Bagging and Boosting

Evgeny Burnaev

Skoltech, Moscow, Russia

Outline

Classification Problem I

- A predictor, feature $\mathbf{x} \in \mathbb{R}^p$ has distribution D
- $h(\mathbf{x})$ is a deterministic function from some concept class
- **Goal**:
  - Based on m training pairs $(\mathbf{x}_i, y_i = h(\mathbf{x}_i))$ drawn from D produce a classifier $\hat{h}(\mathbf{x}) \in \{0, 1\}$
  - Choose $\hat{h}$ to have low generalization error $R(\hat{h}) = \mathbb{E}_D \left[1_{\hat{h}(\mathbf{x}) \neq h(\mathbf{x})} \right]$

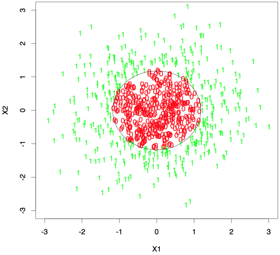
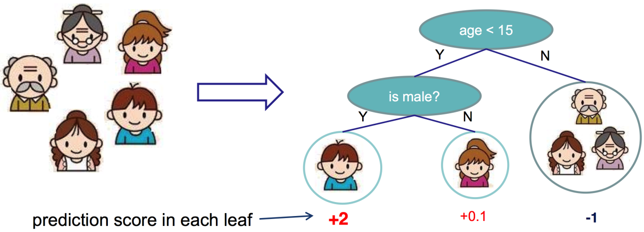


Figure : “Sphere” in $\mathbb{R}^{10}$

Classification and Regression Trees (CART) I

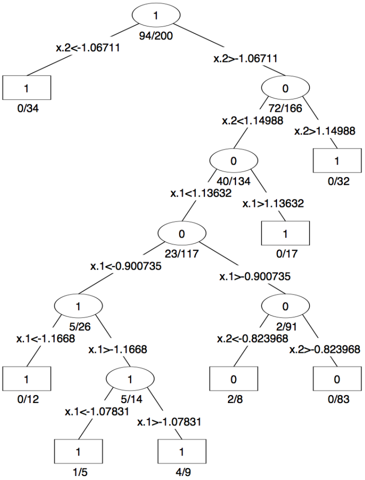
- Classification and Regression Trees:
  - Decision rules
  - Contains one score in each leaf value

Input: age, gender, occupation,... $\Rightarrow$ Does the person like computer games?



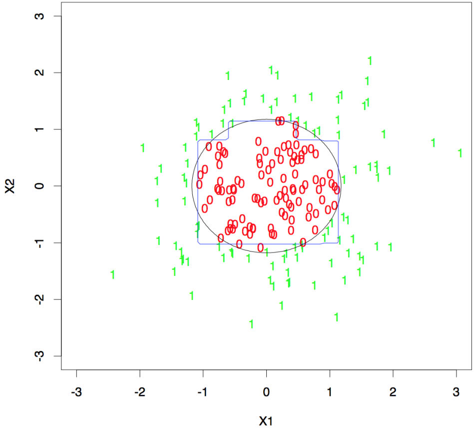
CART II

Sample of size 200



Classification Problem I

Sample of size 200



In case of "Sphere" in $\mathbb{R}^{10}$ CART produces a rather noisy and inaccurate rule $\hat{h}(\mathbf{x})$, with error rates around 30%

Model Averaging

Classification trees can be simple, but often produce noise (bushy) or weak (stunted) classifiers

- Bagging (Breiman, 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote

- Boosting (Freund & Shapire, 1996): Fit many large or small trees to reweighted versions of the training data. Classify by weighted majority vote

In general

$$\text{Boosting} \succ \text{Bagging} \succ \text{Single Tree}$$

Statistics: bootstrap

- Model:
  - we have i.i.d. sample $\{\mathbf{x}_i\}_{i=1}^m \subset \mathbb{R}^1$, generated by some distribution function F
  - We consider some statistics $T_m = g(\mathbf{x}_1, \ldots, \mathbf{x}_m)$.
- Problem: estimate variance $\mathbb{V}_F(T_n)$, which depends on some unknown distribution function F

Example

Let us consider $T_m = \overline{\mathbf{x}}_m$. Then $\mathbb{V}_F(T_m) = \sigma^2/m$, where $\sigma^2 = \int (\mathbf{x} - \mu)^2 dF(\mathbf{x})$ and $\mu = \int \mathbf{x} dF(\mathbf{x})$. Thus, the variance T_m is a function of F

Bootstrap Idea

Step 1. Estimate $\mathbb{V}_F(T_m)$ using $\mathbb{V}_{\hat{F}_m}(T_m)$, where

$$\hat{F}_m(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} 1(\mathbf{x}_i \leq \mathbf{x})$$

