Lecture 22: ensemble methods

CS 475: Machine Learning

Raman Arora

April 26, 2017

Slides credit: Greg Shakhnarovich

Review: CART

$$C_\lambda(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \lambda |T|$$

- Q_m is the "lack of purity" of leaf m; measured differently in regression vs. classification
- T is obtained from T_0 by collapsing some internal nodes (merging multiple leaves)
- For a given $\lambda \geq 0$, there exists a unique $T_\lambda = \text{argmin}_T \, C_\lambda(T)$
- Weakest link pruning: keep collapsing the internal nodes that produce the *smallest increase* in $\sum_m N_m Q_m(T)$, going from T_0 to a single node.
- The resulting sequence contains T_λ, found explicitly
- Tune λ, e.g., by (cross) validation

Roadmap

- Decision trees: partition space hierarchically
- Can see a tree as a combination of very simple models; each model is responsible for one area (region)
- Today: boosting – a way to build a combination of models greedily

Combining classifiers

- Classifying a point by decision tree can be seen as a sequence of classifiers, refined as we follow the path to a leaf.
- A more general formulation: combine classifiers $h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})$

$$H(\mathbf{x}) \;=\; \alpha_1 h_1(\mathbf{x}) + \ldots + \alpha_m h_m(\mathbf{x}),$$

- α_j is the *vote* assigned to classifier h_j.
  - Votes should be higher for more reliable classifiers.
- Prediction:

$$\hat{y}(\mathbf{x}) \;=\; \operatorname{sign} H(\mathbf{x}).$$

- Classifiers h_j can be simple (e.g., based on a single feature).

Greedy assembly of classifier combination

- Consider a family of classifiers $\mathcal{H}$ parametrized by θ.
- Setting θ_1: minimize the training error

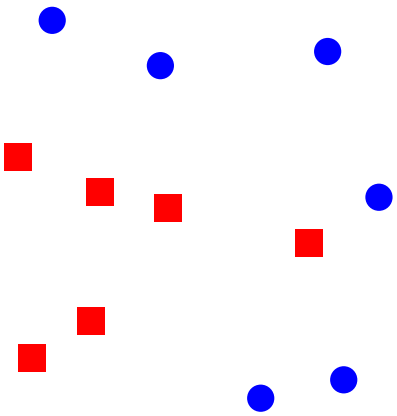
$$\sum_{i=1}^{N} L(h(\mathbf{x}_i; \theta_1),\, y_i),$$

  where L is some surrogate for the 0/1 loss.
- How do we set θ_2?
- We would like to minimize the (surrogate) loss of the combination,

$$\sum_{i=1}^{N} L(H(\mathbf{x}_i),\, y_i),$$

  where $H(\mathbf{x}) \;=\; \operatorname{sign}\left(\alpha_1 h(\mathbf{x}; \theta_1) + \alpha_2 h(\mathbf{x}; \theta_2)\right)$.

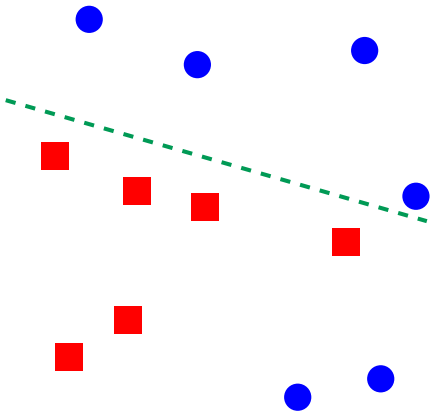
AdaBoost: intuition



Greedy alg. for $m = 1, \ldots, M$

- Maintain weights $W_i^{(m)}$, initially all $1/N$
- Pick a weak classifier h_m minimizing error ϵ_m weighted by $W^{(m-1)}$
- Set $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
- Update weights $W_i^{(m)}$ based on mistakes of h_m and on α_m
- Final (strong) classifier $\operatorname{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

