Machine Learning

MDI343- P2

Ensemble methods: bagging, random forests, boosting

Florence d'Alché-Buc

Contact: `florence.dalche@telecom-paris.fr`,
Télécom Paris, Institut Polytechnique de Paris, France

Table of contents

Outline

Ensemble methods for classification and regression

1. Remark:
    - Machine Learning not so "automatic": too many hyperparameters to tune
2. **meta-learning**: a procedure that learns to learn
3. **committee learning** or **wisdom of the crowd**: better results are obtained by combining the predictions of a set of **diverse** classifiers/regressors
4. **ensemble learning**: Improve upon a single base predictive model by building an ensemble of predictive model (with no hyperparameter)

Ensemble methods for regression

Let $f_t, t = 1, \dots, T$ be T different regressors.
Notations:

$$\begin{aligned}
\epsilon_t(x) &= y - f_t(x) \\
MSE(f_t) &= \mathbb{E}[\epsilon_t(x)^2] \\
f_{ens}(x) &= \frac{1}{T}\sum_t f_t(x) \\
&= y - \frac{1}{T}\sum_t \epsilon_t(x).
\end{aligned}$$

Encourage the diversity of base models

$$MSE(f_{ens}) = \mathbb{E}[(y - f_{ens}(x))^2]$$

If ϵ_t are mutually independent with zero mean, then we have:

$$MSE(f_{ens}) = \frac{1}{T^2}\mathbb{E}[\sum_t \epsilon_t(x)^2]$$

The more diverse are the models, the more we reduce the mean square error !

Ensemble methods for supervised classification

Binary classification

$$h_{ens}(x) = \text{sign}(\sum_t h_t(x))$$

Multiclass classification

$$h_{ens}(x) = \arg\max_c \text{vote}(c, h_1, \ldots, h_T)$$

with : $\text{vote}(c, h_1, \ldots, h_T) = \sum_t 1_{h_t(x)=c}(h_t(x))$

Ensemble methods

- **Encourage the diversity of base predictors by:**
  - using bootstrap samples (Bagging and Random forests)
  - using randomized predictors (ex: Random forests)
  - using weighted version of the current sample (Boosting) with weights dependent on the previous predictor (adaptive sampling)

Ensemble methods at a glance

- 1995: Boosting, Freund and Schapire
- 1996: Bagging, Breiman
- 2001: Random forests, Breiman
- 2006: Extra-trees, Geurts, Ernst, Wehenkel

Outline

Reminder: Decomposition bias/variance in regression

Given x,

$$\mathbb{E}_S \mathbb{E}_{y|x}(y - f_S(x))^2 = \textit{noise}(x) + \textit{bias}^2(x) + \text{variance}(x) \qquad (1)$$

noise(x): $E_{y|x}[(y - E_{y|x}(y))^2]$:
quantifies the error made by the Bayes model $(E_{y|x}(y))$
$\textit{bias}^2(x) = (E_{y|x}(y) - E_S[f_S(x)])^2$
measures the difference between minimal error (Bayes error) and the average model
$\textit{variance}(x) = E_S[(f_S(x) - E_S[f_S(x)])^2]$
measures how much $h_S(x)$ varies from one training set to another

Introduction to bagging (regression) - 1

Assume we can generate several training independent samples $\mathcal{S}_1, \dots, \mathcal{S}_T$ from P(x,y).
A first algorithm:

- draw T training independent samples $\{\mathcal{S}_1, \dots, \mathcal{S}_T\}$
- learn a model $f_t \in \mathcal{F}$ from each training sample $\mathcal{S}_t; t = 1, \dots, T$
- compute the average model : $f_{ens}(x) = \frac{1}{T} \sum_{t=1}^{T} f_t(x)$

Introduction to bagging - 2

The bias $(E_{\mathcal{S}_1,\dots,\mathcal{S}_T}[f_{ens}(x)] - f_{target}(x))$ remains the same because :
$E_{\mathcal{S}_1,\dots,\mathcal{S}_T}[f_{ens}(x)] = \frac{1}{T}\sum_t E_{\mathcal{S}_t}[f_t(x)] = E_{\mathcal{S}}[f_{\mathcal{S}}(x)]$
But the variance is divided by T:
$E_{\mathcal{S}_1,\dots,\mathcal{S}_T}[(f_{ens}(x) - E_{\mathcal{S}_1,\dots,\mathcal{S}_T}[f_{ens}(x)])^2] = \frac{1}{T}E_{\mathcal{S}}[(f_{\mathcal{S}}(x) - E_{\mathcal{S}}[f_{\mathcal{S}}(x)])^2]$
When is it useful? When the learning algorithm is unstable, producing high variance estimators such as trees !