Step 2. Approximate $\mathbb{V}_{\hat{F}_m}(T_m)$ using Monte-Carlo sampling from $\hat{F}_m$

Example

For $T_m = \overline{\mathbf{x}}_m$, $\mathbb{V}_{\hat{F}_m}(T_m) = \hat{\sigma}^2/m$, where $\hat{\sigma}^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i - \overline{\mathbf{x}}_m)^2$. Thus Step 1 is sufficient in this case. However, often we can not provide an explicit expression for $\mathbb{V}_{\hat{F}_m}(T_m)$. Thus we can use Step 2

General Scheme

Step 1. In "Real" World

$$F \Rightarrow \mathbf{x}_1, \dots, \mathbf{x}_m \Rightarrow T_m = g(\mathbf{x}_1, \dots, \mathbf{x}_m)$$

Step 2. In "Bootstrap" World

$$\hat{F}_m \Rightarrow \{\mathbf{x}_1^*, \dots, \mathbf{x}_m^*\} \Rightarrow T_m^* = g(\mathbf{x}_1^*, \dots, \mathbf{x}_m^*)$$

- **Problem**: how to generate $\mathbf{x}_1^*, \dots, \mathbf{x}_m^*$ from $\hat{F}_m$?
- **Solution**: $\hat{F}_m$ has a mass $\frac{1}{m}$ in each of sample point $\mathbf{x}_i$, $i = 1, \dots, m \Rightarrow$ generating from $\hat{F}_m$ is equivalent to selection with replacement from the initial sample $\{\mathbf{x}_i\}_{i=1}^m$

Bootstrap Variance Estimation

In order to estimate variance of a functional using bootstrap:

1. Select $\mathbf{x}_1^*, \dots, \mathbf{x}_m^* \sim \hat{F}_m$
2. Calculate $T_m^* = g(\mathbf{x}_1^*, \dots, \mathbf{x}_m^*)$
3. Repeat steps 1 and 2 until you get $T_m^{*,1}, \dots, T_m^{*,B}$
4. Set

$$v_{boot} = \frac{1}{B} \sum_{b=1}^{B} \left(T_m^{*,b} - \frac{1}{B} \sum_{r=1}^{B} T_m^{*,r} \right)^2$$

Thus we get that

$$\mathbb{V}_F(T_m) \approx \mathbb{V}_{\hat{F}}(T_m) \approx v_{boot}$$

Bagging

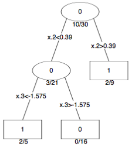
- Bagging or bootstrap averaging averages a given procedure over many samples to reduce its variance
- Let us denote by
  - $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ a sample of size m
  - $\hat{h}_S(\mathbf{x})$ a classifier, such as a tree, trained using the sample S
- To bag $\hat{h}$ we draw bootstrap samples $S^{*,1}, \ldots, S^{*,B}$ each of size m with replacement from the training data
- Then

$$\hat{h}_{\text{bag}}(\mathbf{x}) = \text{MajorityVote}\left\{\hat{h}_{S^{*,b}}(\mathbf{x})\right\}_{b=1}^{B}$$

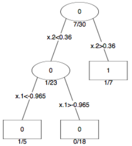
- Bagging can dramatically reduce the variance of unstable procedures (like trees), leading to improved prediction
- However any simple structure in h (e.g. a tree) is lost

Example: Bagging

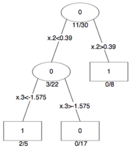
Original Tree



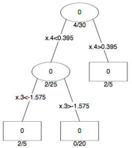
Bootstrap Tree 1



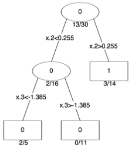
Bootstrap Tree 2



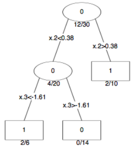
Bootstrap Tree 3



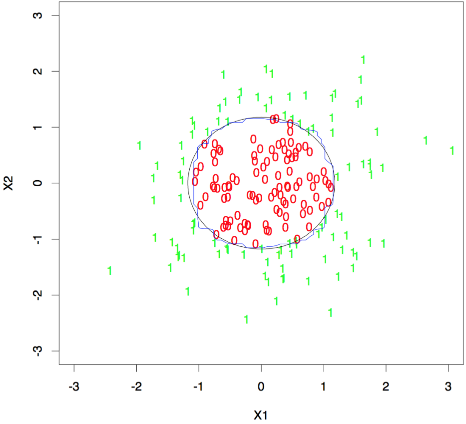
Bootstrap Tree 4



Bootstrap Tree 5



Decision Boundary: Bagging



"Sphere" in $\mathbb{R}^{10}$: Bagging averages many trees, and produces smoother decision boundaries

Example: Spam Filtering

- **problem**: filter out spam (junk email)
- gather large collection of examples of spam and non-spam

