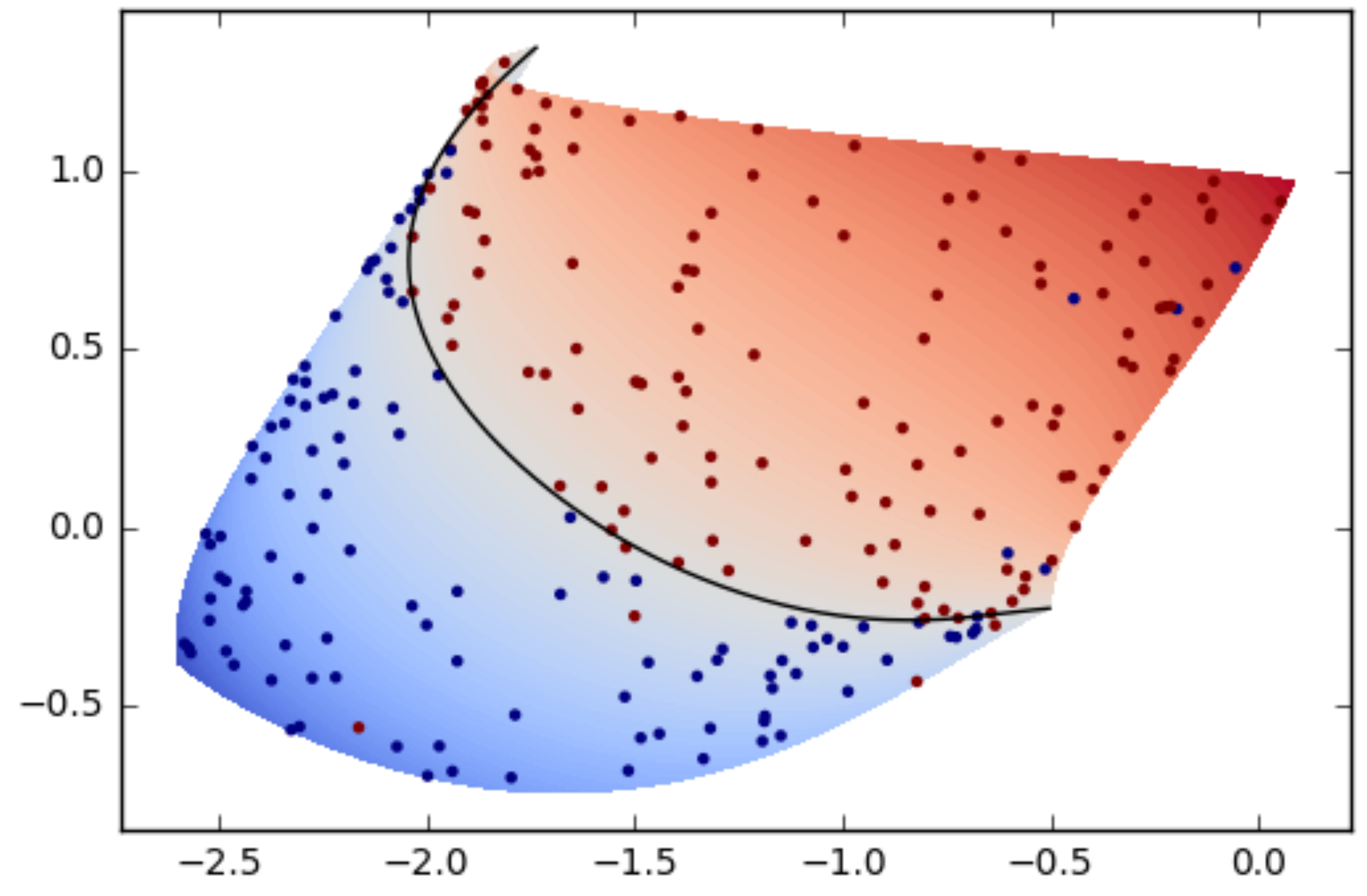
CS 273A: Machine Learning

Winter 2021

Lecture 13: Ensemble Methods

Roy Fox
Department of Computer Science
Bren School of Information and Computer Sciences
University of California, Irvine

All slides in this course adapted from Alex Ihler & Sameer Singh

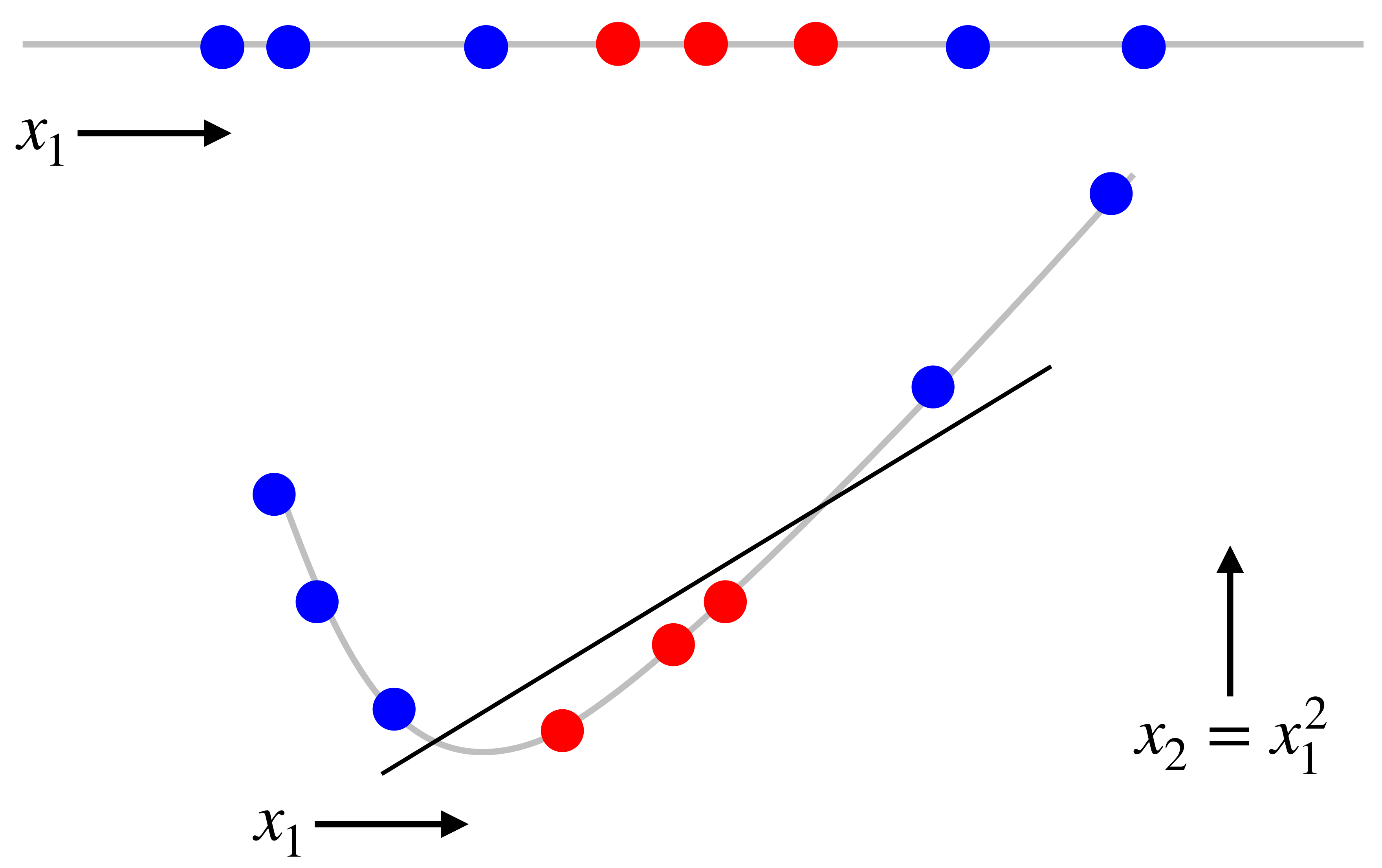


Today's lecture

- Kernel Machines
- Bagging
- Gradient boosting
- AdaBoost

Adding features

- So far: linear SVMs, not very expressive
  - $\implies$ add features $x \mapsto \Phi(x)$
- Linearly non-separable:
- Linearly separable in quadratic features:



