ECE 5424: Introduction to Machine Learning

Topics:

- Ensemble Methods: Bagging, Boosting
- PAC Learning

Readings: Murphy 16.4; Hastie 16

Stefan Lee

Virginia Tech

Fighting the bias-variance tradeoff

- **Simple (a.k.a. weak) learners**
  - e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees)
  - **Good:** Low variance, don't usually overfit
  - **Bad:** High bias, can't solve hard learning problems

- **Sophisticated learners**
  - Kernel SVMs, Deep Neural Nets, Deep Decision Trees
  - **Good:** Low bias, have the potential to learn with Big Data
  - **Bad:** High variance, difficult to generalize

- Can we make combine these properties
  - **In general, No!!**
  - **But often yes…**

Ensemble Methods

Core Intuition: A combination of multiple classifiers will perform better than a single classifier.

Bagging

Boosting

Ensemble Methods

- Instead of learning a single predictor, learn **many predictors**
- **Output class:** (Weighted) combination of each predictor
- With sophisticated learners
  - Uncorrelated errors → expected error goes down
  - On average, do better than single classifier!
  - **Bagging**
- With weak learners
  - each one good at different parts of the input space
  - On average, do better than single classifier!
  - **Boosting**

Bagging

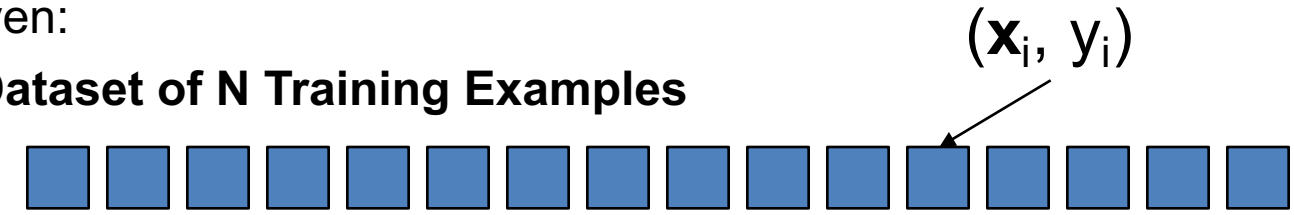
(**B**ootstrap **Ag**gregating / **B**ootstrap **A**veragin**g**)

Core Idea: Average multiple strong learners trained from resamples of your data to reduce variance and overfitting!