From: yoav@att.com	Rob, can you review a paper...	non-spam
From: xa412@hotmail.com	Earn money without working!!!! ...	spam
⋮	⋮	⋮

- **goal**: get computer learn from examples to distinguish spam from non-spam
- **main observation**:
  - — easy to find "rules of thumb" that are "often" correct

    if 'v1agr@' occurs in message, then predict "spam"

  - — hard to find single rule that is very highly accurate

The Boosting Approach I

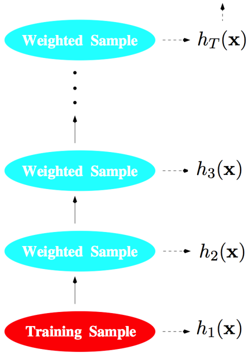
- devise computer program for deriving rough rules of thumb
- apply procedure to subset of emails
- obtain rule of thumb
- apply to 2nd subset of emails
- obtain 2nd rule of thumb
- repeat T times
- aggregate

The Boosting Approach II

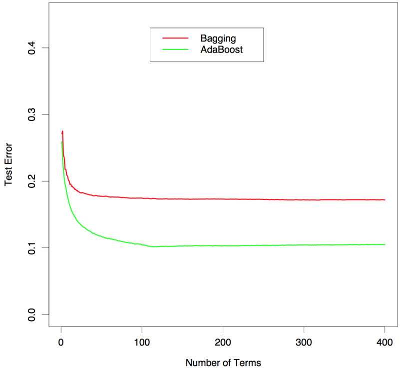
1. How to choose examples on each round?
    - concentrate on "hardest" examples (those most often misclassified by previous rules of thumb)
2. How to combine rules of thumb into single prediction rule?
    - take (weighted) majority vote of rules of thumb

Final Classifier

$$h(\mathbf{x}) = \mathrm{sign}\left[\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right]$$



Bagging and Boosting



- 2000 points, "Sphere" in $\mathbb{R}^{10}$; Bayes error rate is 0%
- Trees are grown Best First without pruning
- Leftmost iteration is a single tree

PAC Learning Model: Notations

- X: set of all possible instances or examples, e.g. the set of all men and women characterized by their hight and weight
- $c : X \to \{0, 1\}$: the target concept to learn; can be identified with its support $\{\mathbf{x} \in X : c(\mathbf{x}) = 1\}$
- C: concept class, a set of target concepts c
- D: target distribution, a fixed probability distribution over X. Training and test examples are drawn according to D

PAC Learning Model: Notations

- S: training sample
- H: set of concept hypothesis, e.g. the set of all linear classifiers
- The learning algorithm receives sample S and selects a hypothesis h_S from H approximating c

PAC Learning Model: Errors

- **True error or generalization error** of h with respect to the target concept c and distribution D

$$R(h) = \mathbb{P}_{\mathbf{x} \sim D} \left[h(\mathbf{x}) \neq c(\mathbf{x}) \right] = \mathbb{E}_{\mathbf{x} \sim D} \left[1_{h(\mathbf{x}) \neq c(\mathbf{x})} \right]$$

- **Empirical error**: average error of h on the training sample S drawn according to distribution D

$$\begin{aligned} \hat{R}_S(h) &= \mathbb{P}_{\mathbf{x} \sim \hat{D}} \left[h(\mathbf{x}) \neq c(\mathbf{x}) \right] \\ &= \mathbb{E}_{\mathbf{x} \sim \hat{D}} \left[1_{h(\mathbf{x}) \neq c(\mathbf{x})} \right] = \frac{1}{m} \sum_{i=1}^{m} 1_{h(\mathbf{x}_i) \neq c(\mathbf{x}_i)} \end{aligned}$$

- Note:

$$R(h) = \mathbb{E}_{S \sim D^m} \left[\hat{R}_S(h) \right]$$

PAC Learning Model: Definition

- **PAC Learning**: Probably Approximately Correct Learning (Valiant, 1984)
- **Definition**: concept class C is PAC-learnable if there exists a learning algorithm L such that
  — for all $c \in C$, $\epsilon > 0$, $\delta > 0$, and all distributions D,

$$\mathbb{P}_{S \sim D^m}\left[R(h_S) \leq \epsilon\right] \geq 1 - \delta,$$

  — for samples S of size $m =$ poly$(1/\epsilon, 1/\delta)$ for a fixed polynomial
- Such L is called a strong Learner

PAC Learning Model: comments

- Concept class C is known to the algorithm
- Distribution-free model: no assumption on D
- Both training and test examples drawn $\sim D$
- Probably: confidence $1 - \delta$
- Approximately correct: accuracy $1 - \epsilon$
- Efficient PAC-Learning: L runs in time poly$(1/\epsilon, 1/\delta, N, \mathrm{size}(c))$

