

CS598 – Machine Learning for Signal Processing

Classification: The rest of the story

Today's lecture

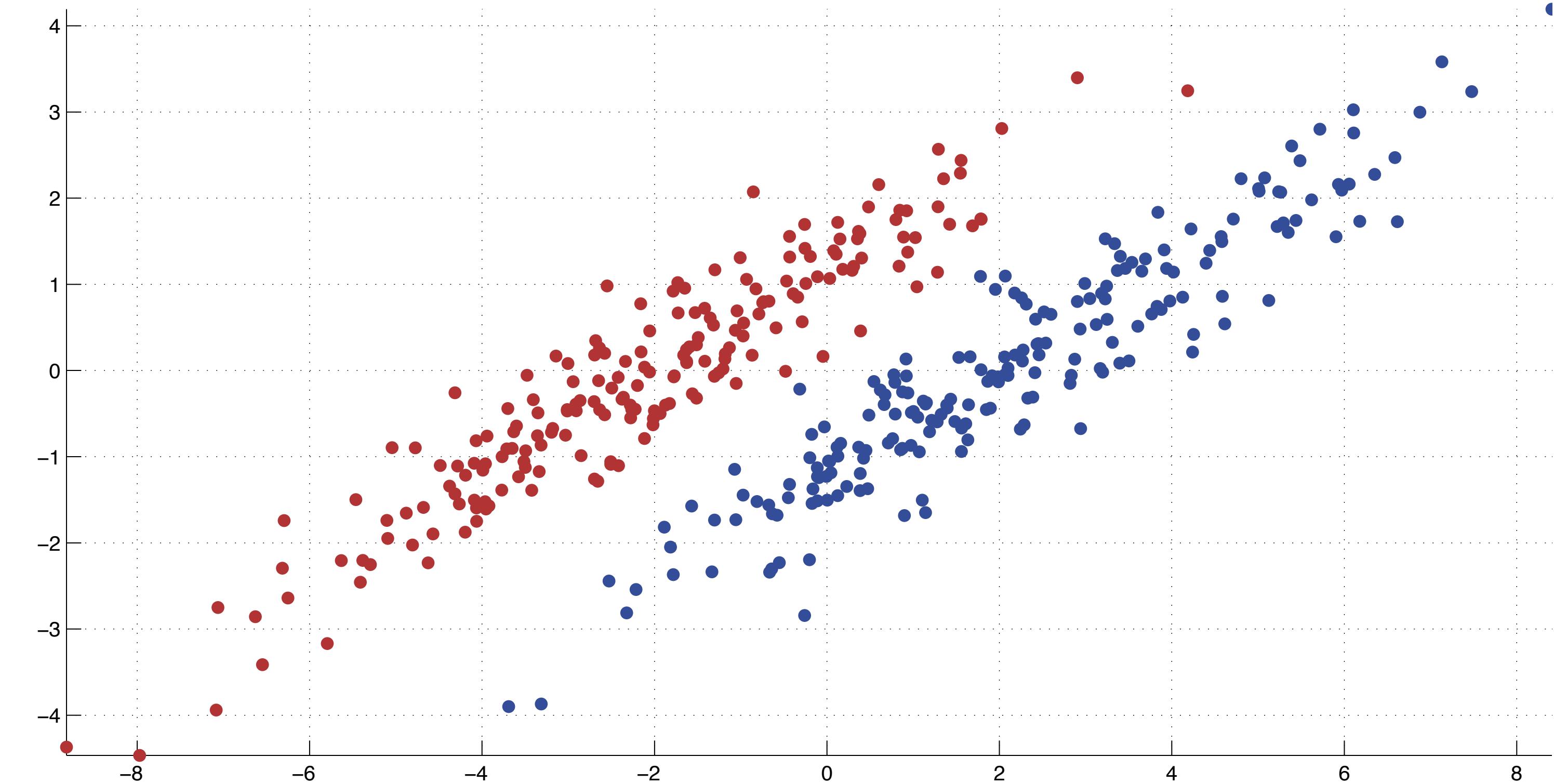
- Important things we haven't covered yet
 - Fisher Linear Discriminant Analysis
 - Nearest-neighbors classification
 - Combining multiple classifiers
 - Multiple class classification
- Setting up experiments
 - Strategy
 - Tradeoffs

Optimal feature discovery

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

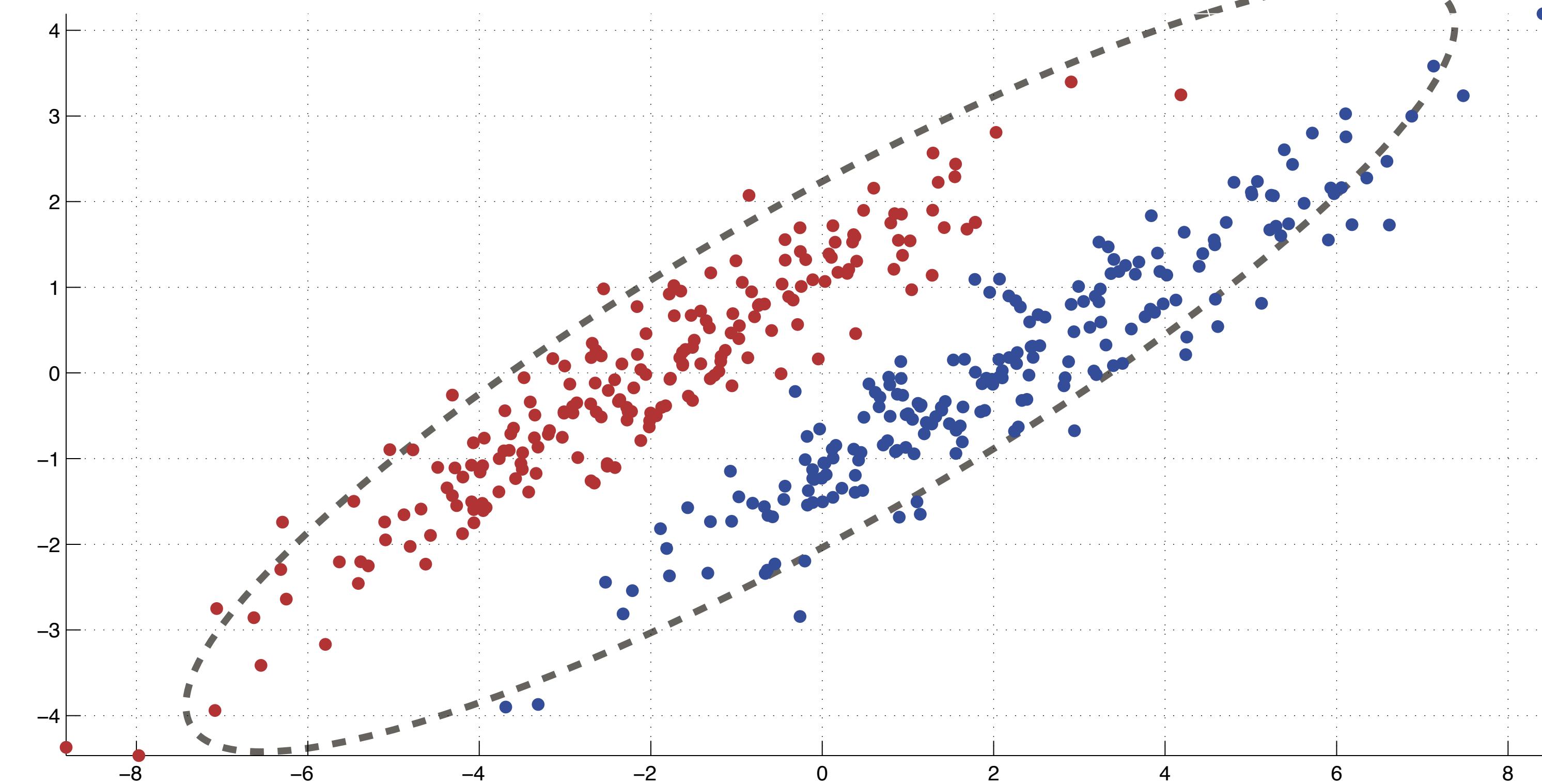
An example

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



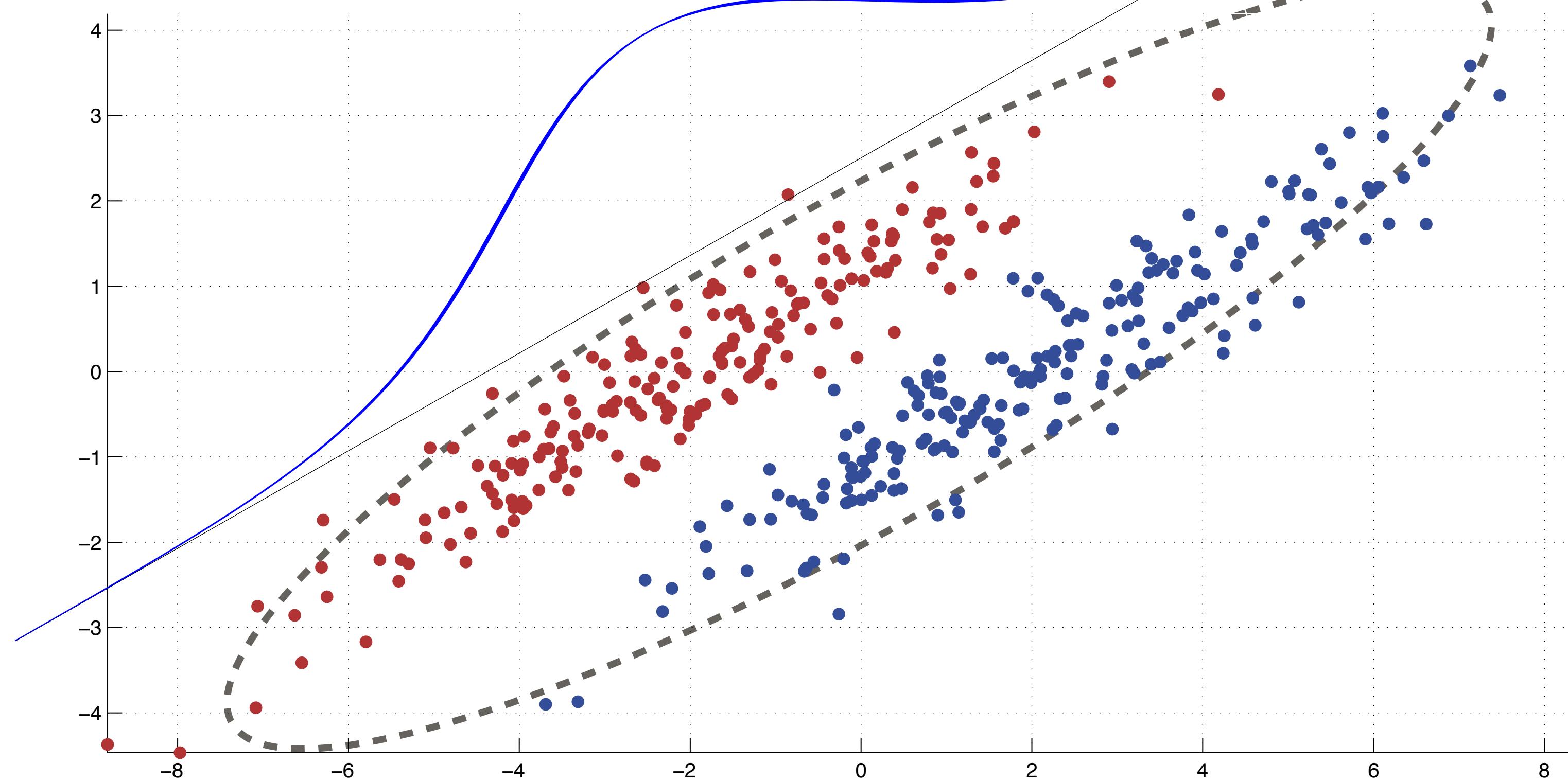
An example

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



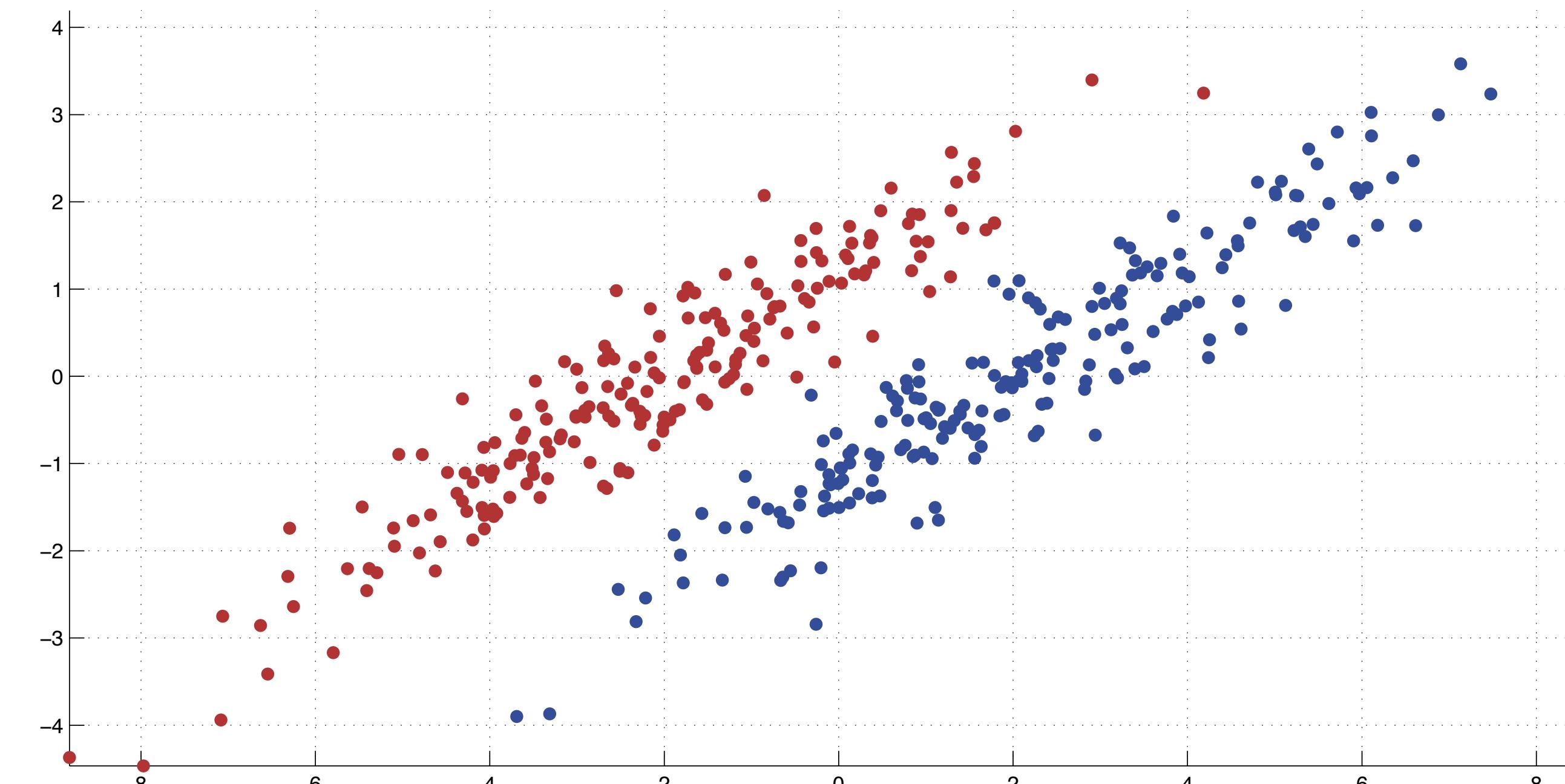
An example

- 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 doing the right thing
- What's the right thing to do?



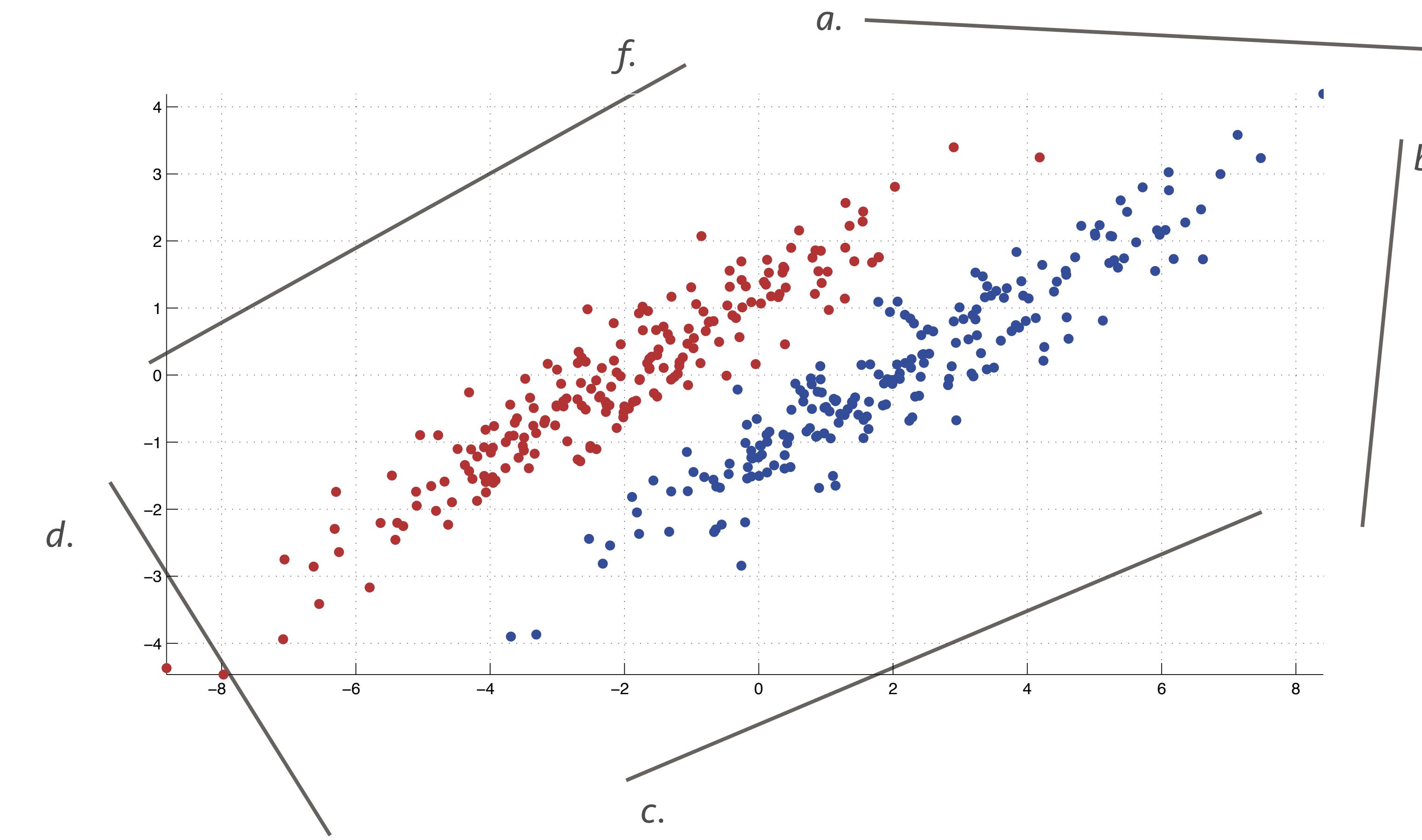
New plan

- 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:

$$\underset{\text{Maximize}}{\longrightarrow} F = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \begin{array}{l} \leftarrow \text{Maximize} \\ \leftarrow \text{Minimize} \end{array}$$

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

$$\mu_i = E\{y^{(i)}\} = E\{\mathbf{w}^\top \cdot \mathbf{x}^{(i)}\} = \mathbf{w}^\top \cdot E\{\mathbf{x}^{(i)}\} = \mathbf{w}^\top \cdot \mathbf{m}_i$$

$$\sigma_i^2 = E\{(y^{(i)} - \mu_i)^2\} = E\{\mathbf{w}^\top \cdot (\mathbf{x}^{(i)} - \mathbf{m}_i)(\mathbf{x}^{(i)} - \mathbf{m}_i)^\top \cdot \mathbf{w}\} = \mathbf{w}^\top \cdot \Sigma_i \cdot \mathbf{w}$$

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:

$$S_w = \sum P_i \Sigma_i$$

- “Within-class scatter matrix”
- P_i is i^{th} class prior
- Σ_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 w that maximizes:

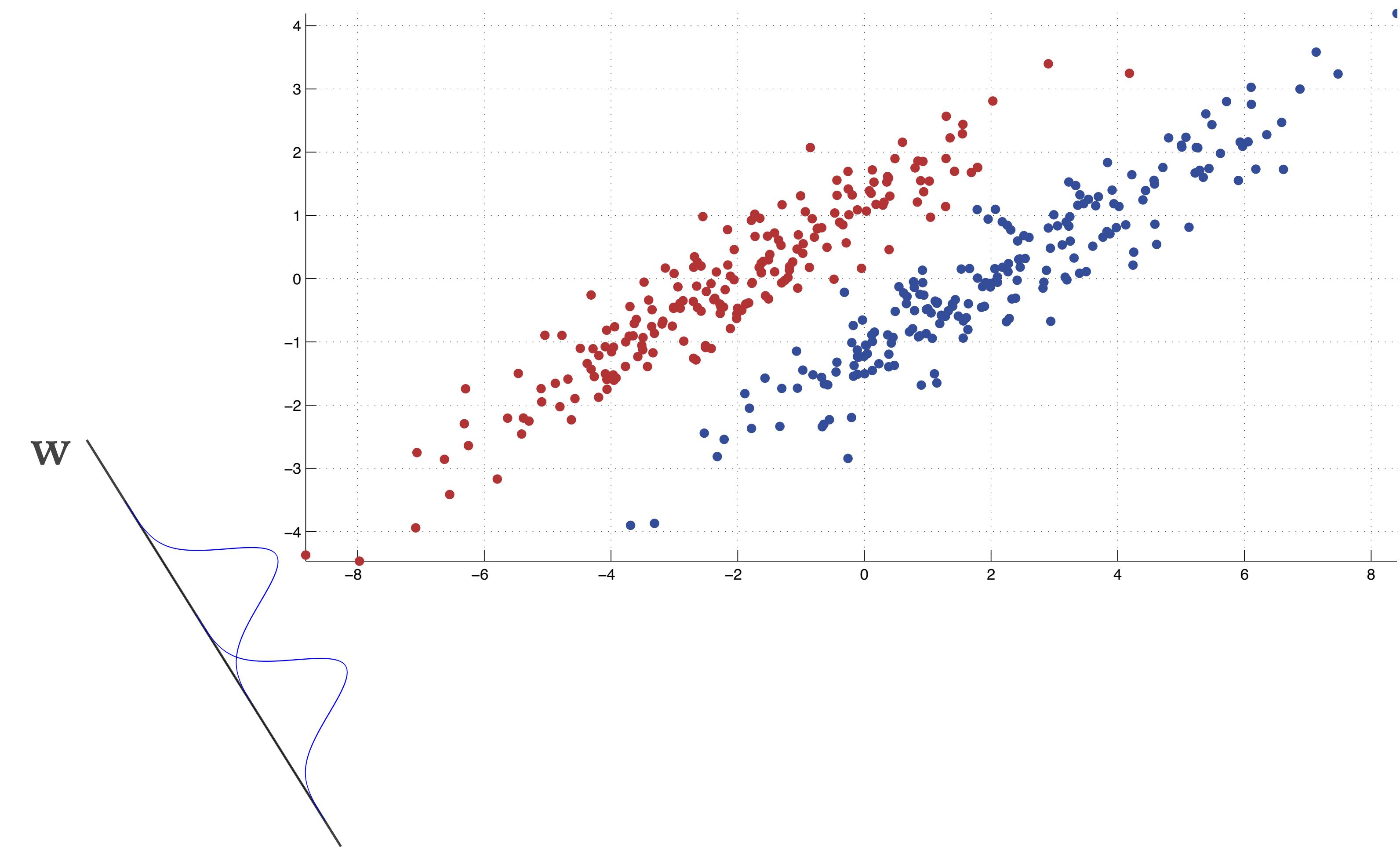
$$F(w) = \frac{\mathbf{w}^\top \cdot S_b \cdot \mathbf{w}}{\mathbf{w}^\top \cdot S_w \cdot \mathbf{w}}$$

- 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

- Transform is now a matrix:

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

- New objective is defined as:

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

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 offers no advantage (zero eigenvalues)
 - Fewer produces a loss in discrimination

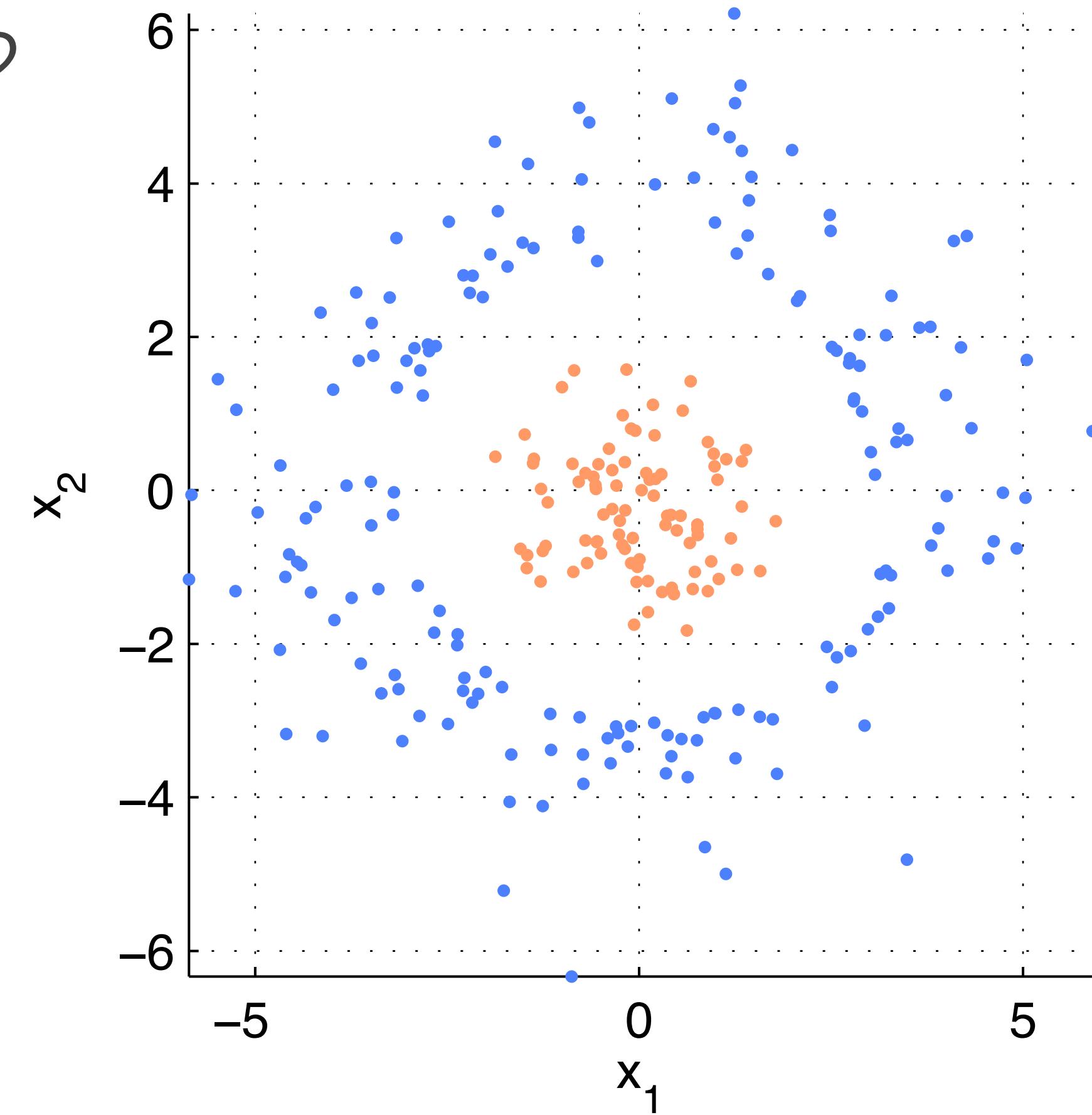
What about non-linear projections?

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

- Use a kernelized 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 ...

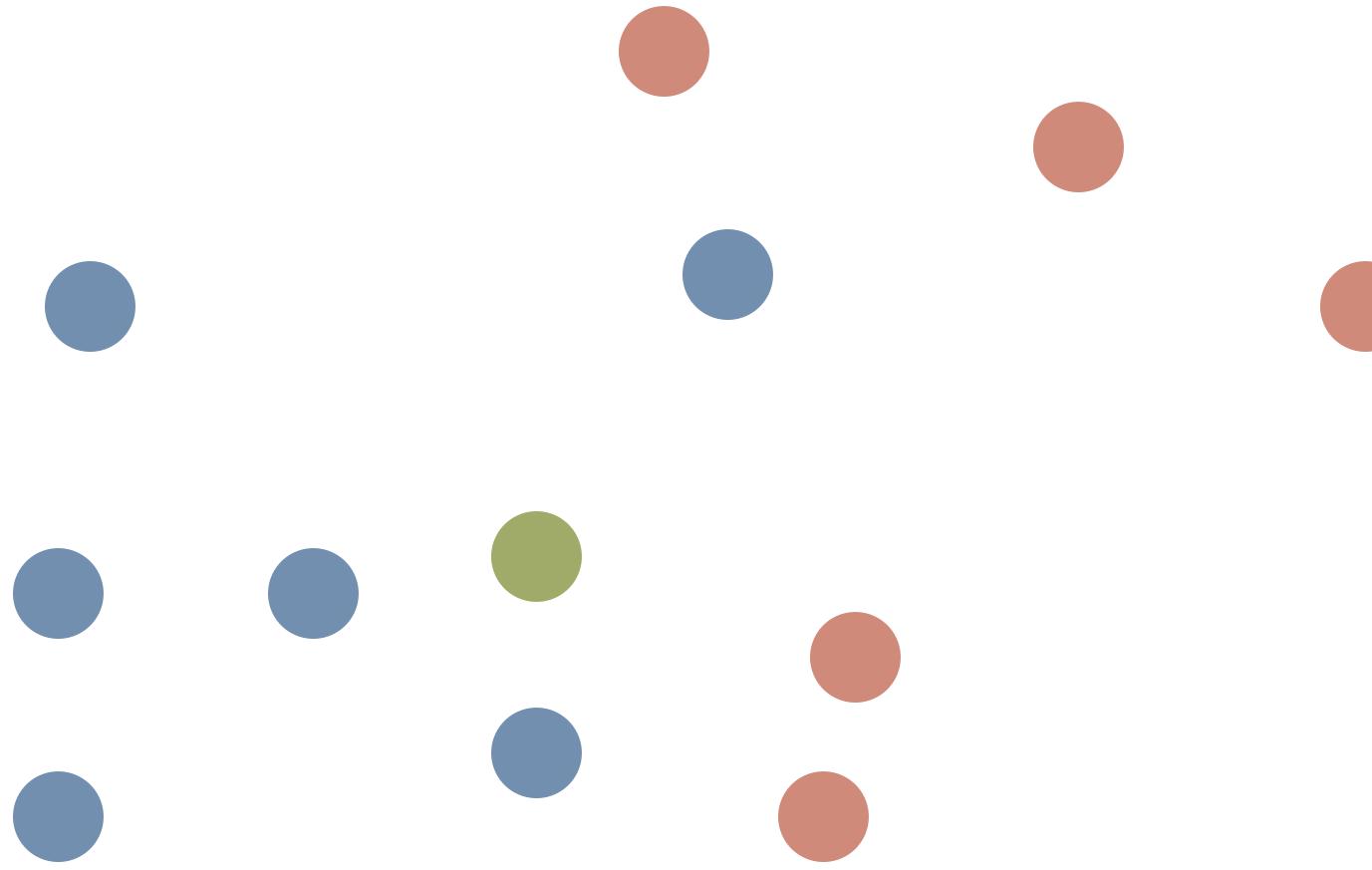


Non-parametric approaches

- 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?

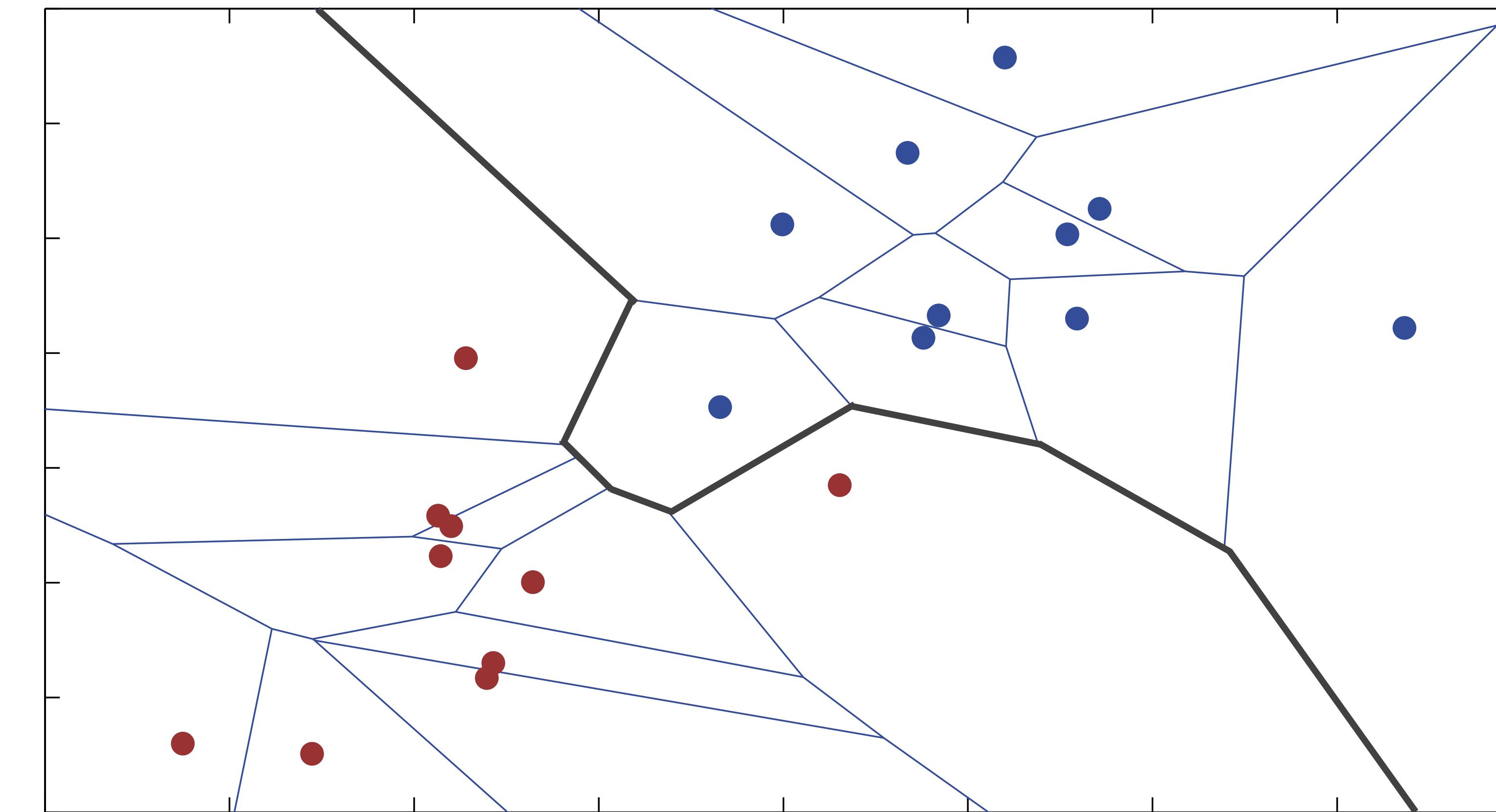


Nearest neighbor rule

- 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 tessellation



Generalizing

- What class is the green sample now?

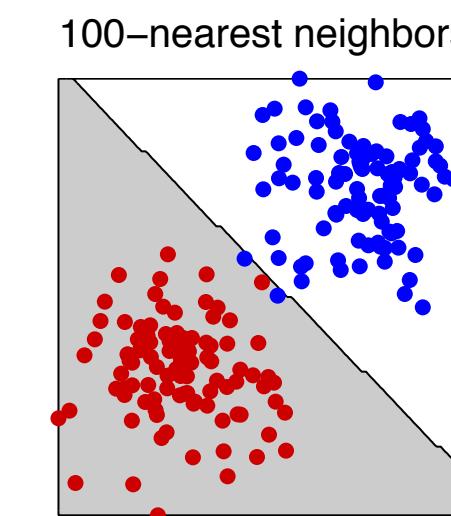
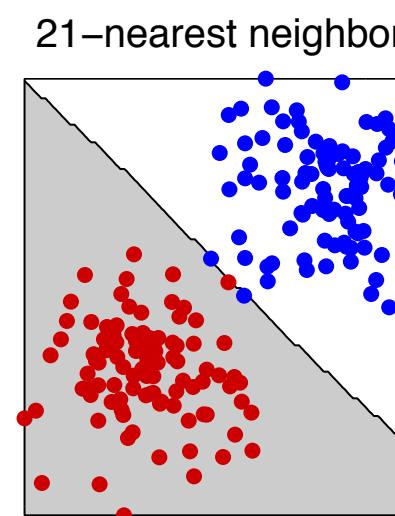
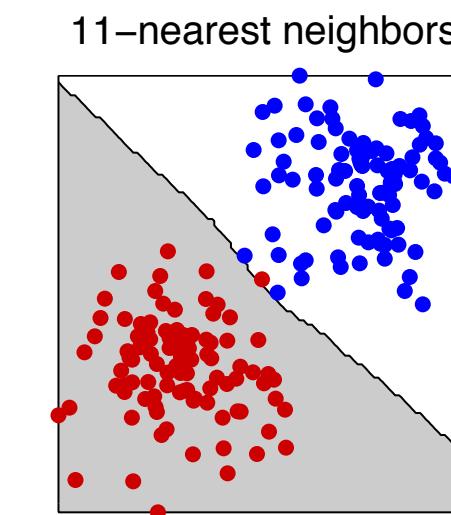
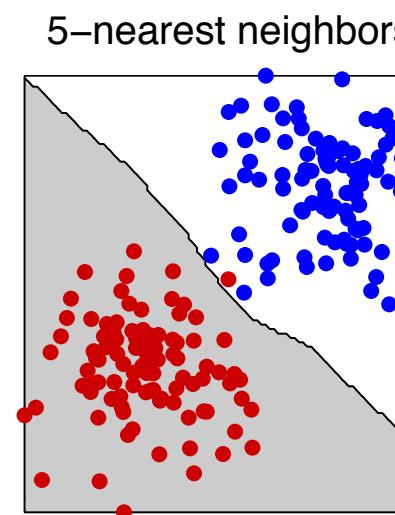
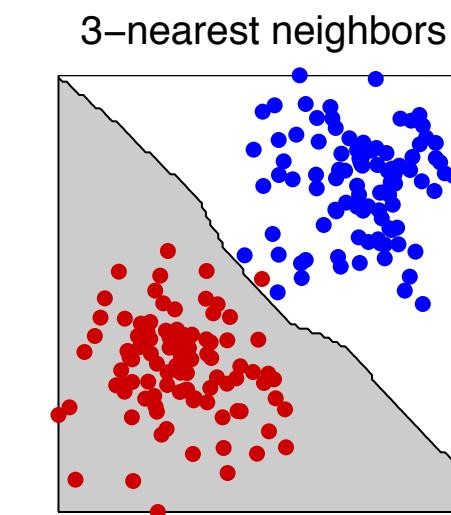
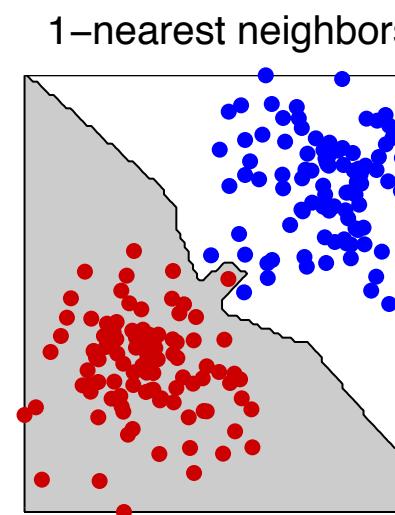