Adding features

- Prediction: $\hat{y}(x) = \operatorname{sign}(w \cdot \Phi(x) + b)$
- Dual problem: $\max_{0 \le \lambda \le R} \sum_j \left(\lambda_j - \frac{1}{2} \sum_k \lambda_j \lambda_k y^{(j)} y^{(k)} \Phi(x^{(j)}) \cdot \Phi(x^{(k)}) \right)$ s.t. $\sum_j \lambda_j y^{(j)} = 0$
- Example: quadratic features $\Phi(x) = \begin{bmatrix} 1 & \sqrt{2} x_i & x_i^2 & \sqrt{2} x_i x_{i'} \end{bmatrix}$
  - n features $\mapsto O(n^2)$ features
  - Why $\sqrt{2}$? Next slide... But just scale corresponding weights

Implicit features

- For dual problem, we need $K_{jk} = \Phi(x^{(j)}) \cdot \Phi(x^{(k)})$

- Kernel trick: with $\Phi(x) = \begin{bmatrix} 1 & \sqrt{2}x_i & x_i^2 & \sqrt{2}x_i x_{i'} \end{bmatrix}$:

$$K_{jk} = 1 + \sum_i 2x_i^{(j)}x_i^{(k)} + \sum_i (x_i^{(j)}x_i^{(k)})^2 + \sum_{i<i'} 2(x_i^{(j)}x_i^{(k)})(x_{i'}^{(j)}x_{i'}^{(k)})$$

$$= \left(1 + \sum_i x_i^{(j)}x_i^{(k)}\right)^2$$

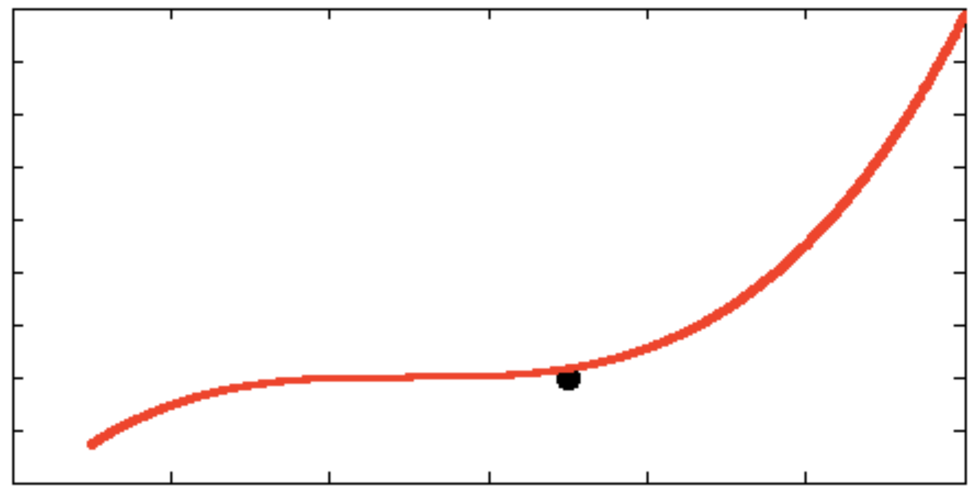
  - Each of m^2 elements computed in $O(n)$ time (instead of $O(n^2)$)

Mercer's Theorem

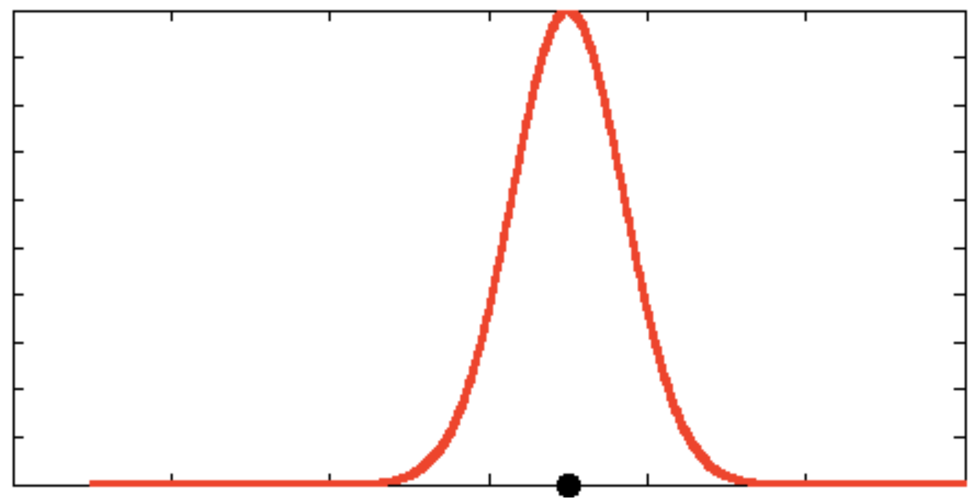
- Reminder: positive semidefinite matrix $A \succeq 0$: $v^\intercal A v \geq 0$ for all vectors v
- Positive semidefinite kernel $K \succeq 0$: matrix $K(x^{(j)}, x^{(k)}) \succeq 0$ for all datasets
- Mercer's Theorem: if $K \succeq 0 \implies K(x, x') = \Phi(x) \cdot \Phi(x')$ for some $\Phi(x)$
- Φ may be hard to calculate
  - May even be infinite dimensional (Hilbert space)
  - Not an issue, only the kernel $K(x, x')$ should be easy to compute ($O(m^2)$ times)

Common kernel functions

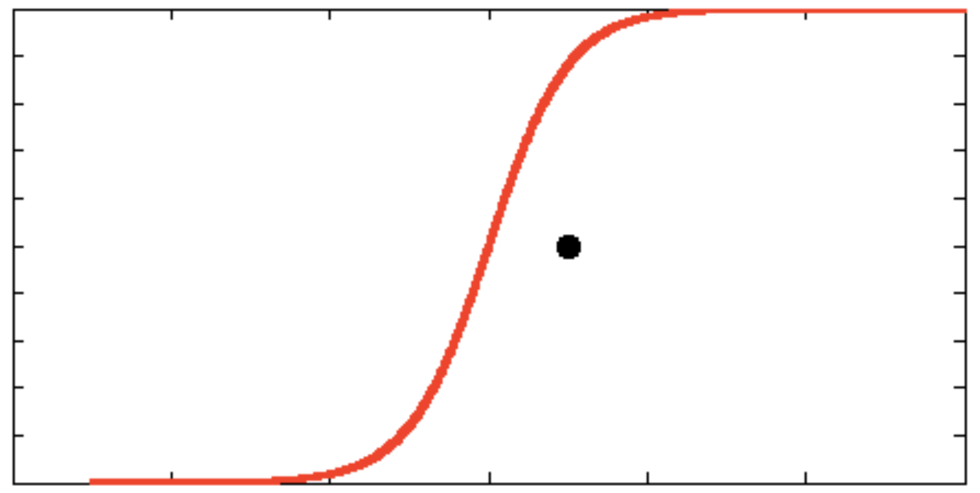
- Polynomial: $K(x, x') = (1 + x \cdot x')^d$