Weak Learning

- **Definition** : concept class C is weakly PAC-learnable if there exists a (weak) learning algorithm L and $\gamma > 0$ such that:
  - for all $c \in C$ and $\delta > 0$, and all distributions D,

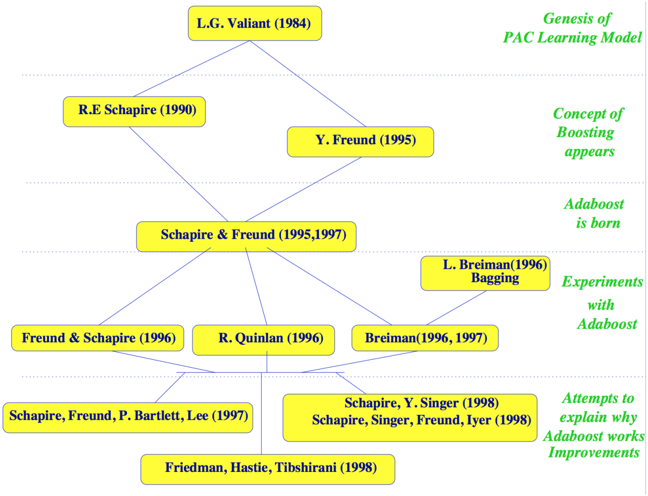
$$\mathbb{P}_{S \sim D^m}\left[R(h_S) \le \frac{1}{2} - \gamma\right] \ge 1 - \delta,$$

  - for samples S of size $m = \text{poly}(1/\delta)$ for a fixed polynomial

The Boosting Approach III

- Finding simple relatively accurate base classifiers often not hard $\leftarrow$ weak learner
- Main ideas:
  - — use weak learner to create a strong learner
  - — combine base classifiers returned by weak learner (ensemble method)
- But how should the base classifiers be combined?

History of Boosting



Boosting a Weak Learner

- Weak learner L produces an h with error rate $\beta = (\frac{1}{2} - \gamma) < \frac{1}{2}$ with $\Pr \geq (1 - \delta)$ for any D
- L has access to continuous stream of training data a class oracle
  - — L learns h_1 on first m training points
  - — L randomly filters the next batch of training points, extracting $m/2$ points correctly classified by h_1, $m/2$ incorrectly classified, and produces h_2
  - — L builds a third training set of m points for which h_1 and h_2 disagree, and produces h_3
  - — L outputs

$$h = \mathrm{MajorityVote}(h_1, h_2, h_3)$$

- **Theorem** (Schapire, 1990): "The Strength of Weak Learnability"

$$R(h) \leq 3\beta^2 - 2\beta^3 < \beta$$

AdaBoost

$H \subseteq \{-1, +1\}^X$

$\text{AdaBoost}(S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\})$

for $i \leftarrow 1$ **to** m **do**

    $D_1(i) \leftarrow \frac{1}{m}$

for $t \leftarrow 1$ **to** T **do**

    $h_t \leftarrow$ base classif. with small $\epsilon_t = \Pr_{i \sim D_t}\left[h_t(\mathbf{x}_i) \neq y_i\right]$

    $\alpha_t \leftarrow \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$

    $Z_t \leftarrow 2\left[\epsilon_t(1-\epsilon_t)\right]^{\frac{1}{2}}$ (normalization factor)

    for $i \leftarrow 1$ **to** m **do**

        $D_{t+1}(i) \leftarrow \frac{D_t(i) \exp(-\alpha_t y_t h_t(\mathbf{x}_i))}{Z_t}$

    $f_t \leftarrow \sum_{s=1}^{t} \alpha_s h_s$

return $h = \operatorname{sign}(f_T)$

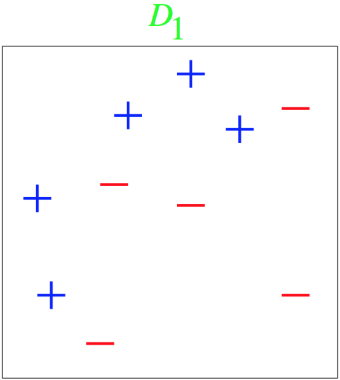
AdaBoost: Comments

- Distribution D_t over training sample:
  - — originally uniform
  - — at each round, the weight of a misclassified example is increased
  - — observation: $D_{t+1}(i) = \frac{e^{-y_i f_t(\mathbf{x}_i)}}{m \prod_{s=1}^{t} Z_s}$, since

$$
\begin{aligned}
D_{t+1}(i) &= \frac{D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{Z_t} \\
&= \frac{D_{t-1}(i) e^{-\alpha_{t-1} y_i h_{t-1}(\mathbf{x}_i)} e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{Z_{t-1} Z_t} \\
&= \frac{1}{m} \frac{e^{-y_i \sum_{s=1}^{t} \alpha_s h_s(\mathbf{x}_i)}}{\prod_{s=1}^{t} Z_s}
\end{aligned}
$$

