Machine Learning

SD-TSIA210

Decision and regression trees - Ensemble methods

Ekhine Irurozki

irurozki@telecom-paris.fr,
Télécom Paris, IP Paris, France

Supervised learning

X a random vector that takes its value in $\mathcal{X} = \mathbb{R}^p$

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

$\mathcal{Y} = \{1, \ldots, C\}$ in case of supervised multiclass classification

Minimizing the true risk

Denote $\mathcal{H}$ the hypothesis class: e.g. the set of models we consider

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

One wishes to solve this problem:

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

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

Minimizing the empirical risk Since we do not access to the true distribution $\mathcal{S} = \{(x_i, y_i), i = 1, \dots, n\}$, i.i.d. sample

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

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:

$$f_{reg}(x) = \mathbb{E}[Y|x]$$

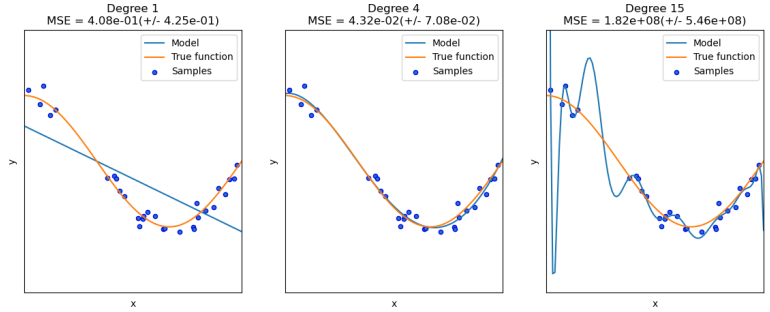
Minimizing the (regularized) empirical risk

$\mathcal{S} = \{(x_i, y_i), i = 1, \ldots, n\}$, i.i.d. sample; Ω: regularization term

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

$$s.c. \Omega(f) \leq C$$

Or equivalently: with $\lambda > 0$, a hyperparameter term, $\arg\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, f(x_i)) + \lambda \Omega(f)$



Source: SKlearn

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

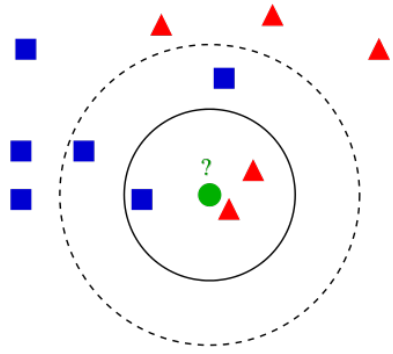
Prediction: given a new x, and compute $f_n(x)$

Statistical Learning with local average models and trees

- Nonparametric approach to machine learning
- Training data are the main parameters !
- Motivations: Local average / Majority vote: prediction is made locally, looking at the neighbors, like k-nearest neighbors (KNN) algorithm

Locallity

Exercise Which is the complexity?



Source: Wikipedia

Problem: **the curse of dimensionality**, as the number of dimensions grows, the amount of data we need to generalize accurately grows exponentially

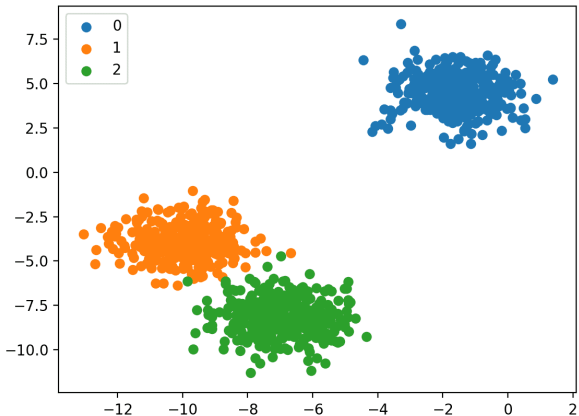
Statistical Learning with local average models and trees

- **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 data
- **Advantage**: the model itself contains the training data on which it is based (transparency, pre-training), interpretability

Decision trees: learning and prediction

Learning: construct the regions

Prediction: find the region



Decision trees

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

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

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 (multiclass and binary) is done by majority voting

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

with $\hat{p}_{\ell c} = \frac{1}{N_\ell} \sum_{x_i \in \mathcal{R}_\ell} \mathbb{I}(y_i = c)$, 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:

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

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 $\mathcal{R}_\ell$.

