

Boosting

Machine Learning - 10601

Geoff Gordon, Miroslav Dudík

[partly based on slides of Rob Schapire and Carlos Guestrin]

<http://www.cs.cmu.edu/~ggordon/10601/>

November 9, 2009

Ensembles of trees

BAGGING and RANDOM FORESTS

- learn many **big trees**
- each tree aims to fit the **same target concept**
 - random training sets
 - randomized tree growth
- voting \approx averaging:
DECREASE in VARIANCE

BOOSTING

- learn many **small trees**
(**weak** classifiers)
- each tree 'specializes' to a **different part** of target concept
 - reweight training examples
 - higher weights where still errors
- voting increases expressivity:
DECREASE in BIAS

Boosting

- **boosting** = general method of converting rough rules of thumb (e.g., decision stumps) into highly accurate prediction rule

Boosting

- **boosting** = general method of converting rough rules of thumb (e.g., decision stumps) into highly accurate prediction rule
- **technically:**
 - assume given **“weak” learning algorithm** that can consistently find classifiers (**“rules of thumb”**) at least slightly better than random, say, **accuracy $\geq 55\%$** (in two-class setting)
 - given sufficient data, a boosting algorithm can provably construct **single classifier** with **very high accuracy**, say, **99%**

A Formal Description of Boosting

- given **training set** $(x_1, y_1), \dots, (x_m, y_m)$
- $y_i \in \{-1, +1\}$ correct label of instance $x_i \in X$
- for $t = 1, \dots, T$:
 - construct distribution D_t on $\{1, \dots, m\}$
 - find **weak classifier** ("rule of thumb")

$$h_t : X \rightarrow \{-1, +1\}$$

with small **error** ϵ_t on D_t :

$$\epsilon_t = \Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$$

- output **final classifier** H_{final}

AdaBoost

[Freund-Schapire 1995]

- **constructing** D_t :
 - $D_1(i) = 1/m$
 - given D_t and h_t :

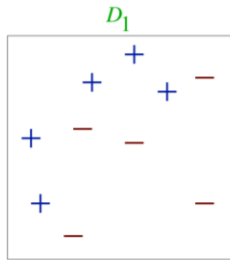
$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } y_i = h_t(x_i) \\ e^{\alpha_t} & \text{if } y_i \neq h_t(x_i) \end{cases} \\ &= \frac{D_t(i)}{Z_t} \exp(-\alpha_t y_i h_t(x_i)) \end{aligned}$$

where $Z_t =$ normalization constant

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right) > 0$$

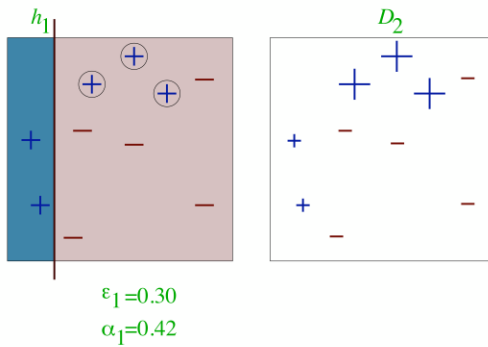
- **final classifier**:
 - $H_{\text{final}}(x) = \text{sign} \left(\sum_t \alpha_t h_t(x) \right)$

Toy example

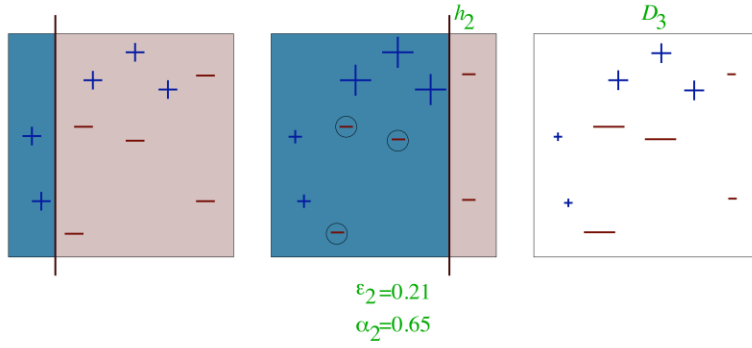


weak classifiers = decision stumps
 (vertical or horizontal half-planes)