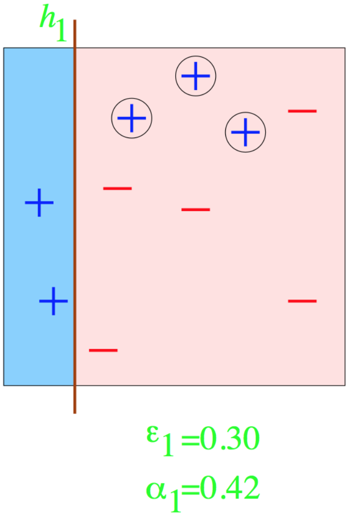
- Weight assigned to base classifier h_t: α_t directly depends on the accuracy of h_t at round t

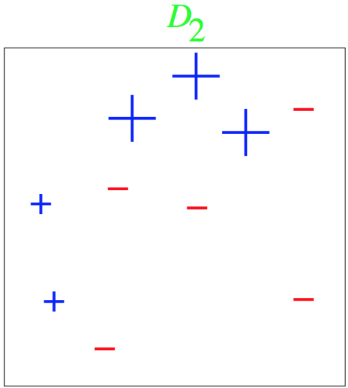
Toy Example



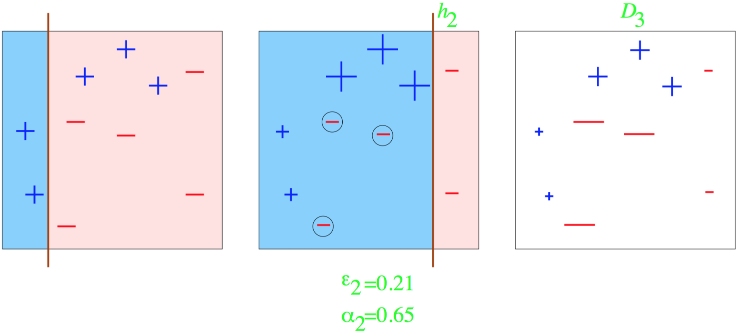
Weak classifiers = vertical or horizontal half-planes