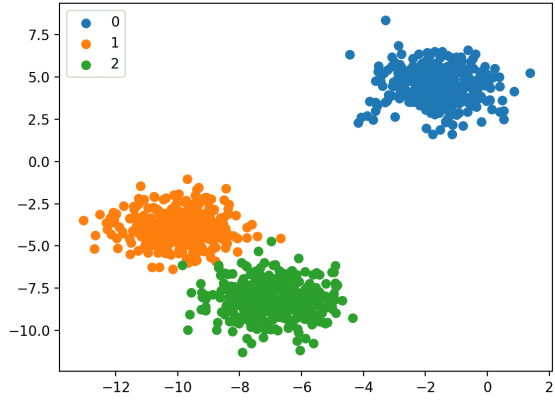
Decision and regression trees: issues

- 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 trees

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

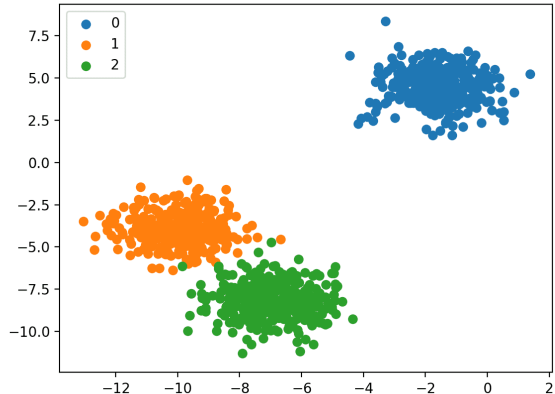
- Local criterion and Constructive algorithm
- Build a binary tree by choosing at each node, a splitting rule that splits the current dataset in two, minimizing a local criterion
- Greedy algorithm



Source: SKlearn

Decision trees

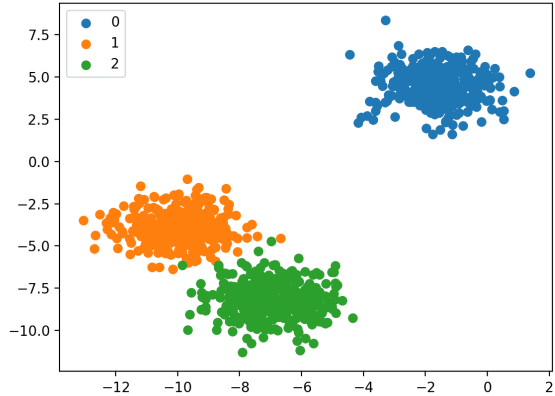
- Use several linear separators to build nonlinear decision frontiers
- These linear separators are chosen to be orthogonal to each basis vector, e.g. defining hyperplane of equation: $x^j = \theta$ to allow an interpretability of the decision function
- At the end of training, we know the features that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



Source: SKlearn

Decision trees

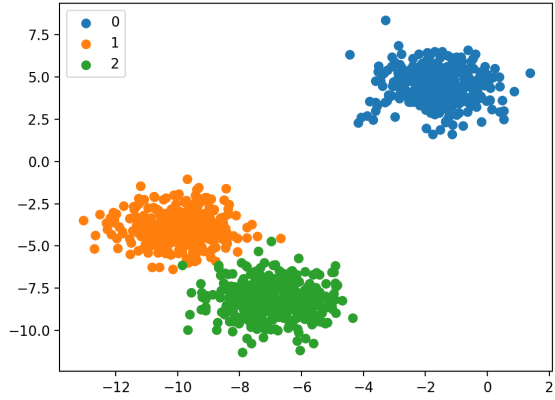
The decision function can be represented by a tree (binary tree) whose each intermediary node is associated to a separating hyperplane $x^j = c$ and each leaf is associated to a majority vote (classification) or a local average (regression).



Source: SKlearn

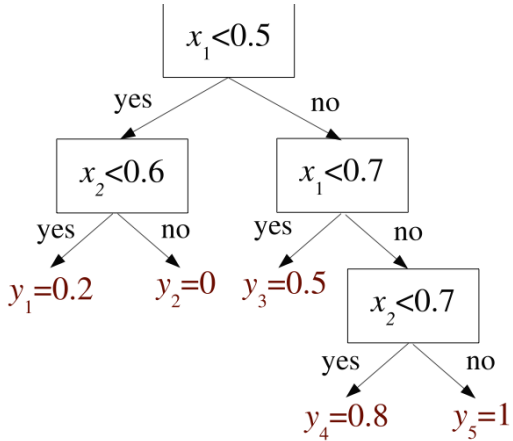
Decision trees

The tree codes for a set of logical rules (symbolic logic of order zero+): each leaf is associated to a logical rule if $(x^1 - 4)$ and $(x^2 < -6)$ then x is predicted to belong to the green class.



