Machine Learning

SD-TSIA-210 - P3

Lecture 2 - From decision and regression trees to ensemble methods

Florence d'Alché-Buc, Ekhine Irurozki

Contact: `florence.dalche,ekhine.irurozki@telecom-paris.fr`,
Télécom Paris, IP Paris, France

Outline

Supervised learning

- Let's call X a random vector that takes its value in $\mathcal{X} \subset \mathbb{R}^p$
- Y a random variable that takes its value in $\mathcal{Y}$
- $\mathcal{D}$ is the joint probability distribution of (X, Y)
- $\mathcal{Y} = \mathbb{R}$ in case of regression
- $\mathcal{Y} = \{-1, 1\}$ in case of supervised binary classification
- $\mathcal{Y} = \{1, \dots, C\}$ in case of supervised multiclass classification

Minimizing the true risk

- Define a local loss function $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$,

One wishes to solve this problem:

$$\arg\min_{f:\mathcal{X}\to\mathcal{Y}} \mathbb{E}_{(X,Y)\sim\mathcal{D}}[\ell(Y, f(X))]$$

from the knowledge of $\{(x_i, y_i), i = 1, \dots n\}$.

Target functions for a given loss

Classification
For $\ell(y, f(x))$: $(0/1)$ prediction loss, the best classifier is the **Bayes classifier**: $f_{Bayes}(x) = \arg\max_c \mathbb{P}(Y = c|x)$

Regression
for $\ell(y, f(x))) = (y - f(x))^2$: ℓ_2 loss, the best solution in regression is the regression function: $f_{reg}(x) = \mathbb{E}[Y|x]$

Minimizing the (regularized) empirical risk and focusing on ahypothesis space $\mathcal{H}$

Denote $\mathcal{H}$ the hypothesis class: e.g. the set of models we consider, $\mathcal{S} = \{(x_i, y_i), i = 1, \ldots, n\}$, i.i.d. sample; Ω: regularization term (constraint)

$$\begin{aligned} \arg\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)) \\ s.c.\ \Omega(f) \ \leq \ C \end{aligned}$$

Or equivalently: with $\lambda > 0$, a hyperparameter term,

$$\arg\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ \ell(y_i, f(x_i)) + \lambda \ \Omega(f)$$

Statistical Learning in a nutshell

The most important slide of the whole course

- **Learning**: given $\mathcal{H}$,ℓ, Ω, $\mathcal{S}_n$, define a learning algorithm $\mathcal{A}$ allowing to build $f_n = \mathcal{A}(\mathcal{S}_n, \mathcal{H}, \ell, \Omega)$ with
  - $\mathcal{A}$: learning algorithm
  - $\mathcal{S}_n$: training data
  - $\mathcal{H}$: class of functions
  - Ω: some complexity/capacity measure on $f \in \mathcal{H}$
  - ℓ : local loss function
- **Prediction**: given a new x, and compute $f_n(x)$

Statistical Learning with nonperametric approaches

- Parametric model: prior to learning, the function is completely defined by a set of parameters, requires an assumption (often hidden) about data complexity, fewer training sample required
- Nonparametric model: the number of parameters grows with the size of the training (training data can be considered as part of the set of parameters), many training samples required, no or smaller assumption about data complexity
- **What you already know**:
    1. linear separators (neurons)
    2. K-nearest-neighbour
    3. Support Vector Machine

More non-parametric methods: tree-based approaches

- Local average / Majority vote: prediction is made locally, looking at the neighbors
- **Why is it important ?** Simple way to handle a learning problem, very general, can be extended to many kinds of outputs/ many ML tasks
- Limited number of hyperparameters
- **Drawback:** heavily depends on the number of training data
- **Advantage**: the model itself contains the training data on which it is based (transparency, pre-training), interpretability

Outline

Outline

Decision trees

Trees belong to the family of models of the following form:

$$f(x) = \sum_{\ell=1}^{m} \mathbf{1}(x \in \mathcal{R}_\ell) f_\ell,$$

In practise, the $\mathcal{R}_\ell$'s form a partition and are estimated/learned using the training dataset as well as the f_ℓ's.
f_ℓ only depends on regions $\mathcal{R}_\ell \subset \mathcal{X}$ in the input space and on training data in this region.
Classification into C classes

$$f_\ell = \arg\max_{c=1,\dots,C} \hat{p}_{\ell c},$$

with $\hat{p}_{\ell c} = \frac{1}{N_\ell} \sum_{x_i \in \mathcal{R}_\ell} \mathbf{1}(y_i = c)$, and N_ℓ is the number of training data falling into region $\mathcal{R}_\ell$.

