Lecture 12: Ensemble Methods

TTIC 31020: Introduction to Machine Learning

Instructor: Kevin Gimpel

TTI-Chicago

November 7, 2019

Review: CART

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

- 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 $T_\lambda = \operatorname{argmin}_T C_\lambda(T)$
- 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

Ensembles of models

- So far, we have considered a single model approach: train a model, apply it on test data
- What if we have multiple (different) models?
- **Ensemble** of models: given M models $\{f_1(\mathbf{x}), \ldots, f_M(\mathbf{x})\}$, at test time combine them to make a single prediction
- Simplest way to combine: average

$$\widehat{f}(\mathbf{x}) = \frac{1}{M} \sum_j f_j(\mathbf{x})$$

- When does it help?
  How do we come up with the ensemble?
  Can we do better than just average?

Combining trees

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

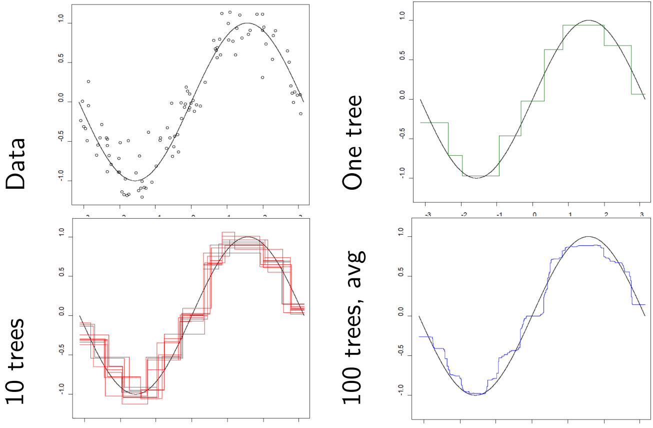


figure: A. Cutler

- 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

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

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

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: little to no benefit from bagging
- Useful property of bagging: "out of bag" (OOB) data
  in each tree, treat the $\approx$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.

Combining classifiers

- Combine classifiers as follows: $h_1(\mathbf{x}), \ldots, h_m(\mathbf{x})$

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

  where α_j is the weight of the vote assigned to classifier h_j
- Votes should have higher weight for more reliable classifiers
- Prediction (for binary classification):

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

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

Greedy assembly of classifier combination

- Setting h_1: minimize the training error

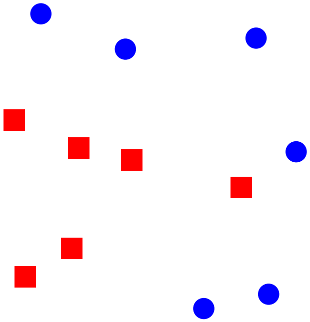
$$\sum_{i=1}^{n} L(h_1(\mathbf{x}_i),\, y_i)$$

  where L is some surrogate for the 0/1 loss
- How do we learn h_2?
- We would like to minimize the (surrogate) loss of the combination,

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

  where $H(\mathbf{x}) \;=\; \operatorname{sign}\left(\alpha_1 h_1(\mathbf{x}) + \alpha_2 h_2(\mathbf{x})\right)$

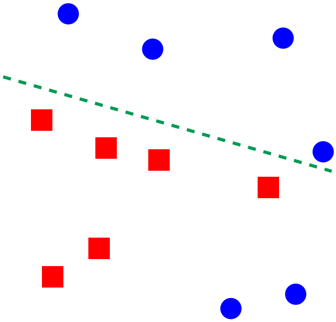
AdaBoost: intuition



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

- Maintain weights $W_i^{(m)}$, initially all $1/n$
- Learn 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 based on α_m
- Final (strong) classifier $\operatorname{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

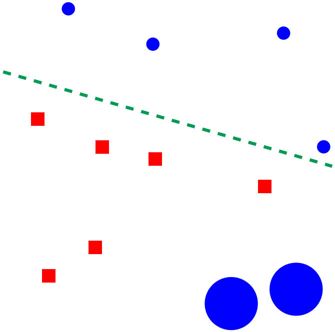
AdaBoost: intuition



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

- Maintain weights $W_i^{(m)}$, initially all $1/n$
- Learn 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 based on α_m
- Final (strong) classifier $\text{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