Source: SKlearn

Regression tree



Separator or split t

Huge combinatorial problem

Continuous Variable x^j:

$$t_{j,s}(\mathbf{x}) = \text{sign}(x^j - s) \tag{2}$$

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

$$t_{j,\mathbf{v},k}(\mathbf{x}) = 1(x^j = v_k^j) \tag{3}$$

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 classification task ?

We seek the splitting feature j and the split threshold s that minimizes an **impurity criterion**.
We want to have children 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.

Impurity criteria

For a given $\mathcal{S}$ with n labeled data:$p_k(\mathcal{S}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(y_i = k)$ with $k(\mathcal{S})$: majority class in $\mathcal{S}$.

Misclassification

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

Cross-entropy:

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

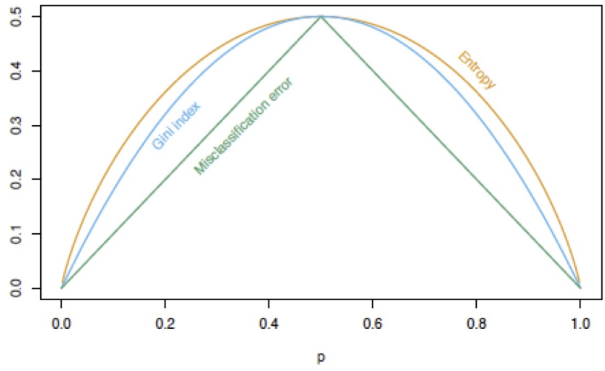
Gini index

$$H(\mathcal{S}) = 1 - \sum_{k=1}^{C} p_k(\mathcal{S})^2$$

Exercise (300,100),(100,300) vs (400,200),(200,0)

Exercise Relate the cross entropy with the Kullback-Leibler divergence, $KL(p, q) = \sum p_i \log \frac{p_i}{q_i}$

Impurity criteria



Are we maximizing or minimizing?

Impurity criteria: computational aspects

All impurity measures depend on $\mathbb{I}(y_i = k)$ via the probability

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

Exercise Finding the best split (naively) has complexity $O(dn^2c)$ for a sample of n points on d dimensions and c clases. Propose an incremental version and give the complexity.

Exercise Propose the extension of Gini and the combination of nodes for weighted samples.

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

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

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:

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

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.

Stopping criterion

- A maximal depth is reached
- All the leaves are pure
- No improvement
- 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.

Exercise why is XOR not adequate for "No improvement" rule?

Exercise Which is the predicted function by a regression tree for the given (see board) function and splits?

Exercise See the given code. Which are the **main disadvantages and advantages of trees**?

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

Reminder: Decomposition bias/variance in regression

Given x,

$$\mathbb{E}_S \mathbb{E}_{y|x}(y - f_S(x))^2 = noise(x) + bias^2(x) + \text{variance}(x) \tag{4}$$

$noise(x)$: $\mathbb{E}_{y|x}[(y - \mathbb{E}_{y|x}(y))^2]$: quantifies the error made by the Bayes model ($\mathbb{E}_{y|x}(y)$)

$bias^2(x) = (\mathbb{E}_{y|x}(y) - \mathbb{E}_S[f_S(x)])^2$ measures the difference between minimal error (Bayes error) and the average model

$variance(x) = \mathbb{E}_S[(f_S(x) - \mathbb{E}_S[f_S(x)])^2]$ 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:

Algorithme : Reducing variance

Données : T training independent samples $\{\mathcal{S}_1, \ldots, \mathcal{S}_T\}$
Résultat : An ensemble model f_{ens}
pour *sample* $\mathcal{S}_t$ *in* $\{\mathcal{S}_1, \ldots, \mathcal{S}_T\}$ **faire**
  $f_t \leftarrow$ learn a model with $\mathcal{S}_t$
retourner $f_{ens}(x) = \frac{1}{T} \sum_{t=1}^{T} f_t(x)$