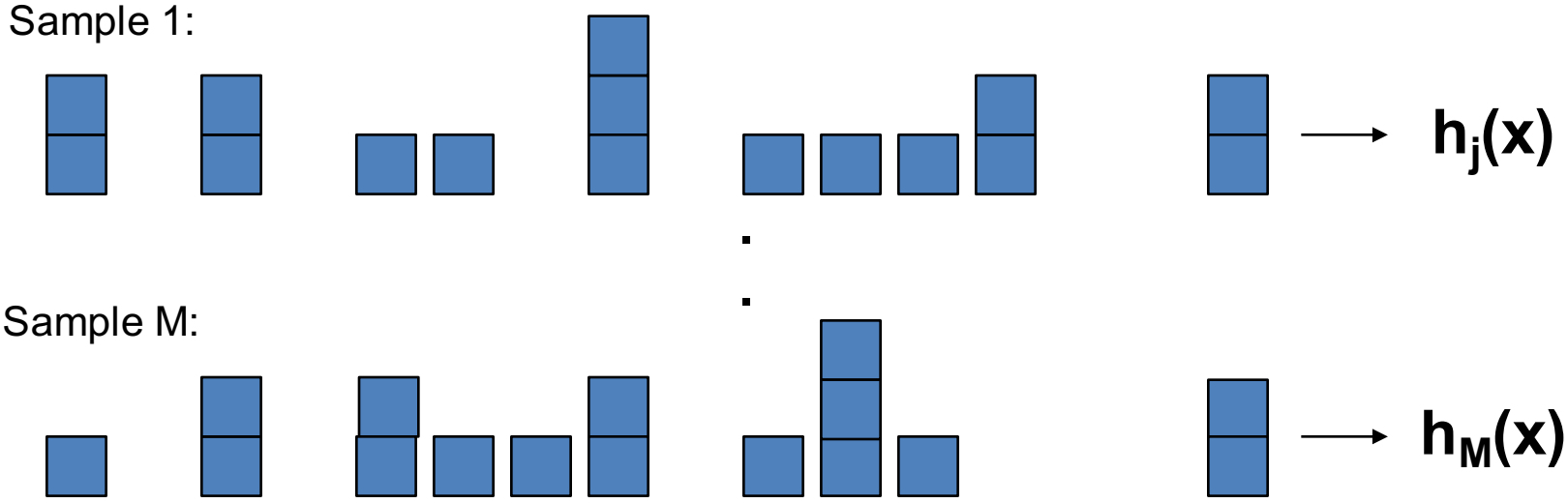
Bagging

Given:

Dataset of N Training Examples



Sample N training points **with replacement** and train a predictor, repeat M times:



At test time, output the (weighted) average output of these predictors.

Why Use Bagging

Let $\mathbf{e^m}$ be the error for the $\mathbf{m^{th}}$ predictor trained through bagging and e^{avg} be the error of the ensemble. If

$E[e^m] = 0$ (unbiased) and

$E[e^m e^k] = E[e^m]E[e^k]$ (uncorrelated) then..

$$E[e^{avg}] = \frac{1}{M}\frac{1}{M}\sum E[e^m]$$

The expected error of the average is a faction of the average expected error of the predictors!

When To Use Bagging

In practice, completely uncorrelated predictors don't really happen, but there also wont likely be perfect correlation either, so bagging may still help!

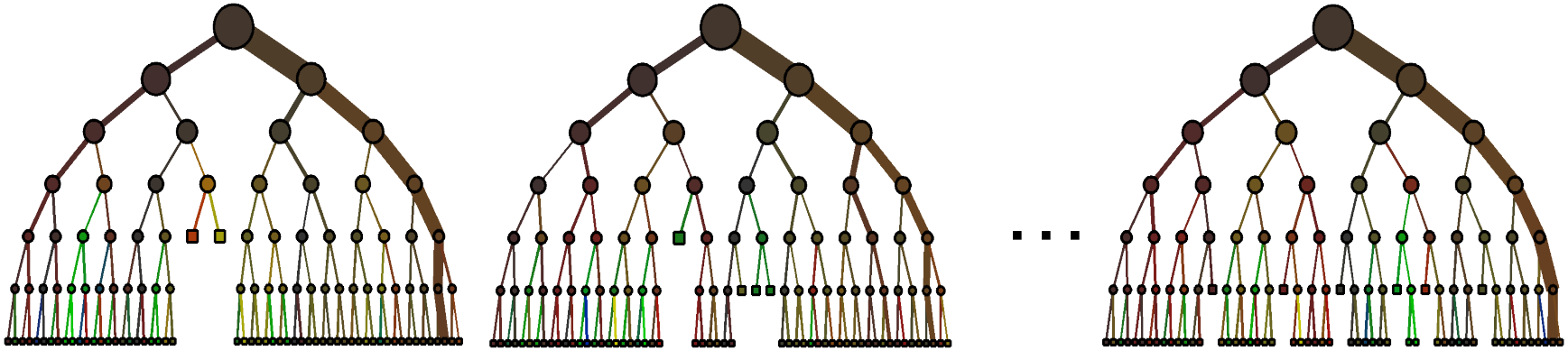
Use bagging when…

… you have overfit sophisticated learners (averaging lowers variance)

… you have a somewhat reasonably sized dataset

… you want an extra bit of performance from your models

Example: Decision Forests



We've seen that single decision trees can easily overfit!

- Train a M trees on different samples of the data and call it a forest.

Uncorrelated errors result in better ensemble performance. Can we force this?

- Could assign trees random max depths
- Could only give each tree a random subset of the splits
- Some work to optimize for no correlation as part of the object!

A Note From Statistics

Bagging is a general method to reduce/estimate the variance of an estimator.