k-nearest neighbors

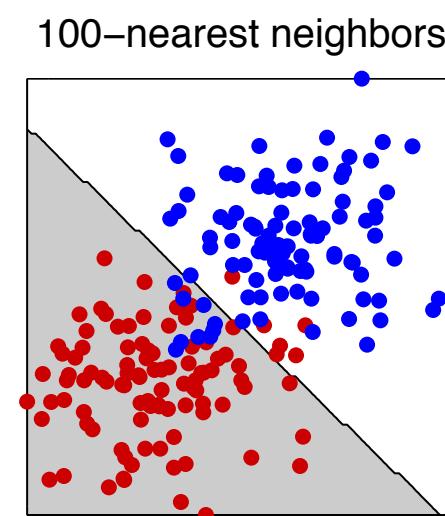
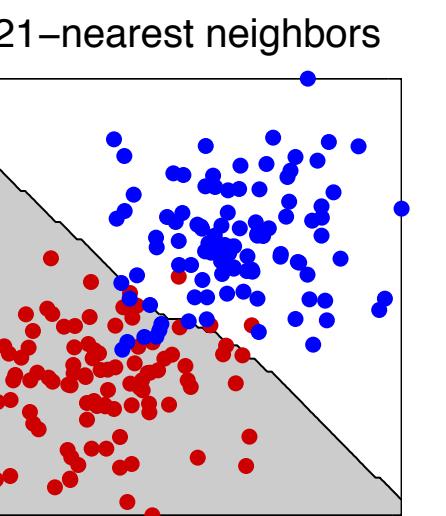
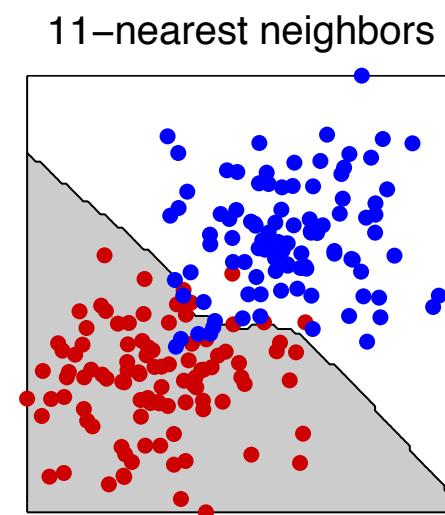
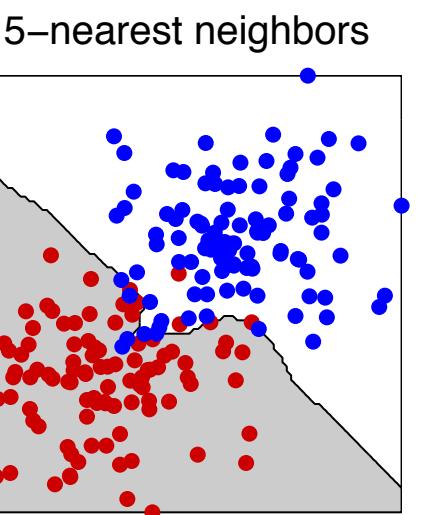
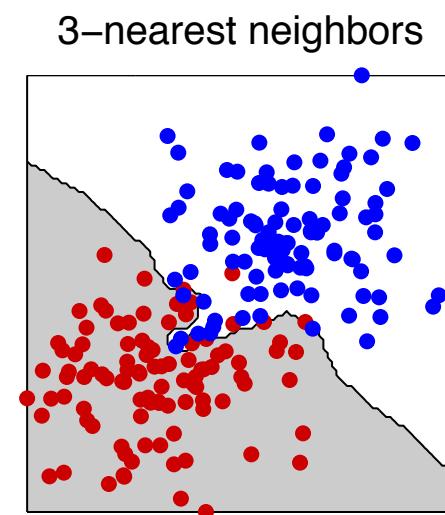
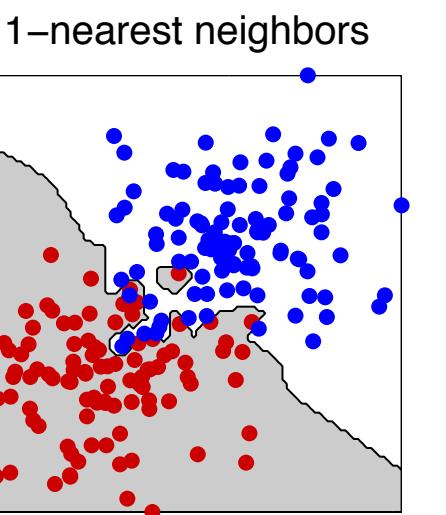
- Same as before, but count K neighbors
 - Compare input to all training data
 - Find K closest neighbors
 - Assign class to the majority of neighbors
- Pick an odd K to avoid ties!
- Results in varying decision boundaries
 - A bigger K generalizes more

Example cases

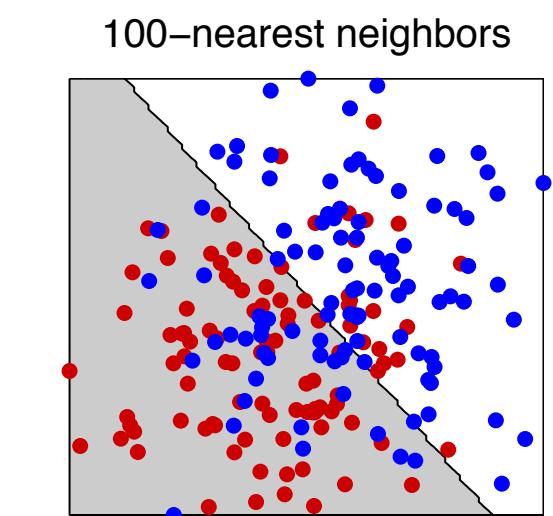
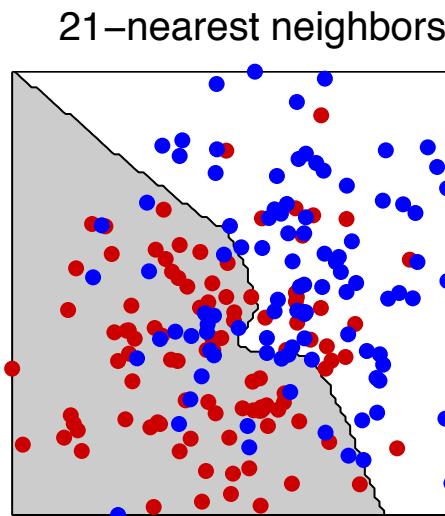
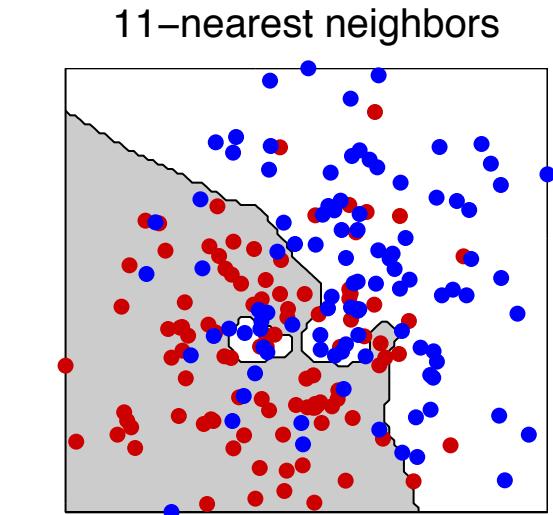
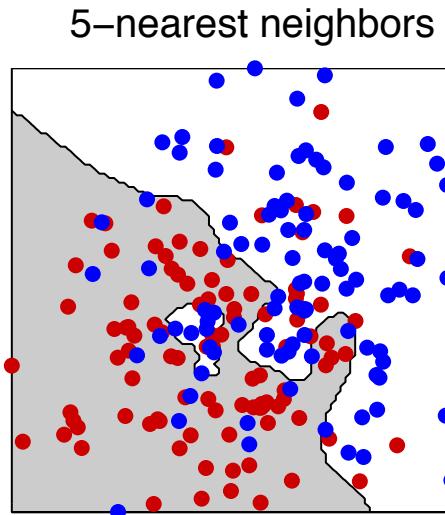
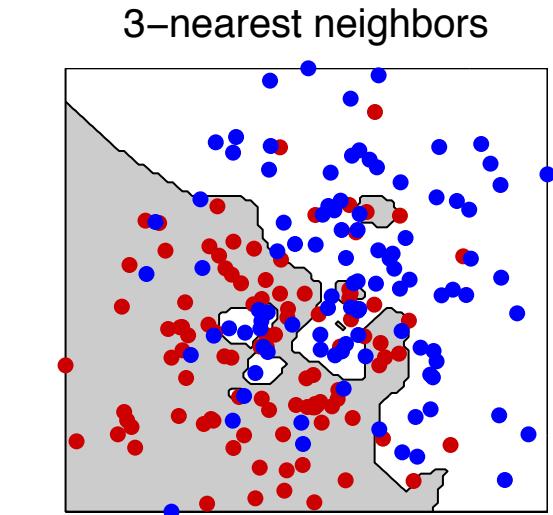
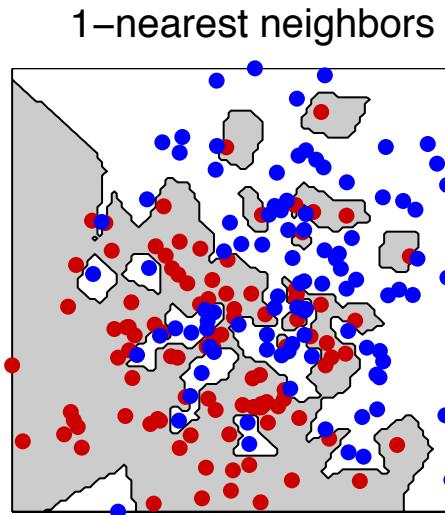
Easy classification problem



Harder classification problem



Really hard classification problem

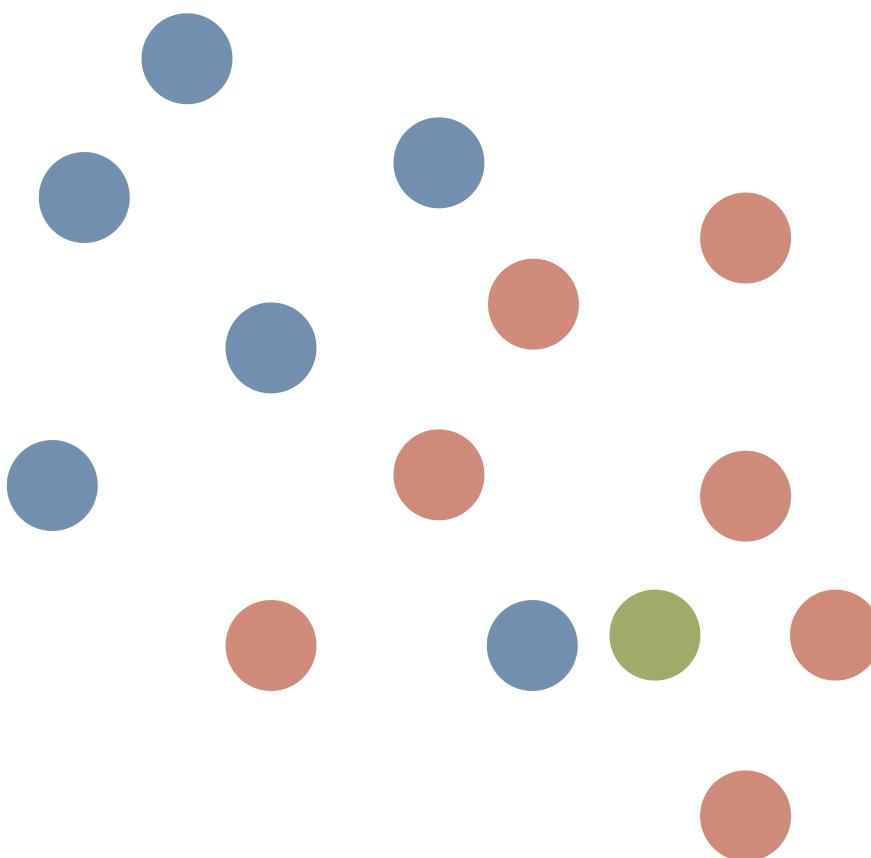


kNN pros and cons

- 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
 - Will choke at high dimensions (more later)
 - 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?



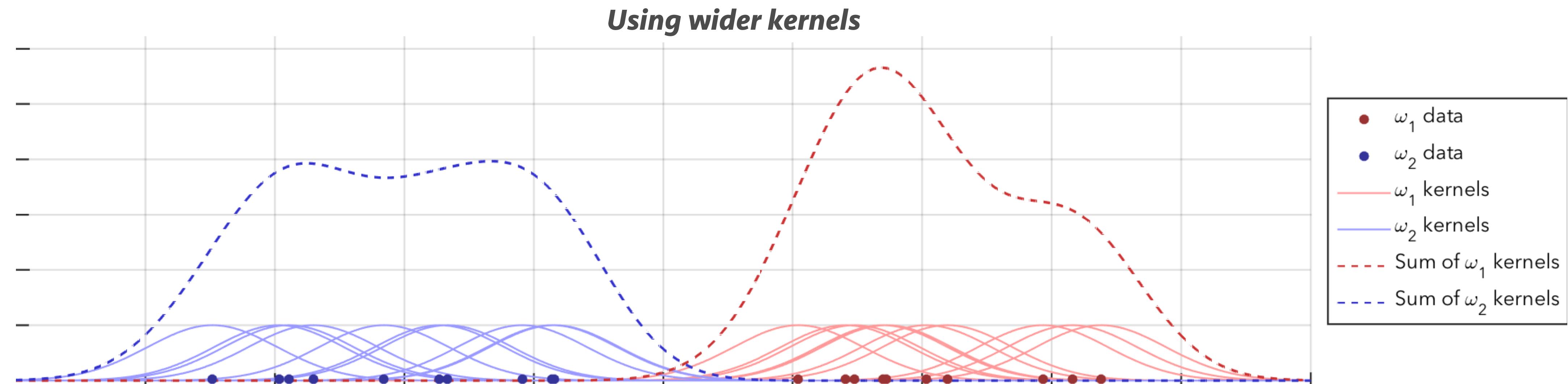
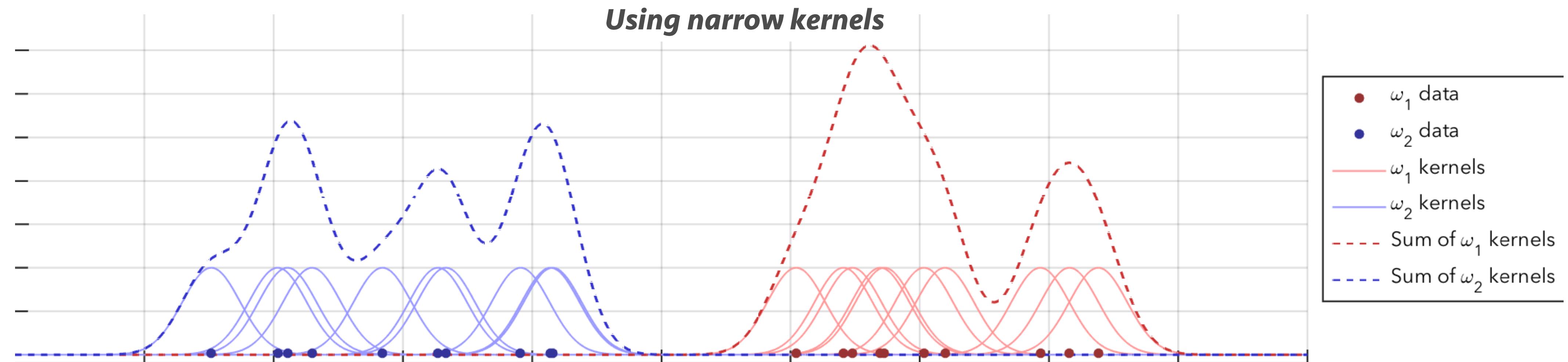
Parzen estimation

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

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

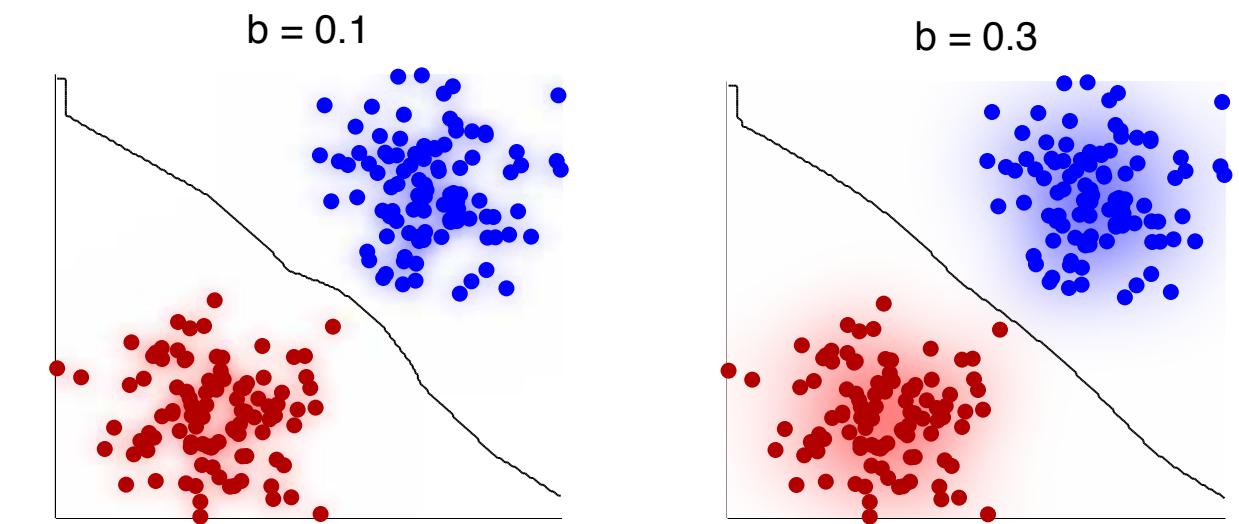
- These kernels get superimposed to approximate the distribution of the data