Toy Example: Round 1

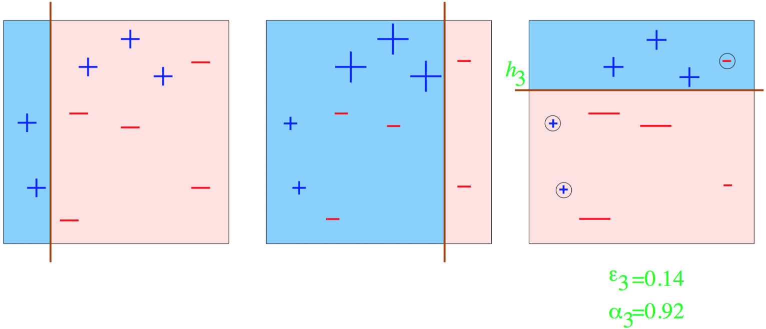




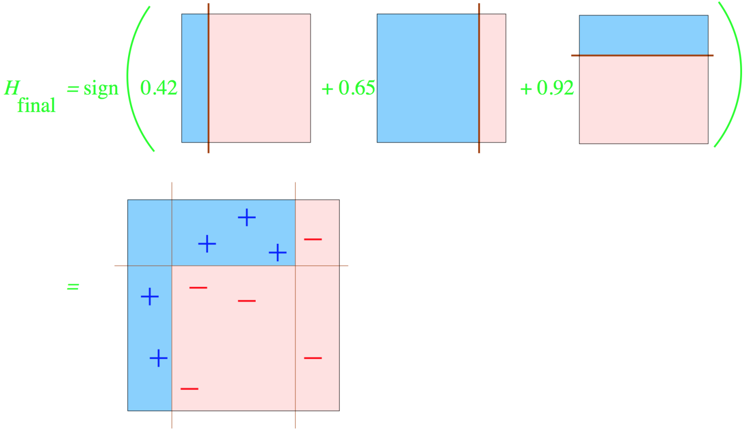
Toy Example: Round 2



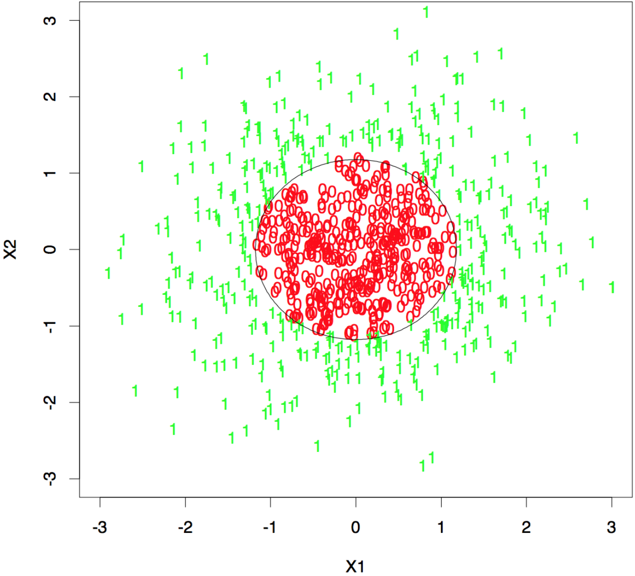
Toy Example: Round 3



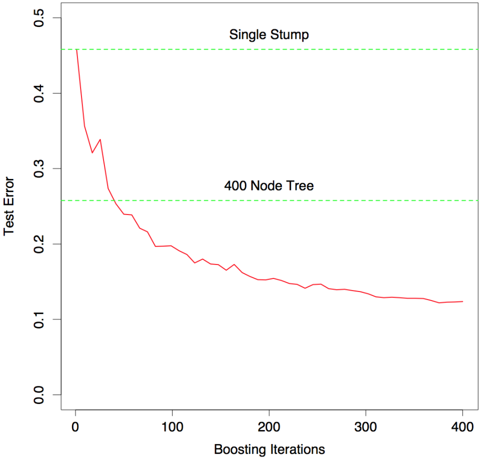
Toy Example: Final Classifier



Example: "Sphere" in $\mathbb{R}^{10}$

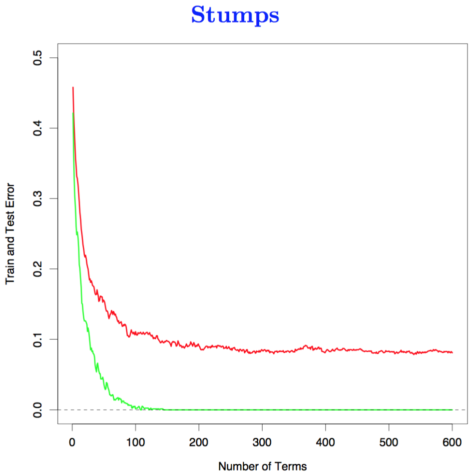


Boosting Stumps



"Sphere" in $\mathbb{R}^{10}$: A stump is a two-node tree, after a single split. Boosting stumps works remarkably well on this problem

Training & Test Error



"Sphere" in $\mathbb{R}^{10}$: Boosting drives the training error to zero. Further iterations continue to improve test error in many examples

Boosting Noisy Problems I

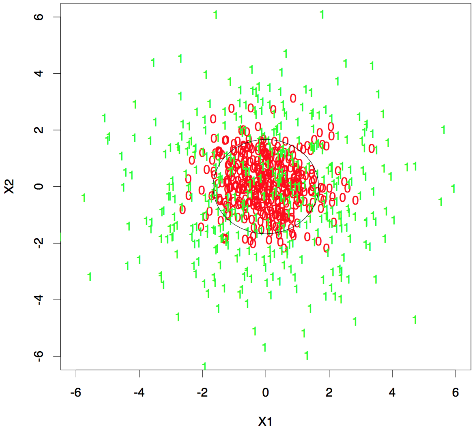
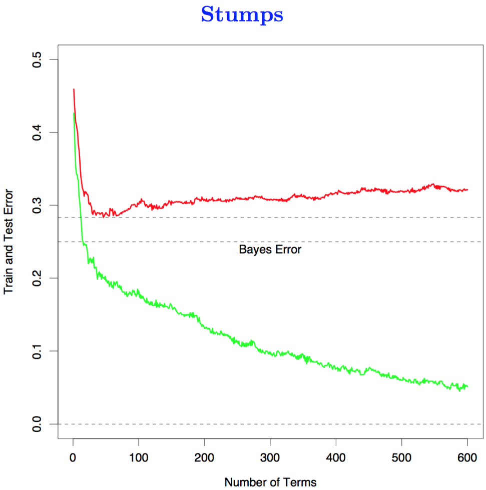


Figure : "Gaussians" in $\mathbb{R}^{10}$. Bayes error is 25%

Boosting Noisy Problems II



"Gaussians" in $\mathbb{R}^{10}$. Bayes error is 25%. Here the test error does increase, but quite slowly

Bound on Empirical Error

- **Theorem**: The empirical error of the classifier output by AdaBoost verifies:

$$\hat{R}(h) \leq \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2}-\epsilon_t\right)^2\right]$$