Introduction to bagging - 2

The bias $(\mathbb{E}_{\mathcal{S}_1,\ldots,\mathcal{S}_T}[f_{ens}(x)] - f_{target}(x))$ remains the same because :

$$E_{\mathcal{S}_1,\ldots,\mathcal{S}_T}[f_{ens}(x)] = \frac{1}{T}\sum_t \mathbb{E}_{\mathcal{S}_t}[f_t(x)] = \mathbb{E}_{\mathcal{S}}[f_{\mathcal{S}}(x)]$$

But the variance is divided by T:

$$\mathbb{E}_{\mathcal{S}_1,\ldots,\mathcal{S}_T}[(f_{ens}(x) - \mathbb{E}_{\mathcal{S}_1,\ldots,\mathcal{S}_T}[f_{ens}(x)])^2] = \frac{1}{T}\mathbb{E}_{\mathcal{S}}[(f_{\mathcal{S}}(x) - \mathbb{E}_{\mathcal{S}}[f_{\mathcal{S}}(x)])^2]$$

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, i.e., random samples with replacement of the original dataset.

Bagging (or Bootstrap aggregating) is a machine learning ensemble meta-algorithm based on resampling the dataset used here to reduce variance

---

Algorithme : Bagging = Bootstrap Aggregating

Données : Sample $\mathcal{S}$
Résultat : An (bootstrap) ensemble model f_{bag}
pour t *in* $\{1, \ldots, T\}$ **faire**
  $\mathcal{B} \leftarrow$ bootstrap sample from $\mathcal{S}$
  $f_t \leftarrow$ learn a model (a tree) with $\mathcal{B}$
retourner $f_{bag}(x) = \frac{1}{T} \sum_{t=1}^{T} f_t(x)$

Exercises

- Define sampling with and without replacement.
- How do we average discrete labels? See Equation (1).
- Give the pseudocode for bagging
- (Effective sample) Show that each bootstrap sample (or bagged tree) will contain $1 - e^{-1} \approx 0.632$ of the original sample $\mathcal{S}$.
- Is $\mathcal{B}_i$ drawn from the same distribution as $\mathcal{S}$?
- Are the samples i.i.d.?

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

What if there is a feature with large predictive power? Correlated trees

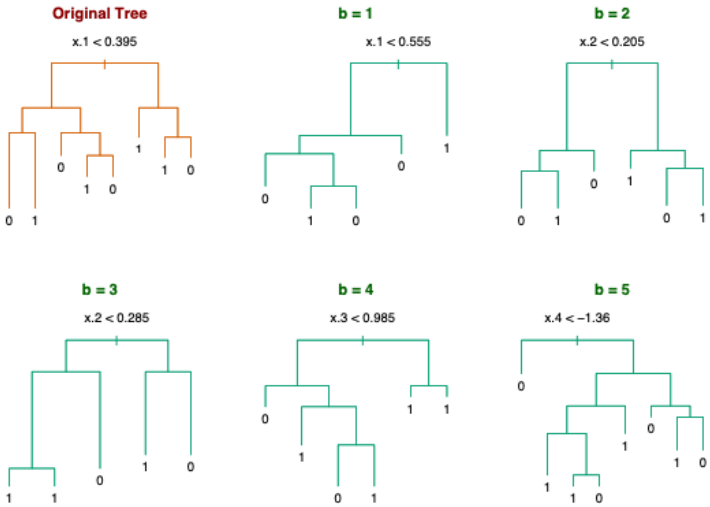


Figure: [Book: The elements of statistical learning, Hastie, Tibshirani, Friedman, 2001]

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

Algorithme : Random forest

Données : Sample $\mathcal{S}$
Résultat : ...
pour t *in* $\{1, \dots, T\}$ **faire**
  $\mathcal{B} \leftarrow$ bootstrap sample from $\mathcal{S}$
  $f_t \leftarrow$ learn a **randomized** tree with $\mathcal{B}$
retourner $f_{bag}(x) = \frac{1}{T} \sum_{t=1}^{T} f_t(x)$

Algorithme : Randomized tree

Données : Sample $\mathcal{S}$
Résultat : Randomized tree
tant que *not halting condition* **faire**
  $\mathcal{F} \leftarrow$ randomly keep $k = \sqrt{p}$ features
  find best split in $\mathcal{F}$
  $\vdots$
retourner randomized tree

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

To select a split at a node:

