Ensemble Methods (Machine Learning, SIR)

Matthieu Geist (CentraleSupélec)
matthieu.geist@centralesupelec.fr

- (base) learner: predictions are made based on an estimator built with a given learning algorithm
- ensemble methods: improve generalization and robustness by combining the predictions of several base learners

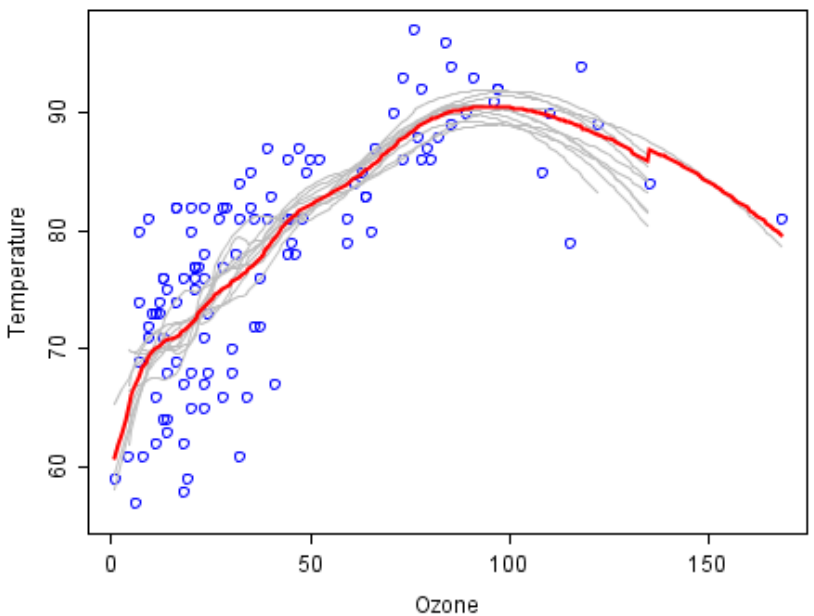


Figure : An illustration of bagging.

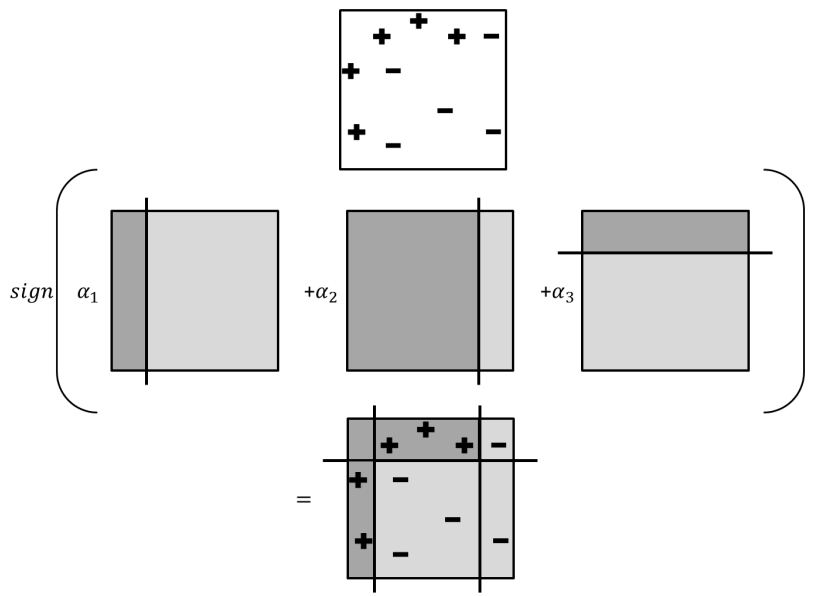
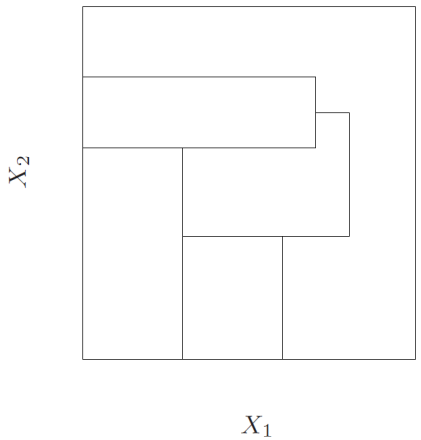
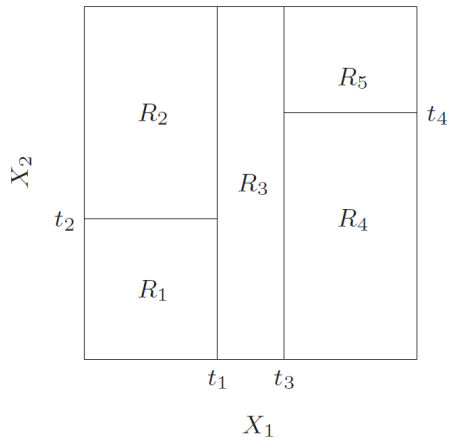