  — if further for all $t \in [1, T]$, $\gamma \leq \left(\frac{1}{2}-\epsilon_t\right)$, then

$$\hat{R}(h) \leq \exp(-2\gamma^2 \cdot T)$$

  — $\gamma > 0$ does not need to be known in advance: adaptive boosting

- **Proof**: Since, as we saw, $D_{t+1}(i) = \frac{e^{-y_i f_t(\mathbf{x}_i)}}{m \prod_{s=1}^{t} Z_s}$,

$$\hat{R}(h) = \frac{1}{m}\sum_{i=1}^{m} 1_{y_i f_T(\mathbf{x}_i) \leq 0} \leq \frac{1}{m}\sum_{i=1}^{m} \exp\left(-y_i f_T(\mathbf{x}_i)\right)$$
$$\leq \frac{1}{m}\sum_{i=1}^{m}\left[m\prod_{t=1}^{T} Z_t\right] D_{T+1}(i) = \prod_{t=1}^{T} Z_t$$

- Now, since Z_t is a normalization factor,

$$Z_t = \sum_{i=1}^{m} D_t(i) e^{-\alpha_t y_i h_t(\mathbf{x}_i)}$$
$$= \sum_{i:\, y_i h_t(\mathbf{x}_i) \geq 0} D_t(i) e^{-\alpha_t} + \sum_{i:\, y_i h_t(\mathbf{x}_i) < 0} D_t(i) e^{+\alpha_t}$$
$$= (1-\epsilon_t)e^{-\alpha_t} + \epsilon_t e^{\alpha_t}$$
$$= (1-\epsilon_t)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

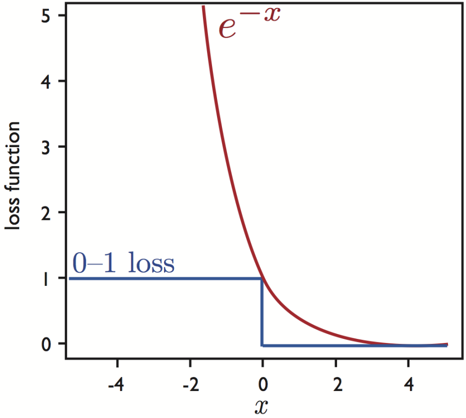
- Thus

$$\prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} = \prod_{t=1}^{T} \sqrt{1-4\left(\frac{1}{2}-\epsilon_t\right)^2}$$
$$\leq \prod_{t=1}^{T} \exp\left[-2\left(\frac{1}{2}-\epsilon_t\right)^2\right] = \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2}-\epsilon_t\right)^2\right]$$

- Comments:
  - α_t is a minimizer of $\alpha \to (1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}$
  - since $(1-\epsilon_t)e^{-\alpha_t} = \epsilon_t e^{\alpha_t}$, at each round Ada Boost assigns the same probability mass to correctly classified and misclassified instances

AdaBoost = Coordinate Descent

- Objective Function: convex and differentiable



- **Direction** unit vector $\mathbf{e}_k$ with best directional derivative

$$F'(\overline{\alpha}_{t-1}, \mathbf{e}_k) = \lim_{\eta \to 0} \frac{F(\overline{\alpha}_{t-1} + \eta \mathbf{e}_k) - F(\overline{\alpha}_{t-1})}{\eta}$$

- Since $F(\overline{\alpha}_{t-1} + \eta \mathbf{e}_k) = \sum_{i=1}^{m} e^{-y_i \sum_{j=1}^{K} \overline{\alpha}_{t-1,j} h_j(\mathbf{x}_i) - \eta y_i h_k(\mathbf{x}_i)}$,

$$\begin{aligned}
F'(\overline{\alpha}_{t-1}, \mathbf{e}_k) &= -\frac{1}{m} \sum_{i=1}^{m} y_i h_k(\mathbf{x}_i) e^{-y_i \sum_{j=1}^{K} \overline{\alpha}_{t-1,j} h_j(\mathbf{x}_i)} \\
&= -\frac{1}{m} \sum_{i=1}^{m} y_i h_k(\mathbf{x}_i) \overline{D}_t(i) \overline{Z}_t \\
&= -\left[\sum_{i=1}^{m} \overline{D}_t(i) 1_{y_i h_k(\mathbf{x}_i) = +1} - \sum_{i=1}^{m} \overline{D}_t(i) 1_{y_i h_k(\mathbf{x}_i) = -1} \right] \frac{\overline{Z}_t}{m} \\
&= -\left[(1 - \overline{\epsilon}_{t,k}) - \overline{\epsilon}_{t,k} \right] \frac{\overline{Z}_t}{m} = \left[2\overline{\epsilon}_{t,k} - 1 \right] \frac{\overline{Z}_t}{m}
\end{aligned}$$

Here $[2\overline{\epsilon}_{t,k} - 1]$ is a direction corresponding to the base classifier with the smallest error