- Radial Basis Functions (RBF): $K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$



- Saturating: $K(x, x') = \tanh(ax \cdot x' + c)$



- Domain-specific: textual similarity, genetic code similarity, ...
  - May not be positive semidefinite, and still work well in practice

Kernel SVMs

- Define kernel $K : (x, x') \mapsto \mathbb{R}$
- Solve dual QP: $\max_{0 \le \lambda \le R} \sum_j \left(\lambda_j - \frac{1}{2} \sum_k \lambda_j \lambda_k y^{(j)} y^{(k)} K(x^{(j)}, x^{(k)}) \right)$ s.t. $\sum_j \lambda_j y^{(j)} = 0$
- Learned parameters = λ (m parameters)
  - But also need to store all support vectors (having $\lambda_j > 0$)
- Prediction: $\hat{y}(x) = \operatorname{sign}(w \cdot \Phi(x))$

$$= \operatorname{sign}\left(\sum_j \lambda_j y^{(j)} \Phi(x^{(j)}) \cdot \Phi(x) \right) = \operatorname{sign}\left(\sum_j \lambda_j y^{(j)} K(x^{(j)}, x) \right)$$

Demo

- https://cs.stanford.edu/people/karpathy/svmjs/demo/

Linear vs. kernel SVMs

- Linear SVMs
  - $\hat{y} = \text{sign}(w \cdot x + b) \implies n + 1$ parameters
  - Alternatively: represent by indexes of SVs; usually, #SVs = #parameters
- Kernel SVMs
  - $K(x, x')$ may correspond to high- (possibly infinite-) dimensional $\Phi(x)$
  - Typically more efficient to store the SVs $x^{(j)}$ (not $\Phi(x^{(j)})$)
    - And their corresponding λ_j

Recap

- Maximize margin for separable data
  - Primal QP: minimize $||w||^2$ subject to linear constraints
  - Dual QP: m variables, m^2 dot products
- Soft margin for non-separable data
  - Primal problem: regularized hinge loss
  - Dual problem: m-dimensional QP
- Kernel Machines
  - Dual form involves only pairwise similarity
  - Mercer kernels: equivalent to dot products in implicit high-dimensional space