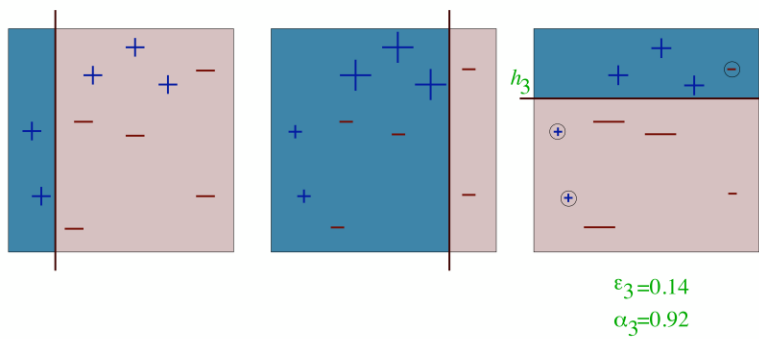
Round 1



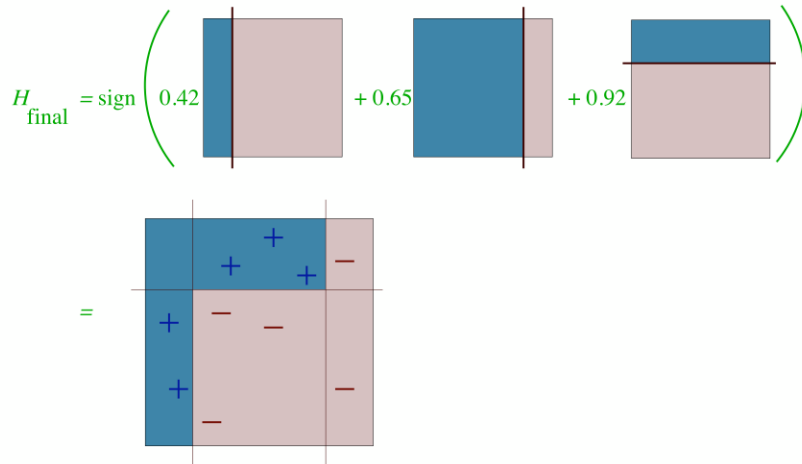
Round 2



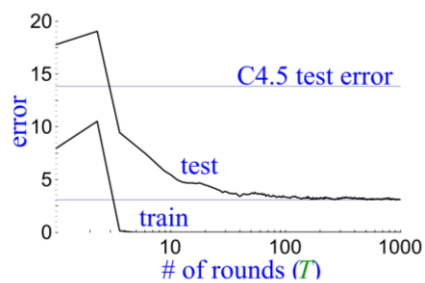
Round 3



Final classifier



A typical run of AdaBoost



(boosting C4.5 on "letter" dataset)

- **training error rapidly drops**
(combining weak learners increases expressivity)
- **test error does not increase** with number of trees T
(robustness to overfitting)

Bounding *training* error

Training error of final classifier is bounded by:

$$\text{err}_{\text{train}}(H) = \frac{1}{m} \sum_{i=1}^m \delta(H(x_i) \neq y_i) \leq \frac{1}{m} \sum_{i=1}^m \exp(-y_i f(x_i)) = \prod_{t=1}^T Z_t$$

where $f(x) = \sum_{t=1}^T \alpha_t h_t(x)$; $H(x) = \text{sign}(f(x))$.

Last step can be proved by unraveling the definition of $D_t(i)$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-y_i \alpha_t h_t(x_i))}{Z_t}$$

$$D_{T+1}(i) = \frac{1 \exp(-y_i f(x_i))}{m \prod_{t=1}^T Z_t}$$

Optimizing α_t

By minimizing $\prod_t Z_t$, we minimize the training error.