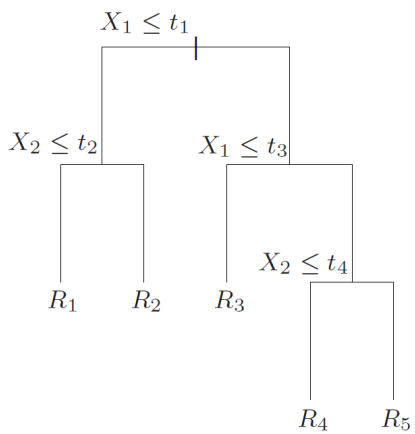
Figure : An illustration of boosting.

Basic idea

$$f(x) = \sum_{m=1}^{5} c_m \mathbb{I}_{\{x \in R_m\}}.$$







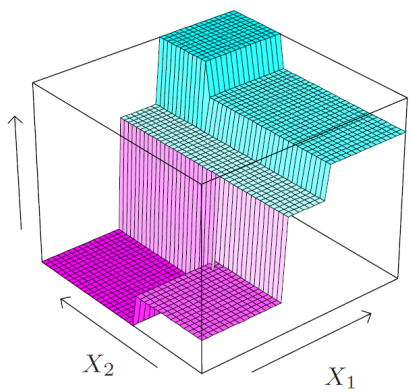


Figure : Illustration of a regression tree.

Building a regression tree

- Assume the partition $(R_1, \ldots, R_M)$ is fixed, the regression model associates a constant to each leaf:

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{I}_{\{x \in R_m\}}$$

- Minimize the empirical risk based on the ℓ_2-loss:

$$\min_{c_1,\ldots,c_M} \mathcal{R}_n(f) = \min_{c_1,\ldots,c_M} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2.$$

- Solution obtained by setting the gradient to zero:

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m) = \frac{\sum_{i=1}^{n} y_i \mathbb{I}_{\{x_i \in R_m\}}}{\sum_{i=1}^{n} \mathbb{I}_{\{x_i \in R_m\}}}.$$

Building a regression tree (cont.)

- How to find the best binary partition in term of $\mathcal{R}_n$?
  - in general, computationally infeasible;
  - idea: use a (heuristic) greedy approach.
- Let $\mathcal{D}$ be the dataset:

$$\mathcal{D} = \{(x_i, y_i)_{1 \leq i \leq n}\} \text{ with } x_i = (x_{i,1}, \ldots, x_{i,d})^\top \in \mathbb{R}^d.$$

- For a splitting variable j and a split point s, define the pair of half planes

$$R_1(j,s) = \{x_i \in \mathcal{D} : x_{i,j} \leq s\} \text{ and } R_2(j,s) = \{x_i \in \mathcal{D} : x_{i,j} > s\}.$$

- Then, search for (j,s) solving

$$\min_{j,s} \left(\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right).$$

Building a regression tree (cont.)

