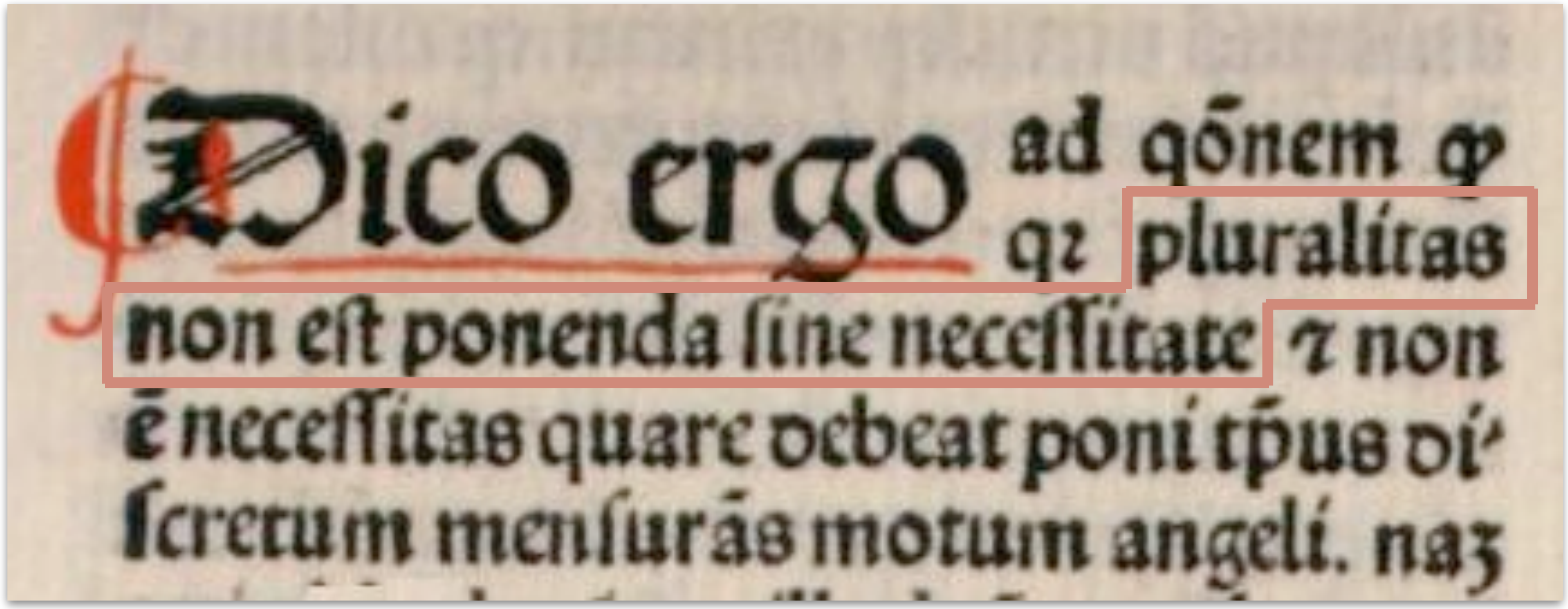
- Garbage in, Gospel out
  - Beware of the bias!

No Free Lunch Theorem

- There is no overall “best” algorithm
  - On average over all possible problems and data, all algorithms will perform the same
- Your goal is to find the best approach for the specific problem you are tackling
  - And that approach will likely differ each time
  - So try many things, there’s no silver bullet

Occam’s Razor

- *“Plurality must never be posited without necessity”*
  - aka K.I.S.S.



Dico ergo ad qonem q q2 pluralitas non est ponenda sine necessitate ⁊ non ē necessitas quare debeat poni tpus di scretum mensuras motum angeli. naz

Occam's Razor

- Don't overcomplicate things!
  - The simplest answer is the better one
    - Assuming both work well
- Don't start with a 10 layer 2,000 node graph neural net boosted by an kernel SVM, after doing LLE
  - Maybe start with a linear classifier …
  - Beware of overfitting

Ockham chooses a razor

A generic gameplan

- Carefully pick your data
- Reduce dimensionality if it’s too high
- Start with simple classifiers
  - Complicate things only if needed
- Evaluate on train and test data
  - Find optimal parameter # and training time spot
- But also break the rules if you have to!

Recap

- Linear Discriminant Analysis
- Non-parametric methods
- Boosting
- Some general advice

Next lecture

- What if we don’t have class labels?
- Clustering and segmentation
- The EM algorithm

Reading

- Textbook reading, sections:
  - 5.8, 2.6, 3.5.3, 3.7.3, 2.5.6, 4.21, 4.22