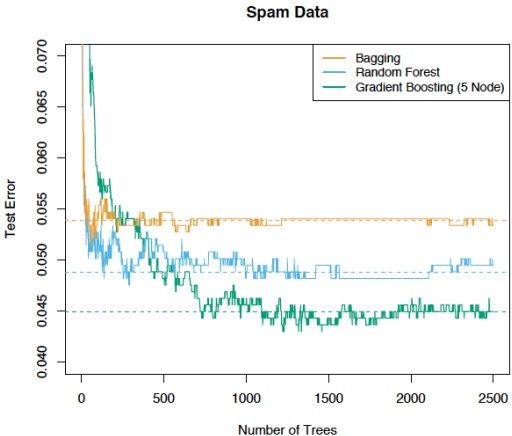
Algorithme : Randomized extra-tree

Données : Sample $\mathcal{S}$
Résultat : Extra-tree
tant que *not halting condition* **faire**
randomly select (without replacement) K feature with $K << p$
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.
$\vdots$
retourner randomized tree

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

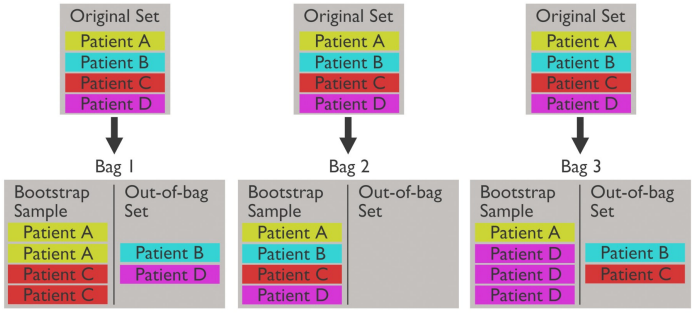
- Fast, parallelizable and appropriate for a large number of features
- Relatively easy to tune
- Frequently the winner in challenges

Cons

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

Out-of-bag samples

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



1

Used for

- Out-of-bag error: evaluating at training time
- Variable importance

Variable importance

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

Let $\{\bar{\mathcal{S}}_n^t = \mathcal{S}_n - \mathcal{S}_n^t, t = 1, \ldots, n_{tree}\}$ **out-of-bag samples**

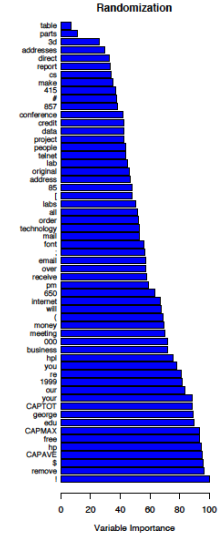
Let $\{\bar{\mathcal{S}}_n^{t,j}, t = 1, \ldots, n_{tree}\}$: permuted out-of-bag-samples (the values of the jth variable have been randomly permuted).

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

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

Variable importance: spam data

Spam dataset :



Boosting: 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

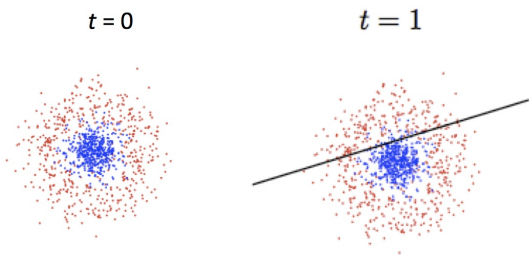
Idea: train a sequence of learners on weighted datasets with weights depending on the loss obtained so far.

Freund and Schapire received the Gödel prize in 2003 for their work on AdaBoost.

A **weak classifier** is 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...)

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

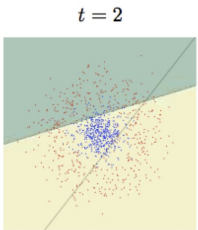


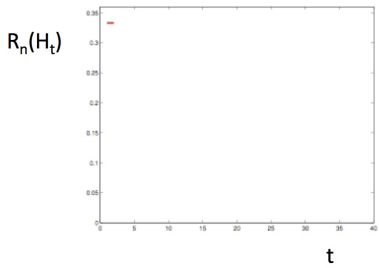
Source Jiri Matas (Oxford U.)