- There is a finite number of (j, s) couples:
  - d splitting variables: $1 \leq j \leq d$;
  - $n - 1$ split points for each splitting variable j: obtained by ordering the j^{th} components.
- The inner minimization problem is easily solved for any (j, s):

$$\hat{c}_1 = \text{ave}(y_i | x_i \in R_1(j, s)) \text{ and } \hat{c}_2 = \text{ave}(y_i | x_i \in R_2(j, s)).$$

- Therefore, by scanning through each dimension of all the inputs, determination of the best pair (j, s) is feasible.
- Then, having found the best split, we partition the data into the two resulting regions and repeat the splitting procedure for each of these regions.
- And so on, until a stopping criterion is reached.

Building a regression tree (cont.)

- Possible stopping criterion:
  - max. depth;
  - max. number of leaves;
  - max. number of samples per leaf;
  - etc.
- Stopping criterion is important:
  - a very large tree will overfit the data (consider a tree with as many leaves as samples);
  - a too small tree might not capture the important structure (*e.g.*, decision stump, tree with two nodes).

Building a classification tree

- classification tree
  - $y \in \{1, \ldots, K\}$;
  - the ℓ_2-loss is not the best choice, we should consider another criteria for splitting.
- For a node m representing a region R_m, consider:
  - the number of data points $n_m = \sum_{i=1}^{n} \mathbb{I}_{\{x_i \in R_m\}}$;
  - the associated dataset $\mathcal{D}_m = \{(x_i, y_i) \in \mathcal{D} : x_i \in R_m\}$;
  - the proportion of class k observations in node m

$$\hat{p}_{m,k} = \frac{1}{n_m} \sum_{x_i \in R_m} \mathbb{I}_{\{y_i = k\}};$$

  - the majority class $k(m) = \text{argmax}_{1 \leq k \leq K} \hat{p}_{m,k}$.
- in regression, partitioning was based on the node impurity

$$Q(\mathcal{D}_m) = \frac{1}{n_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2,$$

what node impurity should one choose for classification?

Building a classification tree (cont.)

- misclassification error,

$$Q(\mathcal{D}_m) = \frac{1}{n_m} \sum_{x_i \in R_m} \mathbb{I}_{\{y_i \neq k(m)\}}$$