We can tighten this bound greedily, by choosing α_t on each round to minimize Z_t :

$$Z_t = \sum_{i=1}^m D_t(i) \exp(-y_i \alpha_t h_t(x_i))$$

Since $h_t(x_i) \in \{-1, 1\}$, we can find this explicitly:

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right)$$

Intuition: In each round, we adjust example weights, so that the accuracy of the last rule of thumb h_t drops to 50%.

Weak learners to Strong learners

Plugging the optimal α_t in the error bound:

$$\frac{1}{m} \sum_{i=1}^m \delta(H(x_i) \neq y_i) \leq \prod_{t=1}^T Z_t \leq \exp\left(-2 \sum_{t=1}^T (0.5 - \epsilon_t)^2\right)$$

Now, if each rule of thumb (at least slightly) better than random:

$$\epsilon_t \leq 0.5 - \gamma \text{ for } \gamma > 0$$

then training error drops exponentially fast:

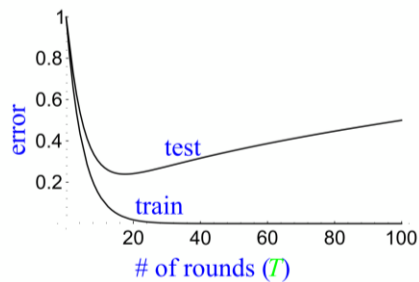
$$\frac{1}{m} \sum_{i=1}^m \delta(H(x_i) \neq y_i) \leq \prod_{t=1}^T Z_t \leq \exp\left(-2 \sum_{t=1}^T (0.5 - \epsilon_t)^2\right) \leq e^{-2T\gamma^2}$$

Bounding *true* error [Freund-Schapire 1997]

$$\text{err}_{\text{true}}(H) \leq \text{err}_{\text{train}}(H) + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right)$$

- **T** = number of rounds
- **d** = VC dimension of weak learner
- **m** = number of training examples

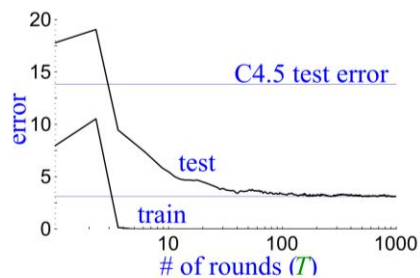
Bounding *true* error (a first guess)



expect:

- training error to continue to drop (or reach zero)
- test error to **increase** when H_{final} becomes “too complex”

A typical run **contradicts** a naïve bound



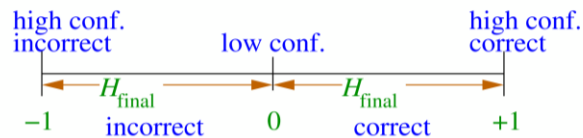
(boosting C4.5 on
“letter” dataset)

- test error does **not** increase, even after 1000 rounds
 - (total size > 2,000,000 nodes)
- test error continues to drop even after training error is zero!

Finer analysis: margins

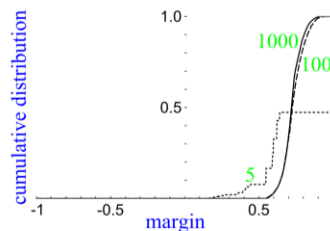
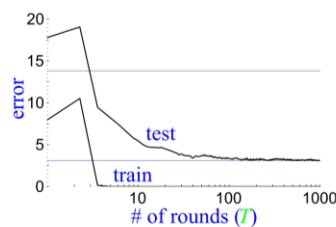
[Schapire et al. 1998]

- key idea:
 - training error only measures whether classifications are right or wrong
 - should also consider confidence of classifications
- recall: H_{final} is weighted majority vote of weak classifiers
- measure confidence by margin = strength of the vote
= (fraction voting correctly) – (fraction voting incorrectly)