Regression trees

Similarly, regression trees belong to the family of models of the following form:

$$f(x) = \sum_{\ell=1}^{M} \mathbf{1}(x \in \mathcal{R}_\ell) f_\ell,$$

with

$$f_\ell = \frac{1}{N_\ell} \sum_{x_i \in \mathcal{R}_\ell} y_i,$$

N_ℓ is the number of training data falling into region $\mathcal{R}_\ell$.
M is the number of the different regions where the output of the model differs

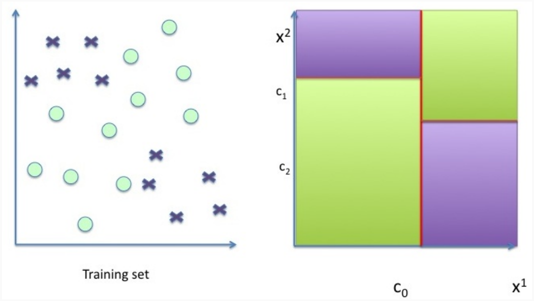
Decision and regression trees: issues

Issues

- How to build automatically these regions ? Loss function ? Learning Algorithm ?
- How to control the complexity of the model ? How to choose m the size of the partition ?
- Why is this model ubiquitous in bank/marketing/medecine fields ?
- Why trees are so important in Machine Learning ?

Decision tree

Invented simultaneously between 1979 and 1983 par L. Breiman et al. (CART, Berkeley, USA) et R. Quinlan (ID3, Sydney, Australia) in two different communities: applied statistics and computer science.

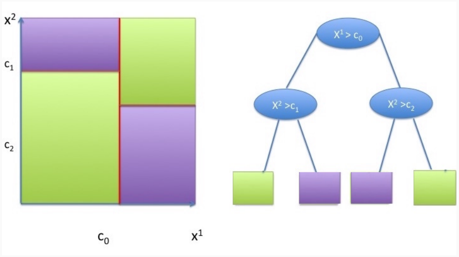


Decision trees

Decision and regression trees have been proposed to build automatically the partition $\mathcal{R}_1 \cup \ldots \mathcal{R}_m$ from the training set.

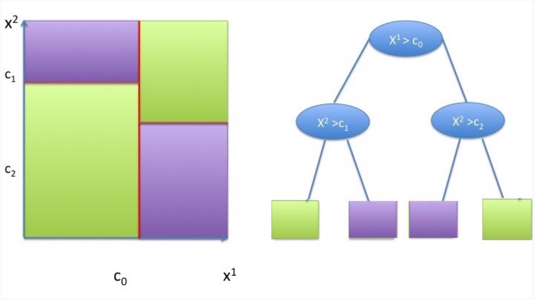
- Local criterion and Constructive algorithm
- Build a binary tree by choosing at each node, a splitting rule that splits the current dataset in two, minimizing a local criterion
- Greedy algorithm

Decision tree 1



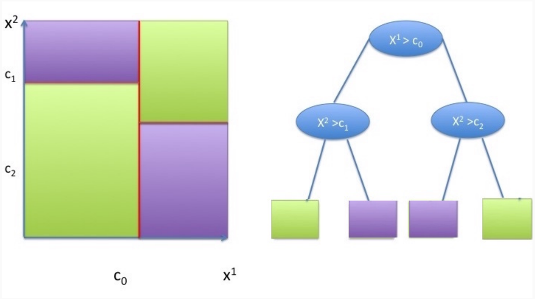
- Use several linear separators to build nonlinear decision frontiers
- These linear separators are chosen to be orthogonal to each basis vector, e.g. defining hyperplane of equation: $x^j = \theta$ to allow an interpretability of the decision function
- At the end of training, we know the features that take part to the decision.

Decision tree 2



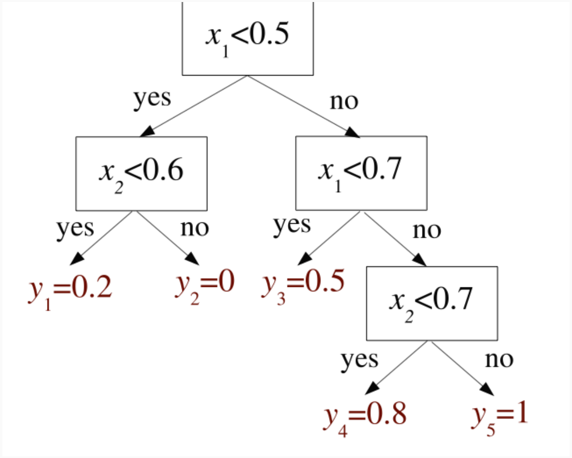
The decision function can be represented by a tree (binary tree) whose each intermediary node is associated to a separating hyperplane $x^j = c$ and each leaf is associated to a majority vote (classification) or a local average (regression).