Bagging (Breiman 1996)

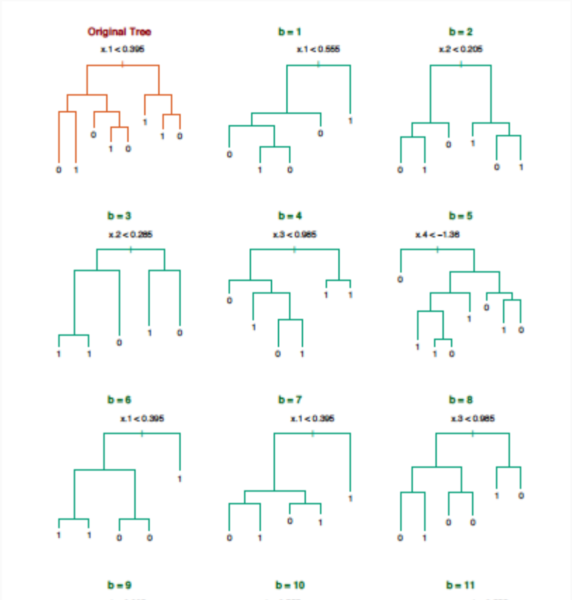
In practice, we do not know P(x,y) and we have only **one training sample** $\mathcal{S}$: we are going to use Bootstrap samples !

Bagging = Bootstrap Aggregating

- draw T bootstrap samples $\{\mathcal{B}_1 \ldots, \mathcal{B}_T\}$ from $\mathcal{S}$ (bootstrap: uniform sampling with replacement)
- Learn a model f_t for each $\mathcal{B}_t$
- Build the average model: $f_{bag}(x) = \frac{1}{T}\sum_t f_t(x)$

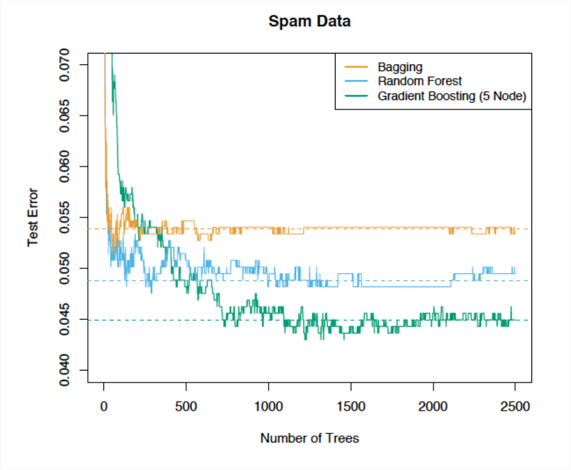
Example of bagged trees

[Book: The elements of statistical learning, Hastie, Tibshirani, Friedman,



Example of bagged trees

[Book: The elements of statistical learning, Hastie, Tibshirani, Friedman, 2001]



Bagging in practise

- Variance is reduced but the bias can increase a bit (the effective size of a bootstrap sample is 30% smaller than the original training set $\mathcal{S}$
- The obtained model is however more complex than a single model
- Bagging works for unstable predictors (neural nets, trees)
- In supervised classification, bagging a good classifier usually makes it better but bagging a bad classifier can make it worse

Outline

Other ensemble methods

- Perturbe and combine algorithms
  - Perturbe the base predictive model
  - Combine the perturbed predictive model

REFS: Random forests: Breiman 2001

Geurts, Ernst, Wehenkel, Extra-trees, 2006

Random forests: Breiman 2001

Random forests algorithm

- INPUT: F= set of p candidate feature splits, $\mathcal{S}_{train}$
- for t=1 to T
  - $\mathcal{S}_{train}^{(t)} \leftarrow$ m instance randomly drawn with replacement from $\mathcal{S}_{train}$
  - $h_{tree}^{(t)} \leftarrow$ randomized decision tree learned from $\mathcal{S}_{train}^{(t)}$
- OUTPUT: $H^T = \frac{1}{T} \sum_t h_{tree}^{(t)}$