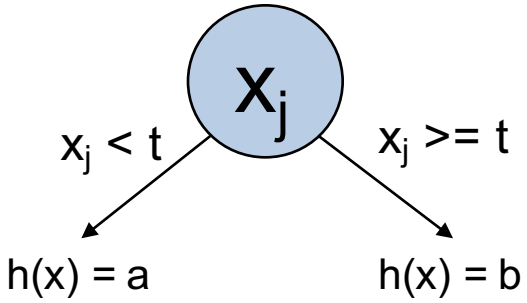
- Looking at the distribution of a estimator from multiple resamples of the data can give confidence intervals and bounds on that estimator.
- Typically just called Bootstrapping in this context.

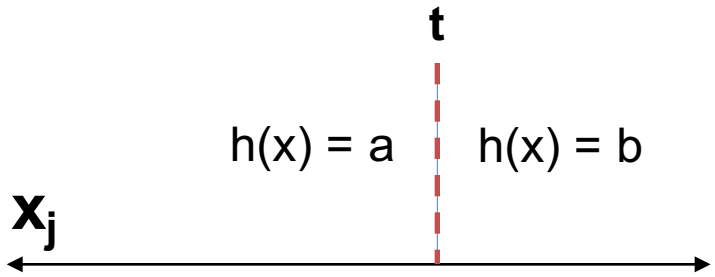
Boosting

Core Idea: Combine multiple weak learners to reduce error/bias by reweighting hard examples!

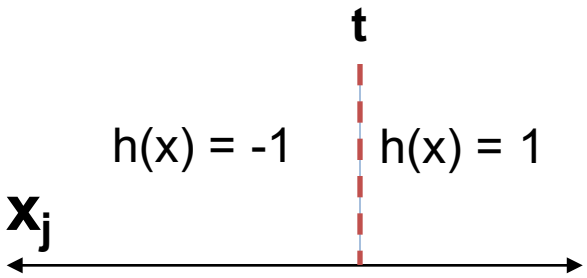
Some Intuition About Boosting

Consider a weak learner h(x), for example a decision stump:

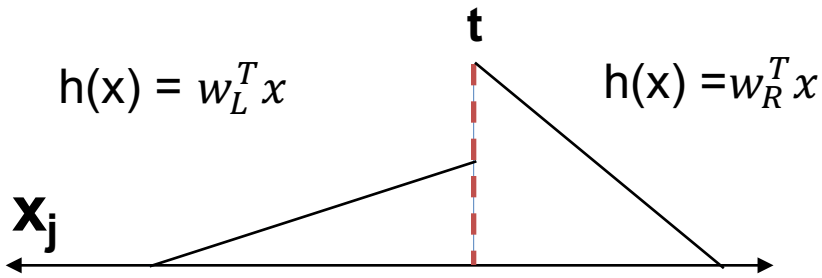




Example for binary classification:

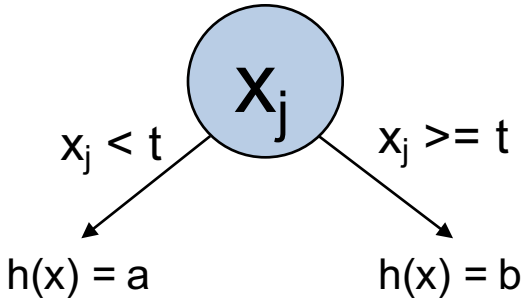


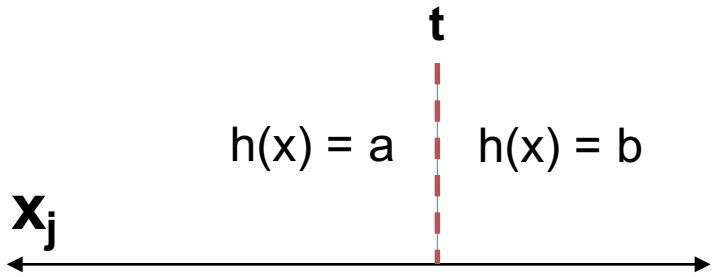
Example for regression:



Some Intuition About Boosting

Consider a weak learner h(x), for example a decision stump:





This learner will make mistakes often but what if we combine multiple to combat these errors such that our final predictor is:

$$f(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x) + \cdots + \alpha_{M-1} h_{m-1}(x) + \alpha_M h_M(x)$$

This is a big optimization problem now!!

$$\min_{\substack{\alpha_1,\ldots,\alpha_M \\ h_i \in \boldsymbol{H}}} \frac{1}{N} \sum_{\boldsymbol{i}} \boldsymbol{L}(y_i, \alpha_1 h_1(x) + \alpha_2 h_2(x) + \cdots + \alpha_{M-1} h_{m-1}(x) + \alpha_M h_M(x))$$

Boosting will do this greedily, training one classifier at a time to correct the errors of the existing ensemble

Boosting Algorithm [Schapire, 1989]

- Pick a class of weak learners $\mathcal{H} = \{h \mid h : X \to Y\}$

- You have a black box that picks best weak learning
  - unweighted sum $h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_i L\left(y_i, h(x_i)\right)$
  - weighted sum $h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_i w_i L\left(y_i, h(x_i)\right)$

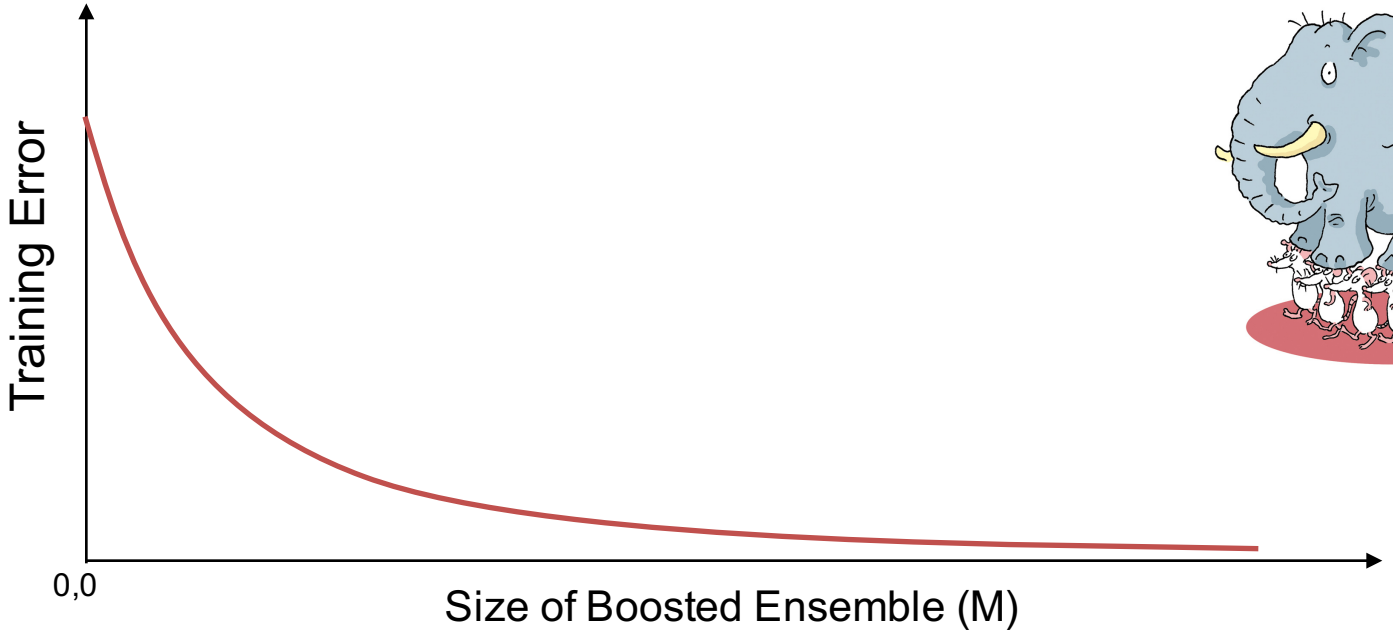
- On each iteration *t*
  - Compute error based on current ensemble $f_{t-1}(x_i) = \sum_{t'=1}^{t} \alpha_{t'} h_{t'}(x_i)$
  - Update weight of each training example based on it's error.
  - Learn a predictor h_t and strength for this predictor α_t