$$= 1 - \hat{p}_{m,k(m)};$$

- Gini index,

$$Q(\mathcal{D}_m) = \sum_{k \neq k'} \hat{p}_{m,k} \hat{p}_{m,k'}$$

$$= \sum_{k=1}^{K} \hat{p}_{m,k}(1 - \hat{p}_{m,k});$$

- cross-entropy,

$$Q(\mathcal{D}_m) = -\sum_{k=1}^{K} \hat{p}_{m,k} \ln \hat{p}_{m,k}$$

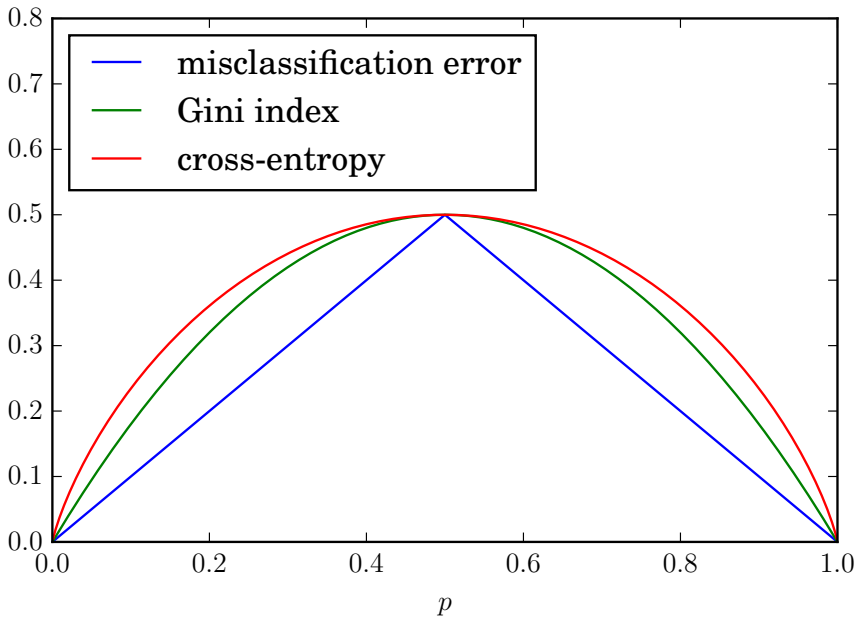


Figure : Measure of node impurity for binary classification, as a function of the proportion p in the second class.

Building a classification tree (cont.)

- The tree is grown as previously.
- For a region R_m:
  - consider a couple (j, s) of splitting variable and split point;
  - write $\mathcal{D}_{m,L}(j, s)$ the resulting dataset of the left node (of size n_{m_L});
  - write $\mathcal{D}_{m,R}(j, s)$ the dataset of the right node (of size n_{m_R}).
- The tree is growth by solving

$$\min_{j,s} \left(n_{m_L} Q(\mathcal{D}_{m,L}(j, s)) + n_{m_R} Q(\mathcal{D}_{m,R}(j, s)) \right).$$

Summary

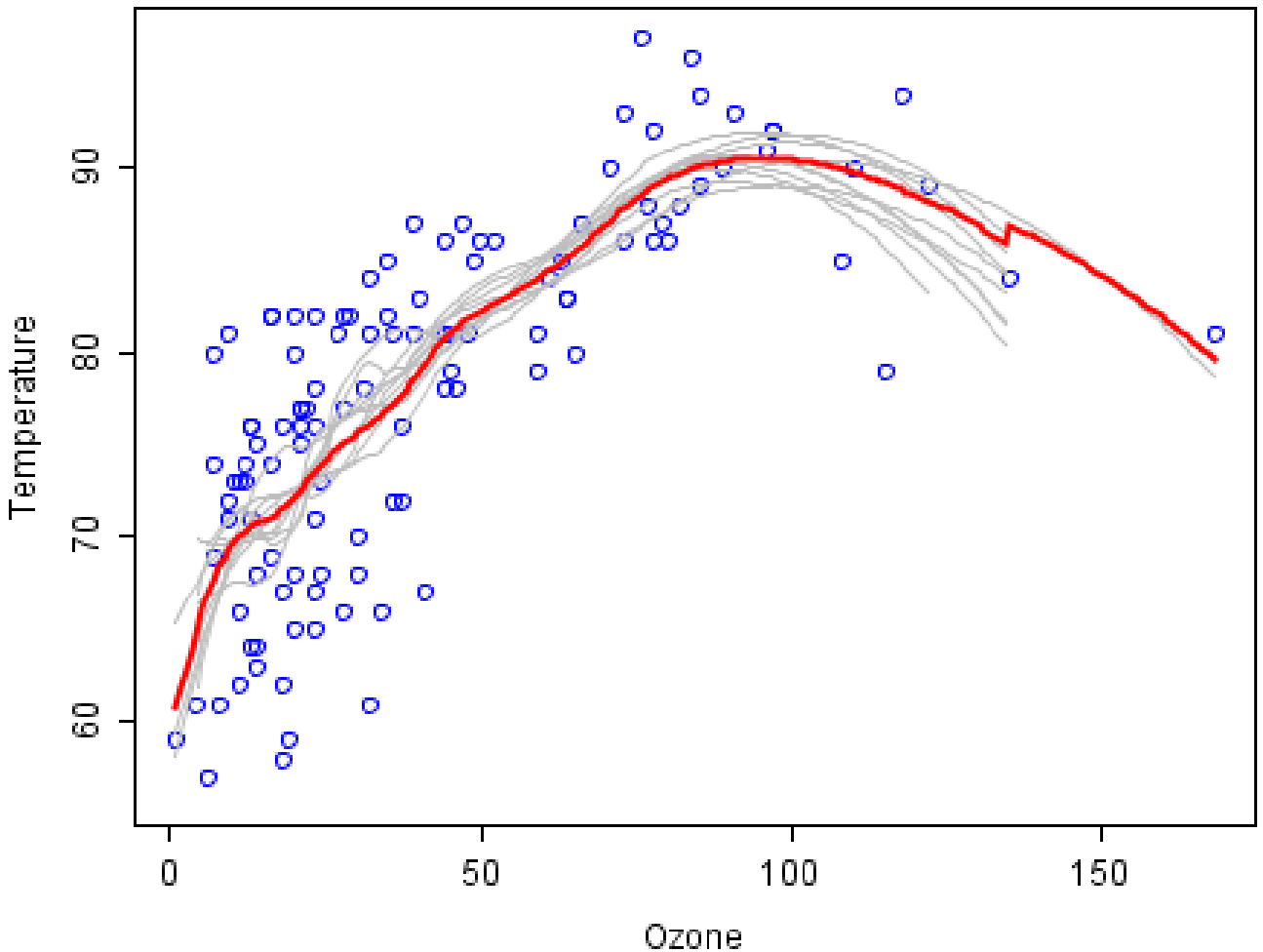
- Trees are interpretable predictors, quite easy to train.
- Other advantages:
  - can handle categorical values;
  - can be extended to cost-sensitive learning;
  - can handle missing input values.
- However, trees are unstable:
  - they have a high variance: a small change in the data can result in a very different tree;
  - mainly caused by the hierarchical nature of the process (the effect of an error is propagated down);
  - the smaller the tree, the more stable it is, but the weaker the learner is.