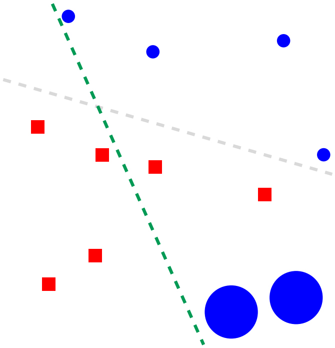
AdaBoost: intuition



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

- Maintain weights $W_i^{(m)}$, initially all $1/n$
- Learn 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 based on α_m
- Final (strong) classifier $\operatorname{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

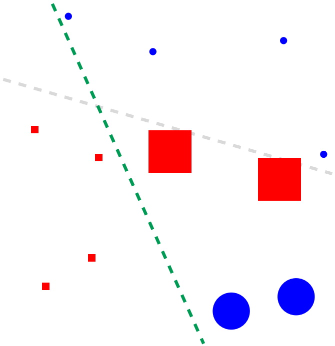
AdaBoost: intuition



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

- Maintain weights $W_i^{(m)}$, initially all $1/n$
- Learn 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 based on α_m
- Final (strong) classifier $\operatorname{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

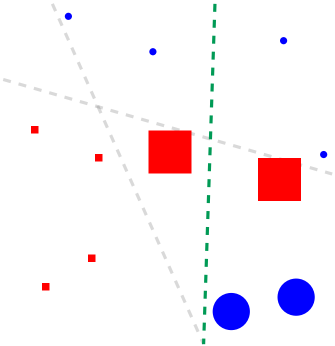
AdaBoost: intuition



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

- Maintain weights $W_i^{(m)}$, initially all $1/n$
- Learn 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 based on α_m
- Final (strong) classifier $\operatorname{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

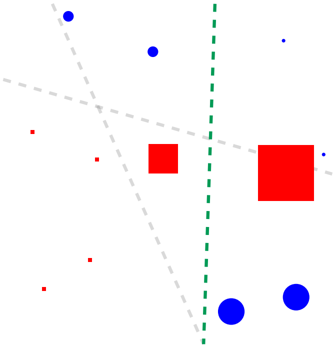
AdaBoost: intuition



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

- Maintain weights $W_i^{(m)}$, initially all $1/n$
- Learn 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 based on α_m
- Final (strong) classifier $\operatorname{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

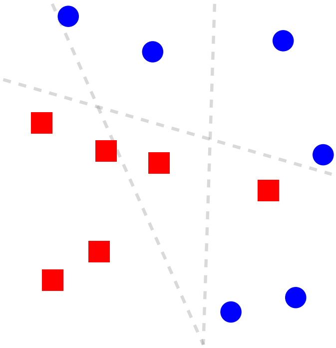
AdaBoost: intuition



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

- Maintain weights $W_i^{(m)}$, initially all $1/n$
- Learn 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 based 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 $1/n$
- Learn 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 based on α_m
- Final (strong) classifier $\operatorname{sign}\left(\sum_m \alpha_m h_m(\cdot)\right)$

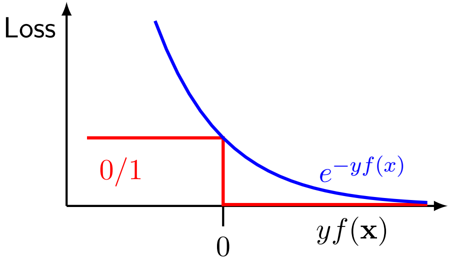
Update the weight for each data point

Exponential loss function

- Another surrogate: **exponential loss** ("exp loss")

$$L(f(\mathbf{x}), y) \;=\; e^{-yf(\mathbf{x})}$$

$$L(f, \mathbf{X}, \mathbf{y}) \;=\; \sum_{i=1}^{n} e^{-y_i f(\mathbf{x}_i)}$$



- Differentiable upper bound on 0/1 loss
- Other choices of loss are possible within this boosting framework

Greedy training of ensemble

- Denote $H_k(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \ldots + \alpha_k h_k(\mathbf{x})$
- Note: the weak classifiers $h_j(\mathbf{x})$ return a value in $\{-1, 1\}$, *not* in $\mathbb{R}$
- Suppose we add $\alpha_m h_m(\mathbf{x})$ to H_{m-1} to get H_m
- We will keep H_{m-1} fixed and we want to learn α_m and h_m