Interpretability of a decision tree in terms of logical rules



The tree codes for a set of logical rules (symbolic logic of order zero+): each leaf is associated to a logical rule
if $(x^1 > c_0)$ and $(x^2 > c_2)$ then x is predicted to belong to the green class .

Regression tree



Outline

Separator or split t

Continuous Variable x^j:

$$t_{j,s}(\mathbf{x}) = \mathrm{sign}(x^j - s) \tag{1}$$

Categorical variable x^j with K values $\{v_1^j, \ldots, v_K^j\}$:

$$t_{j,\mathbf{v},k}(\mathbf{x}) = 1(x^j = v_k^j) \tag{2}$$

To keep the tree binary, every K-value categorical variable is converted into K binary variables.

Recursive building algorithm for trees

1. Let $\mathcal{S}$ the training data set
2. Build a root node
3. At the current node, find the best binary separation defined by the split t to be applied to $\mathcal{S}$ such that $L(t, \mathcal{S})$ be minimal
4. Associate the chosen separator $t(\hat{j}, \hat{s})$ to the current node and split the current training dataset into two datasets at right and left side, $\mathcal{S}_r$ et $\mathcal{S}_l$.
5. Build two child nodes: a node at right, a node at left, the right (resp. left) node is now associated with $\mathcal{S}_r$ (resp. $\mathcal{S}_l$)
6. Compute a stopping criterion on the right node: if it is satisfied, this node becomes a leaf, otherwise, go to 3.
7. Compute a stopping criterion on the left node: if it is satisfied, this node becomes a leaf, otherwise, go to 3.

How to find the best separation/split in a regression task ?

Let us define the pair of half-planes defined by the sign of $t_{j,s}$:
$\mathcal{R}_l(j,s) = \{x, x^j \leq s\}$ and $\mathcal{R}_r(j,s) = \{x, x^j > s\}$.
We seek the splitting feature j and the split threshold s that solve $\min_{j,s} L((j,s),\mathcal{S})$:

$$\min_{j,s}[\min_{f_r} \sum_{x_i \in \mathcal{S}\cap\mathcal{R}_r(j,s)} (y_i - f_r)^2 + \min_{f_l} \sum_{x_i \in \mathcal{S}\cap\mathcal{R}_l(j,s)} (y_i - f_l)^2].$$

You can see that for any choice of j and s that the inner minimization problem can be solved for:

$$\hat{f}_r = \frac{1}{N_r} \sum_{x_i \in \mathcal{S}\cap\mathcal{R}_r} y_i$$

and

$$\hat{f}_l = \frac{1}{N_r} \sum_{x_i \in \mathcal{S}\cap\mathcal{R}_l} y_i$$

with $N_r := |\{x \in \mathcal{S} \cap \mathcal{R}_r(j,s)\}$ and $N_l := |\{x \in \mathcal{S} \cap \mathcal{R}_l(j,s)\}$.
For simplicity we omit (j,s) in the indexing of N_r and N_l.

How to find the best separation/split in a regression task ?

In practise, it is thus sufficient to find the splitting feature j and the split threshold s that solve:

$$\min_{j,s} \sum_{x_k, x_{k'} \in \mathcal{S} \cap \mathcal{R}_r(j,s)} (y_k - y_{k'})^2 + \sum_{x_k, x_{k'} \in \mathcal{S} \cap \mathcal{R}_l(j,s)} (y_k - y_{k'})^2].$$

e.g., we want to find the split that minimizes the sum of the empirical variances on the two parts of the splitted dataset.
N.B. As for s candidates we will take a finite number of candidates in the interval of values of X^j.

Finding the best separation/split in a regression task: additional remark

Denote $\mathrm{Var}(Y|S)$ the empirical variance of Y as computed with data from S.
Let us re-write the optimization problem for finding a split:

$$\min_{j,s} N_r \mathrm{Var}(Y|\mathcal{S} \cap \mathcal{R}_r(j,s)) + N_l \mathrm{Var}(Y|\mathcal{R}_l(j,s)).$$

Nota bene: this is equivalent to: Maximize the variance reduction obtained by splitting, in other words:

$$\max_{j,s} \mathrm{Var}(Y|\mathcal{S}) - \frac{N_r}{N}\mathrm{Var}(Y|\mathcal{S} \cap \mathcal{R}_r(j,s)) - \frac{N_l}{N}\mathrm{Var}(Y|\mathcal{R}_l(j,s)).$$