1-D example

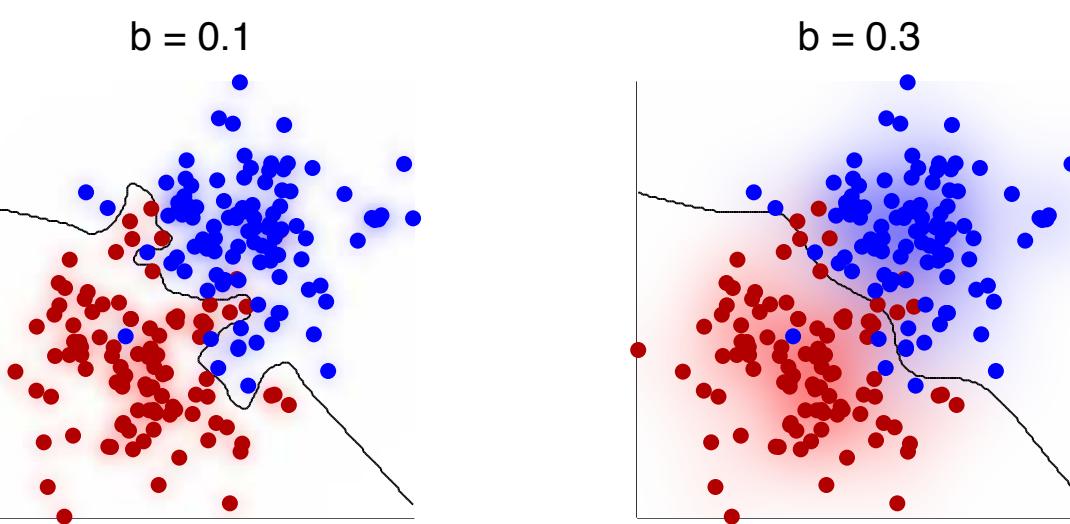


Example cases

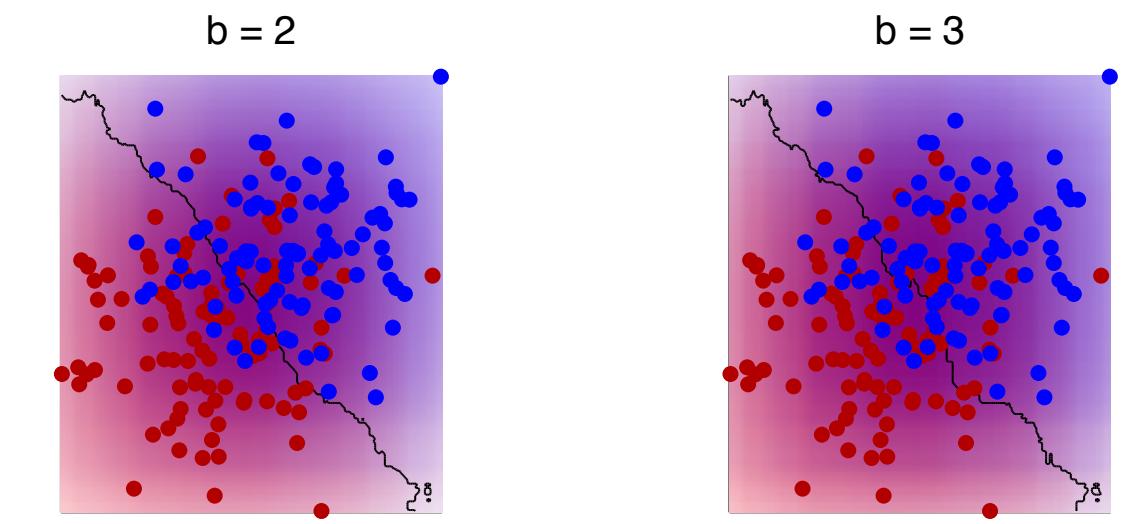
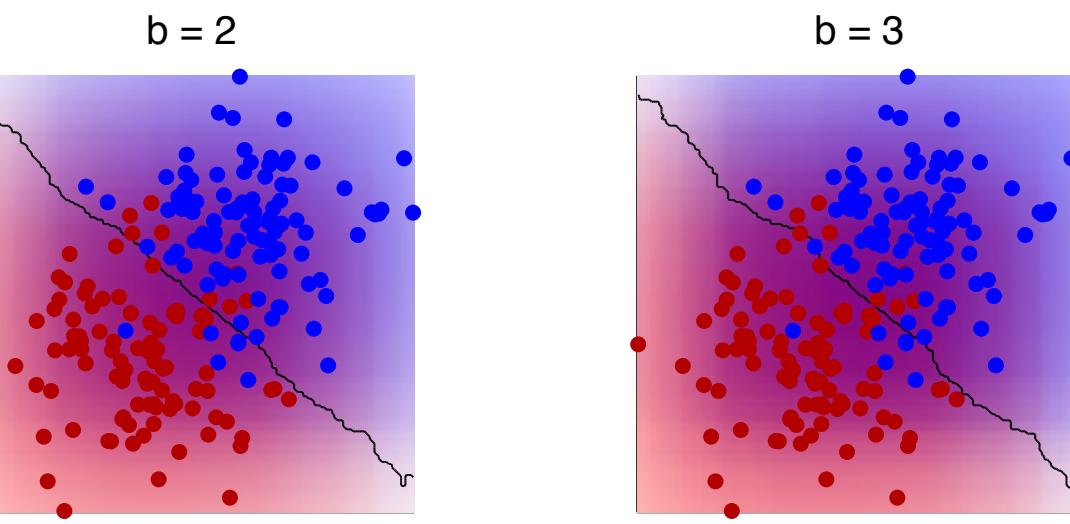
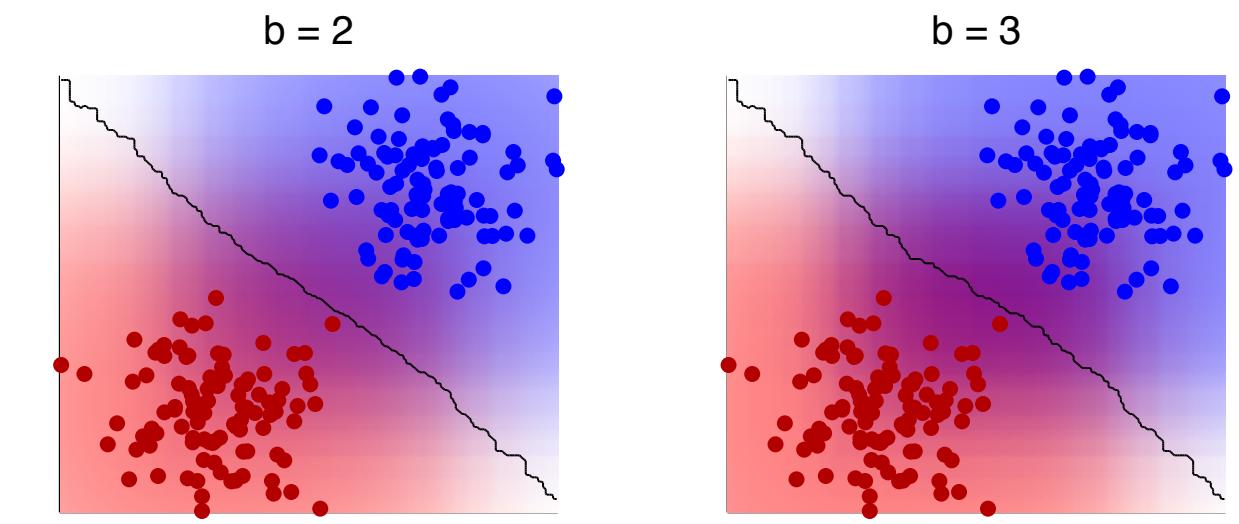
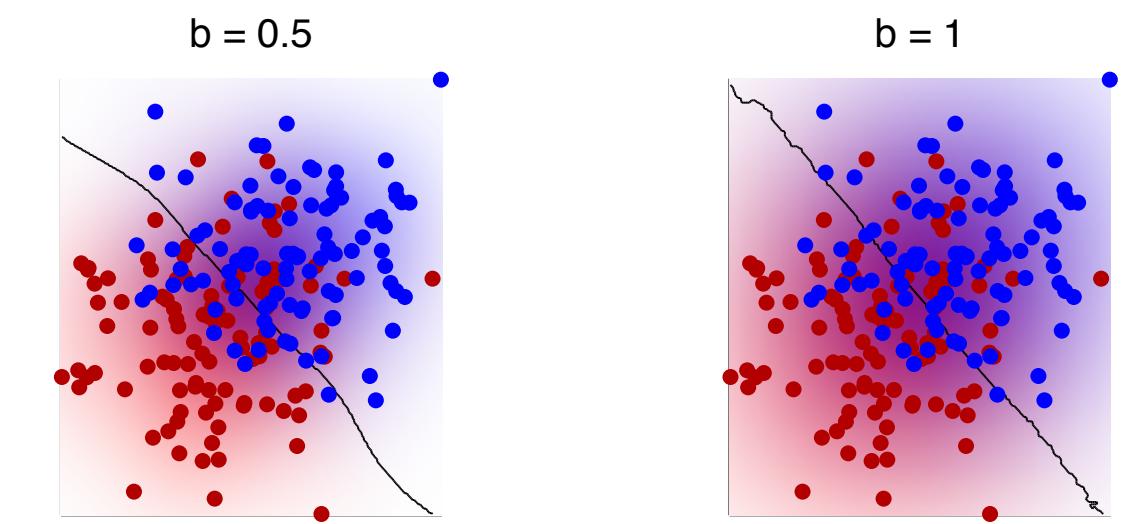
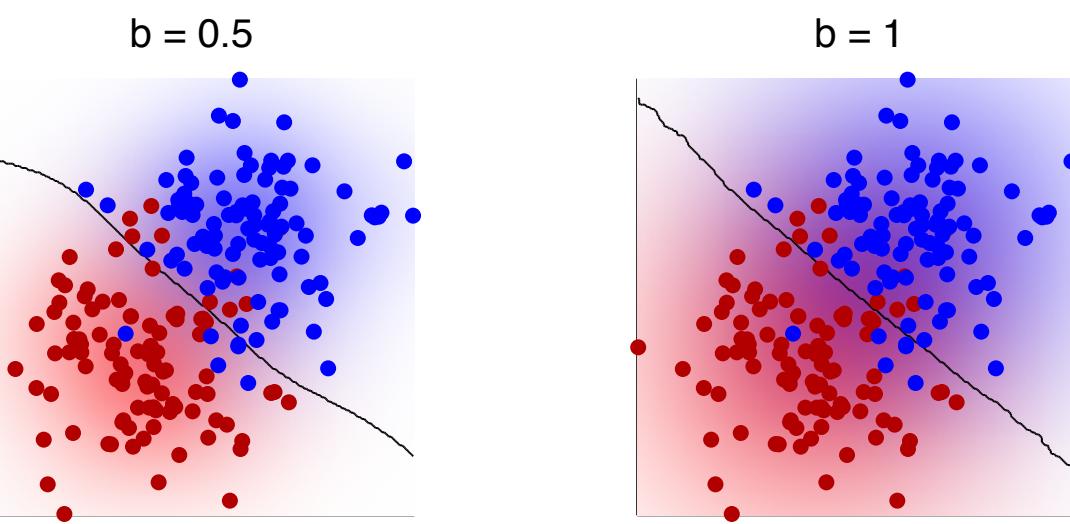
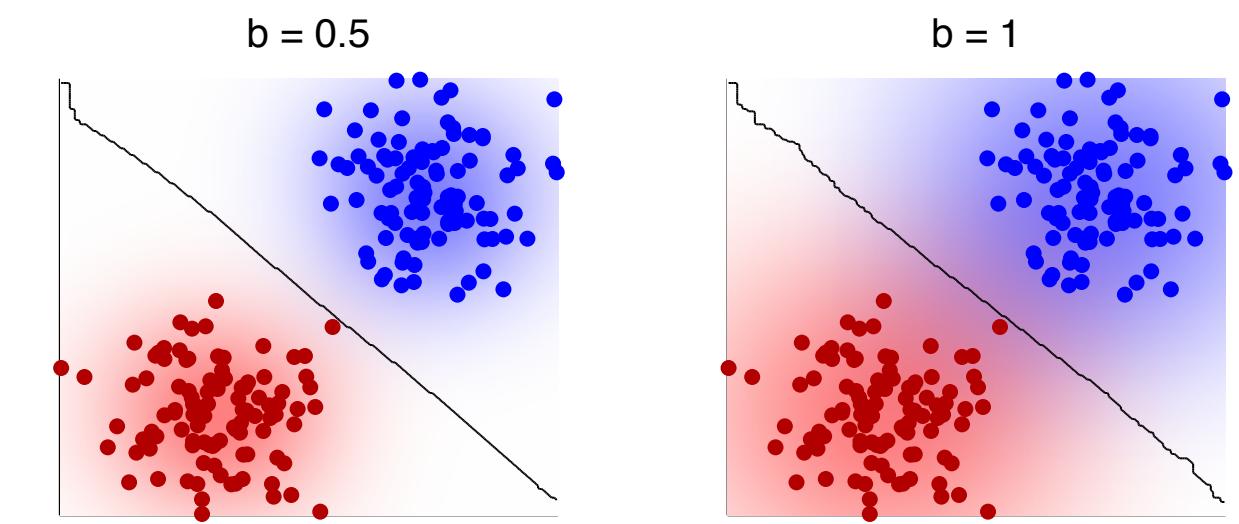
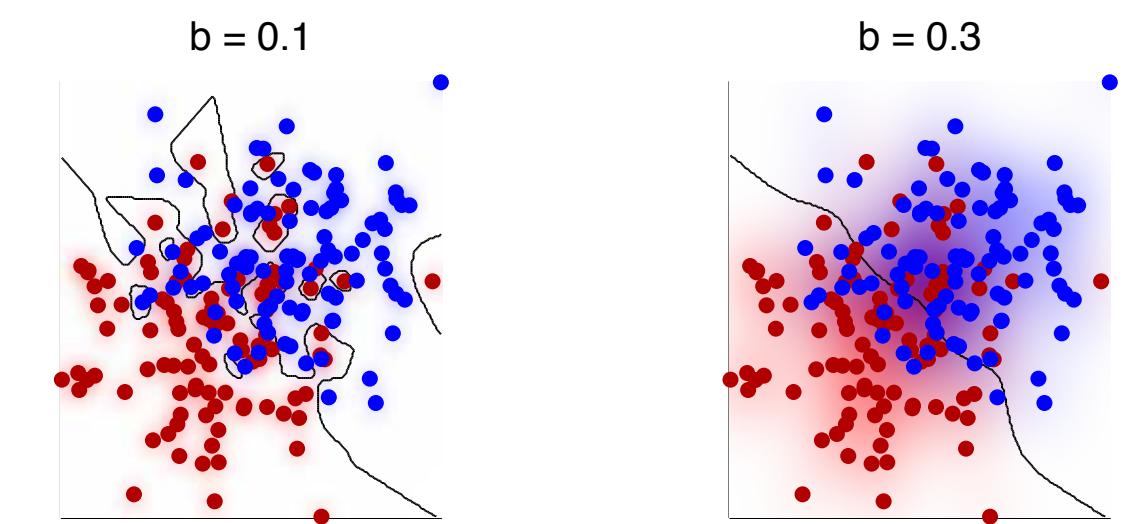
Easy classification problem



Harder classification problem



Really hard classification problem



Parzen vs. kNN

- Parzen
 - Smooth boundaries
 - Probabilistic interpretation
 - Costly in many dimensions
 - Computation heavy
- kNN
 - Unorthodox boundaries
 - Hard assignments
 - Unreliable in high dimensions
 - 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 approach...
 - “Meta-learning”

Boosting

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

AdaBoost formulation

- Classify using: $f(\mathbf{x}) = \text{sgn}\left(F(\mathbf{x})\right)$
$$F(\mathbf{x}) = \sum a_k \phi(\mathbf{x}; \theta_k)$$
- Each ϕ being a classifier with parameter θ_k
- The parameters to optimize:

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

- Errors are being penalized more

An incremental approach

- Split classifier output in current and past contributions:

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

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

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

An incremental approach

- Which separates to:

$$\theta = \arg \min_{\theta} \sum w_i^{(m)} e^{-y_i a \phi(\mathbf{x}_i; \theta)}$$
$$w_i^{(m)} \propto e^{-y_i F_{m-1}(\mathbf{x}_i)}$$

- i.e. learn a classifier that focuses on past failures
- The new weight will be:

$$a_m = \frac{1}{2} \log \frac{1 - \varepsilon_m}{\varepsilon_m}$$
$$\varepsilon_m = \sum_i w_i^{(m)} \left(y_i \neq \text{sgn}(\phi(\mathbf{x}_i; \theta)) \right)$$