- bagging (or more generally averaging methods):
  - build in parallel several estimators, more or less independently, and average their predictions;
  - reduce the variance for the combined estimator;
  - applicable for example to deep trees.
- boosting:
  - build sequentially (weak) base estimators such as reducing the bias of the combined estimators;
  - motivation: combine weak learners to form a strong learner;
  - applicable for example to decision stumps.
- There exist other ensemble methods (Bayesian model averaging, mixture of expert or stacking, for exampe), not addressed here.

Motivation

- Abstract problem:
  - let $X_1, \dots, X_B$ be i.i.d. random variables, of mean $\mu = \mathrm{E}[X_1]$ and of variance $\sigma^2 = \mathrm{var}(X_1) = \mathrm{E}\left[(X_1 - \mu)^2\right]$;
  - consider the empirical mean $\mu_B = \frac{1}{B}\sum_{b=1}^{B} X_b$;
  - the expectation does not change, $\mathrm{E}[\mu_B] = \mu$, while the variance is reduced, $\mathrm{var}(\mu_B) = \frac{1}{B}\sigma^2$.
- Supervised learning:
  - the random quantity is the dataset $\mathcal{D} = \{(x_i, y_i)_{1 \leq i \leq n}\}$;
  - from this, an estimate $f_{\mathcal{D}}$ is computed by minimizing the empirical risk of interest.
- Ideal case:
  - let $\mathcal{D}_1, \dots, \mathcal{D}_B$ be datasets drawn independently, and write $f_b = f_{\mathcal{D}_b}$ the associated minimizer;
  - define $f_{\mathrm{ave}} = \frac{1}{B}\sum_{b=1}^{B} f_b$;
  - this does not change the expectation, $\mathrm{E}[f_{\mathrm{ave}}] = \mathrm{E}[f_1]$, but it reduces the variance, $\mathrm{var}(f_{\mathrm{ave}}) = \frac{1}{B}\mathrm{var}(f_1)$.
- Unfortunately, it is not possible to sample datasets on demand...