Illustration: tree growing=output variance reduction

(We show in blue the input variable that is used to split at each node)

x_1	x_2	y
0.4	0.4	0.2
0.4	0.8	0.0
0.6	0.6	0.5
0.8	0.6	0.8
0.8	0.8	1.0

$\rightarrow$

x_1	x_2	y
0.4	0.4	0.2
0.4	0.8	0.0

$\rightarrow$

$\boxed{\hat{y} = 0.2}$

$\boxed{\hat{y} = 0.0}$

x_1	x_2	y
0.6	0.6	0.5
0.8	0.6	0.8
0.8	0.8	1.0

$\rightarrow$

$\boxed{\hat{y} = 0.5}$

x_1	x_2	$\hat{y}$
0.8	0.6	0.8
0.8	0.8	1.0

$\rightarrow$

$\boxed{\hat{y} = 0.8}$

$\boxed{\hat{y} = 1.0}$

How to find the best separation/split in a classification task ?

We seek the splitting feature j and the split threshold s that minimizes an impurity criterion. We want to have child nodes as "pure" or "homogeneous" as possible in terms of classes.

$$\min_{j,s}[\frac{N_r}{N}H(\mathcal{S}\cap\mathcal{R}_r(j,s))+\frac{N_l}{N}H(\mathcal{S}\cap\mathcal{R}_l(j,s))].$$

where H is an impurity criterion.
Nota bene: if we H is normalized, we must weight the impurity criterion on the righ and on the left to take into account the number of data on each child node.

Impurity criteria

For a given $\mathcal{S}$ with n labeled data:

$$p_k(\mathcal{S}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i = k)$$

Cross-entropy:

$$H(\mathcal{S}) = -\sum_{k=1}^{C} p_k(\mathcal{S}) \log p_k(\mathcal{S})$$

Gini index

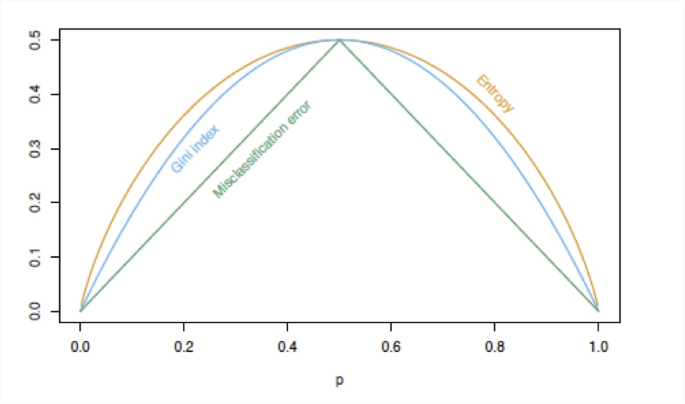
$$H(\mathcal{S}) = \sum_{k=1}^{C} p_k(\mathcal{S})(1 - p_k(\mathcal{S}))$$

Misclassification

$$H(\mathcal{S}) = 1 - p_{k(\mathcal{S})},$$

avec $k(\mathcal{S})$: majority class in $\mathcal{S}$.

Impurity criteria



Finding the best separation/split in a classification task ?

Using for instance Gini criterion:

$$H_{j,s} := [\frac{N_r}{}$$

N $\sum_{k=1}^{C} \frac{N_r(k)}{N_r}(1 - \frac{N_l(k)}{N_l}) + \frac{N_l}{N} \sum_{k=1}^{C} \frac{N_l(k)}{N_l}(1 - \frac{N_l(k)}{N_l})]$. where H is an impurity criterion.

Stopping criterion

- A maximal depth is reached
- OR: A number of minimal number of data is reached at a node

Cross-validation can be used to select that hyperparameter.
Otherwise, the tree is built until the training dataset is perfectly predicted (possible in classification) : **overfitting** !
In the past, people were building deep trees and then used a pruning procedure to erase some branches.

Advantages and drawbacks of Decision trees

Advantages

- Produces an interpretable nonlinear decision function,
- Consistency of decision trees (not all the discrimination rules imply consistency, see Scott, Nowak, IEEE Trans. Information Theory 2006 - for a review)
- Works for multiple classes without any pre-processing
- Efficient prediction stage : $O(\log L)$, L: number of leaves
- Works for continuous and categorial features
- Support many extensions to many other ML tasks