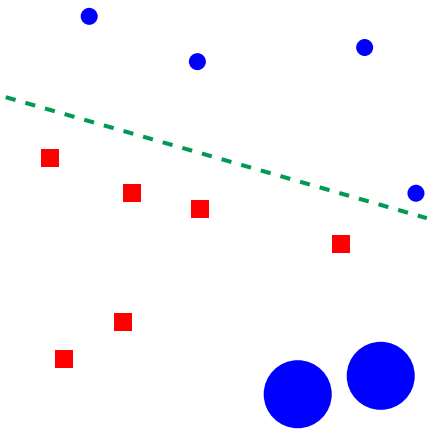
AdaBoost: intuition



Greedy alg. for $m = 1, \ldots, M$

- Maintain weights $W_i^{(m)}$, initially all $1/N$
- Pick a weak classifier h_m minimizing error ϵ_m weighted by $W^{(m-1)}$
- Set $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
- Update weights $W_i^{(m)}$ based on mistakes of h_m and on α_m
- Final (strong) classifier $\text{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

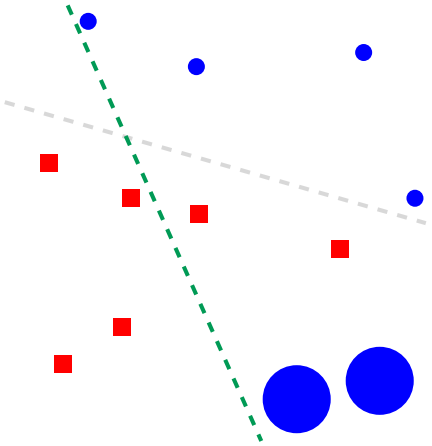
AdaBoost: intuition



Greedy alg. for $m = 1, \ldots, M$

- Maintain weights $W_i^{(m)}$, initially all $1/N$
- Pick a weak classifier h_m minimizing error ϵ_m weighted by $W^{(m-1)}$
- Set $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
- Update weights $W_i^{(m)}$ based on mistakes of h_m and on α_m
- Final (strong) classifier $\text{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

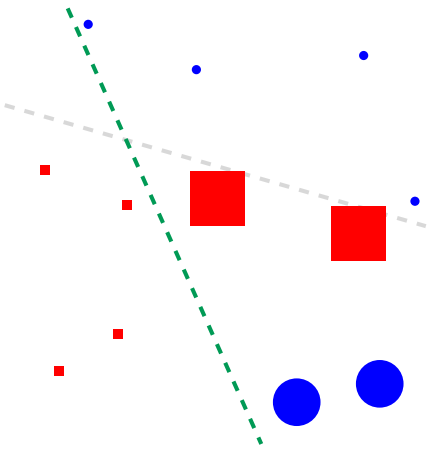
AdaBoost: intuition



Greedy alg. for $m = 1, \ldots, M$

- Maintain weights $W_i^{(m)}$, initially all $1/N$
- Pick a weak classifier h_m minimizing error ϵ_m weighted by $W^{(m-1)}$
- Set $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
- Update weights $W_i^{(m)}$ based on mistakes of h_m and on α_m
- Final (strong) classifier $\text{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

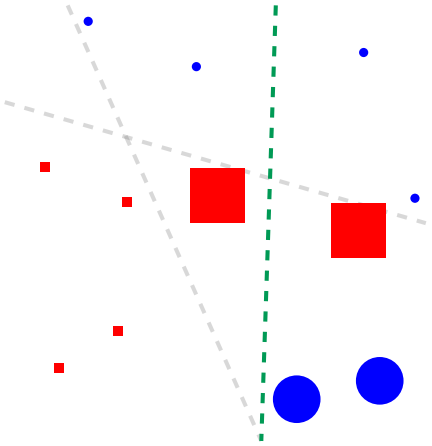
AdaBoost: intuition



Greedy alg. for $m = 1, \ldots, M$

- Maintain weights $W_i^{(m)}$, initially all $1/N$
- Pick a weak classifier h_m minimizing error ϵ_m weighted by $W^{(m-1)}$
- Set $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
- Update weights $W_i^{(m)}$ based on mistakes of h_m and on α_m
- Final (strong) classifier $\text{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

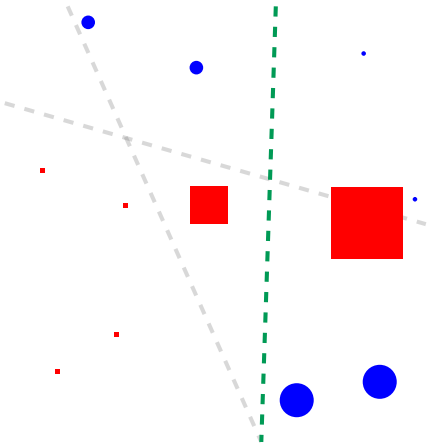
AdaBoost: intuition



Greedy alg. for $m = 1, \ldots, M$

- Maintain weights $W_i^{(m)}$, initially all $1/N$
- Pick a weak classifier h_m minimizing error ϵ_m weighted by $W^{(m-1)}$
- Set $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
- Update weights $W_i^{(m)}$ based on mistakes of h_m and on α_m
- Final (strong) classifier $\text{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