Learning a single randomized tree

- To select a split at a node:
    - $R_f(F) \leftarrow$ randomly select (without replacement) f feature splits from F with $f << p$
    - Choose the best split in $R_f(F)$ (consider the different cut-points)
- Do not prune this tree

Extra-trees: Geurts et al. 2006

Extra-trees

- INPUT: candidate feature splits $F = \{1, \ldots, p\}$, $\mathcal{S}_{train}$
- for t=1 to T
    - Always use $\mathcal{S}_{train}$
    - $h_{tree}^{(t)} \rightarrow$: randomized decision tree learned from $\mathcal{S}_{train}$
- OUTPUT: $H^T = \frac{1}{T} h_{tree}^{(t)}$

Extra-trees: learning a single randomized tree in extra-trees

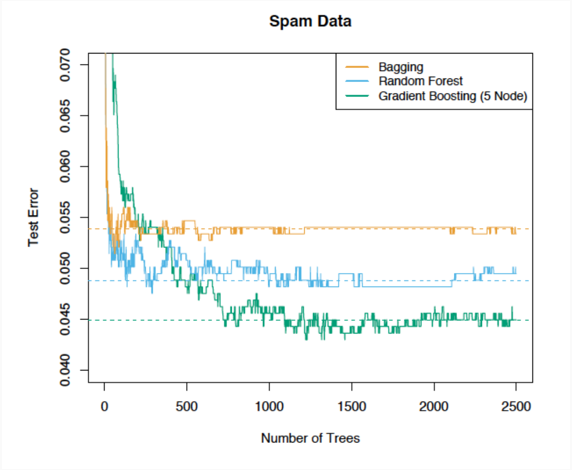
To select a split at a node:

- randomly select (without replacement) K feature splits from F with $K << |F|$
- Draw K splits using the procedure Pick-a-random-split($\mathcal{S}$,i):
    - let a^i_{max} and a^i_{min} denote the maximal and minimal value of x_i in $\mathcal{S}$
    - Draw uniformly a cut-point a_c in $[a^i_{max}, a^i_{min}]$
- Choose the best split among the K previous splits

Do not prune this tree

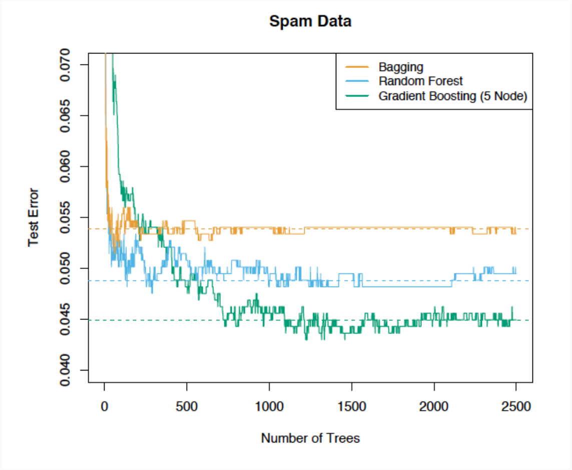
Random forest

Example of decision frontier:



Comparison (just an example)

[Book: The elements of statistical learning, Hastie, Tibshirani, Friedman, 2001]



Random forest

Pros

- Fast, parallelizable and appropriate for a large number of features
- Relatively easy to tune
- Frequently the winner in challenges

Cons

- Overfitting if the size of the trees is too large
- Interpretability is lost (however importance of feature can be measured)

Variable importance

Definition
A variable X^j is important to predict Y if breaking the link between X^j and Y increase the prediction error

$\{\bar{\mathcal{S}}_n^t = \mathcal{S}_n - \mathcal{S}_n^t, t = 1, \ldots, n_{tree}\}$ **out-of-bag samples**: contains the samples not selected by bootstrap