Today's lecture

Kernel Machines

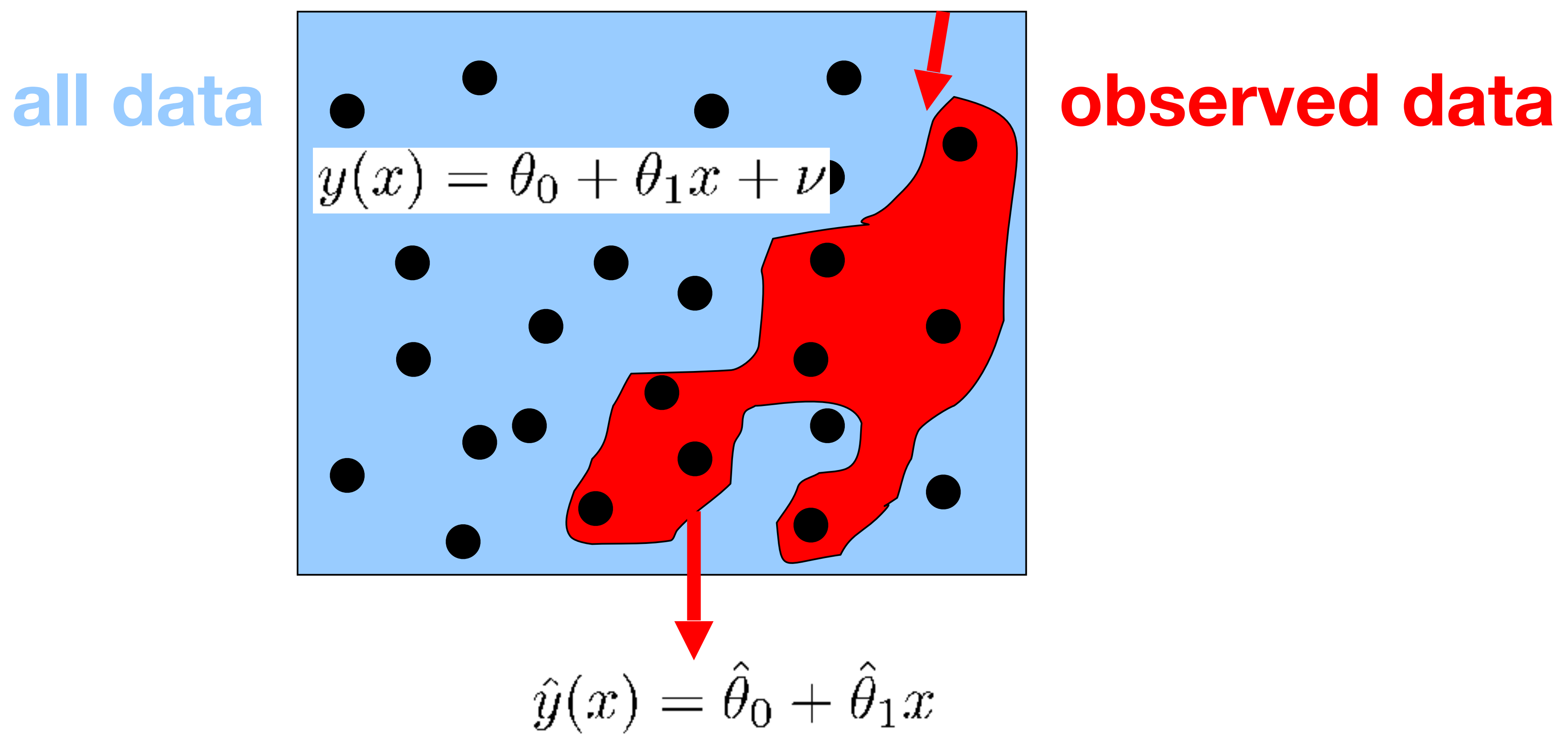
Bagging

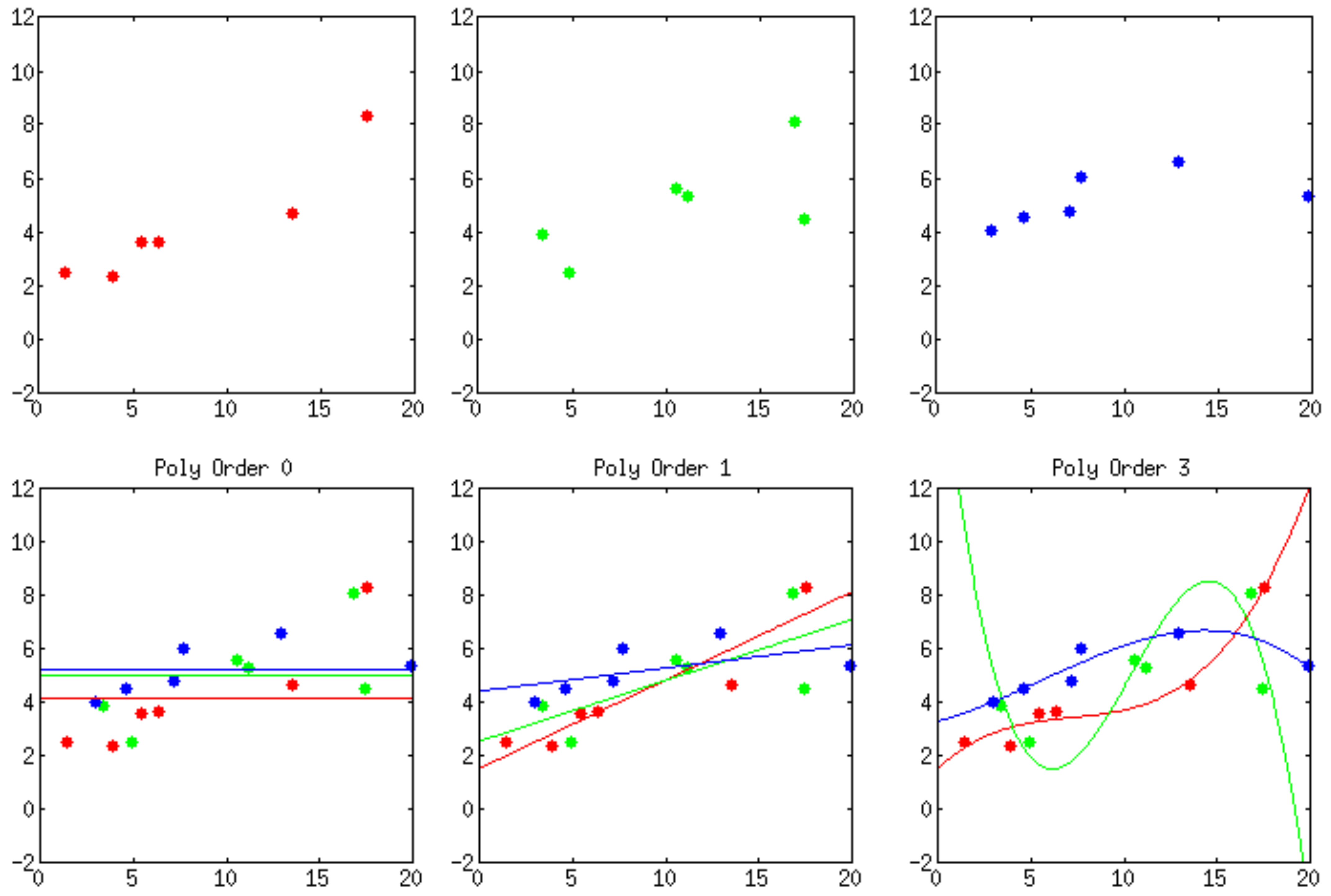
Gradient boosting

AdaBoost

Bias vs. variance

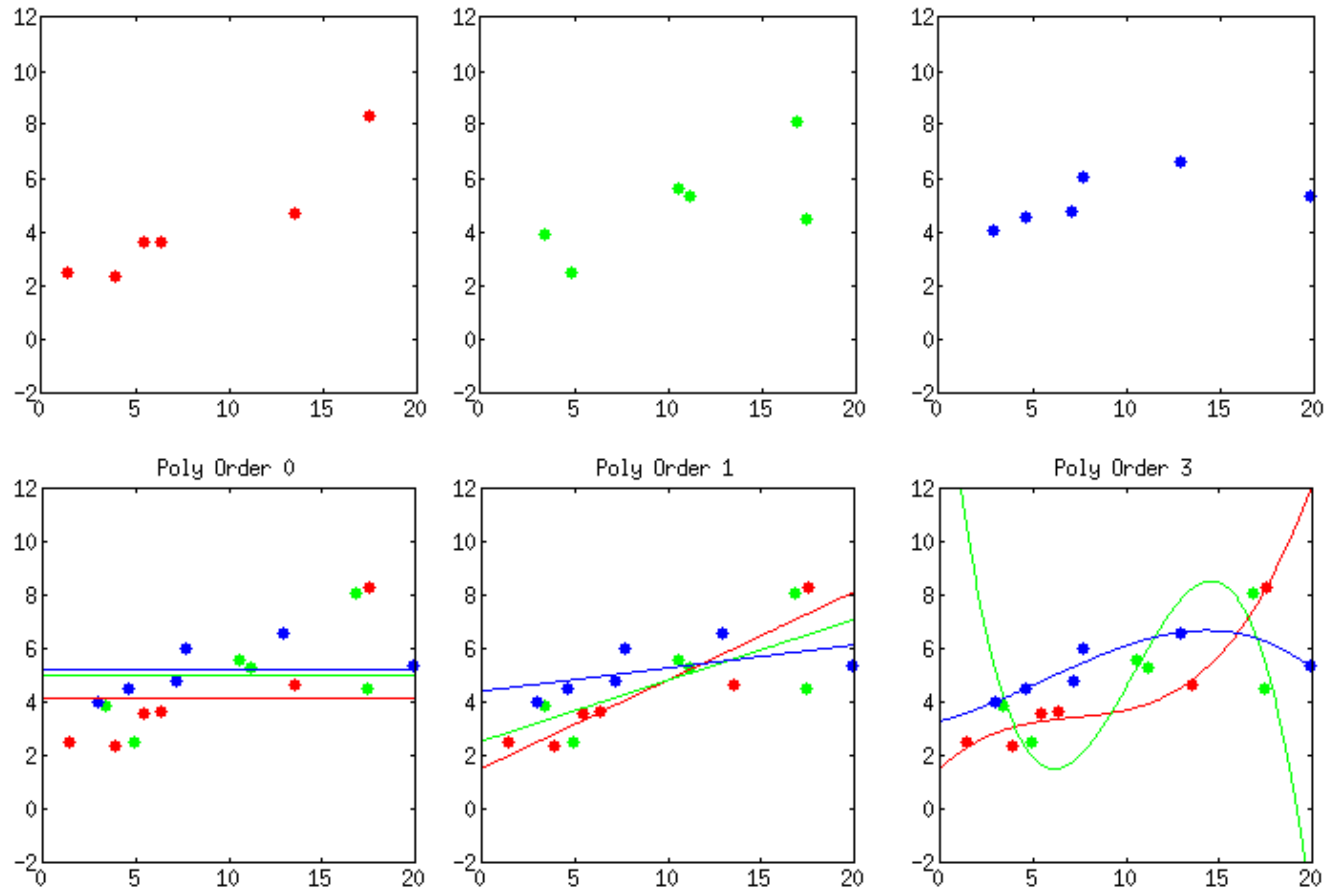
- Imagine 3 universes → 3 datasets
- A simple model:
  - Poor prediction (on average across universes)
    - High bias
  - Doesn't vary much between universes
    - Low variance
- A complex model:
  - Low bias
  - High variance


all data
observed data
$y(x) = \theta_0 + \theta_1 x + \nu$
$\hat{y}(x) = \hat{\theta}_0 + \hat{\theta}_1 x$



Averaging across datasets

- What if we could reach out across universes
  - Average models for different datasets
  - For classification: majority vote of different models
- Same bias
- Lower variance
- But we only have our training set $\mathscr{D}$
  - Idea: resample $\mathscr{D}_1, \ldots, \mathscr{D}_K$ from $\mathscr{D}$
    - Average models trained for each $\mathscr{D}_k$