Empirical evidence: margin distribution

- margin distribution
= cumulative distribution of margins of training examples



	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1
% margins ≤ 0.5	7.7	0.0	0.0
minimum margin	0.14	0.52	0.55

Theoretical evidence:

large margins \Rightarrow simple classifiers

- **Theorem:** large margins \Rightarrow better bound on generalization error (independent of number of rounds)
 - **proof idea:** if all margins are large, then can approximate final classifier by a much smaller classifier (just as polls can predict not-too-close election)
- **Theorem:** boosting tends to increase margins of training examples (given weak learning assumption)
 - **proof idea:** similar to training error proof
- so:
 - although final classifier is getting larger,
 - margins are likely to be increasing,
 - so final classifier actually getting close to a simpler classifier,
 - driving down the test error

More technically...

Previously

$$\text{err}_{\text{true}}(H) \leq \text{err}_{\text{train}}(H) + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right)$$

With high probability, for all $\theta > 0$

$$\text{err}_{\text{true}}(H) \leq \left(\frac{1}{m} \sum_{i=1}^m \delta(y_i f(x_i) < \theta)\right) + \tilde{O}\left(\frac{\sqrt{d/m}}{\theta}\right)$$

Bound depends on:

- **d** = VC dimension of weak learner
- **m** = number of training examples
- **entire distribution** of training margins

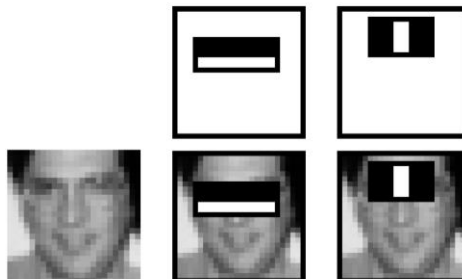
Practical advantages of AdaBoost

- fast
- simple and easy to program
- no parameters to tune (except T)
- flexible — can combine with any learning algorithm
- no prior knowledge needed about weak learner
- provably effective, provided can consistently find rough rules of thumb
 - shift in mind set — goal now is merely to find classifiers barely better than random guessing
- versatile
 - can use with data that is textual, numeric, discrete, etc.
 - has been extended to learning problems well beyond binary classification

Application: detecting faces

[Viola-Jones 2001]

- problem: find faces in photograph or movie
- weak classifiers: detect light/dark rectangles in image

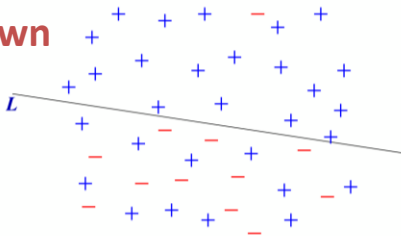


- many clever tricks to make extremely fast and accurate

Caveats

- performance of AdaBoost depends on **data** and **weak learner**
- consistent with theory, AdaBoost can **fail** if
 - weak classifiers too **complex**
 - overfitting
 - weak classifiers too **weak** ($\gamma_t \rightarrow 0$ too quickly)
 - underfitting
 - low margins → overfitting
- empirically, AdaBoost seems especially susceptible to uniform noise

**“Hard” predictions
can slow down
learning!**



- ideally, want weak classifier that says:

$$h(x) = \begin{cases} +1 & \text{if } x \text{ above } L \\ \text{“don't know”} & \text{else} \end{cases}$$

- **problem**: cannot express using “hard” predictions
- if must predict ± 1 below L , will introduce many “bad” predictions
 - need to “clean up” on later rounds
- dramatically increases time to convergence

Confidence-rated Predictions

[Schapire-Singer 1999]

- useful to allow weak classifiers to assign **confidences** to predictions
- formally, allow $h_t : X \rightarrow \mathbb{R}$

$$\begin{aligned} \text{sign}(h_t(x)) &= \text{prediction} \\ |h_t(x)| &= \text{"confidence"} \end{aligned}$$