Illustration



- bagging = bootstrap + aggregating
- bootstrap = random sampling with replacement (statistics)

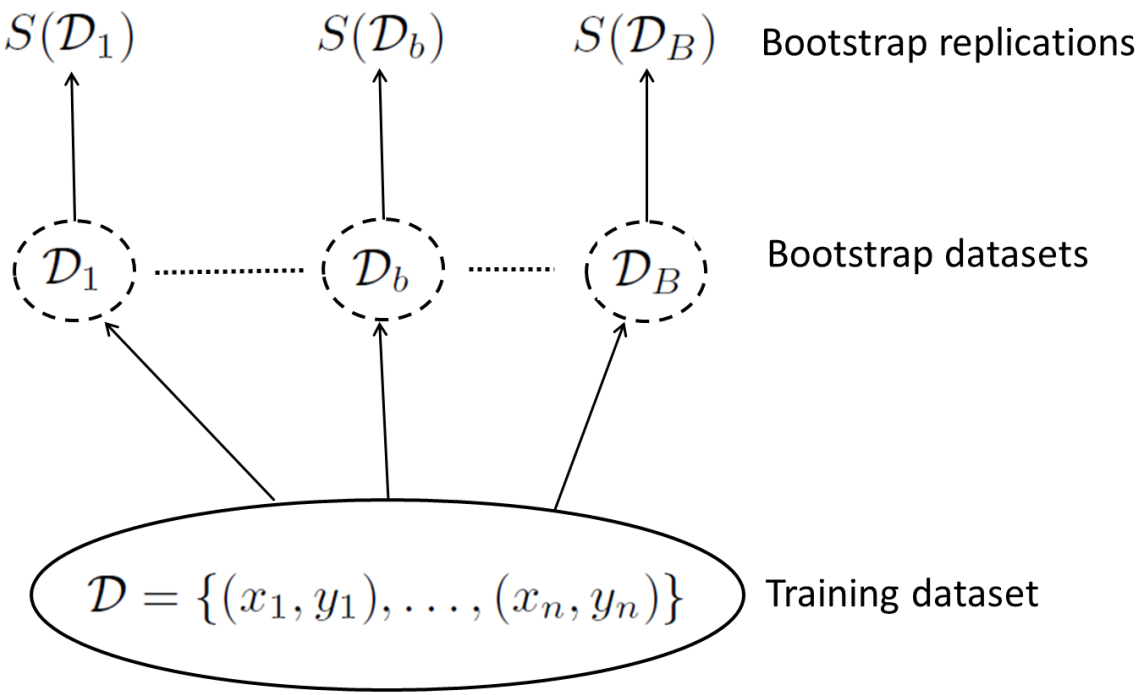


Figure : Illustration of the bootstrapping principle.

- bagging:
  - bootstrap the dataset to get B datasets $\mathcal{D}_b$ (same size as $\mathcal{D}$), $1 \leq b \leq B$;
  - learn a predictor $f_{\mathcal{D}_b} = f_b$ for each of these datasets;
  - average the predictors:

$$f_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} f_b(x)$$

- Apply directly for regression.
- For classification, averaging the predicted classes would not make sense. Possible strategies:
  - do a majority vote over trees:

$$f_{\text{bag}} = \underset{1 \leq i \leq K}{\text{argmax}} \left(\frac{1}{B} \sum_{b=1}^{B} \mathbb{I}_{\{f_b(x)=k\}} \right).$$

  - consider the class proportion for the leave corresponding to the input of interest ($\hat{p}_{m,k}$, with $x \in R_m$) for each tree, average them over all trees and output the class that maximizes this averaged class proportion.

Variations of the bagging approach

- Bagging: draw sample with replacement.
- Pasting: alternatively, random subsets of the dataset can be drawn as random subsets of the samples.
- Random subspaces: one can also selects randomly a subset of the components of the inputs (which is generally multi-dimensional) to learn models.
- Random patches: combination of the two previous methods.