Bootstrap

- Resampling = any method that samples a new dataset from the training set

$$\tilde{\mathscr{D}} = \{(x^{(j_1)}, y^{(j_1)}), \ldots, (x^{(j_b)}, y^{(j_b)})\} \quad j_1, \ldots, j_b \sim \mathrm{U}(1, \ldots, m)$$

  - Subsampling = resampling without replacement (choose a subset)
  - Bootstrap = resampling with replacement (may repeat same datapoint)
    - Preferred for theory that is less sensitive to good choice of b
    - But has higher variance

Bagging

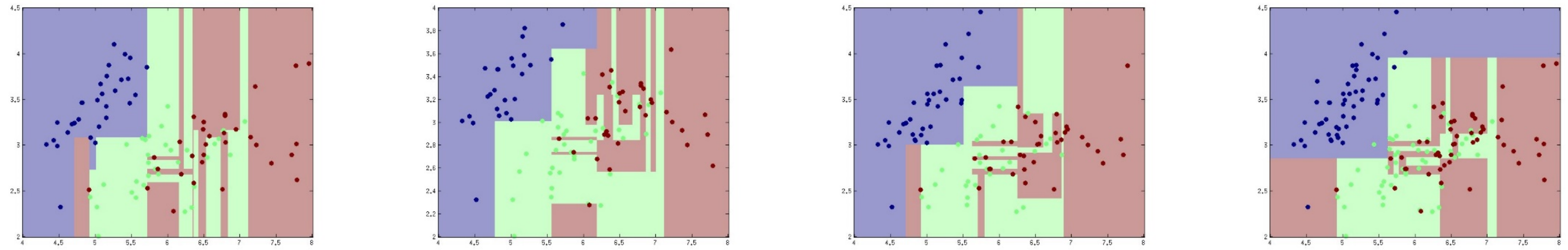
- Bagging = bootstrap aggregating:
  - Resample K datasets $\mathscr{D}_1, \dots, \mathscr{D}_K$ of size b
  - Train K models $\theta_1, \dots, \theta_K$ on each dataset
  - Regression: output $f_\theta : x \mapsto \frac{1}{K} \sum_k f_{\theta_k}(x)$
  - Classification: output $f_\theta : x \mapsto \text{majority}\{f_{\theta_k}(x)\}$
- Similar to cross-validation (for different purpose), but outputs average model
  - Also, datasets are resampled (with replacement), not a partition

Bagging: properties

- Each model is trained from less data
  - More bias
  - More variance
  - Replacement also adds variance (repetitions throw off the data distribution)
- Models are averaged
  - Doesn't affect bias (defined as average over models)
  - Variance reduced a lot (roughly as $\frac{1}{K}$, under some conditions)
- More bias, less variance $\Longrightarrow$ less overfitting = simpler model, in a sense

Bagged decision trees

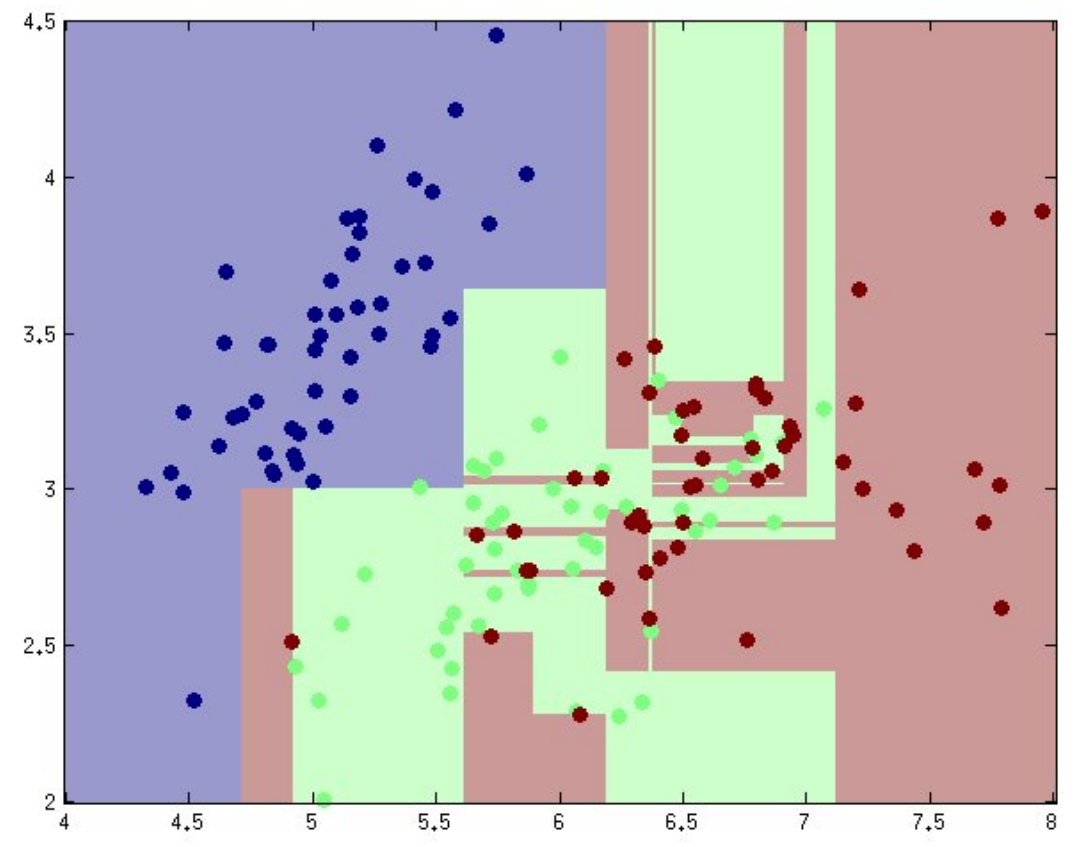
- A model badly in need for complexity reduction: decision trees
  - Very low bias, very high variance
- Randomly resample data
- Train decision tree for each sample; no max depth
  - Still low bias, high variance
- Average / majority decision over models