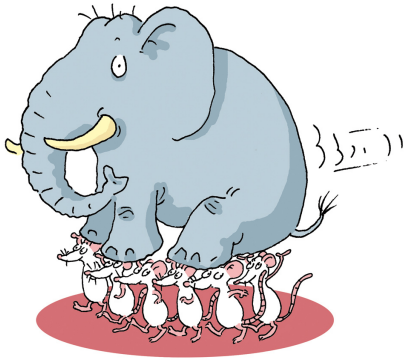
- Update ensemble: $f_t(x) = f_{t-1} + \alpha_t h_t(x)$

Boosting Demo

- Demo
  - Matlab demo by Antonio Torralba
  - http://people.csail.mit.edu/torralba/shortCourseRLOC/boosting/boosting.html

Boosting: Weak to Strong





As we add more boosted learners to our ensemble, error approaches zero (in the limit)

- need to decided when to stop based on a validation set
- don't use this on already overfit strong learners, will just become worse

Boosting Algorithm [Schapire, 1989]

- Pick a class of weak learners $\mathcal{H} = \{h \mid h : X \to Y\}$

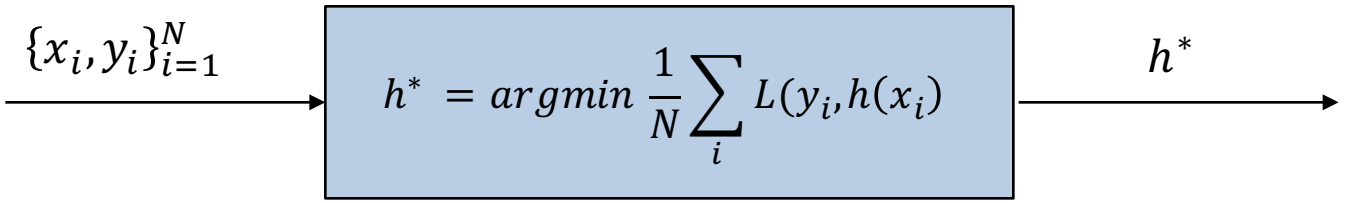
- You have a black box that picks best weak learning
  - unweighted sum $h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_i L\left(y_i, h(x_i)\right)$
  - weighted sum $h^* = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_i w_i L\left(y_i, h(x_i)\right)$

- On each iteration *t*
  - Compute error based on current ensemble $f_{t-1}(x_i) = \sum_{t'=1}^{t} \alpha_{t'} h_{t'}(x_i)$
  - **Update weight of each training example based on it's error.**
  - Learn a predictor h_t and strength for this predictor α_t

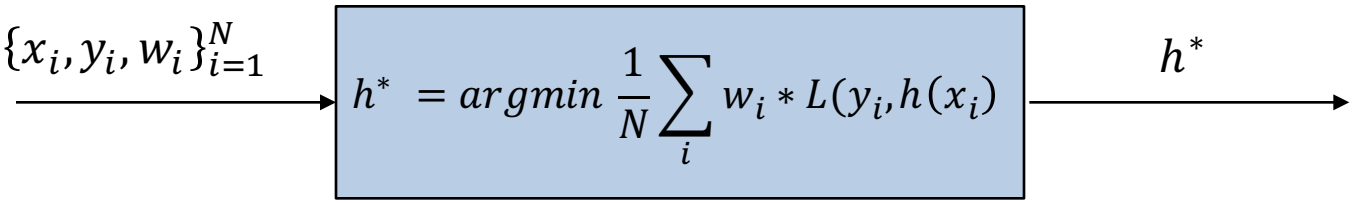
- Update ensemble: $f_t(x) = f_{t-1} + \alpha_t h_t(x)$

Boosting Algorithm [Schapire, 1989]

We've assumed we have some tools to find optimal learners, either



or



To train the t^{th} predictor, our job is to express the optimization for the new predictor in one of these forms

$$\min_{\substack{\alpha_1,\ldots,\alpha_M \\ h_i \in \boldsymbol{H}}} \frac{1}{N} \sum_{\boldsymbol{i}} \boldsymbol{L}(y_i, f_{t-1}(x) + \alpha_t h_t(x))$$

Typically done by either changing y_i or w_i depending on L.