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

Greedy training of ensemble

- Denote $H_k(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \ldots + \alpha_k h_k(\mathbf{x})$
- Note: the weak classifiers $h_j(\mathbf{x})$ return a value in $\{-1, 1\}$, *not* in $\mathbb{R}$
- Suppose we add $\alpha_m h_m(\mathbf{x})$ to H_{m-1} to get H_m
- We will keep H_{m-1} fixed and we want to learn α_m and h_m

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

Greedy training of ensemble

- Denote $H_k(\mathbf{x}) = \alpha_1 h_1(\mathbf{x}) + \ldots + \alpha_k h_k(\mathbf{x})$
- Note: the weak classifiers $h_j(\mathbf{x})$ return a value in $\{-1, 1\}$, *not* in $\mathbb{R}$
- Suppose we add $\alpha_m h_m(\mathbf{x})$ to H_{m-1} to get H_m
- We will keep H_{m-1} fixed and we want to learn α_m and h_m

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

Greedy training of ensemble

- Denote $H_k(\mathbf{x}) \;=\; \alpha_1 h_1(\mathbf{x}) + \ldots + \alpha_k h_k(\mathbf{x})$
- Note: the weak classifiers $h_j(\mathbf{x})$ return a value in $\{-1, 1\}$, *not* in $\mathbb{R}$
- Suppose we add $\alpha_m h_m(\mathbf{x})$ to H_{m-1} to get H_m
- We will keep H_{m-1} fixed and we want to learn α_m and h_m

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

try to make a new h that focus on the problematic examples.

- 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:

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

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

Optimizing weak learner

- Remember: the weak classifiers $h_j(\mathbf{x})$ return a value in $\{-1, 1\}$, *not* in $\mathbb{R}$. Therefore, we can rewrite as follows:

yi

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

"this"

training error of Hm, weighted by Wi

- 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: errors

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

- 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: $W_i^{(0)} = 1/m$ hyperparameter
2. Iterate for $m = 1, \ldots, M$:
   - Find "weak" classifier h_m that attains weighted error ϵ_m
   - Let $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
   - Update the weights and normalize so that $\sum_i W_i^{(m)} = 1$:

y*h(x) will be -1 when classifier is wrong, alpha is larger than 0.

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

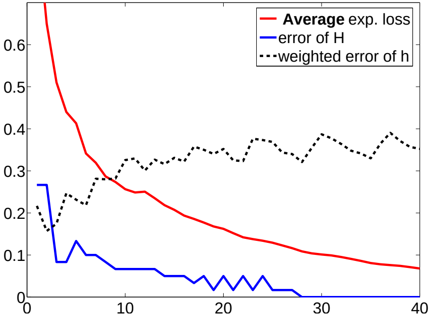
3. The combined classifier: $\text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x})\right)$

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

why weighted error go up:

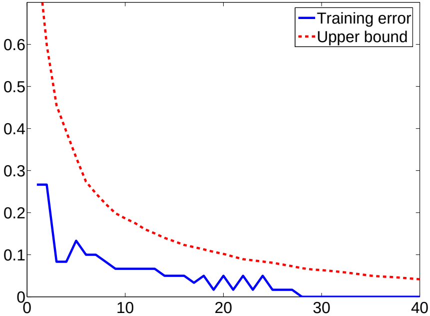
more weight on wrong classified data



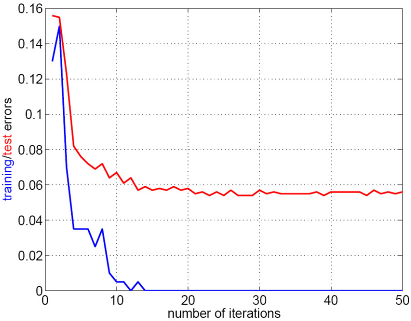
AdaBoost behavior: training error

- 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



- Common: test error continues to decrease after training error is zero
- Intuition: the **normalized margin**

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

of training examples continues to increase⇒more robust classifier

Variations of boosting

- Different surrogate loss functions; e.g., LogitBoost
- 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

Review: AdaBoost

1. Initialize weights: $W_i^{(0)} = 1/n$
2. Iterate for $m = 1, \ldots, M$:
   - Find "weak" classifier h_m that attains weighted error ϵ_m
   - Let $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$
   - Update and renormalize the weights:

$$W_i^{(m)} \propto W_i^{(m-1)} e^{-\alpha_m y_i h_m(\mathbf{x}_i)}$$

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

- Optimizes exponential loss on training data
- Regularization: (a) via early stopping, (b) via regularization of weak learners