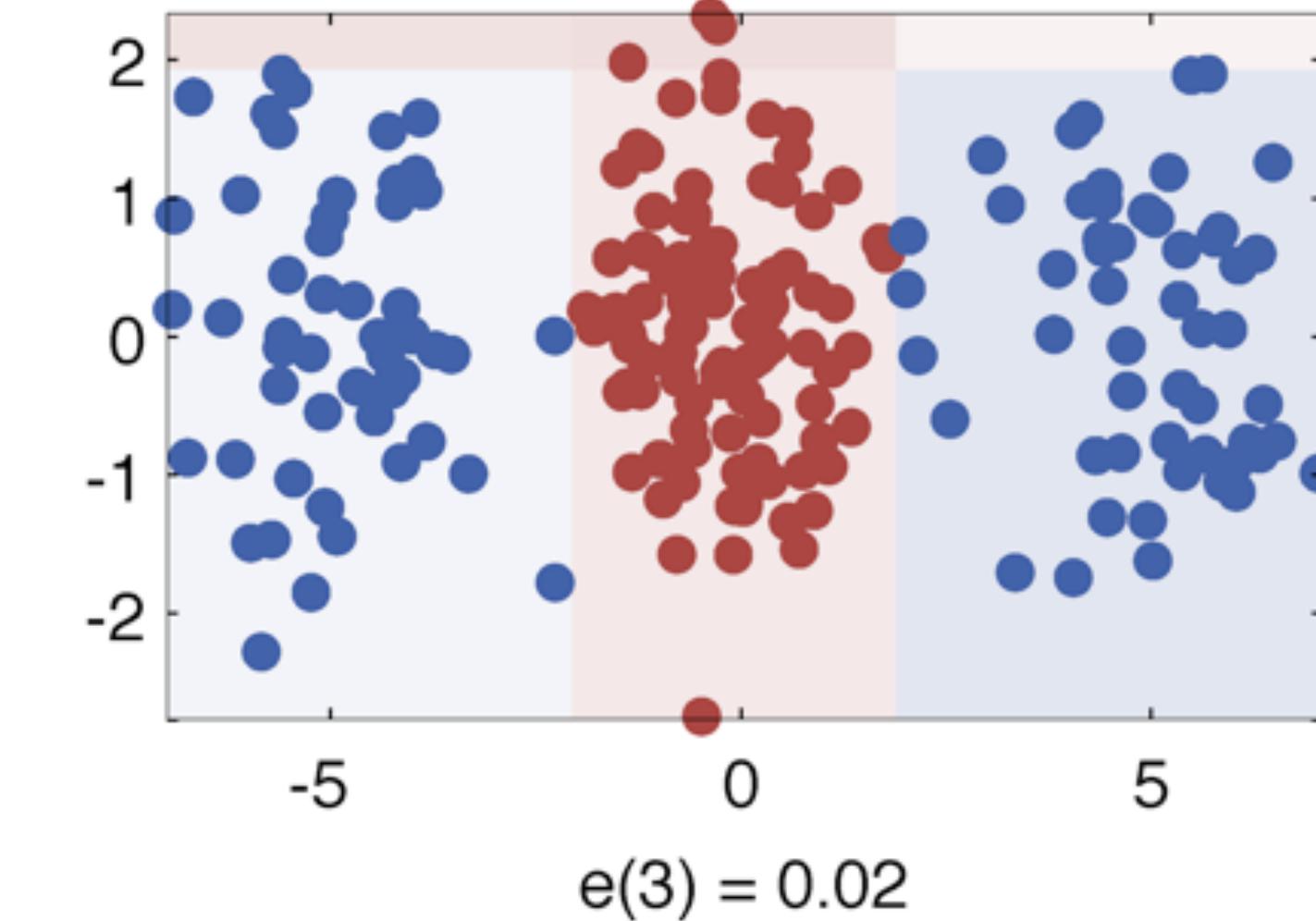
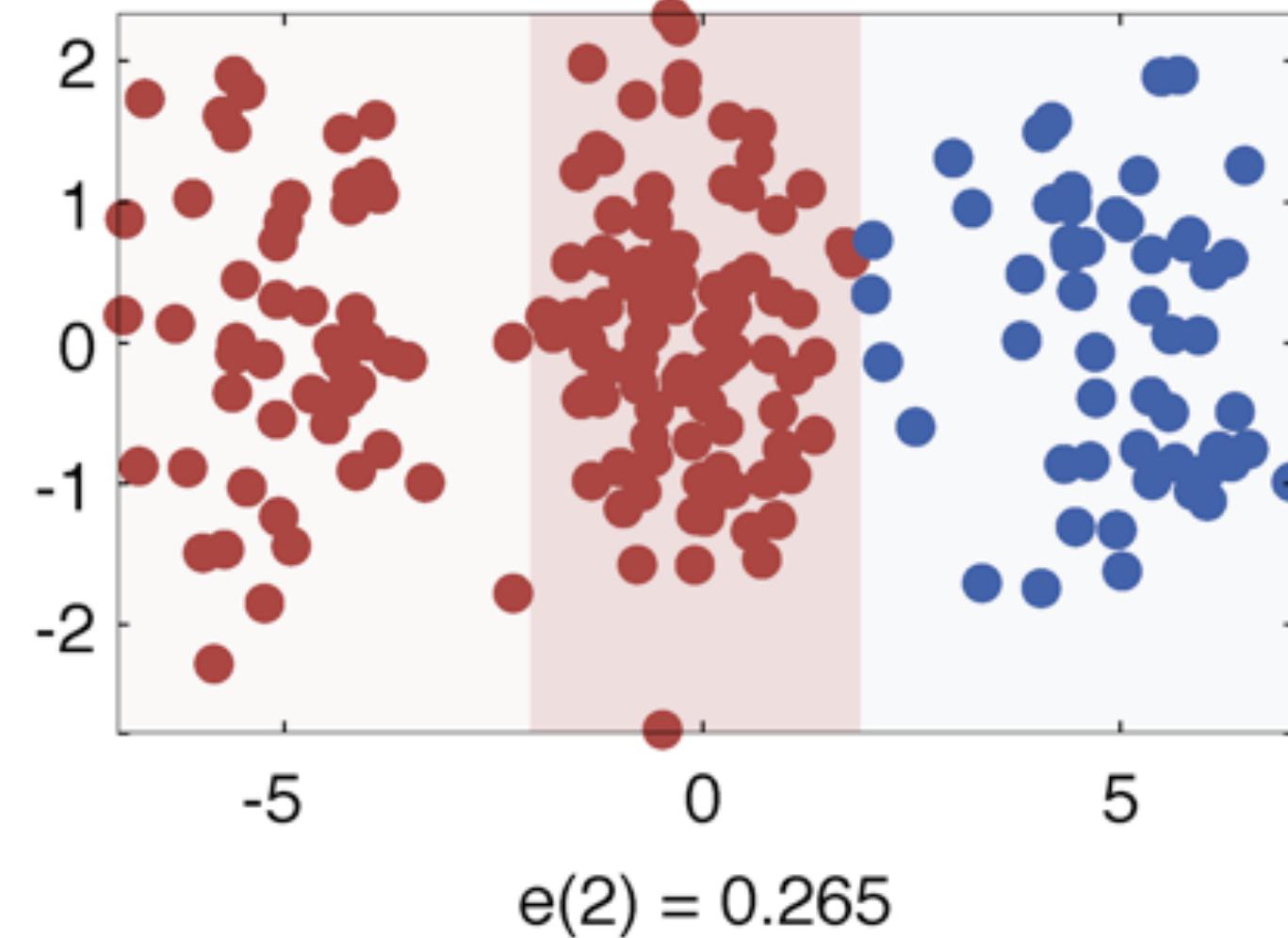
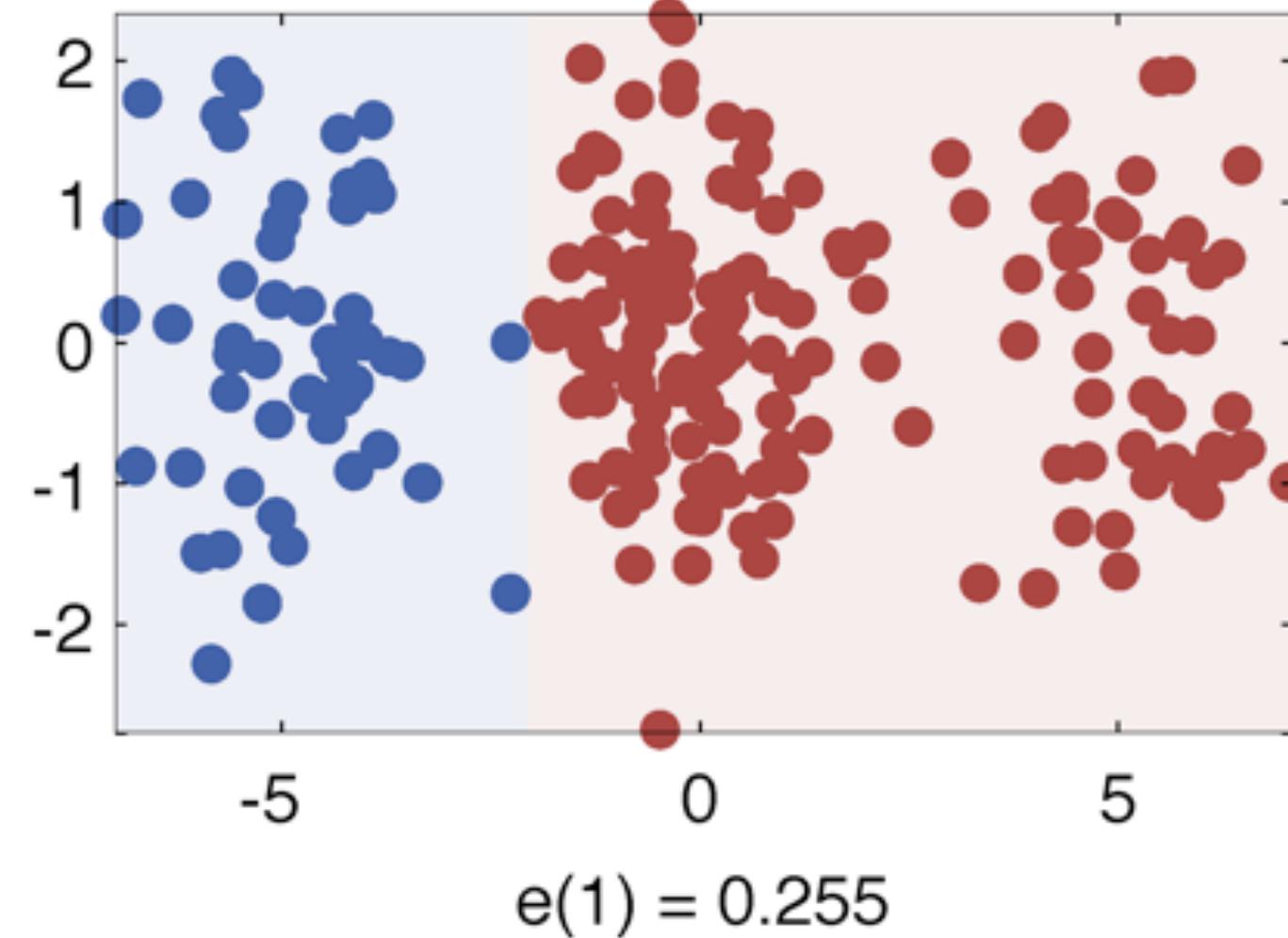
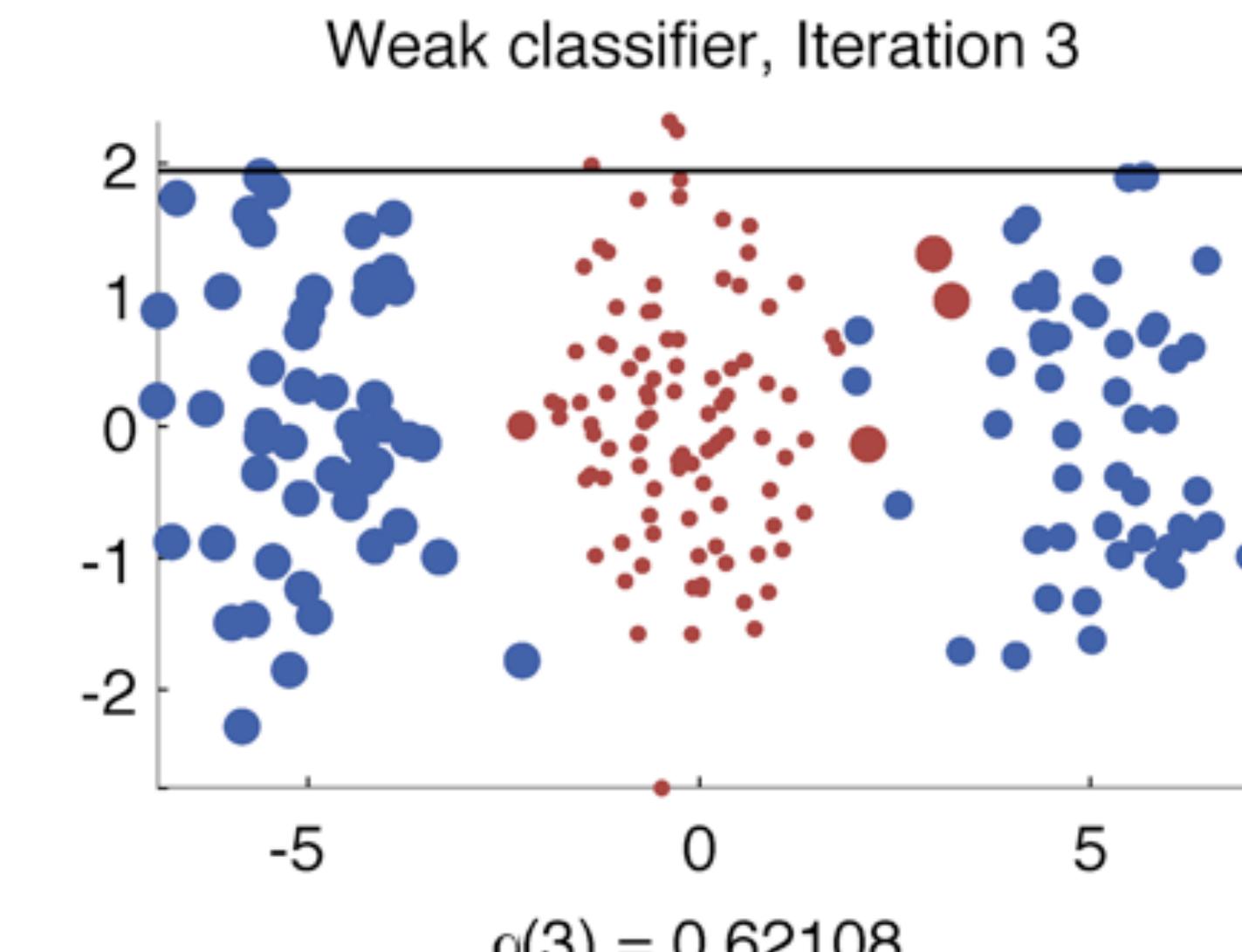
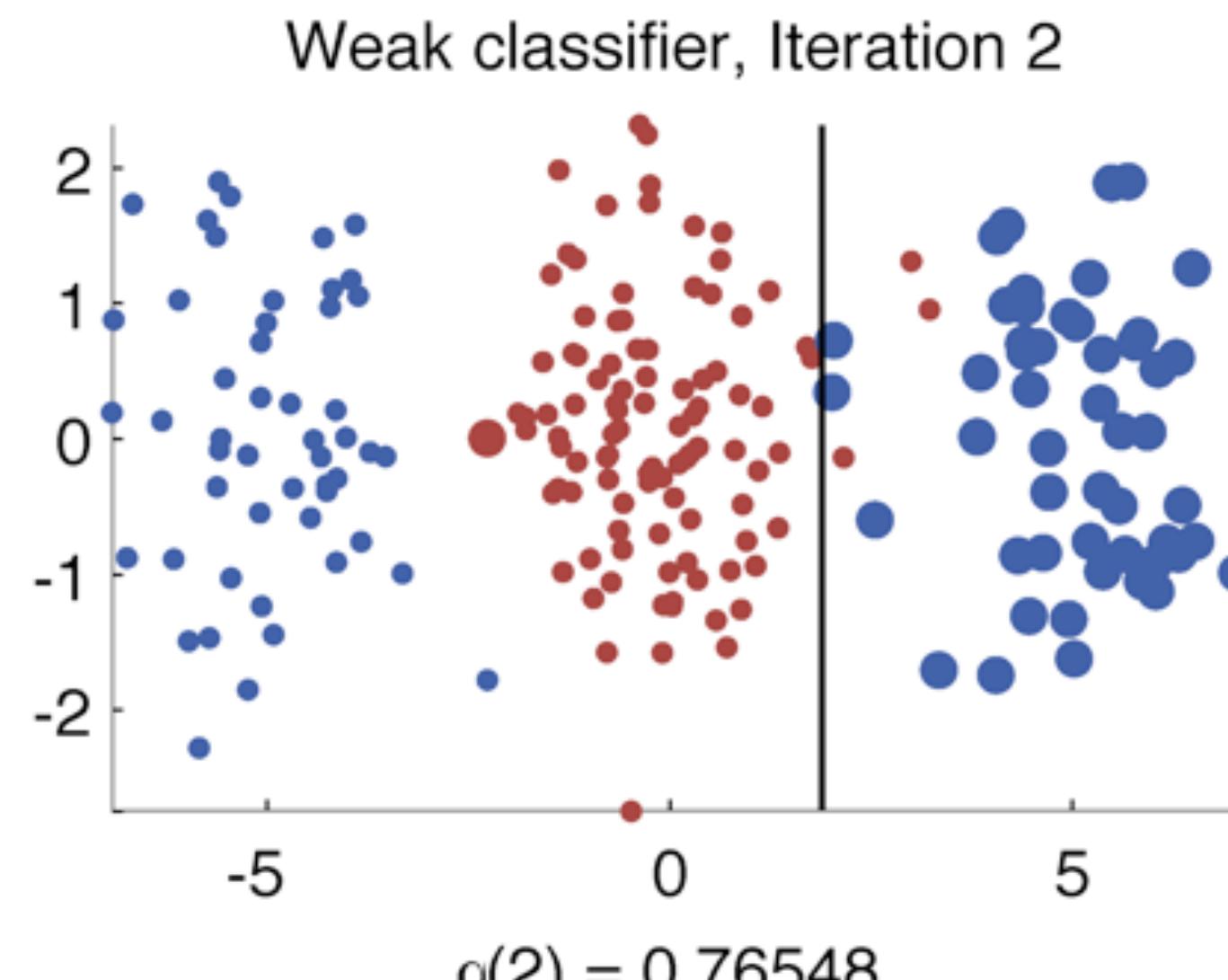
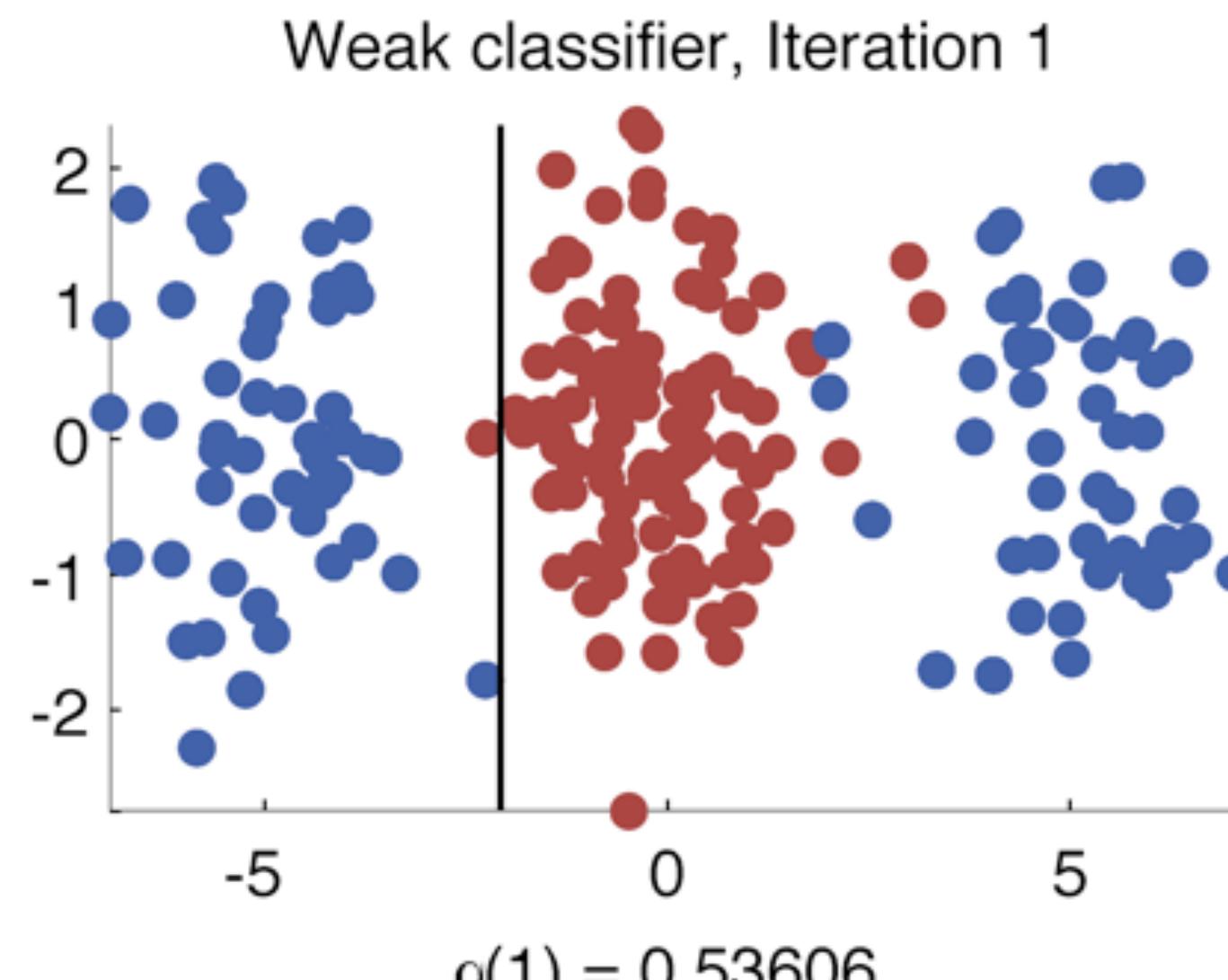
Simply stated

- Learn a new model 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

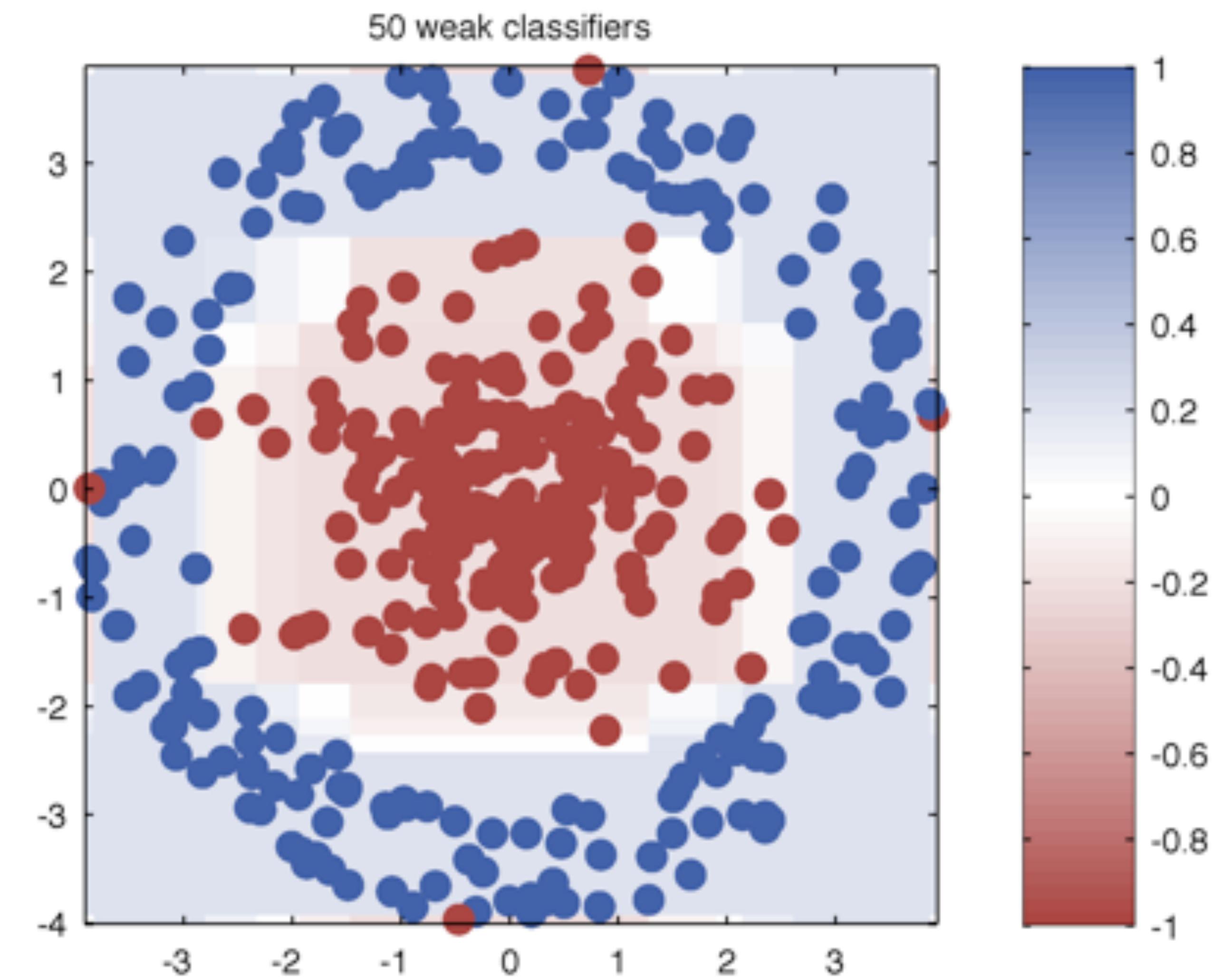
- Simple core models
 - Can be simple stump classifiers (threshold in one dimension)
 - No need for fancy learning and complex code
- Does not overfit
 - More models push the error down
- Works great for huge datasets

Example



Example

- Weak learners combine to form very complex decision boundaries
- Can be highly powerful, yet fairly simple to learn