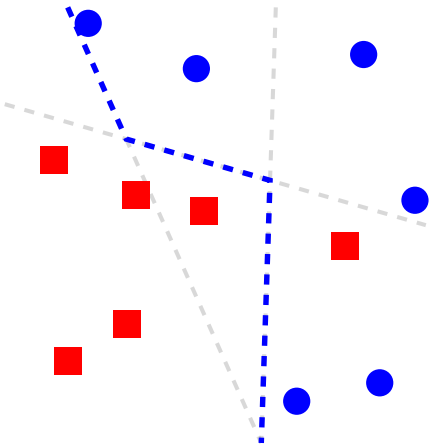
AdaBoost: intuition



Greedy alg. for $m = 1, \ldots, M$

- Maintain weights $W_i^{(m)}$, initially all $1/N$
- Pick a weak classifier h_m minimizing error ϵ_m weighted by $W^{(m-1)}$
- Set $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
- Update weights $W_i^{(m)}$ based on mistakes of h_m and on α_m
- Final (strong) classifier $\text{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

AdaBoost: intuition



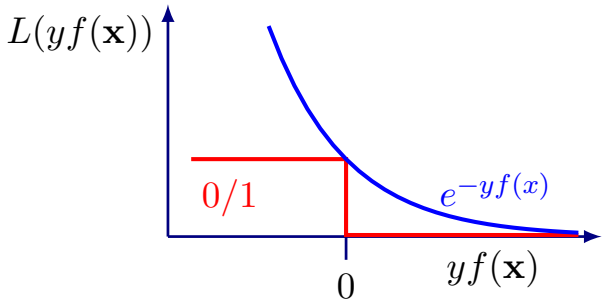
Greedy alg. for $m = 1, \ldots, M$

- Maintain weights $W_i^{(m)}$, initially all $1/N$
- Pick a weak classifier h_m minimizing error ϵ_m weighted by $W^{(m-1)}$
- Set $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
- Update weights $W_i^{(m)}$ based on mistakes of h_m and on α_m
- Final (strong) classifier $\text{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

Exponential loss function

- Another surrogate: *exponential loss* $L(H(\mathbf{x}), y) = e^{-y \cdot H(\mathbf{x})}$

$$L(H, X) = \sum_{i=1}^{N} L(H(\mathbf{x}_i), y_i)$$
$$= \sum_{i=1}^{N} e^{-y_i \cdot H(\mathbf{x}_i)}$$



- Differentiable upper bound on 0/1 loss.
  - Easy to optimize!
- Other choices of loss are possible.

Ensemble loss

- Denote $H_M(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \ldots + \alpha_M h_M(\mathbf{x})$
- Suppose we add $\alpha_m \cdot h_m(\mathbf{x})$ to H_{m-1}:

$$\begin{aligned} L(H_m, X) &= \sum_{i=1}^{N} e^{-y_i \cdot [H_{m-1}(\mathbf{x}_i) + \alpha_m h_m(\mathbf{x}_i)]} \\ &= \sum_{i=1}^{N} e^{-y_i H_{m-1}(\mathbf{x}_i) - \alpha_m y_i h_m(\mathbf{x}_i)} \\ &= \sum_{i=1}^{N} e^{-y_i H_{m-1}(\mathbf{x}_i)} \cdot e^{-\alpha_m y_i h_m(\mathbf{x}_i)} \end{aligned}$$

- Define $W_i^{(m-1)} = e^{-y_i H_{m-1}(\mathbf{x}_i)}$

Weighted loss

- Weights defined $W_i^{(m-1)} = e^{-y_i H_{m-1}(\mathbf{x}_i)}$
- Exponential loss after m-th iteration:

$$L(H_m, X) = \sum_{i=1}^{N} W_i^{(m-1)} \underbrace{e^{-y_i \alpha_m h_m(\mathbf{x}_i)}}_{\text{need to optimize}}$$

- $W_i^{(m-1)}$ captures the "history" of classification of $\mathbf{x}_i$ by H_{m-1}.
- Optimization: choose α_m, $h_m = h(\mathbf{x}; \theta_m)$ that minimize the (weighted) exponential loss at iteration m.
  - Remember: this means minimizing an upper bound on classification error.