Motivation

- There is necessarily some overlap between bootstrapped datasets, so the models corresponding to each of these datasets are correlated.
- Abstract problem
  - let $X_1, \ldots, X_B$ be i.d. but not independent random variables, of mean $\mu = \mathrm{E}[X_1]$ and of variance $\sigma^2 = \mathrm{var}(X_1) = \mathrm{E}\left[(X_1 - \mu)^2\right]$;
  - consider the empirical mean $\mu_B = \frac{1}{B}\sum_{b=1}^{B} X_b$;
  - assume a positive pairwise correlation ρ:

$$\text{for } i \neq j, \quad \rho = \frac{\mathrm{E}[(X_i - \mu)(X_j - \mu)]}{\sqrt{\mathrm{var}(X_i)\,\mathrm{var}(X_j)}}.$$

  - the expectation does not change, $\mathrm{E}[\mu_B] = \mu$, but the variance is now given by

$$\mathrm{var}(\mu_B) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

- Random forest: reduce the correlation between the trees by randomizing their constructions, without increasing the variance too much.

Algorithm 1 Random Forest

Require: A dataset $\mathcal{D} = \{(x_i, y_i)_{1 \leq i \leq n}\}$, the size B of the ensemble, the number m of candidates for splitting.
  for $b = 1$ **to** B **do**
    Draw a bootstrap dataset $\mathcal{D}_b$ of size n from the original training set $\mathcal{D}$.
    Grow a random tree using the bootstrapped dataset:
    repeat
      for all terminal node **do**
        Select m variables among d, at random.
        Pick the best variable and split-point couple among the m.
        Split the node into two daughter nodes.
      end for
    until the stopping criterion is met (*e.g.*, minimum number of sample per node reached)
  end for
  return the ensemble of B trees.

- Extremly randomized forest pushed further the randomization. The two differences with random forests are:
  - split-points are also chosen randomly, one for each random splitting dimension;
  - the full learning dataset is used for growing each tree.
- The rationale for:
  - choosing also the split-point at random is to further reduce the correlation between trees;
  - using the full learning set is to achieve a lower bias (at the price of an increased variance, that should be compensated by the randomization of split-points).

Algorithm 2 Extremely Randomized Forest

Require: A dataset $\mathcal{D} = \{(x_i, y_i)_{1 \leq i \leq n}\}$, the size B of the ensemble, the number m of candidates for splitting.

for $b = 1$ **to** B **do**

Grow a random tree using the original dataset:

repeat

for all terminal node **do**

Select m variables among d, at random.

for all sampled variables **do**

Select a split at random

end for

Pick the best variable and split-point couple among the m candidates.

Split the node into two daughter nodes.