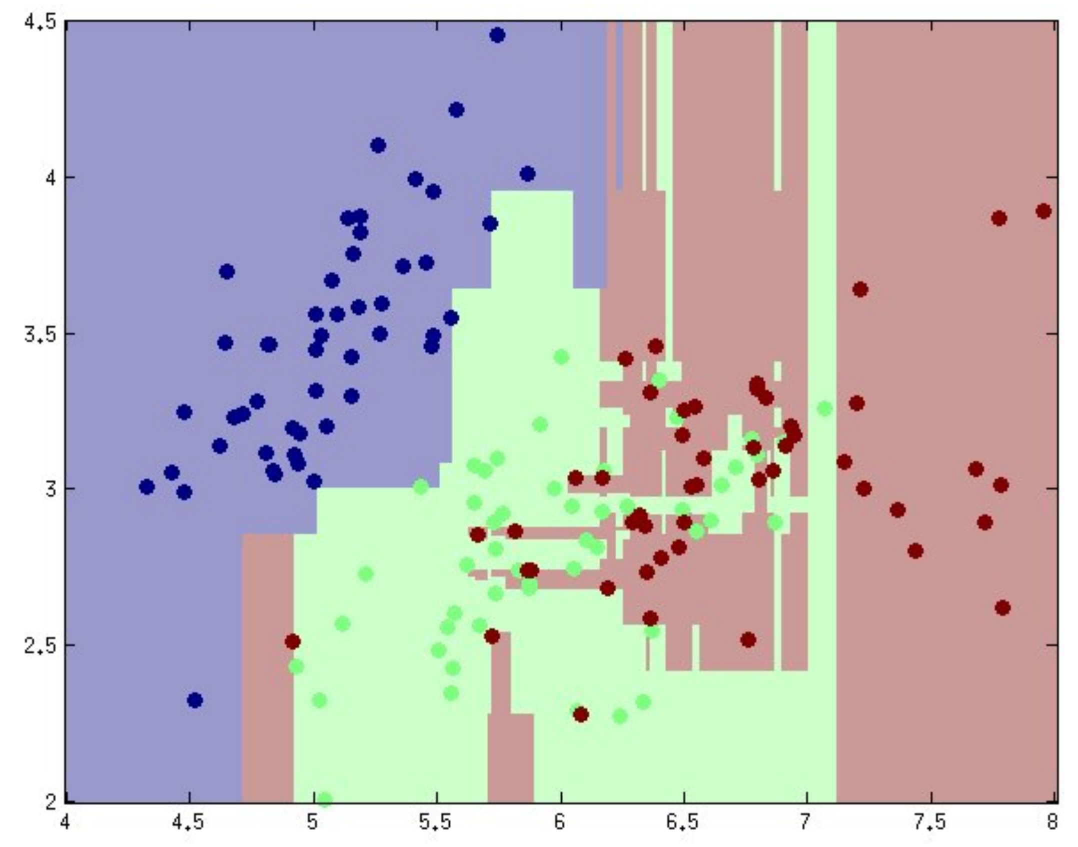
Bagged decision trees

- Average model can't just "memorize" training data
  - Each data point only seen by few models
  - Hopefully still predicted well by majority of other models