Advantages and drawbacks of Decision trees

Drawbacks

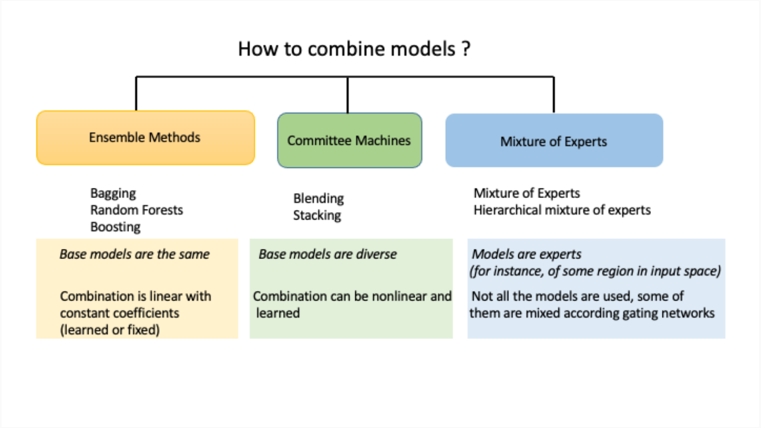
- Large variance estimator, instability : a small variation in the training set produces a very different tree $\rightarrow$ so, ensemble of trees are therefore very attractive
- No global optimization

Outline

Starting point for combining models in Machine Learning

- Remark 1 in the 2000's: Machine Learning not so "automatic": too many hyperparameters to tune
- Remark 2 in the 2020's: Huge models: too many parameters in Large Language Models, inference is too long !

Overview of combination methods in Machine Learning


How to combine models ?
Ensemble Methods
Committee Machines
Mixture of Experts
Bagging
Random Forests
Boosting
Blending
Stacking
Mixture of Experts
Hierarchical mixture of experts
Base models are the same
Combination is linear with constant coefficients (learned or fixed)
Base models are diverse
Combination can be nonlinear and learned
Models are experts (for instance, of some region in input space)
Not all the models are used, some of them are mixed according gating networks

Ensemble and mixture methods

Different ways to cope with this problem

- Meta-learning: Meta-learning algorithms learn from the output of other machine learning algorithms that learn from data. Meta-learning requires the presence of other learning algorithms that have already been trained on data.
- **Committee learning** or **wisdom of the crowd**: better results are obtained by combining the predictions of a set of **diverse** classifiers/regressors
- **Ensemble learning**: Improve upon a single base predictive model by building an ensemble of predictive models **of the same type** (with nearly no hyperparameter)
- **Mixture of experts**: use some of the experts to solve the task depending on the input (gating networks indicate which expert to use)

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
- 2015: Xgboost: Chen and Guestrin.

Outline

Reminder: Decomposition bias/variance in regression

Given x,

$$\mathbb{E}_S \mathbb{E}_{y|x}(y - f_S(x))^2 = noise(x) + bias^2(x) + \text{variance}(x) \qquad (3)$$

noise(x): $E_{y|x}[(y - E_{y|x}(y))^2]$:
quantifies the error made by the Bayes model ($E_{y|x}(y)$)
$bias^2(x) = (E_{y|x}(y) - E_S[f_S(x)])^2$
measures the difference between minimal error (Bayes error) and the average model
$variance(x) = E_S[(f_S(x) - E_S[f_S(x)])^2]$
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 $\mathcal{S}_1, \ldots, \mathcal{S}_T$ from P(x,y).

A first algorithm:

- draw T training independent samples $\{\mathcal{S}_1, \ldots, \mathcal{S}_T\}$
- learn a model $f_t \in \mathcal{F}$ from each training sample $\mathcal{S}_t; t = 1, \ldots, T$
- 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)

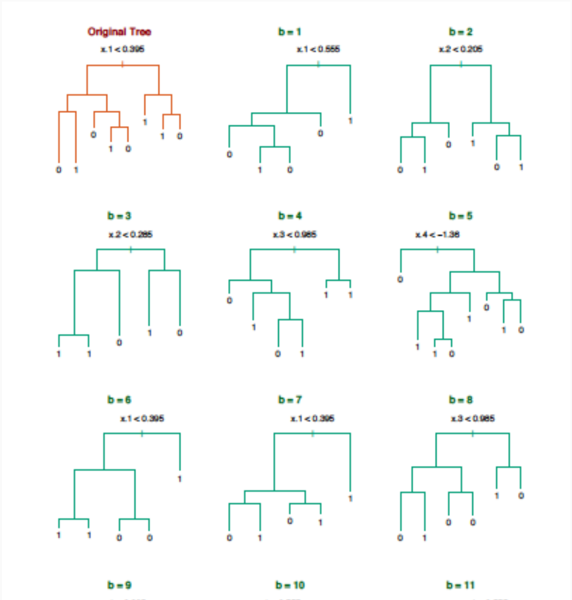
In practice, we do not know P(x,y) and we have only **one training sample** $\mathcal{S}$: we are going to use Bootstrap samples !

