

# Boosting

David Rosenberg

New York University

October 29, 2016

# Ensembles: Parallel vs Sequential

- Ensemble methods combine multiple models
- **Parallel ensembles:** each model is built independently
  - e.g. bagging and random forests
  - Main Idea: Combine many (high complexity, low bias) models to reduce variance
- **Sequential ensembles:**
  - Models are generated sequentially
  - Try to add new models that do well where previous models lack

# The Boosting Question: Weak Learners

- A **weak learner** is a classifier that does slightly better than random guessing.
- Weak learners are like “rules of thumb”:
  - If an email has “Viagra” in it, more likely than not it’s spam.
  - Email from a friend is probably not spam.
  - A linear decision boundary.
- Can we extract wisdom from a committee of fools?
- Can we **combine** a set of weak classifiers to form single classifier that makes accurate predictions?
  - Posed by Kearns and Valiant (1988,1989):
- Yes! **Boosting** solves this problem. [Rob Schapire (1990).]

# AdaBoost: Setting

- Consider  $\mathcal{Y} = \{-1, 1\}$  (binary classification).
- Suppose we have a **weak learner**:
  - Hypothesis space  $\mathcal{F} = \{f : \mathcal{X} \rightarrow \{-1, 1\}\}$ .
  - Algorithm for finding  $f \in \mathcal{F}$  that's better than random on training data.
- Typical weak learners:
  - **Decision stumps** (tree with a single split)
  - Trees with few terminal nodes
  - Linear decision functions

# Weighted Training Set

- Training set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ .
- Weights  $(w_1, \dots, w_n)$  associated with each example.
- **Weighted empirical risk:**

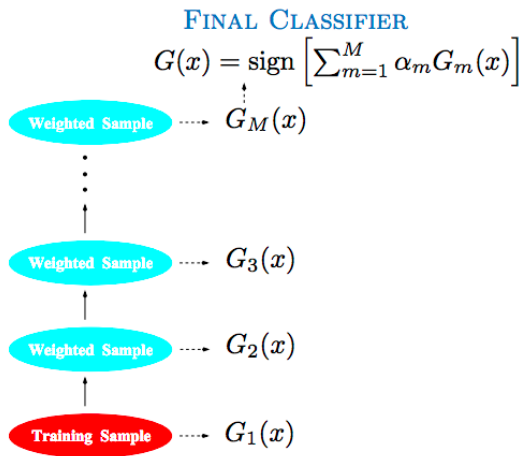
$$\hat{R}_n^w(f) = \frac{1}{W} \sum_{i=1}^n w_i \ell\{f(x_i), y_i\} \quad \text{where } W = \sum_{i=1}^n w_i$$

- Can train a model to minimize weighted empirical risk.
- What if model cannot conveniently be trained to reweighted data?
- Can sample a new data set from  $\mathcal{D}$  with probabilities  $(w_1/W, \dots, w_n/W)$ .

# AdaBoost - Rough Sketch

- Training set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ .
- Start with equal weight on all training points  $w_1 = \dots = w_n = 1$ .
- Repeat for  $m = 1, \dots, M$ :
  - Fit weak classifier  $G_m(x)$  to weighted training points
  - Increase weight on points  $G_m(x)$  misclassifies
- So far, we've generated  $M$  classifiers:  $G_1(x), \dots, G_M(x)$ .

## AdaBoost: Schematic




---

From ESL Figure 10.1

# AdaBoost - Rough Sketch

- Training set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ .
- Start with equal weight on all training points  $w_1 = \dots = w_n = 1$ .
- Repeat for  $m = 1, \dots, M$ :
  - Fit weak classifier  $G_m(x)$  to weighted training points
  - Increase weight on points  $G_m(x)$  misclassifies
- Final prediction  $G(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m G_m(x) \right]$ .
- The  $\alpha_m$ 's are nonnegative,
  - larger when  $G_m$  fits its weighted  $\mathcal{D}$  well
  - smaller when  $G_m$  fits weighted  $\mathcal{D}$  less well