Boosting a linear classifier

$H_2(x) = \alpha_1 h_1(x) + \alpha_2 h_2(x)$

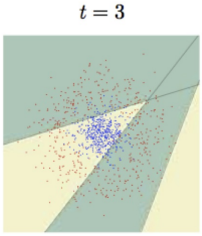
Binary Classifier: $F_2(x) = \text{sign}(H_2(x))$



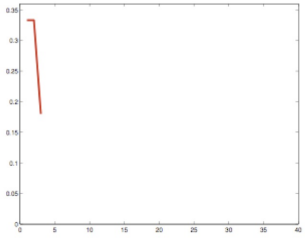


Source Jiri Matas (Oxford U.)

Boosting a linear classifier



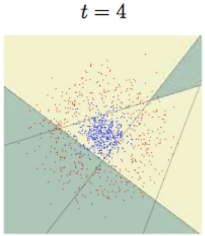
$R_n(H_t)$

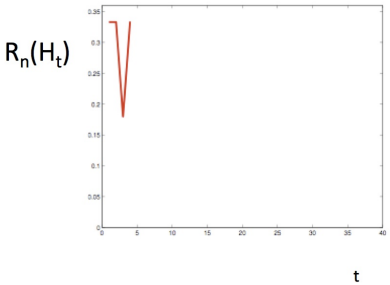


t

Source Jiri Matas (Oxford U.)

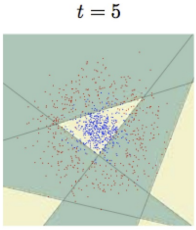
Boosting a linear classifier

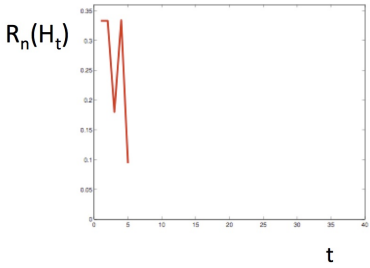




Source Jiri Matas (Oxford U.)

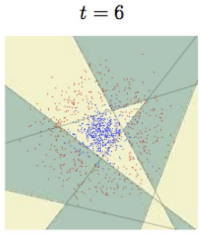
Boosting a linear classifier



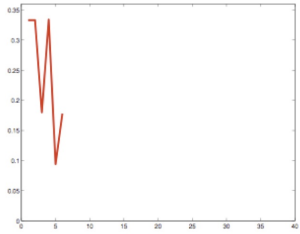


Source Jiri Matas (Oxford U.)

Boosting a linear classifier

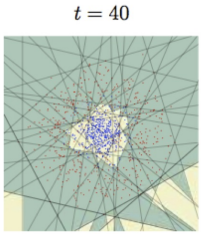


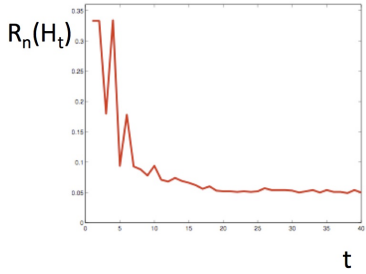
$R_n(H_t)$



Source Jiri Matas (Oxford U.)

Boosting a linear classifier





Source Jiri Matas (Oxford U.)

$H_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$

Greedy version

AdaBoost (Freund and Schapire 1996)

$\mathcal{H}$: a chosen class of "weak" binary classifiers. Learn a sequence of classifiers for weighted samples

---

Données : $l, \alpha, \{(x_i, y_i)\}$
Résultat : AdaBoost model
$H_0 = 0$
$w_i = 1/n \quad \forall i$
pour t *in* $\{1, \ldots, T\}$ **faire**

- $h = \arg\min_{h \in \mathcal{H}} \sum_{i:h(x_i) \neq y_i} w_i$ `/* which algorithm? */`
- $\epsilon \leftarrow \sum_{i:h(x_i) \neq y_i} w_i$ `/* Training error, for a weak learner` $\epsilon > .5$ `*/`
- $\alpha = \frac{1}{2} \log \frac{1-\epsilon}{\epsilon}$
- $H_t = H_{t-1} + \alpha h$
- $w_i \leftarrow \frac{w_i \exp(-\alpha h(x_i) y_i)}{2\sqrt{\epsilon(1-\epsilon)}}$

retourner H_{t-1}

Can we use trees here?

Give the prediction rule

AdaBoost is a forward stage-wise additive model

An additive model can be written as $H(x) = \sum_t \alpha_t h_t(x)$

Typically, these models are learnt by sequentially adding new models without updating those already learnt. Each new model minimizes a loss L, for $t = 1..T$ as

$$(\alpha_t, h_t) = \arg\min_{\alpha,h} L(y_i, H_{t-1}(x) + \alpha h(x))$$

and set $H_t = H_{t-1}(x_i) + \alpha_t h_t(x_i)$.