Optimizing weak learner

$$\sum_{i=1}^{N} W_i^{(m-1)} e^{-\alpha_m y_i h_m(\mathbf{x}_i)} = e^{-\alpha_m} \sum_{i:\, y_i = h_m(\mathbf{x}_i)} W_i^{(m-1)} + e^{\alpha_m} \sum_{i:\, y_i \neq h_m(\mathbf{x}_i)} W_i^{(m-1)}$$

- For any $\alpha_m > 0$, $e^{-\alpha_m} < e^{\alpha_m}$, so minimizing this $\Rightarrow$ minimizing training error, weighted by $W^{(m-1)}$
- We can normalize the weights:

$$W_i^{(m-1)} = \frac{e^{-y_i H_{m-1}(\mathbf{x}_i)}}{\sum_{j=1}^{N} e^{-y_j H_{m-1}(\mathbf{x}_j)}}$$

Optimizing votes

- The weighted error of h_m:

$$\epsilon_m = \sum_{i:\, y_i \neq h_m(\mathbf{x}_i)} W_i^{(m-1)}$$

- Given h_m and its ϵ_m, set α_m that minimizes the exponential loss:

$$\alpha_m = \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m}$$

- As long as $\epsilon_m < \frac{1}{2}$, $\alpha_m > 0$.

AdaBoost: algorithm summary

1. Initialize weights: $W_i^{(0)} = 1/N$
2. Iterate for $m = 1, \ldots, M$:
   - Find (any) "weak" classifier h_m that attains weighted error

$$\epsilon_m \;=\; \frac{1}{2}\left(1 - \sum_{i=1}^{N} W_i^{(m-1)} y_i h_m(\mathbf{x}_i)\right) \;<\; \frac{1}{2}$$

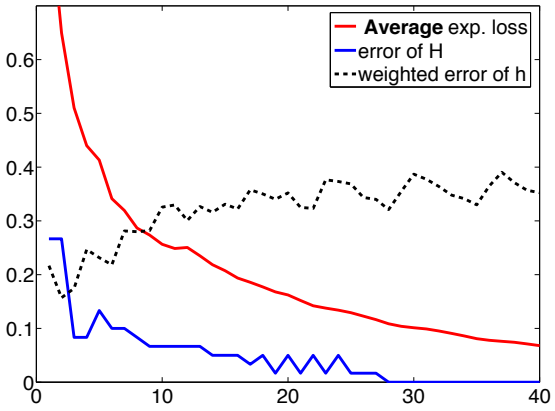
   - Let $\alpha_m \;=\; \frac{1}{2}\log\frac{1-\epsilon_m}{\epsilon_m}$.
   - Update the weights and normalize so that $\sum_i W_i^{(m)} = 1$:

$$W_i^{(m)} \;=\; \frac{1}{Z} W_i^{(m-1)} e^{-\alpha_m y_i h_m(\mathbf{x}_i)},$$

3. The combined classifier: $\operatorname{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$

AdaBoost: typical behavior

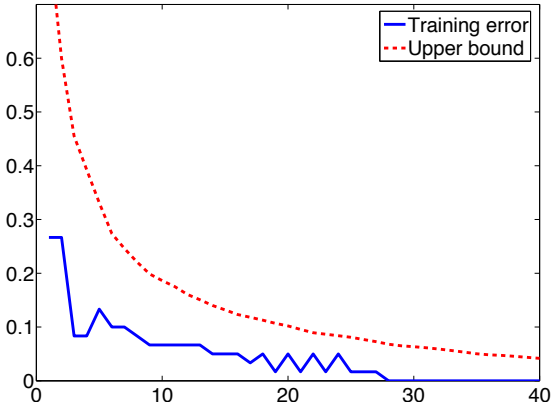
- Training error of H goes down;
- Weighted error ϵ_m goes up;
  $\Rightarrow$ votes α_m go down.
- Exponential loss goes strictly down.