# Adaboost: Weighted Classification Error

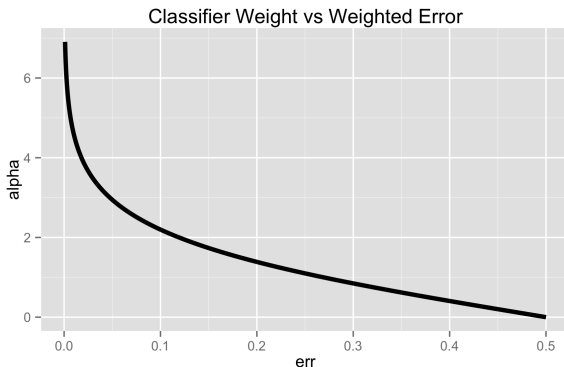
- In round  $m$ , weak learner gets a weighted training set.
  - Returns a classifier  $G_m(x)$  that roughly minimizes weighted 0–1 error.
- The **weighted 0-1 error** of  $G_m(x)$  is

$$\text{err}_m = \frac{1}{W} \sum_{i=1}^n w_i 1(y_i \neq G_m(x_i)) \quad \text{where } W = \sum_{i=1}^n w_i.$$

- Notice:  $\text{err}_m \in [0, 1]$ .
- We treat the weak learner as a black box.
  - It can use any method it wants to find  $G_m(x)$ . (e.g. SVM, tree, etc.)
  - BUT, for things to work, we need at least  $\text{err}_m < 0.5$ .

# AdaBoost: Classifier Weights

- The weight of classifier  $G_m(x)$  is  $\alpha_m = \ln \left( \frac{1 - \text{err}_m}{\text{err}_m} \right)$ .



- Note that weight  $\alpha_m \rightarrow 0$  as weighted error  $\text{err}_m \rightarrow 0.5$  (random guessing).

# AdaBoost: Example Reweighting

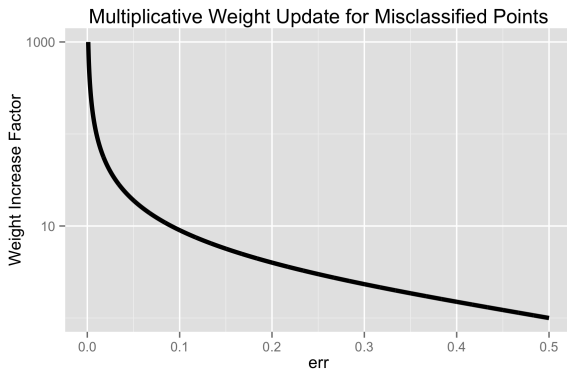
- We train  $G_m$  to minimize weighted error, and it achieves  $\text{err}_m$ .
- Then  $\alpha_m = \ln\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$  is the weight of  $G_m$  in final ensemble.
- Suppose  $w_i$  is weight of example  $i$  before training:
  - If  $G_m$  classifies  $x_i$  correctly, then  $w_i$  is unchanged.
  - Otherwise,  $w_i$  is increased as

$$\begin{aligned}w_i &\leftarrow w_i e^{\alpha_m} \\ &= w_i \left(\frac{1-\text{err}_m}{\text{err}_m}\right)\end{aligned}$$

- [Is it clear why this is always increasing the weight?]

# Adaboost: Example Reweighting

- Any misclassified point has weight adjusted as  $w_i \leftarrow w_i \left( \frac{1 - \text{err}_m}{\text{err}_m} \right)$ .



- The smaller  $\text{err}_m$ , the more we increase weight of misclassified points.

# AdaBoost: Algorithm

Given training set  $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ .

- ➊ Initialize observation weights  $w_i = 1/n$ ,  $i = 1, 2, \dots, n$ .
- ➋ For  $m = 1$  to  $M$ :
  - ➊ Fit weak classifier  $G_m(x)$  to  $\mathcal{D}$  using weights  $w_i$ .
  - ➋ Compute weighted empirical 0-1 risk:

$$\text{err}_m = \frac{1}{W} \sum_{i=1}^n w_i 1(y_i \neq G_m(x_i)) \quad \text{where } W = \sum_{i=1}^n w_i.$$

- ➌ Compute  $\alpha_m = \ln \left( \frac{1 - \text{err}_m}{\text{err}_m} \right)$ .
  - ➍ Set  $w_i \leftarrow w_i \cdot \exp[\alpha_m 1(y_i \neq G_m(x_i))]$ ,  $i = 1, 2, \dots, N$
- ➌ Output  $G(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m G_m(x) \right]$ .

# AdaBoost with Decision Stumps

- After 1 round:

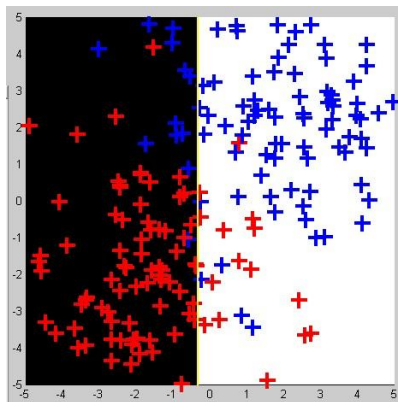


Figure: Plus size represents weight. Blackness represents score for red class.

# AdaBoost with Decision Stumps

- After 3 rounds:

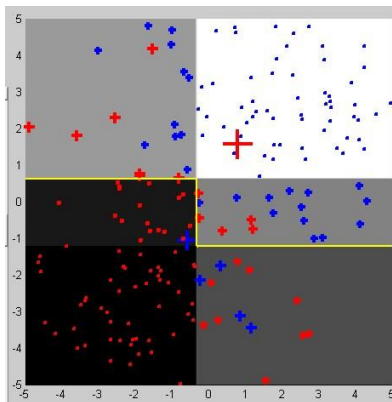


Figure: Plus size represents weight. Blackness represents score for red class.

# AdaBoost with Decision Stumps

- After 120 rounds:

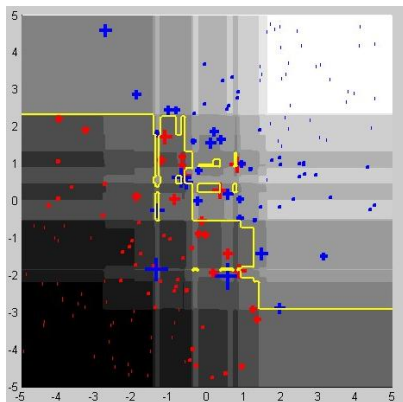


Figure: Plus size represents weight. Blackness represents score for red class.



# AdaBoost: Does it actually minimize training error?

- Methods we've seen so far come in two categories:
  - Convex optimization problems (L1/L2 regression, SVM, kernelized versions)
    - No issue minimizing objective function over hypothesis space
  - Trees
    - Can always fit data perfectly with big enough tree
- AdaBoost is something new - at this point, it's just an algorithm.
- Will  $G(x)$  even minimize training error?
- “Yes”, if our weak classifiers have an “edge” over random.

# AdaBoost: Does it actually minimize training error?

- As a weak classifier,  $G_m(x)$  should have  $\text{err}_m < \frac{1}{2}$ .
- Define the **edge** of classifier  $G_m(x)$  at round  $m$  to be

$$\gamma_m = \frac{1}{2} - \text{err}_m.$$

- Measures how much better than random  $G_m$  performs.

# AdaBoost: Does it actually minimize training error?

## Theorem

*The empirical 0-1 risk of the AdaBoost classifier  $G(x)$  is bounded as*

$$\frac{1}{n} \sum_{i=1}^n 1(y_i \neq G(x)) \leq \prod_{m=1}^M \sqrt{1 - 4\gamma_m^2}.$$

---

For more details, see the book *Boosting: Foundations and Algorithms* by Schapire and Freund.

# AdaBoost: Does it actually minimize training error?

## Example

Suppose  $\text{err}_m \leq 0.4$  for all  $m$ .