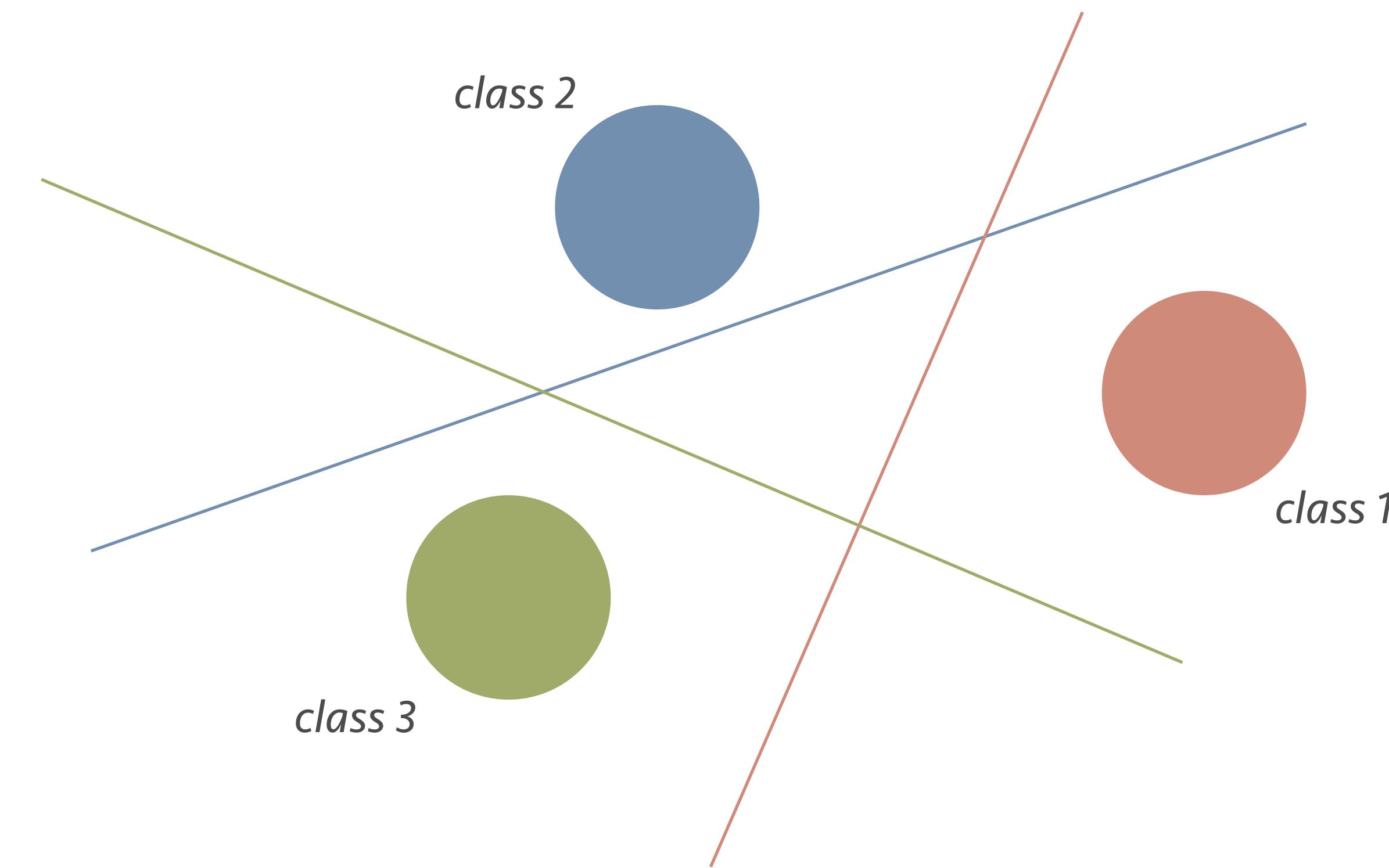


Multiple classes

- 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

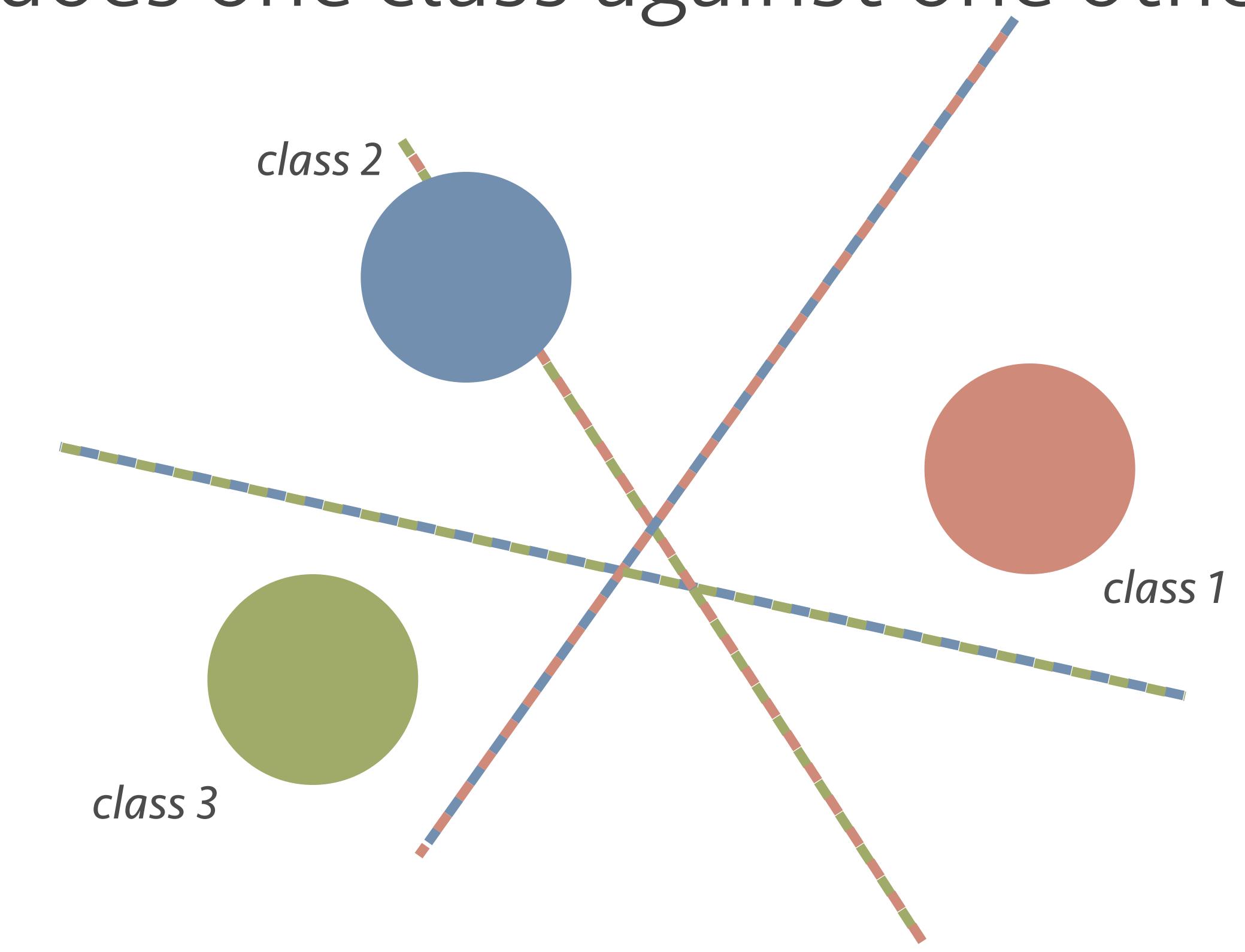
One against all

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



One against one

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



Error 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
- Can use robust codes to help performance

2 classifier bank		
	C ₁	C ₂
Class 1	-1	-1
Class 2	+1	+1
Class 3	-1	+1

Important concepts

- Occam's Razor
- Bias/Variance Tradeoff
- Curse of Dimensionality
- GIGO
- Cross-validation and overfitting

Occam's Razor

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

Dico ergo ad quoniam q[uod] pluralitas non est ponenda sine necessitate et non est necessitas quare debeat ponit p[ro]p[ter]us dicit sicutum mensuram motum angelii. nam



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 neural net boosted by an kernel SVM, after doing LLE
 - Maybe start with a linear classifier ...
 - Beware of overfitting



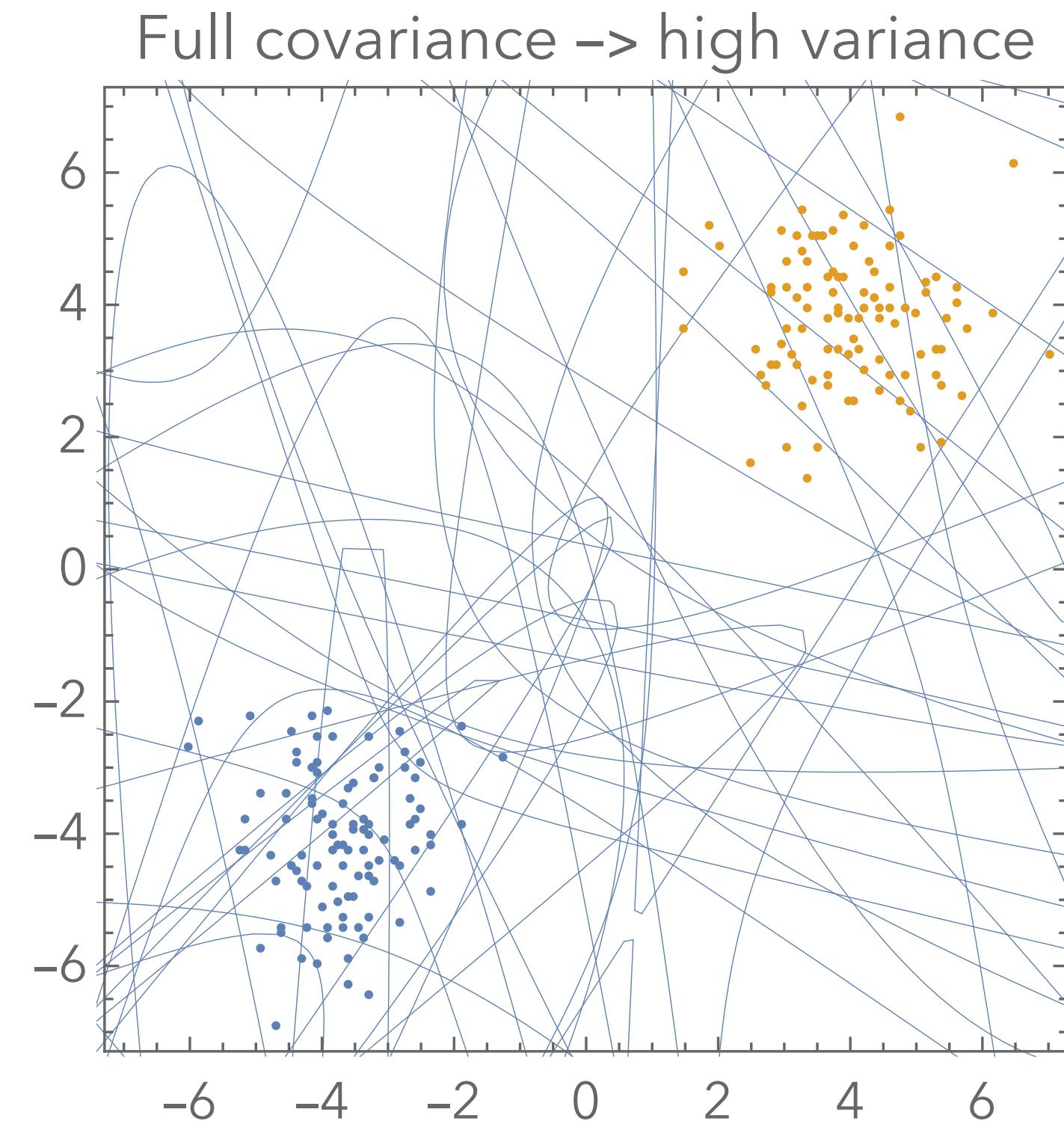
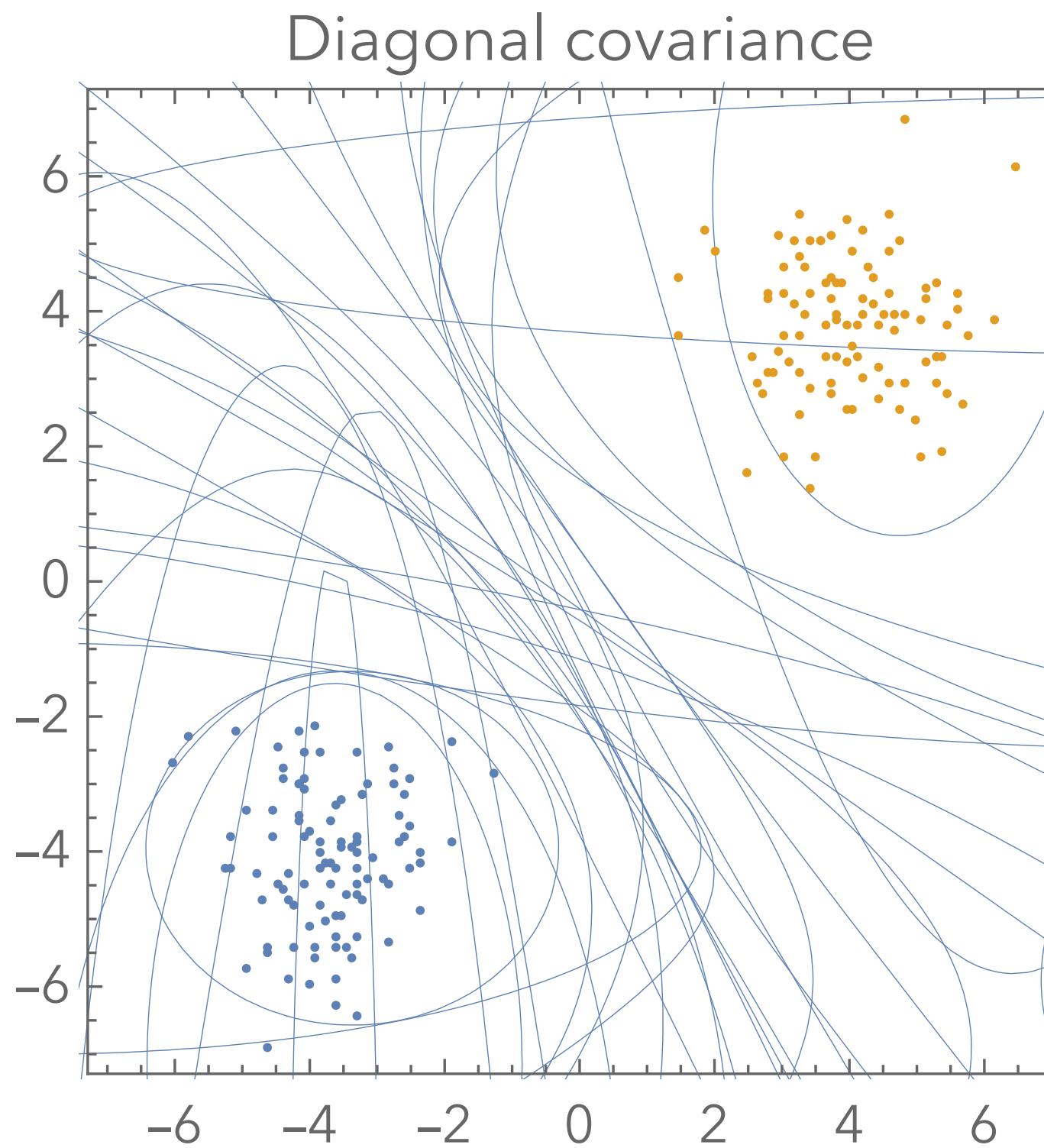
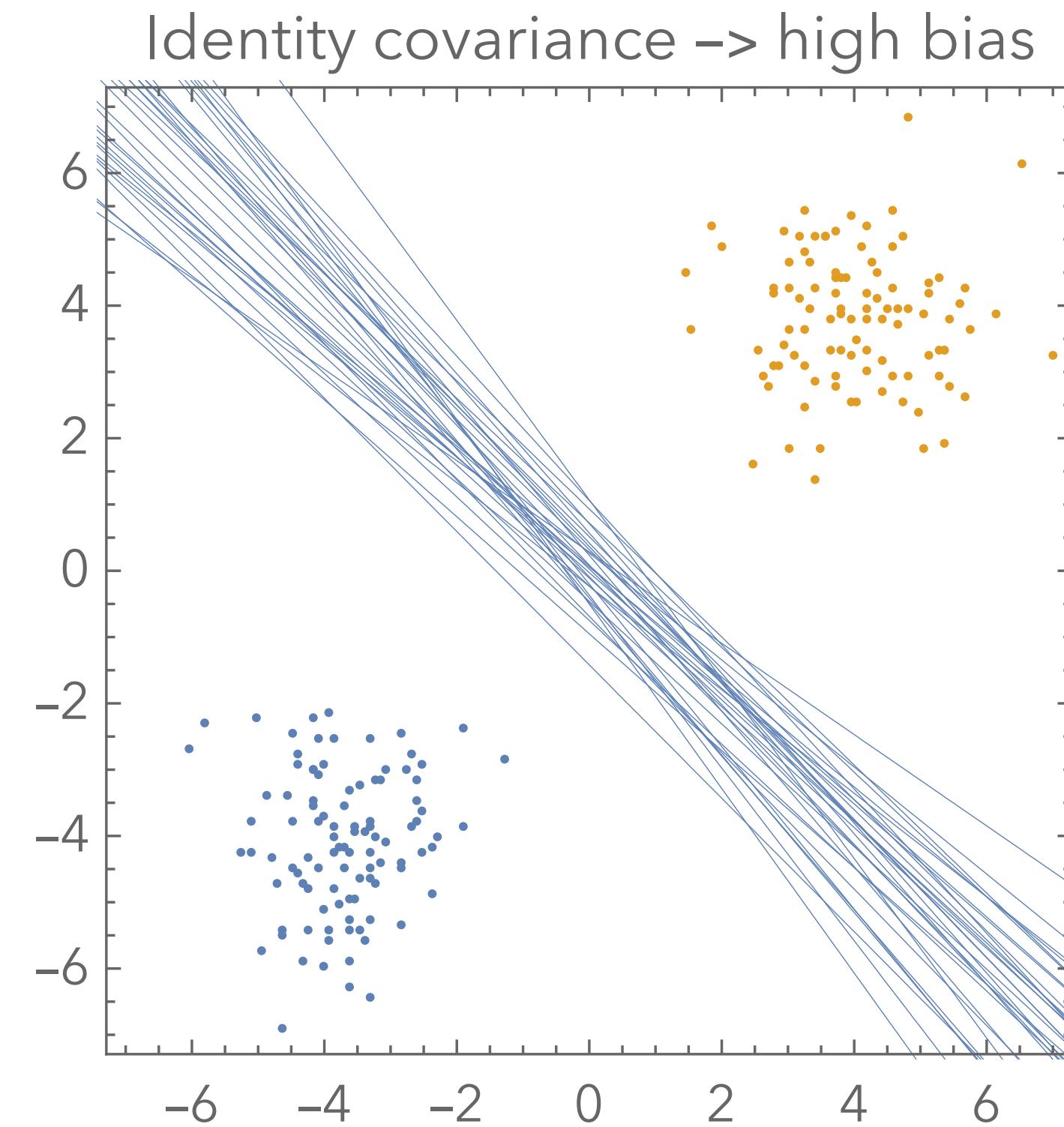
Ockham chooses a razor

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



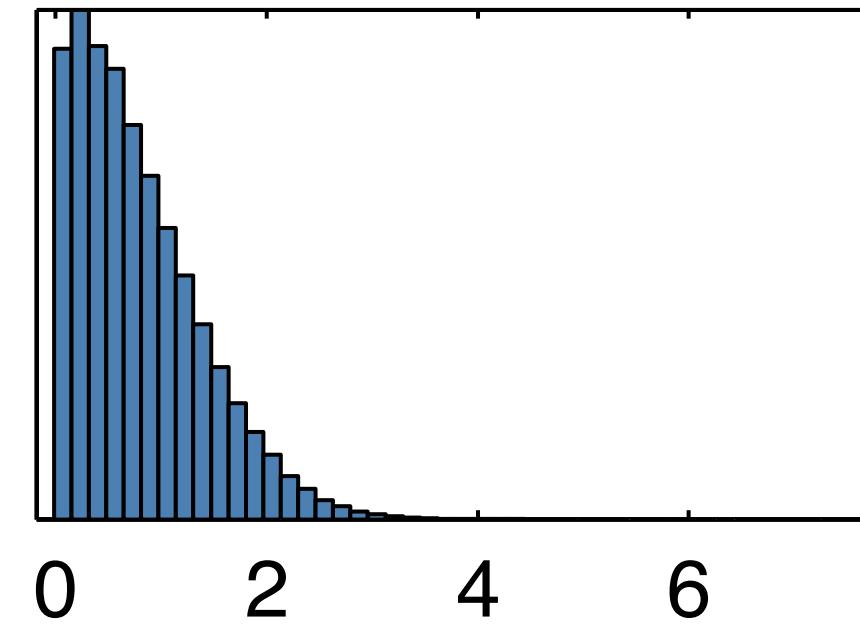
Curse of Dimensionality

- High dimensions are difficult
 - High-D estimators need lots of data!!
 - That's why we do dimensionality reduction
 - Geometry at high dimensions is very counterintuitive!
- Sometimes high-D's are a blessing
 - Surely more data is good
 - Remember the kernel trick ...