Variable importance

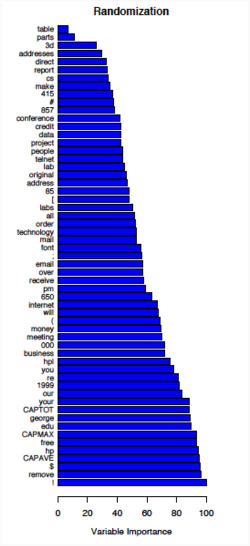
Let $\{\bar{\mathcal{S}}_n^t = \mathcal{S}_n - \mathcal{S}_n^t, t = 1, \dots, n_{tree}\}$ **out-of-bag samples**
Let $\{\bar{\mathcal{S}}_n^{t,j}, t = 1, \dots, n_{tree}\}$: permuted out-of-bag-samples (the values of the jth variable have been randomly permuted).

$$\hat{I}(X^j) = \frac{1}{n_{tree}} \sum_{t=1}^{n_{tree}} R_n(h_t, \bar{\mathcal{S}}_n^{t,j}) - R_n(h_t, \bar{\mathcal{S}}_n^t)$$

with $R_n(h, \mathcal{S})$: empirical loss of h measured on $\mathcal{S}$

Variable importance: spam data

Spam dataset :



Outline

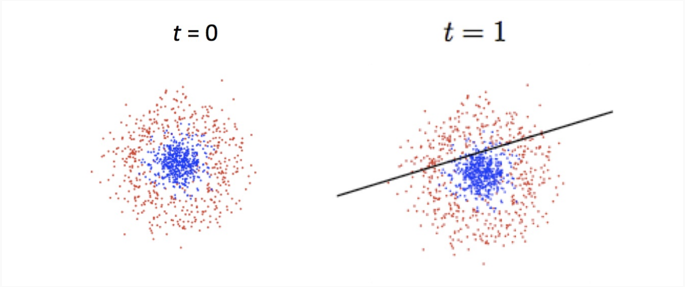
Outline

A preliminary question

- **Is it possible to "boost" a weak learner into a strong learner ?** Michael Kearns
- Yoav Freund and Rob Schapire proposed an iterative scheme, called, Adaboost to solve this problem
  - Idea: train a sequence of learners on weighted datasets with weights depending on the loss obtained so far.
  - Freund and Schapire received the Gödel prize in 2003 for their work on AdaBoost.

Boosting a linear classifier

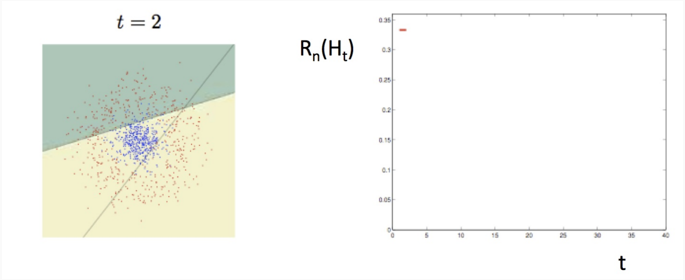
$H_1(x) = h_1(x)$
Binary Classifier: $F_1(x) = \text{sign}(H_1(x))$
Here: h_1: linear classifier
Training error$= R_n$



Source Jiri Matas (Oxford U.)

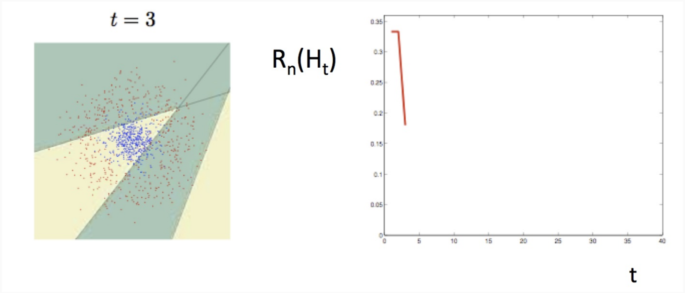
Boosting a linear classifier

$H_2(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x)$
Binary Classifier: $F_2(x) = \text{sign}(H_2(x))$



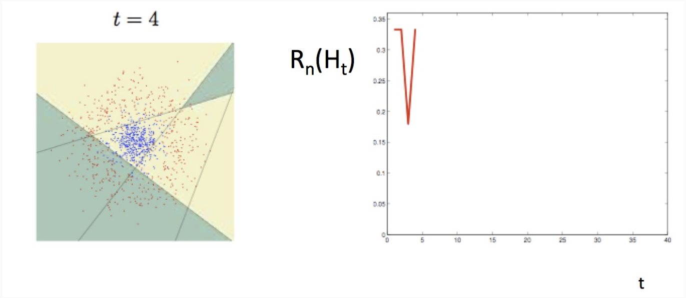
Source Jiri Matas (Oxford U.)