Types of Boosting

Loss Name	Loss Formula	Boosting Name
Regression: Squared Loss	$(y - f(x))^2$	L2Boosting
Regression: Absolute Loss	$\lvert y - f(x) \rvert$	Gradient Boosting
Classification: Exponential Loss	$e^{-yf(x)}$	AdaBoost
Classification: Log/Logistic Loss	$\log\left(1 + e^{-yf(x)}\right)$	LogitBoost

L2 Boosting

Loss Name	Loss Formula	Boosting Name
Regression: Squared Loss	$(y - f(x))^2$	L2Boosting

- Algorithm
  - On Board

Adaboost

Loss Name	Loss Formula	Boosting Name
Classification: Exponential Loss	$e^{-yf(x)}$	AdaBoost

- Algorithm
  - You will derive in HW4!

What you should know

- Voting/Ensemble methods
- Bagging
  - How to sample
  - Under what conditions is error reduced
- Boosting
  - General algorithm
  - L2Boosting derivation
  - Adaboost derivation (from HW4)

Learning Theory

Probably Approximately Correct (PAC) Learning

What does it formally mean to learn?

Learning Theory

- We have explored **many** ways of learning from data

- But…
  - How good is our classifier, really?
  - How much data do I need to make it “good enough”?

A simple setting…

- Classification
  - N data points
  - **Finite** space H of possible hypothesis
    - e.g. decision trees on categorical variables of depth d

- A learner finds a hypothesis *h* that is **consistent** with training data
  - Gets zero error in training – $\text{error}_{\text{train}}(h) = 0$

- What is the probability that *h* has more than true error?
  - $P(\text{error}_{\text{true}}(h) \geq \epsilon)$

Generalization error in finite hypothesis spaces

[Haussler '88]

- ***Theorem***:
  - Hypothesis space *H* finite
  - dataset *D* with *N* i.i.d. samples
  - $0 < \epsilon < 1$

For any learned hypothesis *h* that is consistent (0 training error) on the training data:

$$P(\text{error}_{true}(h) > \epsilon) \leq |H| e^{-N\epsilon}$$

Even if *h* makes zero errors in training data, may make errors in test

Using a PAC bound

$$P(\text{error}_{true}(h) > \epsilon) \leq |H|e^{-N\epsilon}$$

- Let max acceptable $P(error_{true}(g) > \epsilon) = \delta$
- Typically, 2 use cases:
  - 1: Pick ϵ and δ , give you *N*
    - I want no more than ϵ error with probability δ, how much data?
  - 2: Pick N and δ , give you ϵ
    - I have N data points and want to know my error ϵ with δ confidence.

Haussler '88 bound

$$P(\text{error}_{true}(h) > \epsilon) \leq |H| e^{-N\epsilon}$$

- Strengths:
  - Holds for all (finite) H
  - Holds for all data distributions

- Weaknesses
  - Consistent classifier (0 training error)
  - Finite hypothesis space

Generalization bound for |H| hypothesis

- ***Theorem***:
  - Hypothesis space *H* finite
  - dataset *D* with *N* i.i.d. samples
  - $0 < \epsilon < 1$

For any learned hypothesis *h*:

$$P\left(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon\right) \leq |H| e^{-2N\epsilon^2}$$

PAC bound and Bias-Variance tradeoff

$$P\left(\text{error}_{true}(h) - \text{error}_{train}(h) > \epsilon\right) \leq |H| e^{-2N\epsilon^2}$$

or, after moving some terms around, with probability at least 1-δ :

$$\text{error}_{true}(h) \leq \text{error}_{train}(h) + \sqrt{\frac{\ln |H| + \ln \frac{1}{\delta}}{2N}}$$

Important: PAC bound holds for all *h*, but doesn't guarantee that algorithm finds best *h*!!!