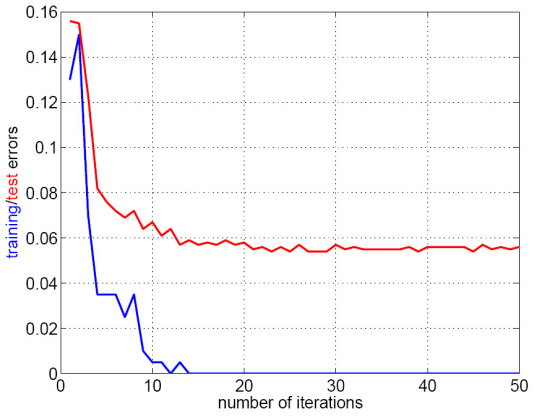
AdaBoost behavior: training error

- Can show: the training error in m-th iteration is bounded

$$err(H_m) \leq \prod_{j=1}^{m} 2\sqrt{\epsilon_j(1-\epsilon_j)}.$$



AdaBoost behavior: test error



- Typical behavior: test error can still decrease after training error is flat (even zero).

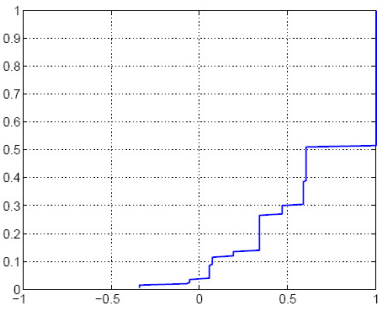
Boosting the margin

- We can define the *margin* of an example

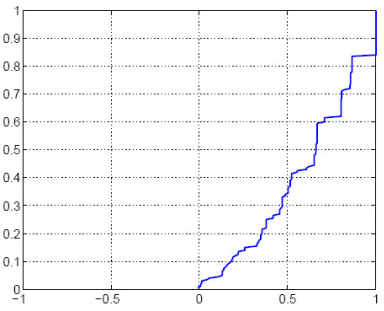
$$\gamma(\mathbf{x}_i) \;=\; y_i \cdot \frac{\alpha_1 h_1(\mathbf{x}) + \ldots + \alpha_m h_m(\mathbf{x})}{\alpha_1 + \ldots + \alpha_m}$$

- $\gamma(\mathbf{x}_i) \in [-1, 1]$, positive iff $H(\mathbf{x}_i) = y_i$; this is a measure of confidence in the correct decision.
- Iterations of AdaBoost *increase* the margin of training examples!
  - Even for correct classification can further improve confidence.

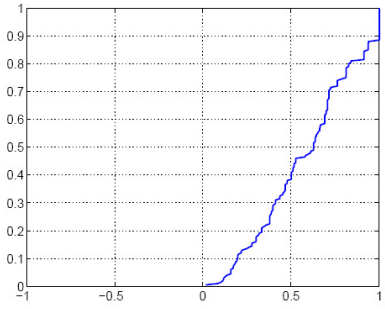
Cumulative distribution of $\gamma(\mathbf{x}_i)$



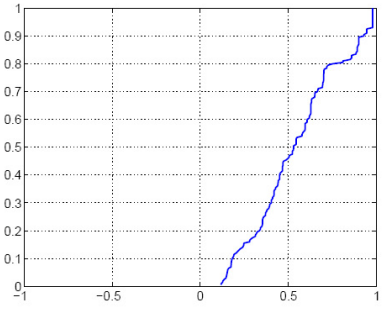
4 iterations



10 iterations



20 iterations



50 iterations

Variations of boosting

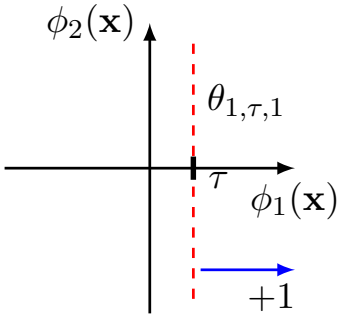
- Different surrogate loss functions
- Confidence rated version: $h(\mathbf{x}) \in [-1, 1]$ instead of $\{\pm 1\}$
- FloatBoost: after each round (having added a weak classifier), see if *removal* of a previously added classifier is helpful.
- Totally corrective AdaBoost: update the αs for all weak classifiers once done.