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: $f_{bag}(x) = \frac{1}{T}\sum_t f_t(x)$

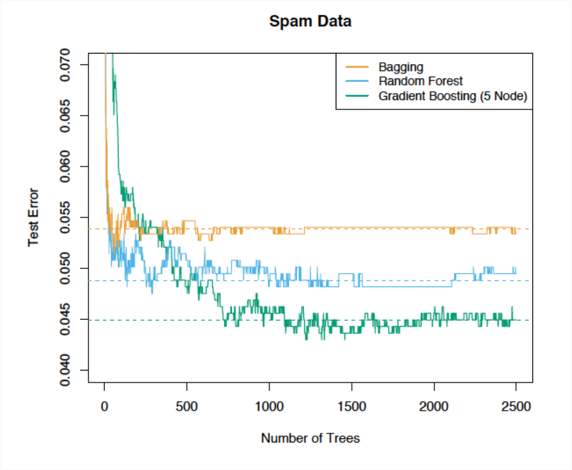
Example of bagged trees

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



Example of bagged trees

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



Bagging: a first step for more involved algorithms

Bagging:

- 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
- Bagging alone is rarely used

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

Remember $X \in \mathcal{X} \subset \mathbb{R}^p$, denote x^j its j-th coordinate

- INPUT: $\mathcal{F} = \{1, 2, \dots, p\}$, $\mathcal{S}_{train}$
- for t=1 to T
    - $\mathcal{S}_{train}^{(t)} \leftarrow n$ instances randomly drawn with replacement from $\mathcal{S}_{train}$
    - $h_{tree}^{(t)} \leftarrow$ randomized decision tree learned from $\mathcal{S}_{train}^{(t)}$
- OUTPUT: $H^T(x) = \frac{1}{T} \sum_t h_{tree}^{(t)}(x)$

Learning a single randomized tree

$x \in \mathbb{R}^p$; Candidate feature index set: $\mathcal{F} = \{1, \ldots, p\}$

- To select a split at a node:
  - $\mathcal{C}_k(\mathcal{F}) \leftarrow$ randomly select (without replacement) k features from $\mathcal{F}$ with $k << p$
  - Choose the variable that provides the best split (in $\mathcal{C}_k(\mathcal{F})$ (consider all different cut-points s for each feature X^j)
- Do not prune this tree

Extra-trees: Geurts et al. 2006

Extra-trees

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

Learning an extremly randomized tree in extra-trees

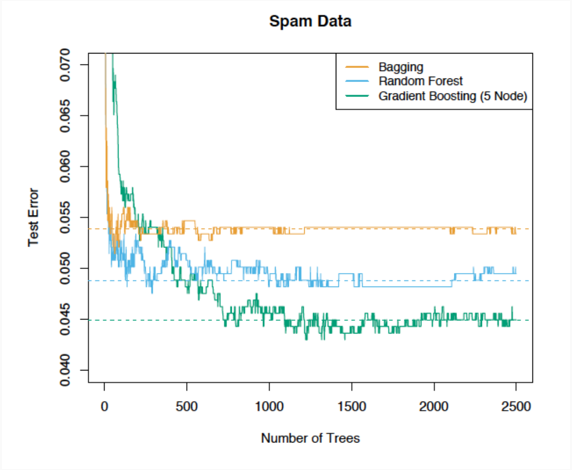
To select a split at a node:

- randomly select (without replacement) k feature with $k << p$
- 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 s_c in $[s^i_{max}, s^i_{min}]$
- Choose the best split among the k previous splits associated to the k cut-points.

Do not prune this tree
k is a hyperparameter, usually 10 to 20% of the number p of features.

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 increases the prediction error

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

Variable importance examplified in Breiman's RF

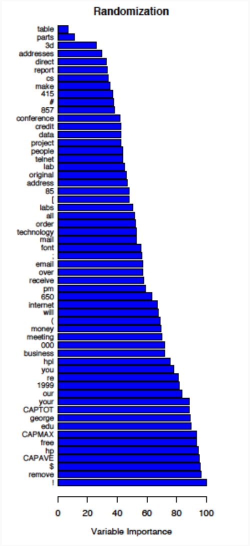
Let $\{\bar{\mathcal{S}}_n^t = \mathcal{S}_n - \mathcal{S}_n^t, t = 1, \ldots, n_{tree}\}$ **out-of-bag samples**
Let $\{\bar{\mathcal{S}}_n^{t,j}, t = 1, \ldots, n_{tree}\}$: permuted out-of-bag-samples (the values of the jth variable have been randomly permuted).

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

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.