This is a greedy approach

Theorem: AdaBoost is a forward stage-wise additive model for the exponential loss $L(y, h(x)) = \exp(-yh(x))$.

A general theory: gradient boosting

Iterative strategy. Given a training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ minimizes the empirical risk.

$H_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$, as initialization, asn then at each iteration,

$H_t(x) = H_{t-1}(x) + (\arg\min_{h_t \in \mathcal{H}} [\sum_{i=1}^n L(y_i, H_{t-1}(x_i) + h_t(x_i))])(x)$, for $t \geq 1$, where $h_t \in \mathcal{H}$ is a base learner function.

Steepest descent step to this minimization problem (functional gradient descent) is equivalent to choosing

$H_t(x) = H_{t-1}(x) - \gamma \sum_{i=1}^n \nabla_{H_{t-1}} L(y_i, H_{t-1}(x_i))$

where $\gamma > 0$. For small γ, this implies that $L(y_i, H_t(x_i)) \leq L(y_i, H_{t-1}(x_i))$.

A general theory: gradient boosting

To prove the following, consider the objective

$$\sum_{i=1}^{n} L(y_i, H_{t-1}(x_i) + h_t(x_i))$$

Doing a Taylor expansion around the fixed point $H_{t-1}(x_i)$ up to first order:

$$\sum_{i=1}^{n} \left(L(y_i, H_{t-1}(x_i)) + h_t(x_i) \nabla_{H_{t-1}} L(y_i, H_{t-1}(x_i)) + \ldots \right)$$

Differentiating with respect to $h_t(x_i)$, only the derivative of the second term remains $\nabla_{H_{t-1}} L(y_i, H_{t-1}(x_i))$.

A general theory: gradient boosting

We can optimize γ by finding the γ value for which the loss function has a minimum:

$$\gamma_t = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, H_t(x_i)) = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, H_{t-1}(x_i) - \gamma \nabla_{H_{t-1}} L(y_i, H_{t-1}(x_i))\right).$$

If we considered the continuous case, i.e., where $\mathcal{H}$ is the set of arbitrary differentiable functions on $\mathbb{R}$, we would update the model in accordance with the following equations:

$$H_t(x) = H_{t-1}(x) - \gamma_t \sum_{i=1}^{n} \nabla_{H_{t-1}} L(y_i, H_{t-1}(x_i))$$

where γ_t is the step length, defined as:

$$\gamma_t = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, H_{t-1}(x_i) - \gamma \nabla_{H_{t-1}} L(y_i, H_{t-1}(x_i))\right).$$

A general theory: gradient boosting

Algorithme : AnyBoost Gradient Boosting Algorithm

Données : Training set $\{(x_i, y_i)\}$, a differentiable loss function $L(y, H(x))$, number of iterations M

Résultat : AnyBoost model

Initialize model with a constant value: $H_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$

pour $t = 1$ *to* T **faire**

$$r_{it} = -\left[\frac{\partial L(y_i, H(x_i))}{\partial H(x_i)}\right]_{H(x)=H_{t-1}(x)} \text{ for } i = 1, \ldots, n$$

$$h_t(x) = \arg\min \sum_i r_i h(x_i)$$

$$\gamma_t = \arg\min_\gamma \sum_{i=1}^n L\left(y_i, H_{t-1}(x_i) + \gamma h_t(x_i)\right)$$

$$H_t(x) = H_{t-1}(x) + \gamma_t h_t(x)$$

retourner H_M

Prop 1: AdaBoost is a gradient descent algorithm in the functional space

Theorem: AdaBoost is a gradient descent algorithm in the functional space where the exponential loss $l(H) = \sum_{i=1}^{n} \exp(-y_i H(x_i))$ and α is obtained via line search

Part 1: What is the next classifier $h_{t+1} = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{n} r_i * h(x_i)$?

Let

$w_i = Z^{-1} \exp(-y_i H(x_i))$ be relative contribution of the point i to the loss,

$Z = \sum_{i=1}^{n} \exp(-y_i H(x_i))$ a normalizing constant,

$r_i = -y_i \exp(-y_i H(x_i))$ the gradient of the loss

Note that Z is the loss!