Boosting stumps

- Suppose $\mathbf{x} = [\phi_0(\mathbf{x}), \phi_1(\mathbf{x}), \ldots, \phi_d(\mathbf{x})]^T$
- Decision stump: a simple classifier

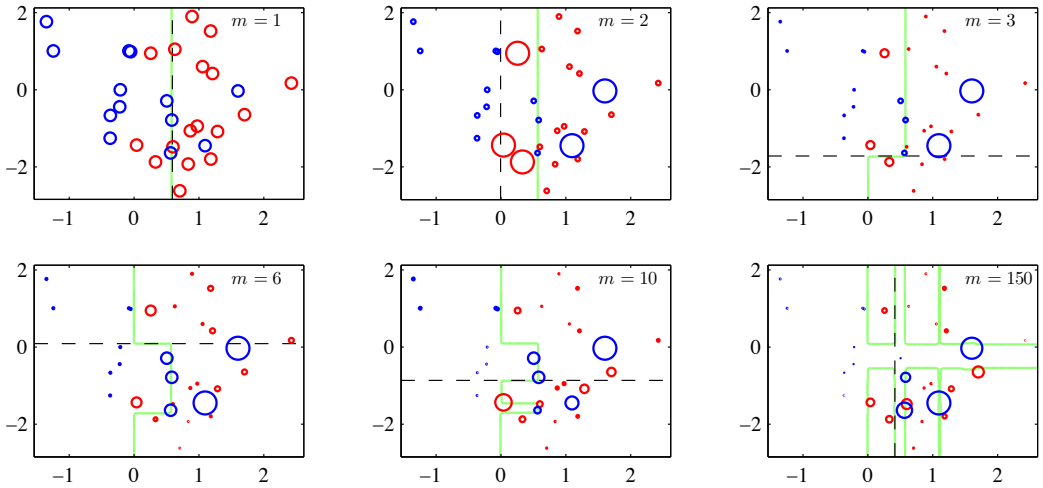
$$\theta_{j,\tau,s}(\mathbf{x}) = \begin{cases} +1 & \text{if } s\phi_j(\mathbf{x}) \geq \tau, \\ -1 & \text{if } s\phi_j(\mathbf{x}) < \tau \end{cases}$$

- Decision boundary: linear, axis parallel



Boosting decision stumps

- Decision stumps: finite family of classifiers
- Example of a boosting run (from Bishop)



Boosting and bias-variance tradeoff

$$H(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})$$

- What determines the complexity of boosted classifier? complexity of weak classifiers h_m, and their number M
- Regularization of H: early stopping
- Typically should prefer simple weak classifiers: stumps, shallow decision trees.

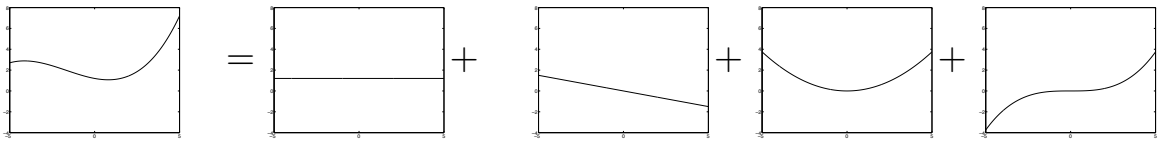
Combination of regressors

- Consider linear regression model

$$y = F(\mathbf{x}; \mathbf{w}) = w_0\phi_0(\mathbf{x}) + w_1\phi_1(\mathbf{x}) + \ldots + w_d\phi_d(\mathbf{x}).$$

- We can see this as a combination of $d+1$ simple regressors:

$$F(\mathbf{x}; \mathbf{w}) = \sum_{j=0}^{d} f_j(\mathbf{x}; \mathbf{w}), \qquad f_j(\mathbf{x}; \mathbf{w}) \triangleq w_j\phi_j(\mathbf{x})$$



Forward stepwise regression

$$F_d(\mathbf{x}; \mathbf{w}) = \sum_{j=0}^{d} f_j(\mathbf{x}; \mathbf{w}), \qquad f_j(\mathbf{x}; \mathbf{w}) = w_j\phi_j(\mathbf{x})$$

- We can build this combination greedily, one function at a time.
- Parametrize the set of functions: $f(\mathbf{x}; \theta)$, $\theta = [w, j]$
- Step 1: fit the first simple model

$$\theta_1 = \operatorname*{argmin}_{\theta} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i; \theta))^2$$

Forward stepwise regression

- Step 1: fit the first simple model

$$\theta_1 \;=\; \operatorname*{argmin}_{\theta} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i;\theta))^2$$