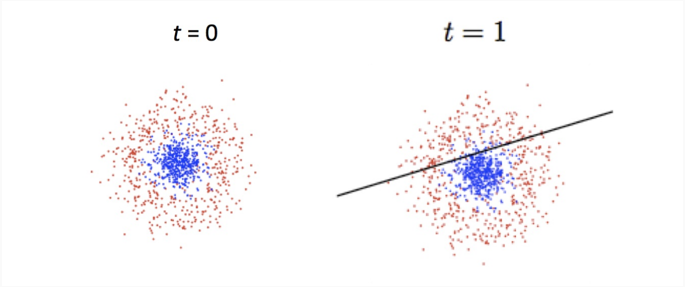
Boosting a linear classifier

$H_1(x) = h_1(x)$

Binary Classifier: $F_1(x) = \text{sign}(H_1(x))$

Here: h_1: linear classifier

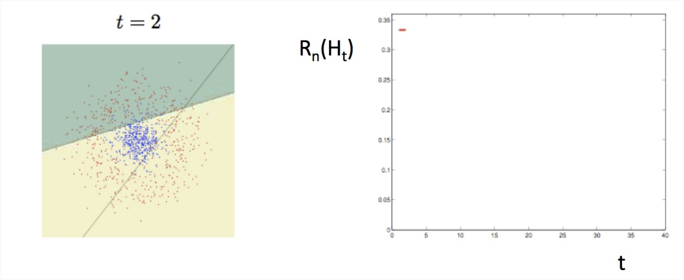
Training error$= R_n$



Source Jiri Matas (Oxford U.)

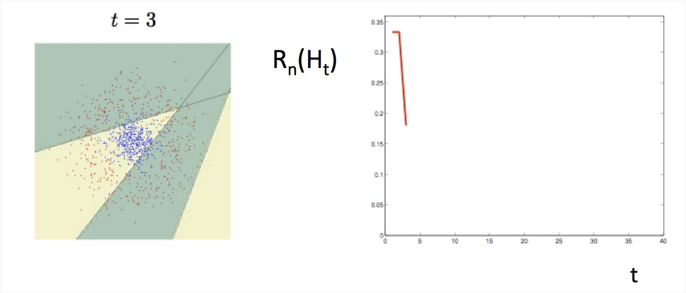
Boosting a linear classifier

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



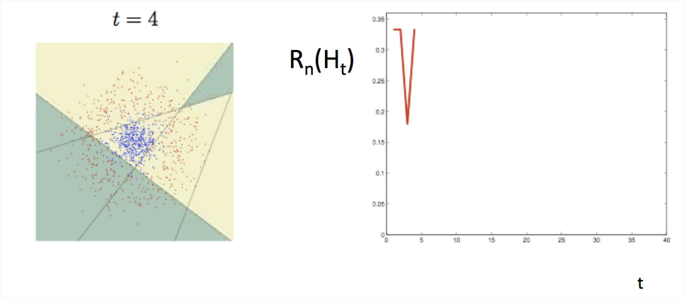
Source Jiri Matas (Oxford U.)

Boosting a linear classifier



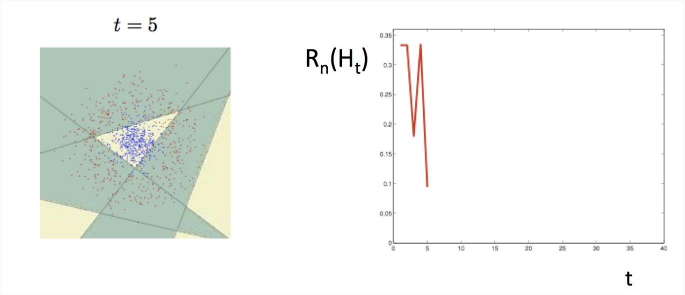
Source Jiri Matas (Oxford U.)

Boosting a linear classifier



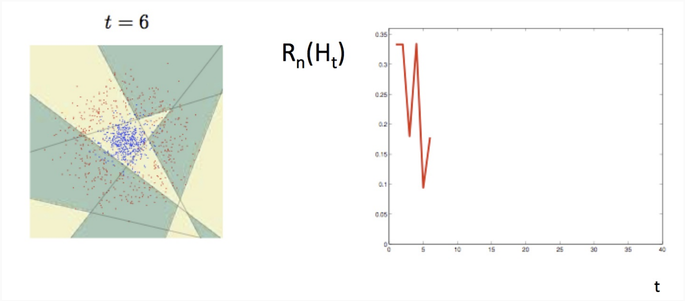
Source Jiri Matas (Oxford U.)

Boosting a linear classifier



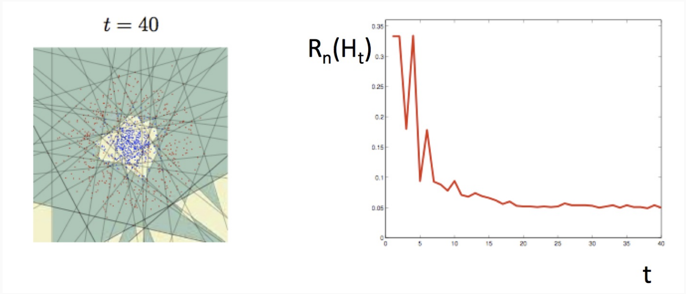
Source Jiri Matas (Oxford U.)