AdaBoost is a gradient descent algorithm in the functional space

Theorem: AdaBoost is a gradient descent algorithm in the functional space

$$
\begin{aligned}
h_{t+1} &= \arg\min_h \sum_{i=1}^{n} r_i h(x_i) = \arg\min_h \sum_{i=1}^{n} y_i \exp(-y_i H(x_i)) h(x_i) \\
&= \arg\min_h \sum_{i=1}^{n} y_i w_i h(x_i) = \arg\min_h \sum_{i:y_i \neq h(x_i)} w_i - \sum_{i:y_i = h(x_i)} w_i \qquad (5) \\
&= \arg\min_h \sum_{i:y_i \neq h(x_i)} w_i = \arg\min_h \epsilon
\end{aligned}
$$

Conclusion: the next classifier h_{t+1} is the one that minimizes the weighted classification error.

AdaBoost is a gradient descent algorithm in the functional space

Part 2: What is the step size (via line search)?

$$\alpha = \arg\min_{\alpha} \ell(H + \alpha h) = \arg\min_{\alpha} \sum_{i=1}^{n} e^{-y_i[H(x_i)+\alpha h(x_i)]}$$

We differentiate w.r.t. α and equate with zero:

AdaBoost, updates

At each step, the weights are updated to the weighted classification error:

$$w_i \leftarrow \frac{w_i \exp(-\alpha h(x_i) y_i)}{2\sqrt{\epsilon(1-\epsilon)}}$$

The normalizitaion constant $Z \leftarrow Z 2\sqrt{\epsilon(i-\epsilon)}$ since

$$\begin{aligned} Z_s =& \sum_{i=1}^{n} w_t(i) e^{-\alpha_t y_i h_t(x_i)} \\ =& \sum_{i:y_i h_t(x_i)=+1} w_t(i) e^{-\alpha_t} + \sum_{i:y_i h_t(x_i)=-1} w_t(i) e^{\alpha_t} \\ =& (1-\epsilon_t) e^{-\alpha_t} + \epsilon_t e^{\alpha_t} \\ =& 2\sqrt{\epsilon_t(1-\epsilon_t)} \end{aligned} \tag{6}$$

AdaBoost, convergence

Rem: The loss equals the normalization constant, $l(H) = Z$. We use this observation to bound the loss.

$$l(H) = n \prod_{i=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

Let $c = \max t\epsilon_t$, then

$$l(H) \leq n(2\sqrt{c(1-c)})^T$$

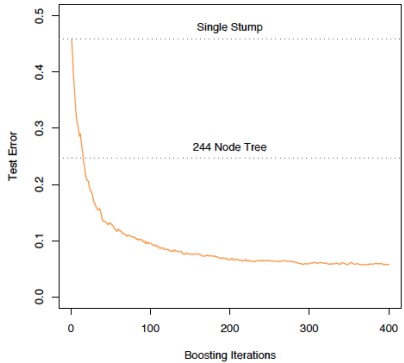
Since $\epsilon_t < .5$ for weak classifiers then $c(1-c) < .25$ or equivalently, $c(1-c) = .25 - \gamma^2$ and thus,

$$l(H) \leq n(1 - 4\gamma^2)^{T/2})$$

.

Typical behavior of boosting

Stump: a two terminal node classification tree



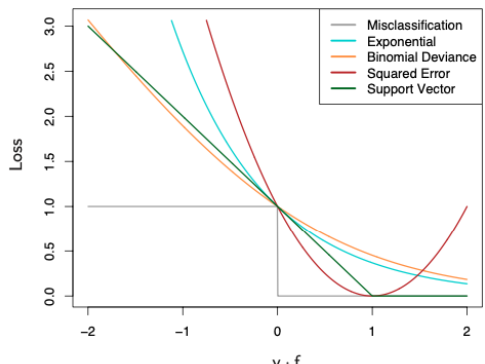
Boosting and regularization

- 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

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...



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

Exercises

- Write a detailed pseudocode for the regression trees for a maximum depth of d
- Code the k-fold cross validation for decision trees for obtaining the best *maximum tree depth* among the values 5,10,15 20
- Define a regularization function for trees.
- Give pros and cons of all algithms
- Describe all algorithms seen today

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.**

Temporary page!

LaTeX was unable to guess the total number of pages correctly. As there was some unproces
data that should have been added to the final page this extra page has been added to receiv

If you rerun the document (without altering it) this surplus page will go away, because LaTeX
now knows how many pages to expect for this document.