Boosting a linear classifier



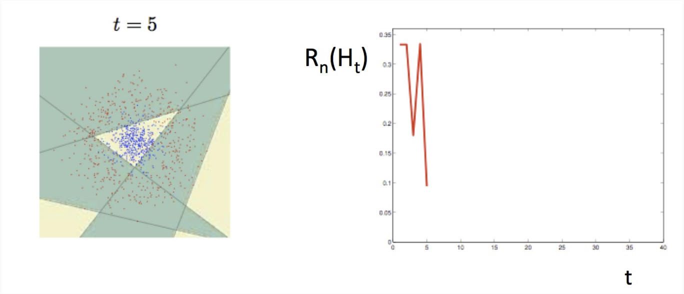
Source Jiri Matas (Oxford U.)

Boosting a linear classifier



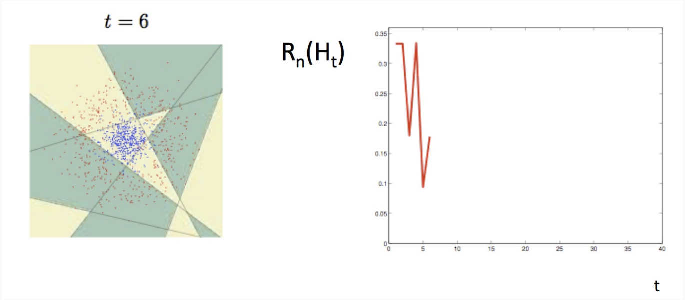
Source Jiri Matas (Oxford U.)

Boosting a linear classifier



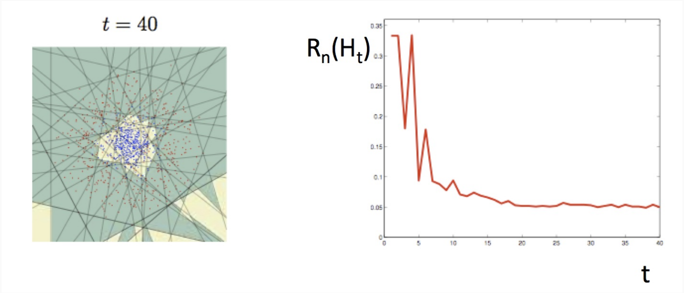
Source Jiri Matas (Oxford U.)

Boosting a linear classifier



Source Jiri Matas (Oxford U.)

Boosting a linear classifier



Source Jiri Matas (Oxford U.)

Weak classifier

Definition: weak classifier
A classifier whose average training error is no more than 0.5

NB : it means that we do not need to have a deep architecture as the base classifier (a ”short” tree will fit for instance, a linear classifier will be perfect and so on...)

Adaboost idea

$\mathcal{H}$: a chosen class of "weak" binary classifiers, $\mathcal{A}$: a learning algorithm for $\mathcal{H}$

- Set $w_1(i) = 1/n$; $H_0 = 0$
- For $t = 1$ to T
    - $h_t = \arg\min_{h \in \mathcal{H}} \epsilon_t(h)$
    - with $\epsilon_t(h) = \mathbb{P}_{i \sim \mathbf{w}_t}[h(x_i) \neq y_i]$
    - Choose α_t
    - Choose w_{t+1}
    - $H_t = H_{t-1} + \alpha_t h_t$
- Output $F_T = \text{sign}(H_t)$

AdaBoost (Freund and Schapire 1996)

$\mathcal{H}$: a chosen class of "weak" binary classifiers

- Set $w_1(i) = 1/n$; $H_0 = 0$
- For $t = 1$ to T
    - $h_t = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} \epsilon_t(h)$ with:
        - $\epsilon_t(h) = \mathbb{P}_{i \sim \mathbf{w}_t}[h(x_i) \neq y_i]$
        - $\epsilon_t = \epsilon_t(h_t)$
        - $\alpha_t = \frac{1}{2} \log \frac{1-\epsilon_t}{\epsilon_t}$
    - let $w_{t+1,i} = \frac{w_t(i) e^{-\alpha_t y_i h_t(x_i)}}{Z_{t+1}}$ where Z_{t+1} is a renormalization constant such that $\sum_{i=1}^{n} w_{t+1,i} = 1$
    - $H_t = H_{t-1} + \alpha_t h_t$
- Output $F_T = \text{sign}(H_t)$

What weight to choose ?