- use identical update:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \cdot \exp(-\alpha_t y_i h_t(x_i))$$

and identical rule for combining weak classifiers

- **question:** how to choose α_t and h_t on each round

Confidence-rated Predictions

- saw earlier:

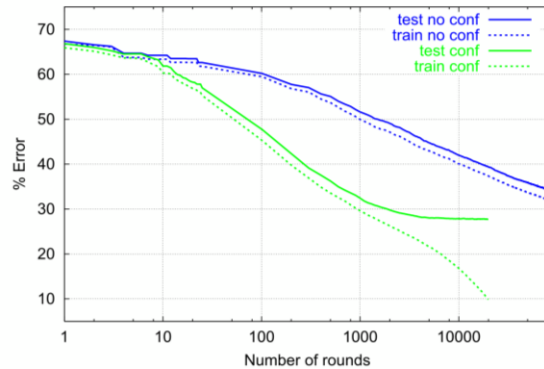
$$\text{training error}(H_{\text{final}}) \leq \prod_t Z_t = \frac{1}{m} \sum_i \exp\left(-y_i \sum_t \alpha_t h_t(x_i)\right)$$

- therefore, on each round t , should choose $\alpha_t h_t$ to minimize:

$$Z_t = \sum_i D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

- in many cases (e.g., decision stumps), best confidence-rated weak classifier has simple form that can be found efficiently

Confidence-rated predictions help a lot!



% error	round first reached		speedup
	conf.	no conf.	
40	268	16,938	63.2
35	598	65,292	109.2
30	1,888	>80,000	—

Loss in logistic regression

Logistic regression assumes:

$$P(Y = 1|X = x) = \frac{1}{1 + \exp(-f(x))}$$

And tries to maximize conditional likelihood:

$$P(y_1, \dots, y_m | x_1, \dots, x_m, H) = \prod_{i=1}^m \frac{1}{1 + \exp(-y_i f(x_i))}$$

Equivalent to minimizing log loss

$$\sum_{i=1}^m \ln(1 + \exp(-y_i f(x_i)))$$

Loss in AdaBoost

Logistic regression equivalent to minimizing log loss

$$\sum_{i=1}^m \ln(1 + \exp(-y_i f(x_i)))$$

Boosting minimizes similar loss function!

$$\frac{1}{m} \sum_{i=1}^m \exp(-y_i f(x_i)) = \prod_t Z_t$$

Logistic regression vs AdaBoost

Logistic regression:

- Minimize *log loss*

$$\sum_{i=1}^m \ln(1 + \exp(-y_i f(x_i)))$$

- Define

$$f(x) = \sum_j w_j x_j$$

where x_j 's predefined

- w_j 's optimized jointly

AdaBoost:

- Minimize *exponential loss*

$$\sum_{i=1}^m \exp(-y_i f(x_i))$$

- Define

$$f(x) = \sum_t \alpha_t h_t(x)$$

where h_t 's defined dynamically to fit the data

- α_t 's optimized one at a time

Benefits of model-fitting view

- immediate generalization to other loss functions
 - e.g. squared error for **regression**
 - e.g. logistic regression (by only changing one line of AdaBoost)
- sensible approach for converting output of boosting into **conditional probability estimates**
- **caveat**: wrong to view AdaBoost as **just** an algorithm for minimizing exponential loss
 - other algorithms for minimizing same loss will (provably) give very **poor** performance
 - thus, this loss function cannot explain why AdaBoost “works”

What you should know about boosting

- **weak classifiers** \Rightarrow **strong classifiers**
 - **weak**: slightly better than random on training data
 - **strong**: eventually zero error on training data
- AdaBoost prevents overfitting by increasing margins
- regimes when AdaBoost overfits
 - **weak learner too strong**: use small trees or stop early
 - **data noisy**: stop early
- **AdaBoost** vs **Logistic Regression**
 - **exponential loss** vs **log loss**
 - **single-coordinate updates** vs **full optimization**