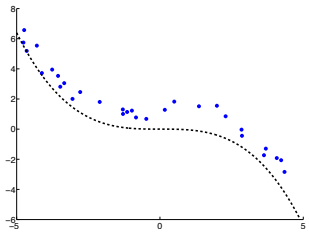
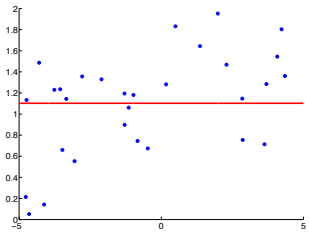
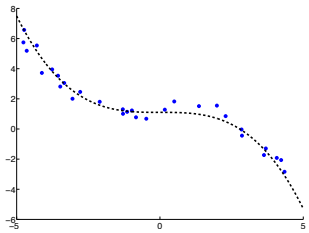
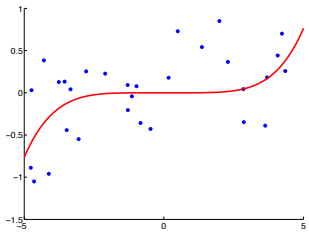
- Step 2: fit second simple model *to the residuals* of the first:

$$\theta_2 \;=\; \operatorname*{argmin}_{\theta} \sum_{i=1}^{N} (\underbrace{y_i - f(\mathbf{x}_i;\theta_1)}_{\text{residual}} - f(\mathbf{x}_i;\theta))^2$$

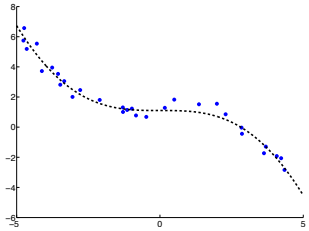
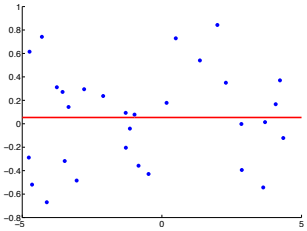
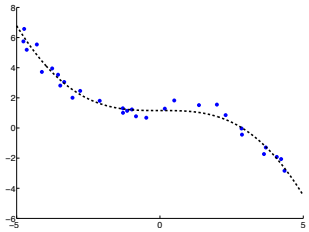
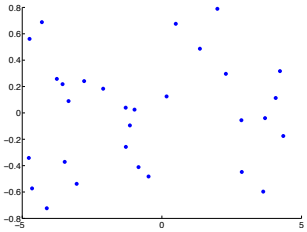
- . . . Step n: fit a simple model to the residuals of the previous step, $y_i - F_{d-1}(\mathbf{x})$
- Stop when no significant improvement in training error.
- Final estimate after M steps:

$$\hat{y}(\mathbf{x}) \;=\; F_M(\mathbf{x};\mathbf{w}) \;=\; f_1(\mathbf{x};\theta_1) + \ldots + f_M(\mathbf{x};\theta_M)$$

Stepwise regression: example

	Fit $\sum_{j=1}^{k} f(\mathbf{x};\theta_j)$	Residuals	θ_j
$k=1$			$d=3, w=-0.0512$
$k=2$			$d=0, w=1.1024$

Stepwise regression: example

	Fit $\sum_{j=1}^{k} f(\mathbf{x};\theta_j)$	Residuals	θ_j
$k=3$			$d=5, w=0.0002$
$k=4$			$d=0, w=0.0536$

Combining regression trees

- Deep decision trees have low bias, high variance
- CART pruning often leads to poor bias/variance tradeoff
- Idea: let trees be deep (low bias), **average** many trees (low variance)