- Then  $\gamma_m = .5 - .4 = .1$ , and

$$\frac{1}{n} \sum_{i=1}^n 1(y_i \neq G(x)) \leq \prod_{m=1}^M \sqrt{1 - 4(.1)^2} \approx (.98)^M$$

- Bound decreases exponentially:

$$.98^{100} \approx .133$$

$$.98^{200} \approx .018$$

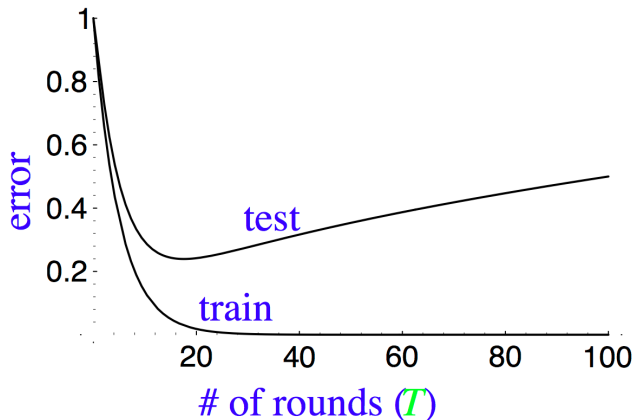
$$.98^{300} \approx .002$$

- With a consistent edge, training error decreases very quickly to 0.

For more details, see the book *Boosting: Foundations and Algorithms* by Schapire and Freund.

# Typical Train / Test Learning Curves

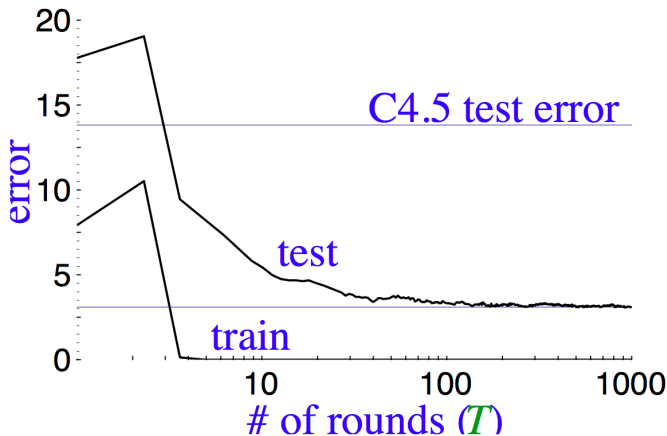
- Might expect too many rounds of boosting to overfit:



From Rob Schapire's NIPS 2007 Boosting tutorial.

# Learning Curves for AdaBoost

- In typical performance, AdaBoost is surprisingly resistant to overfitting.
- Test continues to improve even after training error is zero!



From Rob Schapire's NIPS 2007 Boosting tutorial.

# Adaptive Basis Function Model

- AdaBoost produces a classification score function of the form

$$\sum_{m=1}^M \alpha_m G_m(x)$$

- View this as an **adaptive basis function** model:
  - Linear in the basis functions.
  - But basis functions are learned from the data.

# Adaptive Basis Function Model

- Can write adaptive basis function expansion as

$$f(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m),$$

- where  $\beta_m$  are **expansion coefficients**
  - and  $b(x; \gamma_m)$  is a function of  $x$ , parameterized by  $\gamma_m$ .
- For example, each  $b(x; \gamma_m)$  could be a tree
  - $\gamma_m$  would characterize the splits and the terminal node predictions
- If the  $\gamma_m$ 's were known, this would just be a linear model.
- This type of model is also called an **additive** model.



# Fitting Adaptive Basis Function Model

- Would be nice directly minimize the empirical risk:

$$\min_{\{\beta_m, \gamma_m\}_{m=1}^M} \sum_{i=1}^n \ell \left( y_i, \sum_{m=1}^M \beta_m b(x_i; \gamma_m) \right).$$

- Prediction function more general than what we've seen before.
- Difficult to solve, in general.
- We'll discuss an approximate “greedy” solution, known as
  - **forward stagewise additive modeling**

# Forward Stagewise Additive Modeling

Sequentially add basis functions to the expansion, without adjusting the parameters or coefficients of the functions that have already been added.