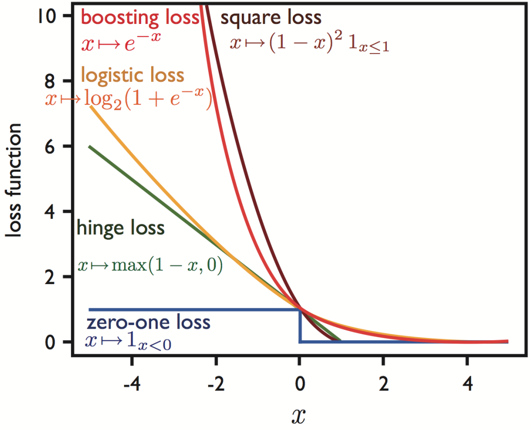
- **Step size**: η is chosen to minimize $F(\overline{\alpha}_{t-1} + \eta \mathbf{e}_k)$

$$
\begin{aligned}
\frac{dF(\overline{\alpha}_{t-1} + \eta \mathbf{e}_k)}{d\eta} = 0 &\Leftrightarrow -\sum_{i=1}^{m} y_i h_k(\mathbf{x}_i) e^{-y_i \sum_{j=1}^{K} \overline{\alpha}_{t-1,j} h_j(\mathbf{x}_i)} e^{-\eta y_i h_k(\mathbf{x}_i)} \\
&\Leftrightarrow -\sum_{i=1}^{m} y_i h_k(\mathbf{x}_i) \overline{D}_t(i) \overline{Z}_t e^{-\eta y_i h_k(\mathbf{x}_i)} = 0 \\
&\Leftrightarrow -\sum_{i=1}^{m} y_i h_k(\mathbf{x}_i) \overline{D}_t(i) e^{-\eta y_i h_k(\mathbf{x}_i)} = 0 \\
&\Leftrightarrow -\left[(1 - \overline{\epsilon}_{t,k}) e^{-\eta} - \overline{\epsilon}_{t,k} e^{\eta}\right] = 0 \\
&\Leftrightarrow \eta = \frac{1}{2} \log \frac{1 - \overline{\epsilon}_{t,k}}{\overline{\epsilon}_{t,k}}
\end{aligned}
$$

Thus, step size matches base classifier weight of AdaBoost

Alternative Loss Functions

- Examples of several convex upper bounds on the zero-one loss



Standard Use in Practice

- **Base Learners**: decision trees, quite often just decision stumps (trees of depth one)
- Boosting stumps
  - — data in $\mathbb{R}^N$, e.g. $N = 2$ (height$(\mathbf{x})$, weight$(\mathbf{x})$)
  - — associate a stump to each component
  - — pre-sort each component: $O(Nm \log m)$
  - — at each round, find best component and threshold
  - — total complexity: $O((m \log m)N + mNT)$
  - — stumps are not weak learners (XOR problem)

Overfitting?

- Assume that $\mathrm{VCdim} = d$ and for a fixed T, define

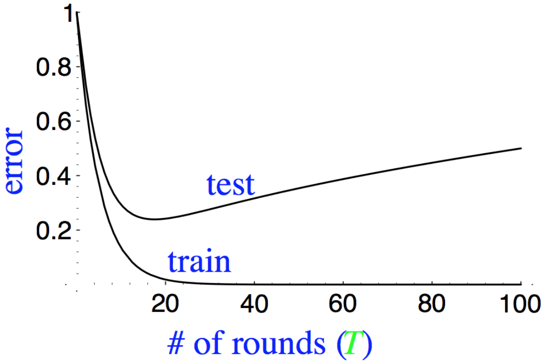
$$\mathcal{F}_T = \left\{ \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t h_t - b \right) : \alpha_t, b \in \mathbb{R},\, h_t \in H \right\}$$

- $\mathcal{F}_T$ can form a very rich family of classifiers. It can be shown (Freund & Shapire, 1997) that

$$\mathrm{VCdim}(\mathcal{F}_T) \leq 2(d+1)(T+1)\log_2((T+1)e)$$

- This suggests that AdaBoost could overfit for large values of T, and that is in fact observed in some cases, but in various others it is not!

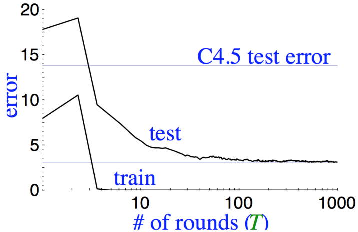
How will Test Error Behave? (A First Guess)



Expect:

- training error to continue to drop (or reach zero)
- test error to increase when h_{final} becomes "too complex"
  - — "Occams razor"
  - — overfitting: hard to know when to stop training

Empirical Observations



(boosting C4.5 on "letter" dataset)

Expect:

- 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!

	# rounds		
	5	100	1000
train error	0.0	0.0	0.0
test error	8.4	3.3	3.1

- Occams razor wrongly predicts "simpler" rule is better