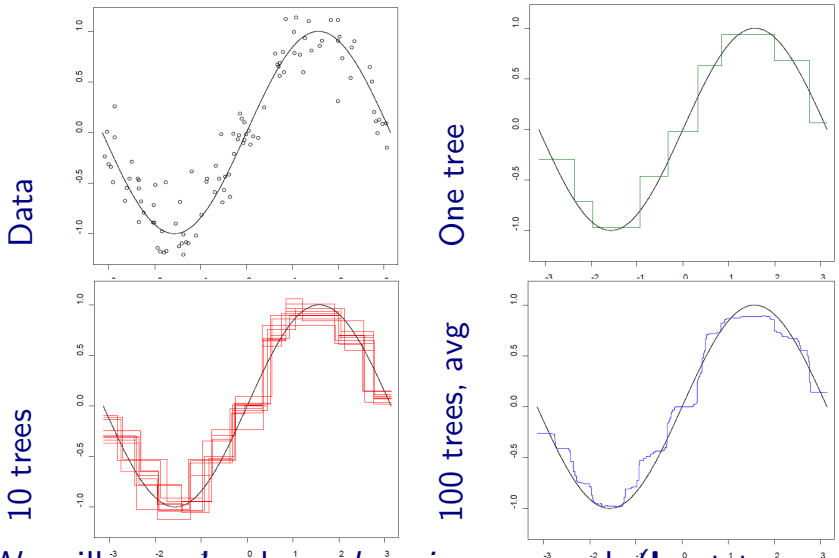


figure: A. Cutler

- We will now develop a *bagging* approach (**b**ootstrap **agg**regation)

Random forests

- In order to benefit from many trees (lower variance), we need them to be different (diverse)
- We will obtain diversity by injecting *randomness* into tree construction
- Two sources of randomness
- **Bootstrap** sampling: out of N training examples, sample N *with replacement*
  some points will appear more than once, some (approx. 37%) will not appear at all
- **Sampling features** in each node, when considering splits, only look at a random $m < d$ features.
- Each tree is less likely to overfit
- The "overfitting quirks" of different trees are likely to cancel out in averaging

Classification with random forests

- In classification, we can't just average labels
- Can either average *scores*, or let trees vote

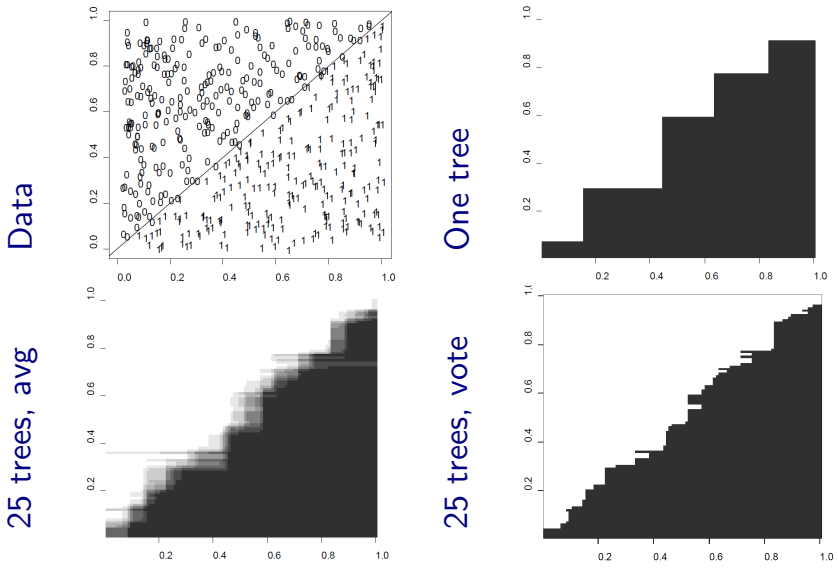


figure: A. Cutler

Random forests summary

- N data points, each with d features
- Build T trees independently (in parallel)
- For each tree:
  sample N points with replacement (or $N' < N$ without)
  Grow CART tree; in each node, only look at a random subset of $m < d$ features
  Do not prune the trees.
- To make a prediction: average (regression), or vote (classification)
- Tuning parameters: number of trees T; feature set size m; tree depth/min number of points in leaves
- Tune on validation; recommended heuristic value of m is $\sqrt{d}$ for classification, $d/3$ for regression

Bagging in general

- Instead of trees, could apply bagging to any predictor family
- Power of bagging: variance reduction through averaging
- Typically, benefit is highest with unstable, highly nonlinear predictors (e.g., trees)
- Linear predictors: no benefit from bagging
- Useful property of bagging: "out of bag" (OOB) data
  in each tree, treat the ≈37% of the examples that didn't make it to the sample as a kind of validation set
- While assembling the trees, keep track of OOB accuracy, stop when see plateau.

Ensemble methods: summary so far

- Main benefit of ensembles: control overfitting
- Boosting: build ensemble of low-variance, high-bias predictors sequentially
  AdaBoost: binary classification, exponential surrogate loss
  gradient boosting: more general framework
- Bagging: build ensemble of high-variance, low-bias predictors in parallel
  use randomness and averaging to reduce variance (e.g., random forests)
- Relatively straightforward to tune; robust
- Not very interpretable
- Computationally expensive (train *and* test time)