- 1 Initialize  $f_0(x) = 0$ .
- 2 For  $m = 1$  to  $M$ :
  - 1 Compute:

$$(\beta_m, \gamma_m) = \arg \min_{\beta, \gamma} \sum_{i=1}^n \ell\{y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)\}.$$

- 2 Set  $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$ .
  - 3 Return:  $f_M(x)$ .
- Note: Actually implementing this minimization is difficult in general. More on this later.

# Exponential Loss and AdaBoost

- Take loss function to be

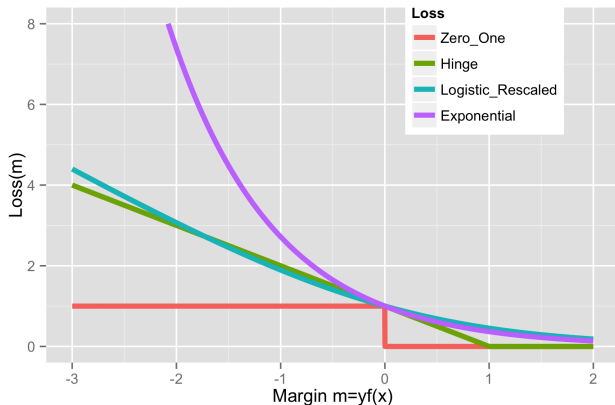
$$\ell(y, f(x)) = \exp(-yf(x)).$$

- Let  $\mathcal{F} = \{b(x; \gamma) \mid \gamma \in \Gamma\}$  be a hypothesis space of weak classifiers.
- Then Forward Stagewise Additive Modeling (FSAM) reduces to AdaBoost!
  - (See HTF Section 10.4 for proof.)
- Only difference:
  - AdaBoost is loose about each  $G_m$  “fitting the weighted training data”
  - For FSAM we’re explicitly looking for

$$G_m = \arg \min_{G \in \mathcal{F}} \sum_{i=1}^N w_i^{(m)} 1(y_i \neq G(x_i))$$

# Exponential Loss

- Note that exponential loss puts a very large weight on bad misclassifications.



# AdaBoost / Exponential Loss: Robustness Issues

- When Bayes error rate is high (e.g.  $\mathbb{P}(f^*(X) \neq Y) = 0.25$ )
  - Training examples with same input, but different classifications.
  - Best we can do is predict the most likely class for each  $X$ .
- Some training predictions **should be wrong** (because example doesn't have majority class)
  - AdaBoost / exponential loss puts a lot of focus on getting those right
- Empirically, AdaBoost has degraded performance in situations with
  - high Bayes error rate, or when there's
  - high “**label noise**”
- Logistic loss performs better with high Bayes error

# Population Minimizers

- In traditional statistics, the **population** refers to
  - the full population of a group, rather than a sample.
- In machine learning, the **population case** is the hypothetical case of
  - an infinite training sample from  $P_{\mathcal{X} \times \mathcal{Y}}$ .
- A **population minimizer** for a loss function is another name for the risk minimizer.
- For the exponential loss  $\ell(m) = e^{-m}$ , the population minimizer is given by

$$f^*(x) = \frac{1}{2} \ln \frac{\mathbb{P}(Y = 1 | X = x)}{\mathbb{P}(Y = -1 | X = x)}$$

- (Short proof in KPM 16.4.1)
- By solving for  $\mathbb{P}(Y = 1 | X = x)$ , we can give probabilistic predictions from AdaBoost as well.

# Population Minimizers

- AdaBoost has the robustness issue because of the exponential loss.
- Logistic loss  $\ell(m) = \ln(1 + e^{-m})$  has the same population minimizer.
  - But works better with high label noise or high Bayes error rate
- Population minimizer of SVM hinge loss is

$$f^*(x) = \text{sign} \left[ \mathbb{P}(Y = 1 \mid X = x) - \frac{1}{2} \right].$$

- Because of the sign, we cannot solve for the probabilities.