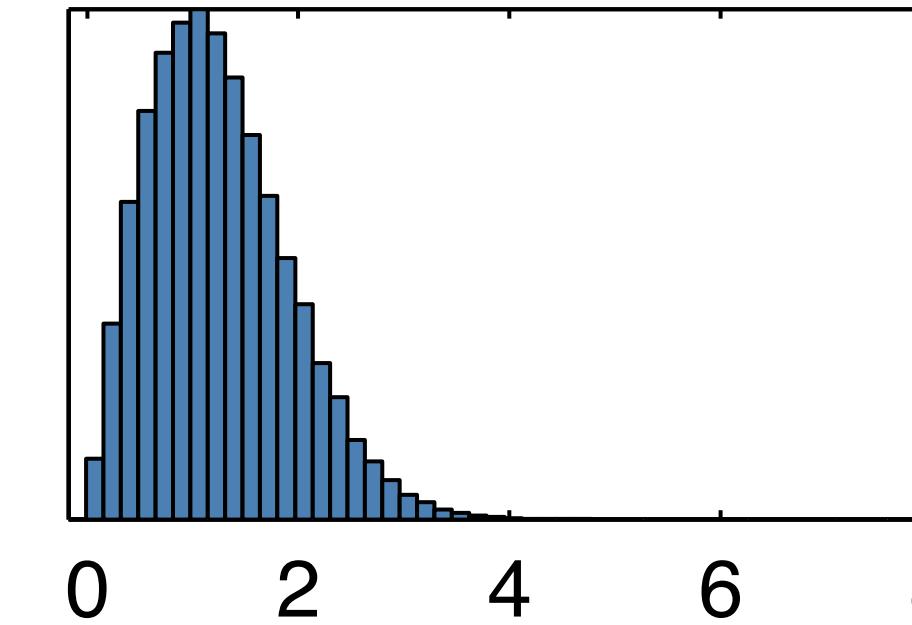
Don't trust your 2D intuition!

- How far from the center are Gaussian samples centered?

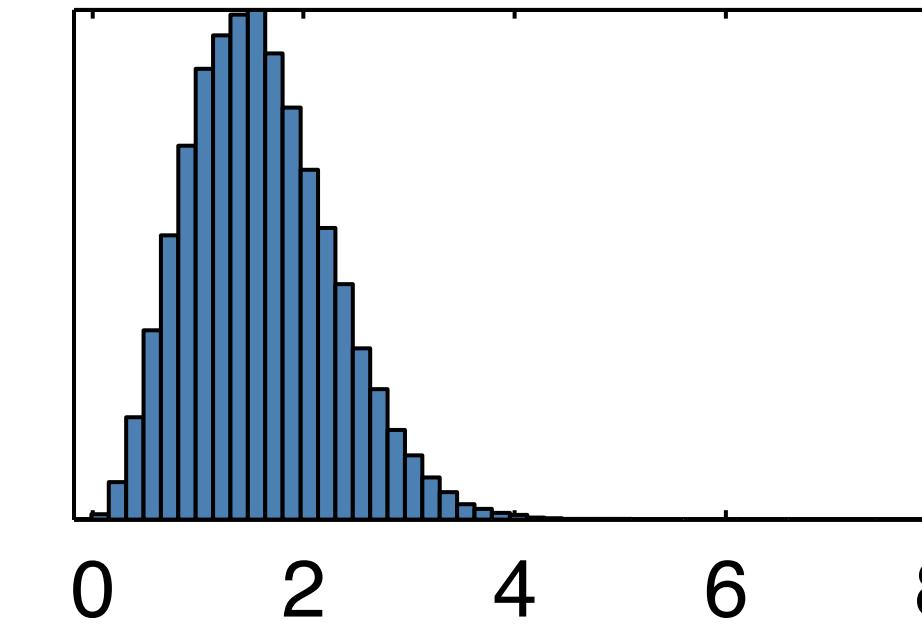
1 dimensions



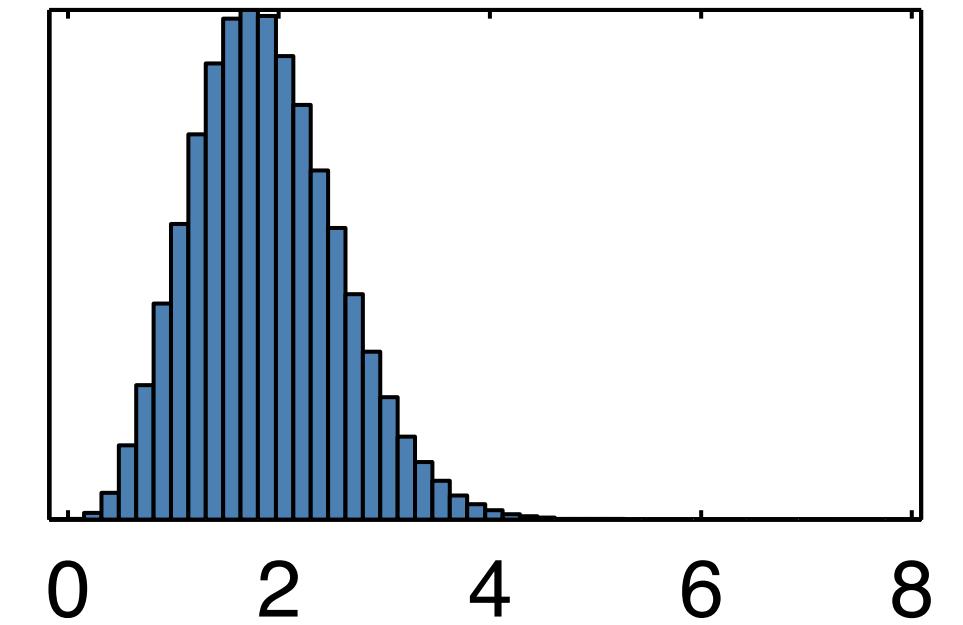
2 dimensions



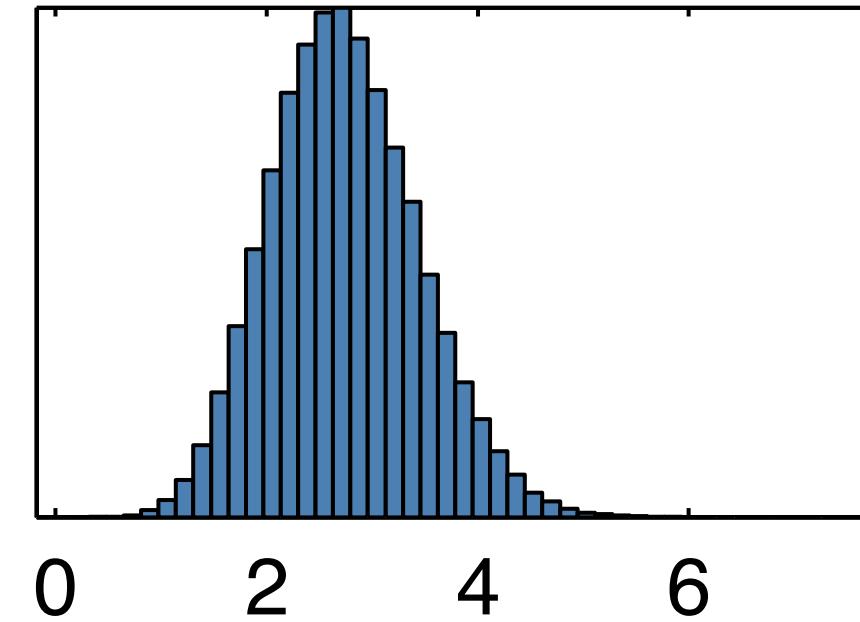
3 dimensions



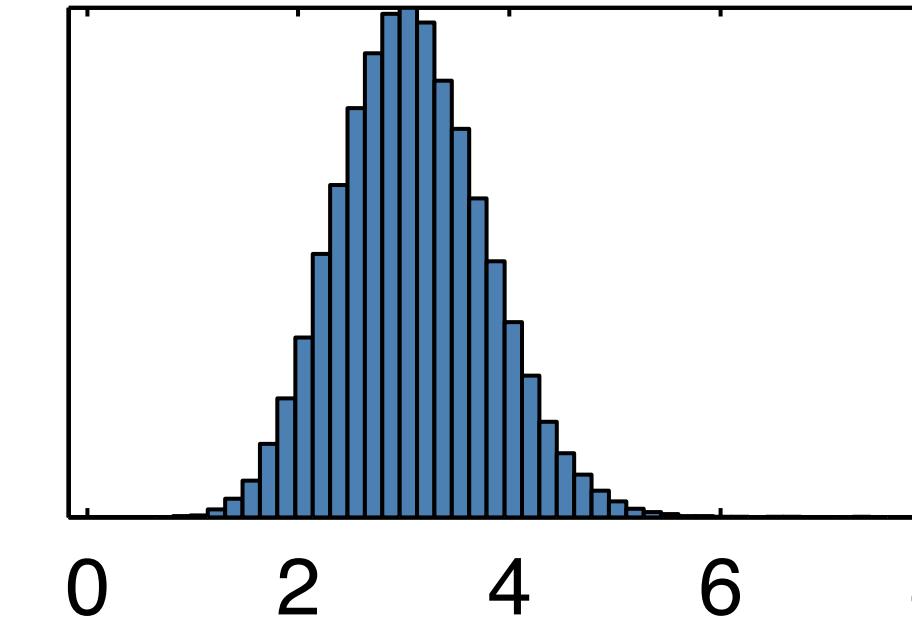
4 dimensions



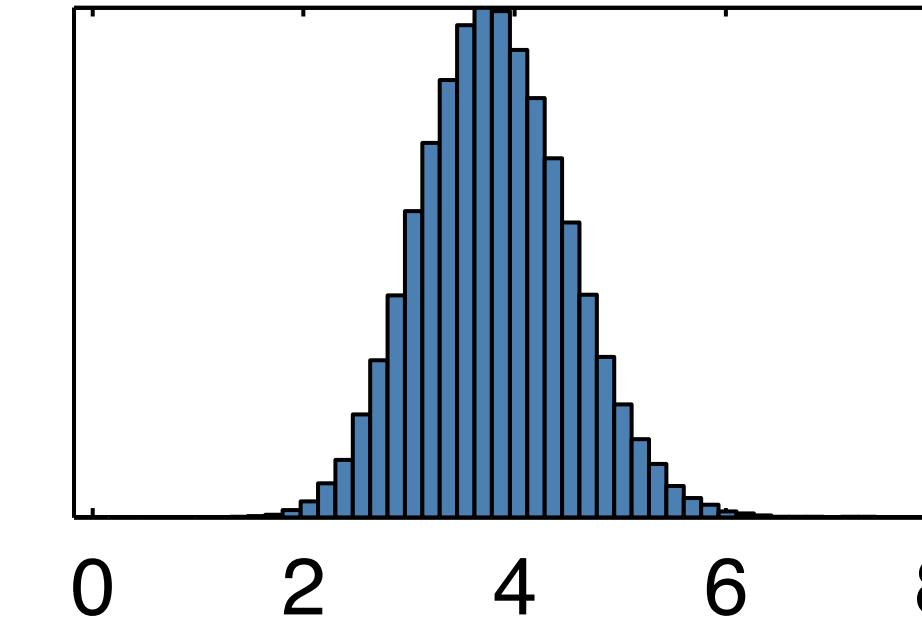
8 dimensions



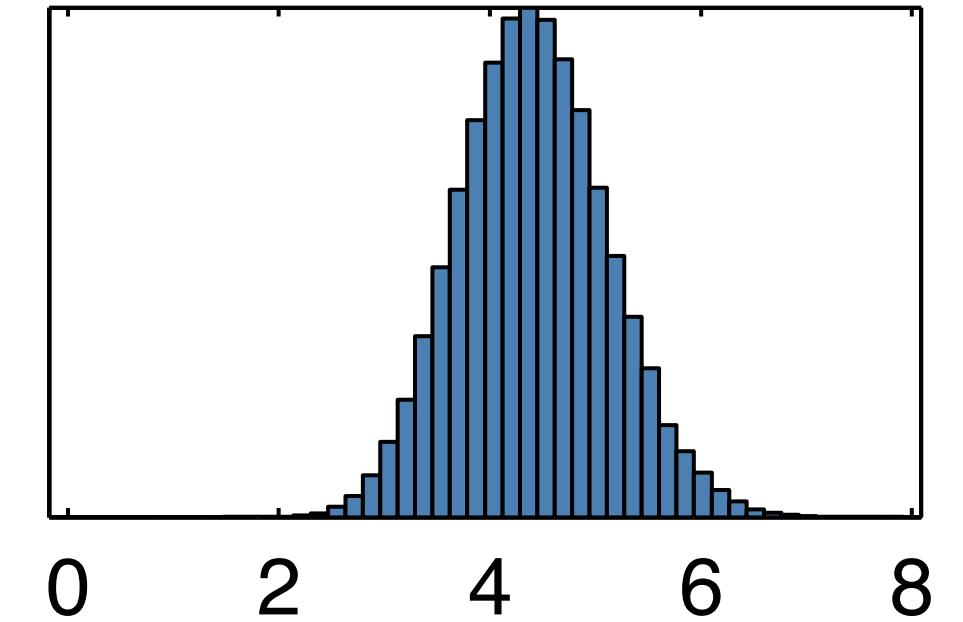
10 dimensions



15 dimensions

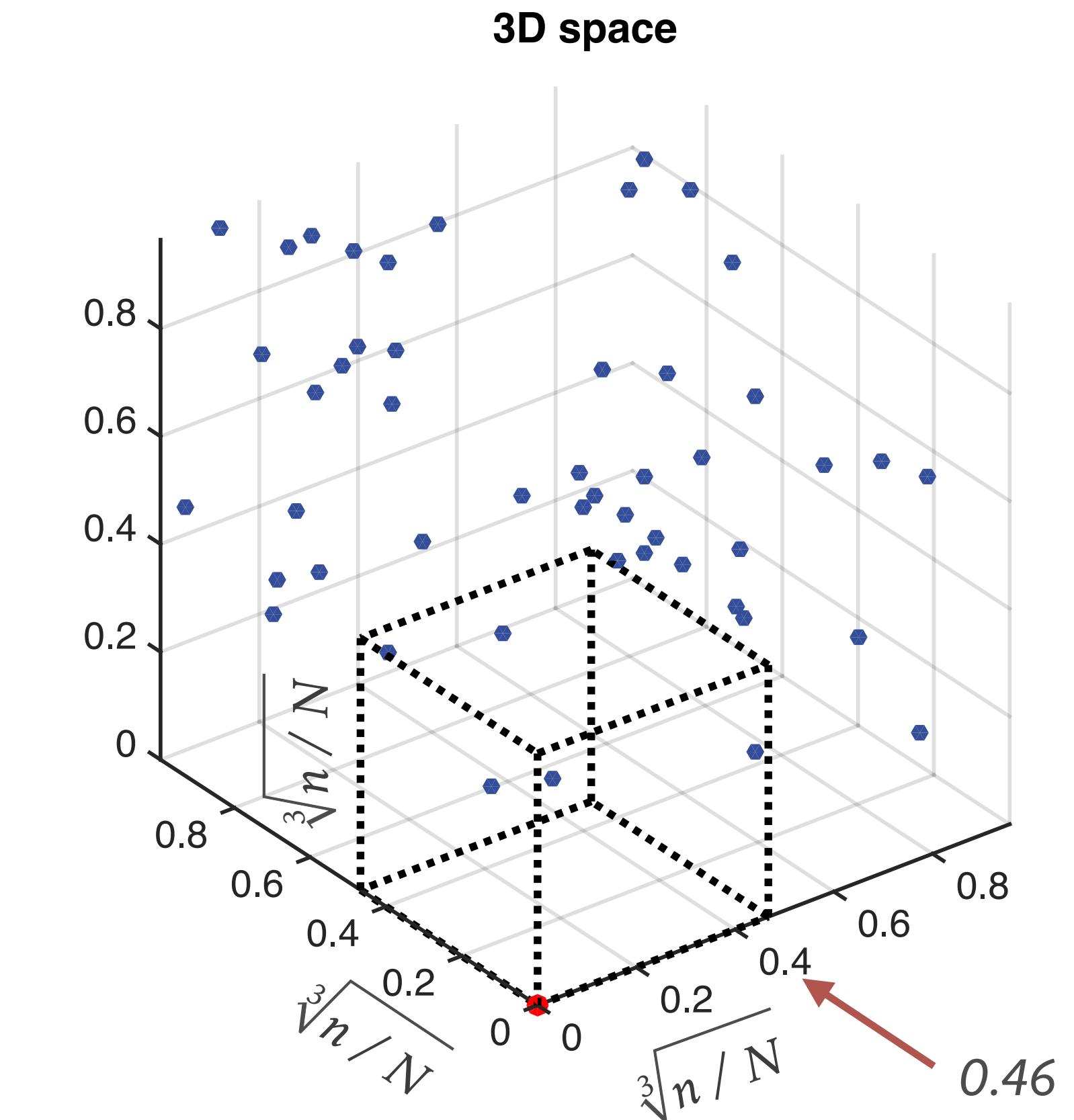
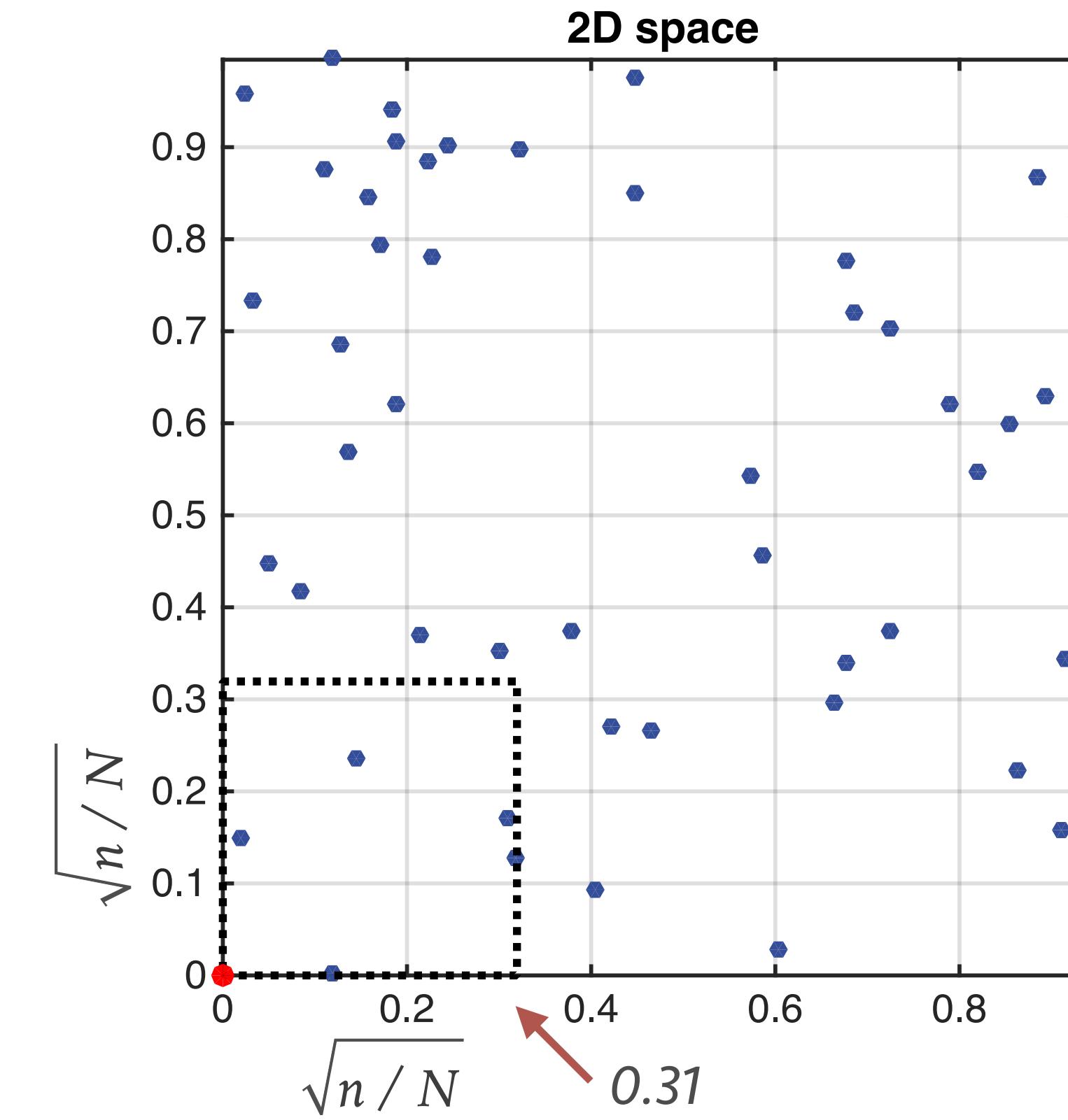
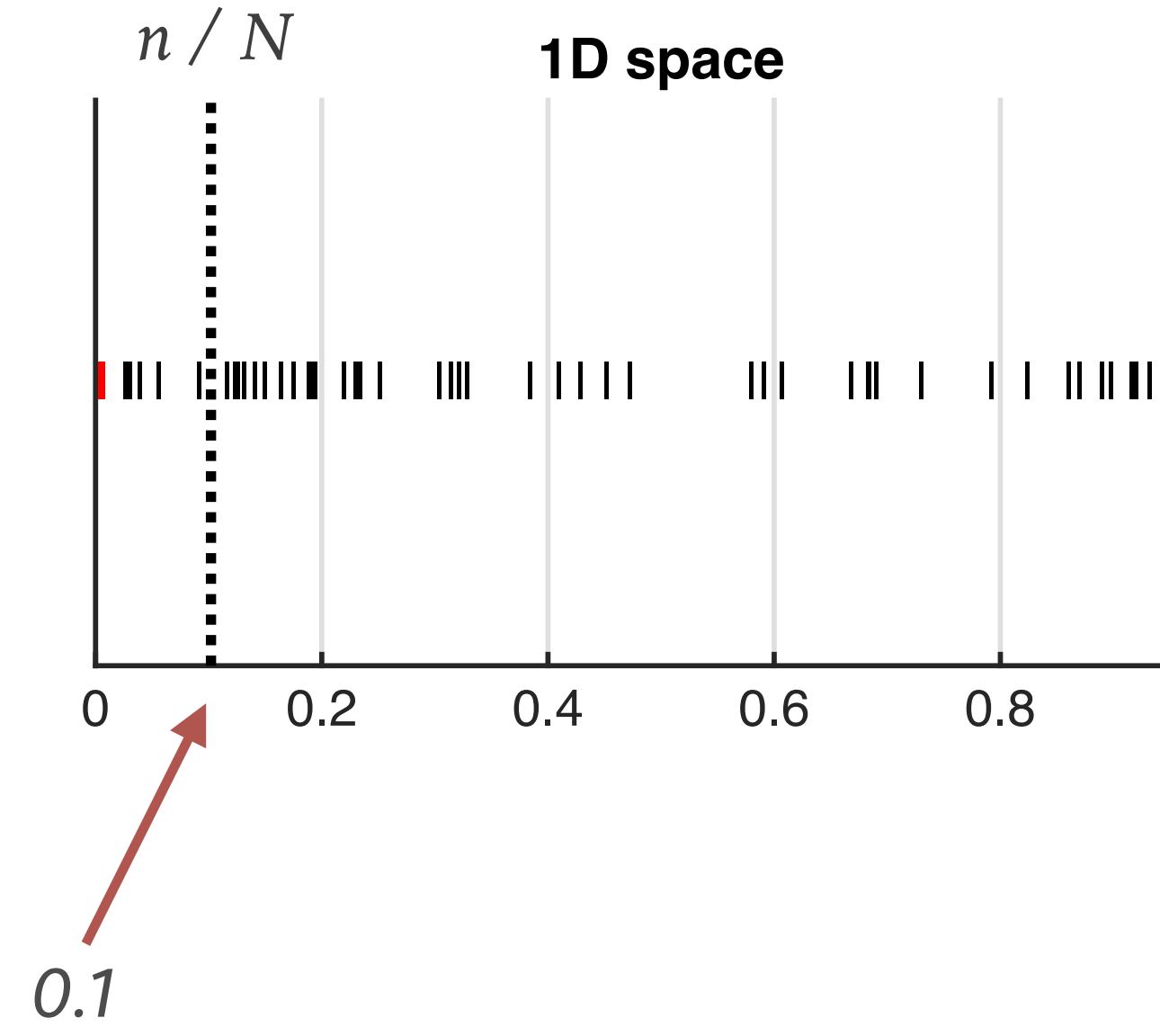