With the chosen definition, we have:

$$\begin{aligned}
w_{t+1,i} &= \frac{w_{t,i} e^{-\alpha_t y_i h_t(x_i)}}{Z_t} \\
&= \frac{w_{t-1,i} e^{-\alpha_{t-1} y_i h_{t-1}(x_i)} e^{-\alpha_t y_i h_t(x_i)}}{Z_{t-1} Z_t} \\
&= \frac{e^{-y_i \sum_{s=1}^{t} \alpha_s h_s(x_i)}}{n \prod_{s=1}^{t} Z_s} \\
&= \frac{e^{-y_i H_t(x_i)}}{n \prod_{s=1}^{t} Z_s}
\end{aligned}$$

You see the weights encourage to correct examples badly classified by the whole combination H_t

First of all let us study Z_t

$$
\begin{aligned}
Z_t &= \sum_{i=1}^{n} w_t(i) e^{-\alpha_t y_i h_t(x_i)} \\
&= \sum_{i=1}^{n} w_t(i) e^{-\alpha_t y_i h_t(x_i)} \\
&= \sum_{i: y_i h_t(x_i) = +1} w_t(i) e^{-\alpha_t} + \sum_{i: y_i h_t(x_i) = -1} w_t(i) e^{\alpha_t} \\
&= (1-\epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t} \\
&= (1-\epsilon_t) \sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \\
&= \ldots \\
&= 2\sqrt{\epsilon_t(1-\epsilon_t)}
\end{aligned}
$$

Bounding the training error

The training error theorem for boosting
The training error of the classifier returned by Adaboost at time T verifies:

$$R_n(F_T) \leq e^{-2\sum_{t=1}^{T}(\frac{1}{2}-\epsilon_t)^2}.$$

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

$$R_n(F_T) \leq e^{-2\gamma^2 T}.$$

Adaboost: Bound on the training error: proof

For all $u \in \mathbb{R}$, we have $1_{u \leq 0} \leq \exp(-u)$.
Then

$$\begin{aligned}
, R_n(F_T) &= \frac{1}{n}\sum_{i=1}^{n} 1_{y_i F_T(x_i) \leq 0} \\
&\leq \frac{1}{n}\sum_{i=1}^{n} \exp(-y_i F_T(x_i)) = \frac{1}{n}\sum_{i=1}^{n}[n\prod_{t=1}^{T} Z_t] w_{t+1,i} = \prod_{t=1}^{T} Z_t
\end{aligned}$$

Bound on the training error: proof ctd'

We can now express $\prod Z_t$ in terms of ϵ_t:

$$\begin{aligned}
\prod_{t=1}^{T} Z_t &= \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} \\
&= \text{by remarkable identity} \\
&= \prod_{t=1}^{T} \sqrt{1-4(1/2-\epsilon_t)^2} \\
&\leq \prod_{t} e^{-2(1/2-\epsilon_t)^2} = e^{-2\sum_{t=1}^{T}(1/2-\epsilon_t)^2}
\end{aligned}$$

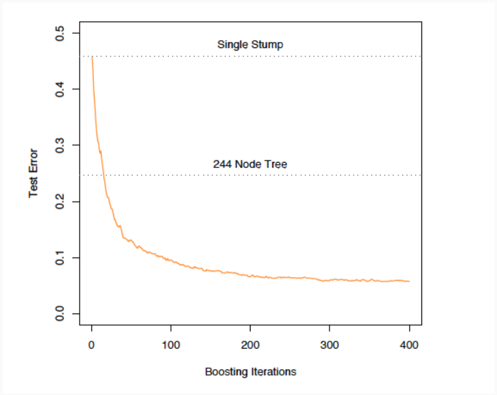
using the identity $1-u \leq \exp(-u)$.