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 $\sum_{i=1}^n w_{t+1} = 1$
  - $H_t = H_{t-1} + \alpha_t h_t$
- 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

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

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:

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

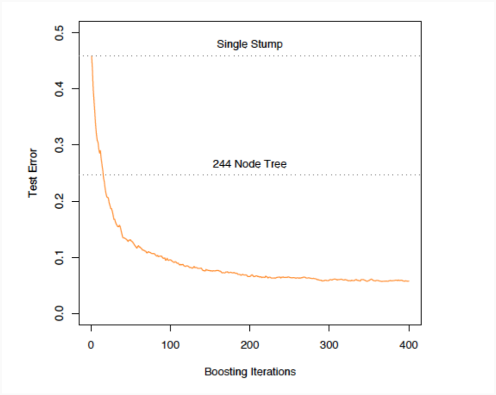
using the identity $1-u \leq \exp(-u)$.

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

- 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
  - The negative gradient is: $-g_t(x_i) = -\frac{\partial \ell(y_i, H_{t-1}(x_i))}{\partial H_{t-1}(x_i)}$
- however, this gradient is only defined at $x_i' s$ values and not for all x values.
- instead, one take the best approximation of the gradient in the space $\mathcal{H}$

$$(h_t, \alpha_t) = \arg\min_{h,\alpha} \sum_{i=1}^{n} (\ell(y_i, H_{t-1}(x_i) + \alpha h) = L(y_i, H_{t-1}(x_i) + \alpha h(x_i)))$$

- Gradient approximation
  $L(y_i, H_{t-1}(x_i) + \alpha h(x_i)) \sim \ell(y_i, H_{t-1}(x_i)) + \alpha \langle \nabla \ell(H_{t-1}), h \rangle.$

with the following definition: in $Lin(\mathcal{H})$, we define the inner product $\langle F, G \rangle = \frac{1}{n} \sum_{i=1}^{n} F(x_i) G(x_i)$

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 $L(y, H + \alpha h_t)$
- Easy if finding the best descent direction is easy!

Gradient boosting and Adaboost

When taking the loss $L(y, H(x)) = e^{-yH(x)}$, 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 Foward 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

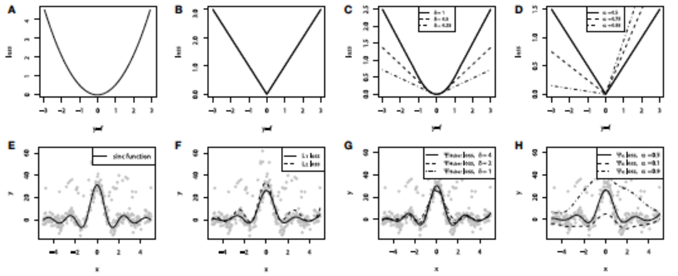


FIGURE 1 | Continuous loss functions: (A) L_2 squared loss function; (B) L_1 absolute loss function; (C) Huber loss function; (D) Quantile loss function. Demonstration of fitting a smooth GBM to a noisy sinc(x) data: **(E)** original sinc(x) function; **(F)** smooth GBM fitted with L_2 and L_1 loss; **(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.

Conclusion about ensemble methods

- Key to learning from tabular data, Random Forests and Gradient Boosting are (still) widely used
- Simplicity of algorithms, parallelization of implementations (RF)

References

- CART : Classification and Regression Trees, Breiman, Olshen, Friedman and Stone, Wadsworth Statistics, 1984.
- Induction and Decision trees, Quinlan, Machine Learning Journal 1, 1986.
- Chapter Additive models and trees : The elements of statistical learning, Hastie, Tibshirani, Friedman, Springer.
- Breiman, L. (2001). Random forests. Machine learning, 45, 5-32.
- Extra-trees: Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. Machine learning, 63, 3-42.
- Freund, Y., & Schapire, R. E. (1996, July). Experiments with a new boosting algorithm. In icml (Vol. 96, pp. 148-156).
- AnyBoost: Mason, L., Baxter, J., Bartlett, P., & Frean, M. (1999). Boosting algorithms as gradient descent. Advances in neural information processing systems, 12.
- XGboost: Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd ACM sigkdd international conference on knowledge discovery and data mining (pp. 785-794).`https://dl.acm.org/doi/pdf/10.1145/2939672.2939785`