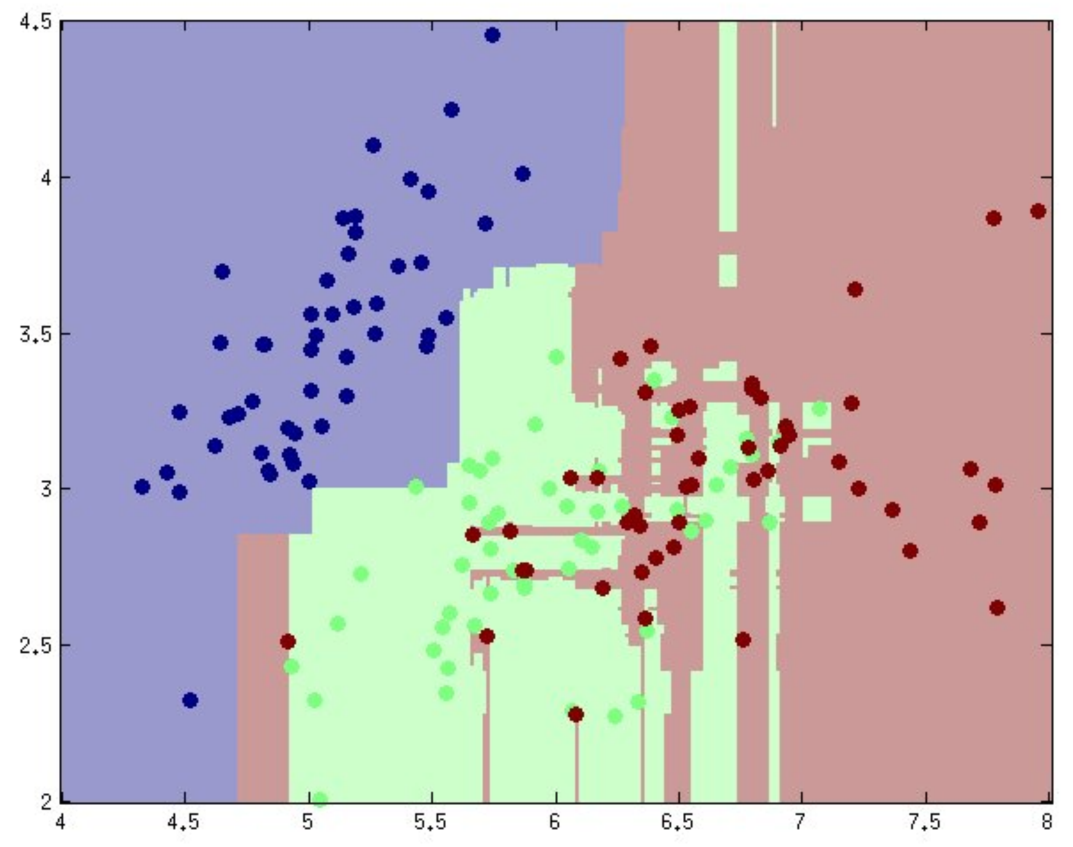
full training dataset



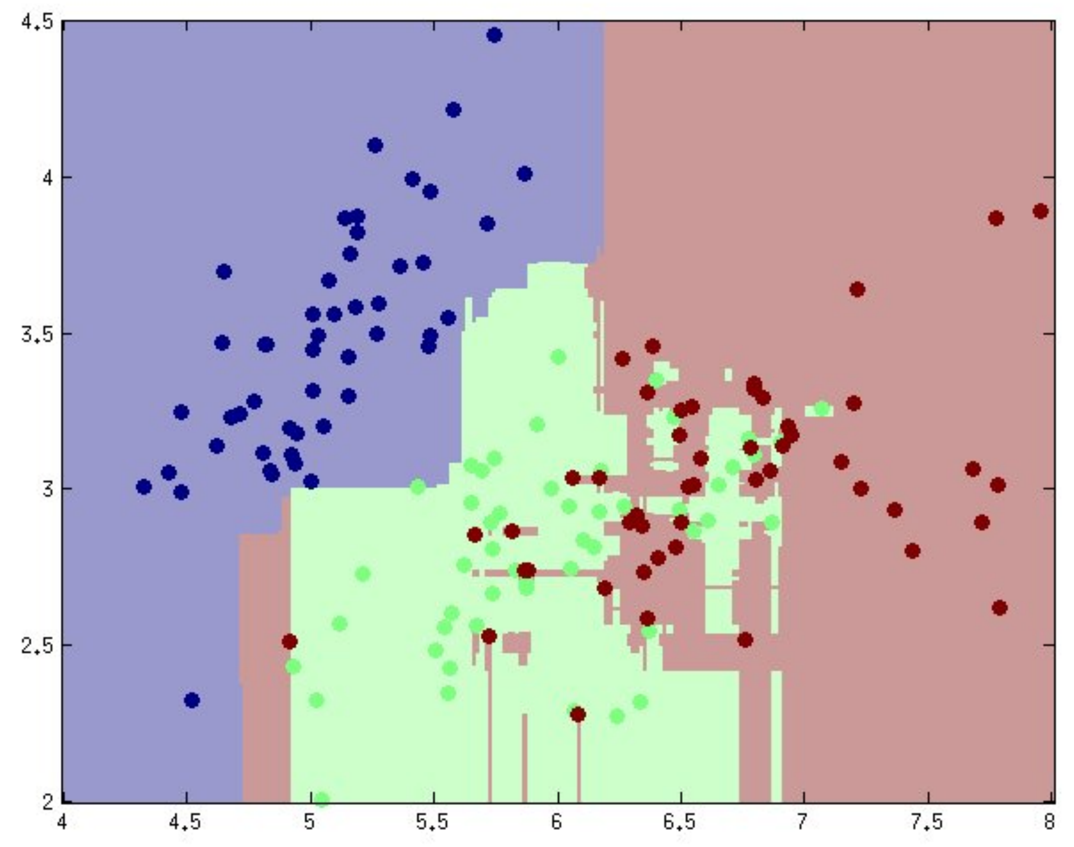
majority of 5 trees



majority of 25 trees



majority of 100 trees



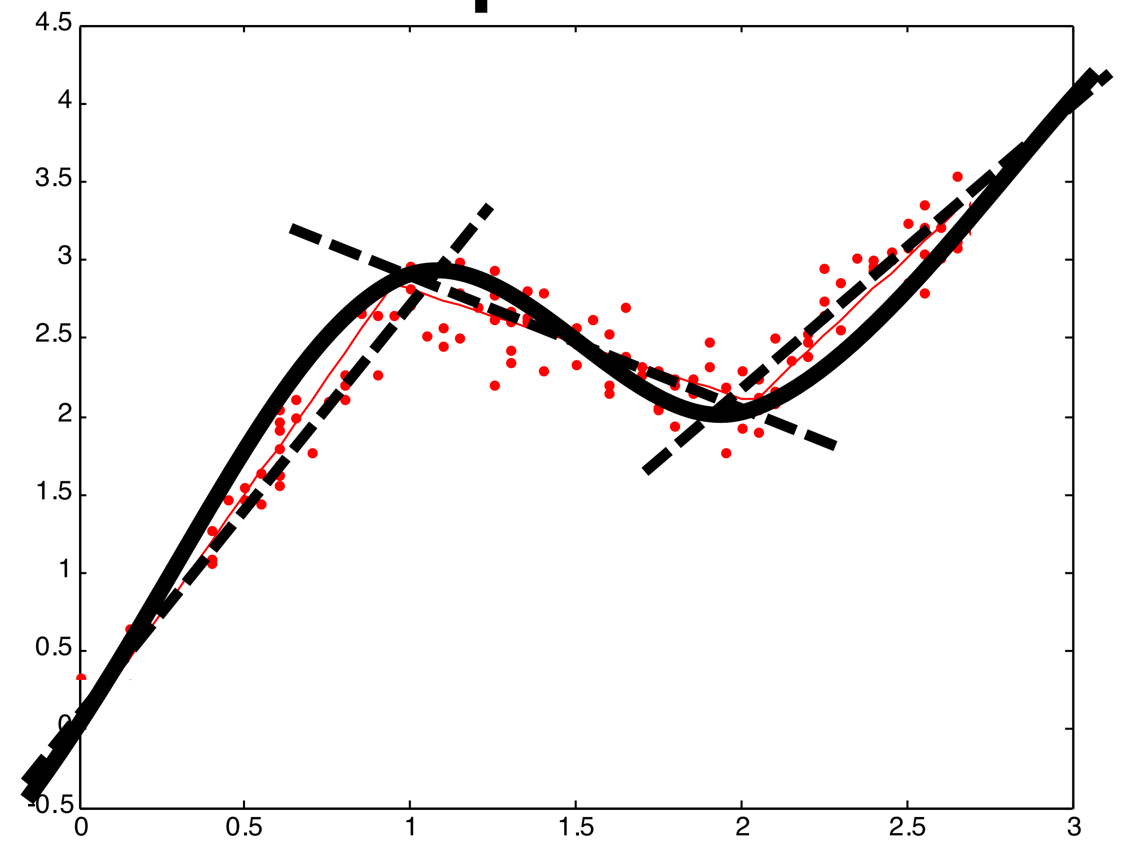
Ensemble methods

- Ensemble = "committee" of models: $\hat{y}_k(x) = f_{\theta_k}(x)$
  - Decisions made by average / majority vote: $\hat{y}(x) = \frac{1}{K}\sum_k \hat{y}_k(x)$
  - May be weighted: better model = higher weight: $\hat{y}(x) = \sum_k \alpha_k \hat{y}_k(x)$
- Stacking = use ensemble as inputs (as in MLP): $\hat{y}(x) = f_\theta(\hat{y}_1(x), \ldots, \hat{y}_K(x))$
  - f_θ trained on held out data = validation of which model should be trusted
  - f_θ linear $\implies$ weighted committee, with learned weights

Mixture of Experts (MoE)

- Experts = models can "specialize", good only for some instances
  - Let weights depend on x: $\hat{y}(x) = \sum_k \alpha_k(x)\hat{y}_k(x)$
- Can we predict which model will perform well?
  - Learn a predictor $\alpha_\phi(k \mid x)$
    - E.g., multilogistic regression (softmax) $\alpha_\phi(k \mid x) = \dfrac{\exp(\phi_k \cdot x)}{\sum_{k'} \exp(\phi_{k'} \cdot x)}$
- Loss, experts, weights differentiable $\Longrightarrow$ end-to-end gradient-based learning

mixture of 3 linear predictors



Random Forests

- Bagging over decision trees: which feature at root?
  - Much data $\Longrightarrow$ max info gain stable across data samples
  - Little diversity among models $\Longrightarrow$ little gained from ensemble
- Random Forests = subsample features
  - Each tree only allowed to use a subset of features
  - Still low, but higher bias
  - Average over trees for lower variance
- Works very well in practice $\Longrightarrow$ go-to algorithm for small ML tasks

Recap

- Ensembles = collections of predictors
  - Combine predictions to improve performance
- Bagging = bootstrap aggregation
  - Reduces model class complexity to mitigate overfitting
  - Resample the data many times (with replacement)
    - For each, train model
  - More bias but less variance
  - Also more compute — both at training time and at test time

Today's lecture

Kernel Machines

Bagging

Gradient boosting

AdaBoost

Growing ensembles

- Ensemble = collection of models: $\hat{y}(x) = \sum_k f_k(x)$
  - Models should "cover" for each other
- If we could add a model to a given ensemble, what would we add?

$$\mathscr{L}(y, \hat{y}') = \mathscr{L}(y, \hat{y} + f_{K+1}(x))$$

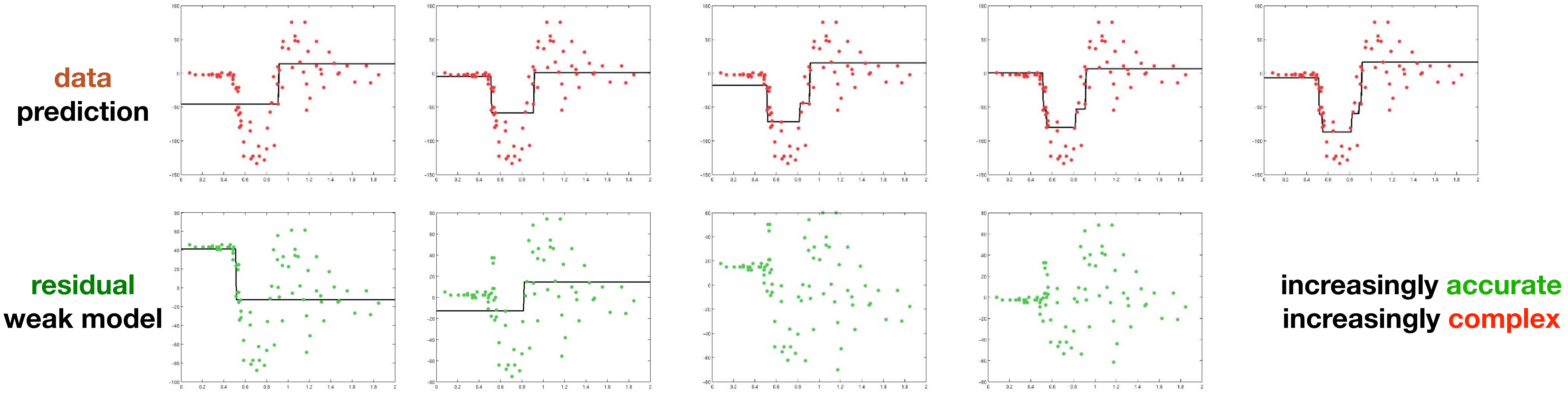
- Let's find $f_{K+1}(x)$ that minimizes this loss
  - If we could do this well — done in one step
  - Instead, let's do it badly but many times → gradually improve