Choice of α_t

The proof reveals several interesting properties:

1. α_t is chosen to minimize $\prod_t Z_t = g(\alpha)$ with $g(\alpha) = (1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}$
    - $g'(\alpha) = -(1-\epsilon_t)e^{-\alpha} + \epsilon_t e^{\alpha}$
    - $g'(\alpha) = 0$ iff $(1-\epsilon_t)e^{-\alpha} = \epsilon_t e^{\alpha}$ iff $\alpha = 1/2 \log \frac{1-\epsilon_t}{\epsilon_t}$
2. The equality $(1-\epsilon_t)e^{-\alpha} = \epsilon_t e^{\alpha}$ means that Adaboost assigns at each time t the same distribution mass to correctly classified examples and incorrectly classified ones. However there is no contradiction because the number of incorrectly examples decreases.

Typical behavior of boosting



Boosting and regularization

- You have to wait a long time to see Boosting overfit. However contrary to first assertions, Adaboost does overfit
- Early stopping: an answer
- or... bound with ℓ_1 norm the magnitude of the weights

Outline

Boosting as a coordinate descent

At the same time, different groups proved that Adaboost writes as a coordinate descent in the convex hull of $\mathcal{H}$.

- Greedy function approximation, Friedman, 1999.
- MarginBoost and AnyBoost : Mason et al. 1999.

Gradient Boosting: the idea on margin-based classification

- Lin($\mathcal{H}$): set of linear combinations of functions of $\mathcal{H}$, endowed with an inner product
- $C : Lin(\mathcal{H}) \to \mathbb{R}$: cost functional on $\mathcal{H}$
- For instance, a margin-based loss $C(H) = \sum_{i=1}^{n} c(y_i H(x_i))$ with $c : \mathbb{R} \to \mathbb{R}$, a differentiable real-valued loss function.
- Imagine that at each boosting step, we would like to find how to correct H_{t-1} in $Lin(\mathcal{H})$ by a gradient descent

Gradient Boosting: the idea on margin-based classification

- Find a new $h \in Lin(\mathcal{H})$ to add to H so that the cost $C(H + \epsilon h)$ decreases the most
- We should compute a functional gradient $-\nabla C(H)$ with respect to function H in $Lin(\mathcal{H})$.
- but this function is not ensured to be in $\mathcal{H}$ so instead, one take the best approximation of the gradient in the space $\mathcal{H}$

$$C(H + \epsilon h) = C(H) + \epsilon \langle \nabla C(H), h \rangle$$

- We take h so as to maximize $-\langle \nabla C(H), h \rangle$.

Gradient Boosting: computations

We use the following definitions:

- in $Lin(\mathcal{H})$, the inner product is : $\langle F, G \rangle = \frac{1}{n}\sum_{i=1}^{n} F(x_i)G(x_i)$
- $\nabla C(H)(x) = \frac{\partial C(H+\alpha 1_x)}{\partial \alpha}\big|_{\alpha=0}$

And then, $\langle \nabla C(H), h \rangle = \frac{1}{n}\sum_{i=1}^{n} y_i h(x_i) c'(y_i H(x_i))$

Gradient Boosting: the algorithm

- Gradient boosting: replace the minimization step by a *gradient descent* type step:
    - Choose h_t as the steepest descent direction in $\mathcal{H}$
    - Choose α_t that minimizes $C(y(H + \alpha h_t))$
- Easy if finding the best descent direction is easy!

Gradient boosting and Adaboost

When taking the loss $L(y, H(x)) = C(yH(x)) = e^{-yH(x)}$, those two algorithms are equivalent!

- Denoting $H_t = \sum_{t'=1}^{t} \alpha_{t'} h_{t'}$,

$$\begin{aligned}
\sum_{i=1}^{n} e^{-y_i(H_{t-1}(x_i)+\alpha h(x_i))} &= \sum_{i=1}^{n} e^{-y_i H_{t-1}(x_i)} e^{-\alpha y_i h(x_i)} \\
&= \sum_{i=1}^{n} w_i'(t) e^{-\alpha y_i h(x_i)} \\
&= (e^{\alpha} - e^{-\alpha}) \sum_{i=1}^{n} w_i'(t) \ell^{0/1}(y_i, h(x_i)) \\
&\quad + e^{-\alpha} \sum_{i=1}^{n} w_i'(t)
\end{aligned}$$

Gradient boosting and adaboost (ctd)

Those two algorithms are equivalent!