20 dimensions



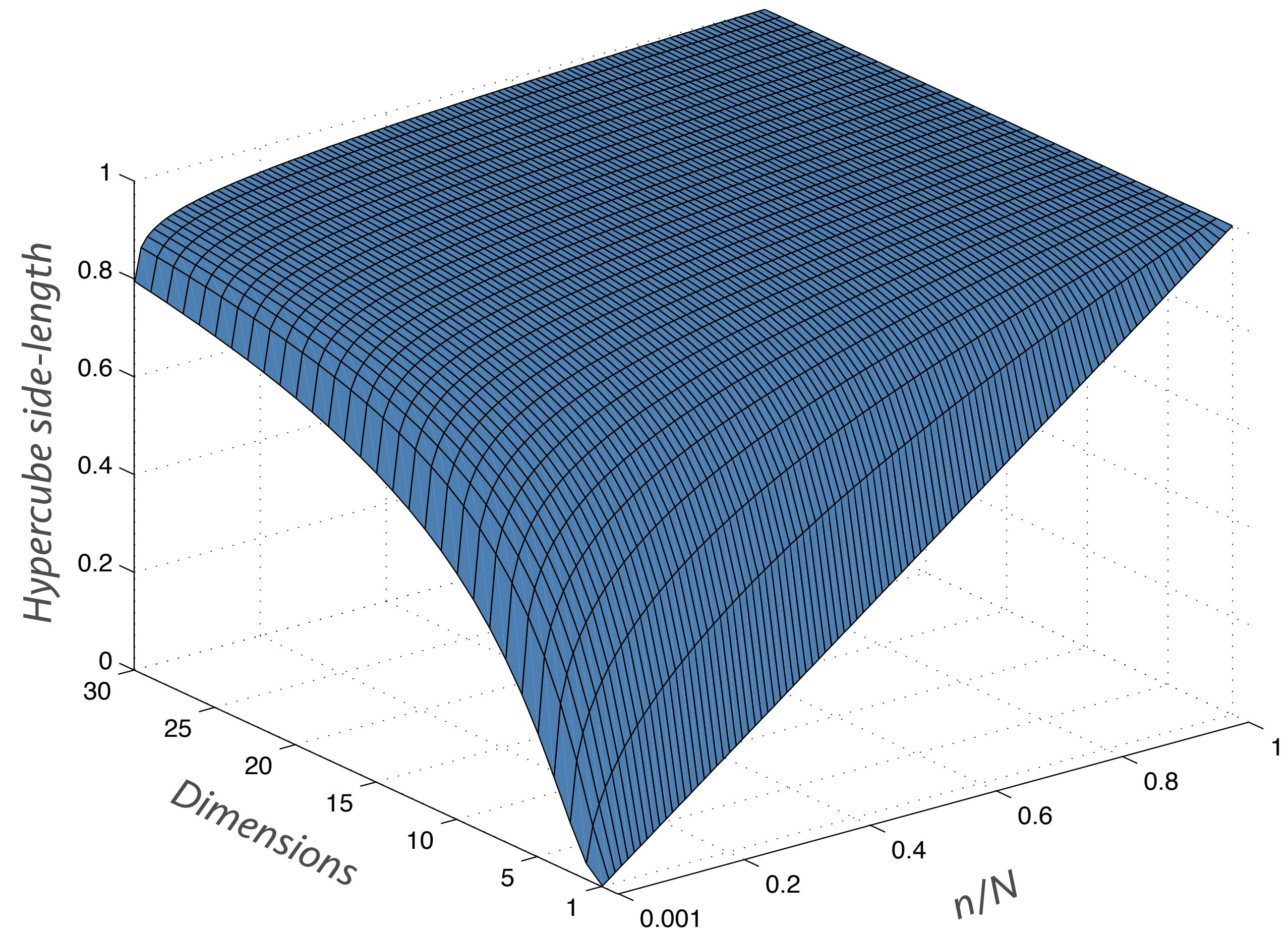
Nearest Neighbor example

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



The curse visualized

- Our search hypercube side-length grows rapidly!
 - We quickly need to cover most of each dimension to find neighbors
- High-dimensional neighbors are not as close as we think!
 - Also not representative of density

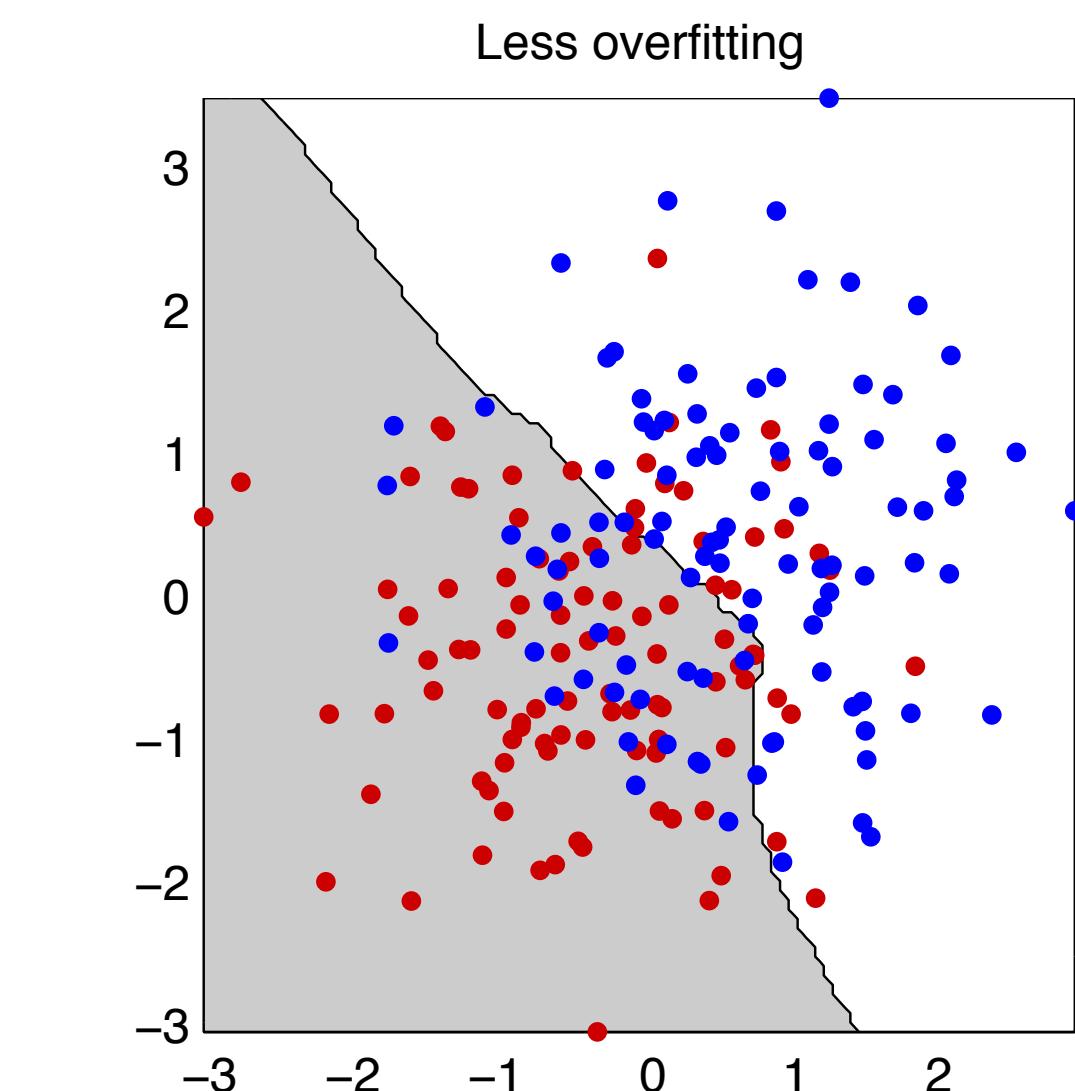
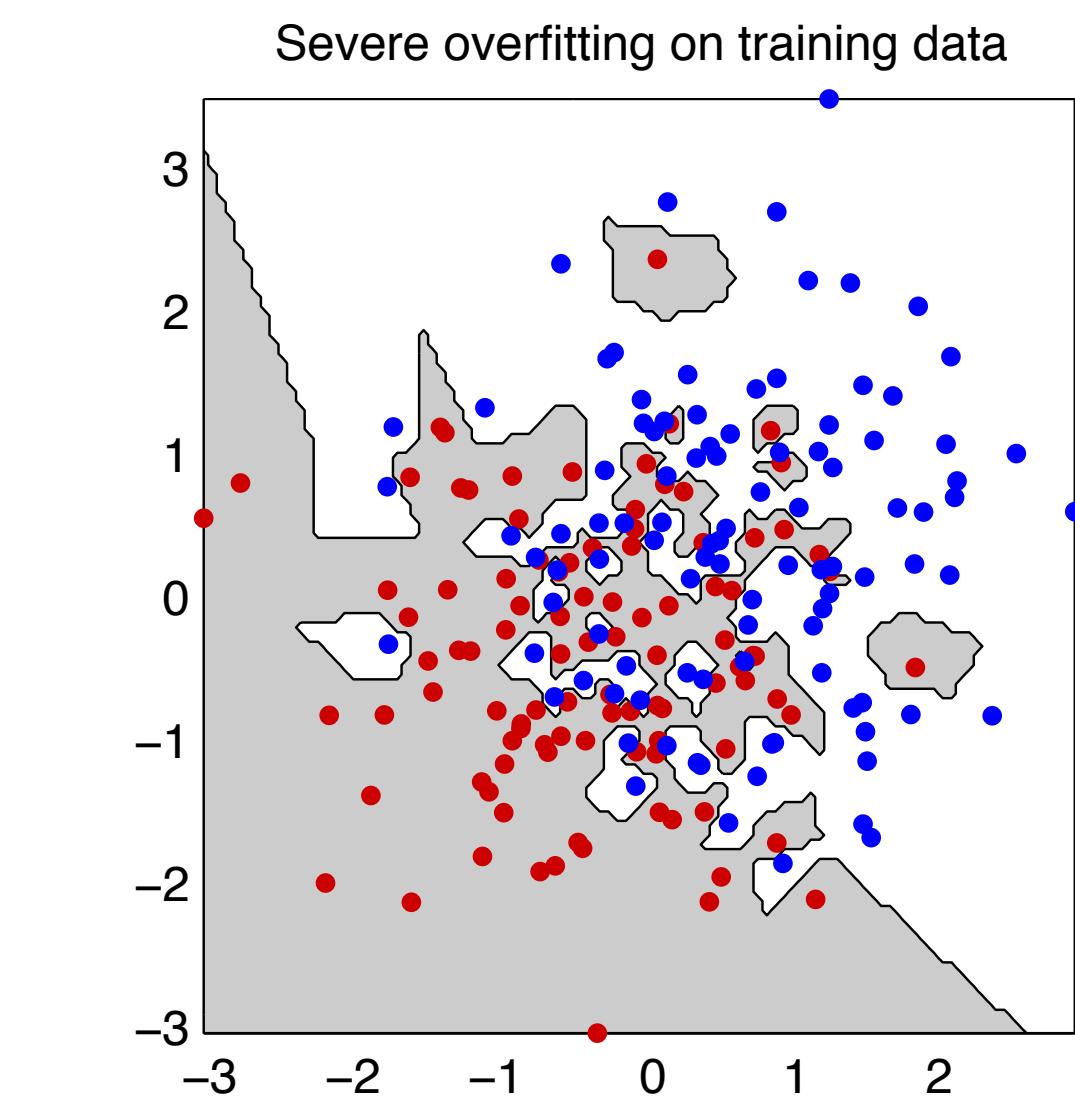


G.I.G.O.

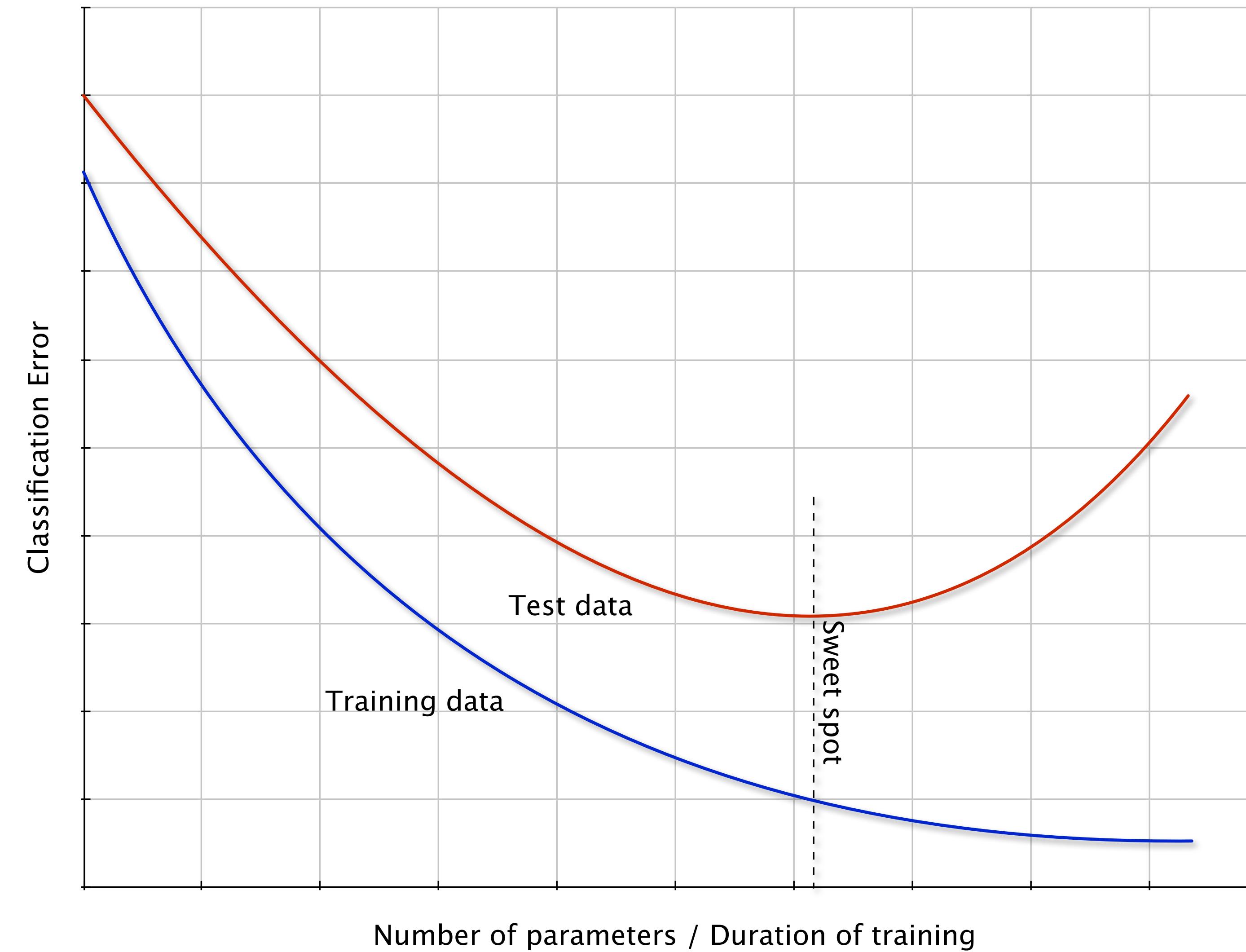
- Garbage in, Garbage out
 - Pick your data carefully!
 - Make sure they are plenty, representative, and relatively clean to facilitate training
- Garbage in, Gospel out
 - Beware of the bias!

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 overfitting looks like



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