Boosting

- Question: can we create a strong learner from many weak learners?
  - Weak learner = underfits, but fast and simple (e.g., decision stump, perceptron)
  - Strong learner = performs well but increasingly complex
- Boosting: focus new learners on instances that current ensemble gets wrong
  - Train new learner
  - Measure errors
  - Re-weight data points to emphasize large residuals
  - Repeat

Example: MSE loss

- Ensemble: $\hat{y}_K = \sum_k f_k(x)$; MSE loss: $\mathscr{L}(y, \hat{y}_k) = \frac{1}{2}(y - \hat{y}_{k-1} - f_k(x))^2$
  - To minimize: have $f_k(x)$ try to predict $y - \hat{y}_{k-1}$
  - Then update $\hat{y}_k = \hat{y}_{k-1} + f_k(x)$



Gradient Boosting

- More generally: pseudo-residuals $r_k^{(j)} = -\partial_{\hat{y}} \mathscr{L}(y^{(j)}, \hat{y})\Big|_{\hat{y}=\hat{y}_{k-1}^{(j)}}$
  - $r_k^{(j)}$ = steepest descent of loss in “prediction space” (how $\hat{y}_{k-1}^{(j)}$ should change)
  - For MSE loss: $r_k^{(j)} = y^{(j)} - \hat{y}_{k-1}^{(j)}$ as before
- Gradient Boosting:
  - Learn weak model to predict $f_k : x^{(j)} \mapsto r_k^{(j)}$
  - Find best step size $\alpha_k = \arg\min_{\alpha} \frac{1}{m} \sum_j \mathscr{L}\left(y^{(j)}, \hat{y}_{k-1}^{(j)} + \alpha f_k(x^{(j)})\right)$ (line search)

Demo

- http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html

Today's lecture

- Kernel Machines
- Bagging
- Gradient boosting
- **AdaBoost**

Growing ensembles

- Ensemble = collection of models: $\hat{y}(x) = \sum_k \alpha_k f_k(x)$
  - Models should "cover" for each other
- If we could add a model to a given ensemble, what would we add?

$$\mathscr{L}(y, \hat{y}_k) = \mathscr{L}(y, \hat{y}_{k-1} + \alpha_k f_k(x))$$

- Let's find $\alpha_k, f_k(x)$ that minimize this loss
  - If we could do this well — done in one step
  - Instead, let's do it badly but many times → gradually improve

Example: exponential loss

- Exponential loss: $\mathscr{L}(y, \hat{y}) = e^{-y\hat{y}}$
  - Optimal $\hat{y}(x)$: $\arg\min_{\hat{y}} \mathbb{E}_{y|x}[\mathscr{L}(y, \hat{y})] = \frac{1}{2} \ln \frac{p(y=+1|x)}{p(y=-1|x)}$ (proof by derivative)
  - If we can minimize the loss $\Longrightarrow$ $\operatorname{sign}(\hat{y})$ is the more likely label
- Let's find model $f_k : x \mapsto \{+1, -1\}$ that minimizes

$$\sum_j \mathscr{L}(y^{(j)}, \hat{y}_k^{(j)}) = \sum_j \mathscr{L}(y^{(j)}, \hat{y}_{k-1}^{(j)} + \alpha_k f_k(x^{(j)})) = \sum_j \overbrace{e^{-y^{(j)}\hat{y}_{k-1}^{(j)}}}^{w_{k-1}^{(j)}} e^{-y^{(j)}\alpha_k f_k(x^{(j)})}$$

independent of f_k

$$= (e^{\alpha_k} - e^{-\alpha_k}) \sum_j w_{k-1}^{(j)} \delta[y^{(j)} \neq f_k(x^{(j)})] + e^{-\alpha_k} \sum_j w_{k-1}^{(j)}$$

independent of f_k

Minimizing weighted loss

- So far, we minimized average loss: $\frac{1}{m}\sum_j \mathcal{L}(y^{(j)}, \hat{y}^{(j)})$
- We can also minimize weighted loss: $\sum_j w^{(j)}\mathcal{L}(y^{(j)}, \hat{y}^{(j)})$
  - Every data point "counts" as $w^{(j)}$
  - E.g., in decision trees, weighted info gain obtained by $p(y=c) \propto \sum_{j:y^{(j)}=c} w^{(j)}$
- In our current case, weighted 0–1 loss: $\sum_j w_{k-1}^{(j)}\delta[y^{(j)} \neq f_k(x^{(j)})]$

Boosting with exponential loss (cont.)

- The best classifier to add to the ensemble minimizes weighted 0–1 loss:

$$\sum_j w_{k-1}^{(j)} \delta[y^{(j)} \neq f_k(x^{(j)})] \quad \text{with } w_{k-1}^{(j)} = e^{-y^{(j)} \hat{y}_{k-1}^{(j)}}$$

- It gives weighted error rate $\epsilon_k = \dfrac{\sum_j w_{k-1}^{(j)} \delta[y^{(j)} \neq f_k(x^{(j)})]}{\sum_j w_{k-1}^{(j)}}$

- Plugging into the loss and solving: $\alpha_k = \dfrac{1}{2} \ln \dfrac{1 - \epsilon_k}{\epsilon_k}$

- Now add the model and update the ensemble $\hat{y}_k(x) = \hat{y}_{k-1}(x) + \alpha_k f_k(x)$

AdaBoost

- AdaBoost = adaptive boosting:
  - Initialize $w_0^{(j)} = \frac{1}{m}$
  - Train classifier f_k on training data with weights w_{k-1}
  - Compute weighted error rate $\epsilon_k = \frac{\sum_j w_{k-1}^{(j)} \delta[y^{(j)} \neq f_k(x^{(j)})]}{\sum_j w_{k-1}^{(j)}}$
  - Compute $\alpha_k = \frac{1}{2} \ln \frac{1 - \epsilon_k}{\epsilon_k}$
  - Update weights $w_k^{(j)} = w_{k-1}^{(j)} e^{-y^{(j)} \alpha_k f_k(x^{(j)})}$ (increase weight for misclassified points)
- Predict $\hat{y}(x) = \operatorname{sign} \sum_k \alpha_k f_k(x)$

Recap

- Ensembles = collections of predictors
  - Combine predictions to improve performance
- Boosting: Gradient Boost, AdaBoost, ...
  - Build strong predictor from many weak ones
  - Train sequentially; later predictors focus on mistakes by earlier
    - Weight “hard” examples more