- The minimizer h_t in h is independent of α and is also the minimizer of

$$\sum_{i=1}^{n} w_i'(t)\ell^{0/1}(y_i, h(x_i))$$

Gradient boosting and Adaboost

- The optimal α_t is then given by

$$\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon'_t}{\epsilon'_t}$$

  with $\epsilon'_t = (\sum_{i=1}^n w'_i(t) \ell^{0/1}(y_i, h_t(x_i)))/(\sum_{i=1}^n w'_i(t))$
- One verify then by recursion that

$$w_i(t) = w'_i(t)/(\sum_{i=1}^n w'_i(t))$$

and thus the two procedures are equivalent!

AnyBoost or Forward Stagewise Additive model

- General greedy optimization strategy to obtain a linear combination of *weak* predictor
    - Set $t = 0$ and $H_0 = 0$.
    - For $t = 1$ to T,
        - $(h_t, \alpha_t) = \arg\min_{h,\alpha} \sum_{i=1}^{n} \ell(y_i, H_{t-1}(x_i) + \alpha h(x_i))$
        - $H_t = H_{t-1} + \alpha_t h_t$
    - Output $H_T = \sum_{t=1}^{T} \alpha_t h_t$

Losses in Forward Stagewise Additive Modeling

- AdaBoost with $\ell(y,h) = e^{-yh}$
- LogitBoost with $\ell(y,h) = \log(1 + e^{-yh})$
- L_2Boost with $\ell(y,h) = (y-h)^2$ (Matching pursuit)
- L_1Boost with $\ell(y,h) = |y-h|$
- HuberBoost with
  $\ell(y,h) = |y-h|^2 \mathbf{1}_{|y-h|<\epsilon} + (2\epsilon|y-h| - \epsilon^2)\mathbf{1}_{|y-h|\geq\epsilon}$

Simple principle but no easy numerical scheme except for AdaBoost and L_2Boost...

Continuous loss functions and gradient boosting

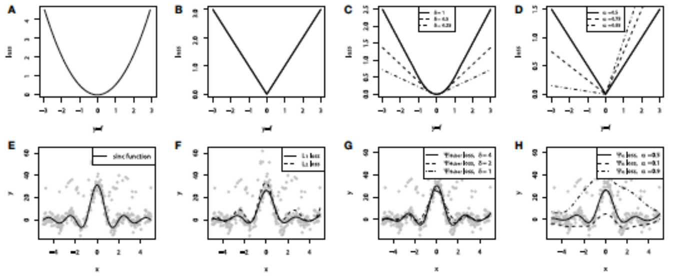


FIGURE 1 | Continuous loss functions: (A) L_2 squared loss function; (B) L_1 absolute loss function; (C) Huber loss function; (D) Quantile loss function. Demonstration of fitting a smooth GBM to a noisy sinc(x) data: **(E)** original sinc(x) function; **(F)** smooth GBM fitted with L_2 and L_1 loss; **(G)** smooth GBM fitted with Huber loss with $\delta = \{4, 2, 1\}$; **(H)** smooth GBM fitted with Quantile loss with $\alpha = \{0.5, 0.1, 0.9\}$.

L_2 Boosting

- Loss function for regression: $\ell(y, h) = (y - h)^2$
- $(h_t, \alpha_t) = \arg\min_{h,\alpha} \sum_{i=1}^{n} (y_i - H_t(x_i) + \alpha h)^2$

Fitting the residuals.

Outline

References

- Perrone, Cooper, When classifiers disagree, 1992
- Tumer and Gosh, 1996
- Breiman, Bagging predictors, 1996
- Further reading: Buhlman and Yu, Analyzing bagging, Annals of stats., 2002
- Breiman, Random Forests, Machine Learning, 2001.
- Geurts, Ernst, Wehenkel, Extra-trees, JMLR, 2006
- Boosting, Freund and Shapire, 1996.
- Friedman, Gradient Boosting, 1999.