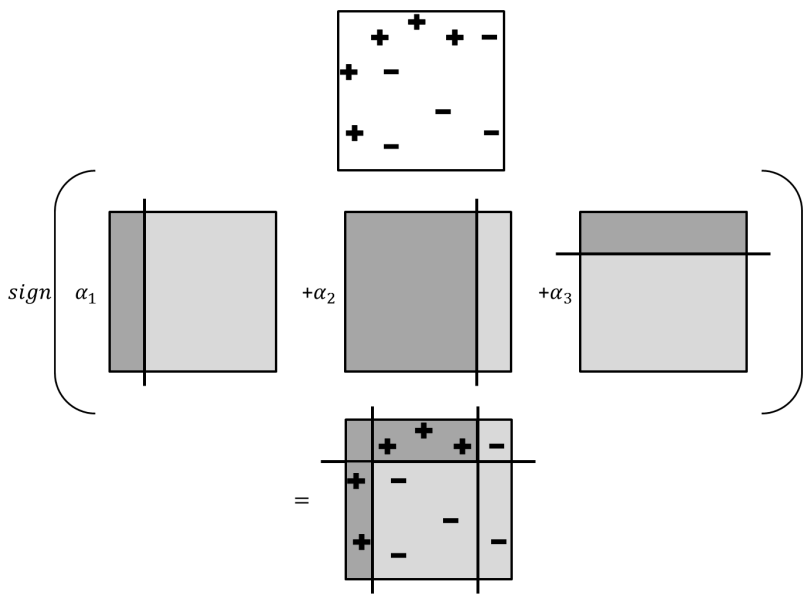
end for

until the stopping criterion is met (*e.g.*, minimum number of sample per node reached)

end for

return the ensemble of B trees.

- Boosting takes a different route from bagging:
  - bagging: learning in parallel a set of models with low bias and high variance (learning being randomized, for example through bootstrapping), the prediction being made by averaging all these models.
  - boosting: add sequentially models with high bias and low variance such as reducing the bias of the ensemble



- For boosting, we will focus on binary classification.

Weighted binary classification

- We consider binary classification:
  - the available dataset is of the form $\mathcal{D} = \{(x_i, y_i)_{1 \leq i \leq n}\}$ with $y_i \in \{-1, +1\}$;
  - the empirical risk of interest is

$$\mathcal{R}_n(f) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}_{\{y_i \neq f(x_i)\}} = \sum_{i=1}^{n} \frac{1}{n} \mathbb{I}_{\{y_i \neq f(x_i)\}};$$

  - all samples have the same importance (each sample has a weight $\frac{1}{n}$)
- Weighted binary classification:
  - we want to associate a different weight w_i to each example (x_i, y_i), such that $\sum_i w_i = 1$ and $w_i \geq 0$;
  - the corresponding empirical risk is

$$\mathcal{R}_n(f) = \sum_{i=1}^{n} w_i \mathbb{I}_{\{y_i \neq f(x_i)\}}.$$

Weighted binary classification (cont.)

- For minimizing the weighted empirical risk:
  - sample a bootstrap replicate according to the discrete distribution $(w_1, \ldots, w_n)$ and minimize the empirical risk on this new dataset;
  - do it more directly, in a problem dependent manner.
- The case of classification trees:
  - the node impurity can be easily adapted
  - for example, for the misclassification error, consider:

$$Q(\mathcal{D}_m) = \sum_{x_i \in R_m} w_i \mathbb{I}_{\{y_i \neq k(m)\}}.$$

In the sequel, we assume that a (weak) base learner is available (typically, a decision stump), and that it is able to minimize the weighted risk using the dataset $\mathcal{D}$ and weights w_i, $1 \leq i \leq n$.

Algorithm 3 The AdaBoost (Adaptive Boosting) algorithm

Require: A dataset $\mathcal{D} = \{(x_i, y_i)_{1 \leq i \leq n}\}$, the size T of the ensemble.

1: Initialize the weights $w_i^1 = \frac{1}{n}$, $1 \leq i \leq n$

2: **for** $t = 1$ **to** T **do**

3: Fit a binary classifier $f_t(x)$ to the training data using weights w_i^t.

4: Compute the error ϵ_t made by this classifier:

$$\epsilon_t = \sum_{i=1}^{n} w_i^t \mathbb{I}_{\{y_i \neq f_t(x_i)\}}.$$

5: Compute the learning rate α_t:

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

6: Update the weights, for all $1 \leq i \leq n$:

$$w_i^{t+1} = \frac{w_i^t e^{-\alpha_t y_i f_t(x_i)}}{\sum_{j=1}^{n} w_j^t e^{-\alpha_t y_j f_t(x_j)}}.$$

7: **end for**

8: **return** the decision rule

$$H_T(x) = \operatorname{sgn}\left(F_T(x)\right) \text{ with } F_T(x) = \sum_{t=1}^{T} \alpha_t f_t(x).$$

Forward stagewise additive modeling

- A convex surrogate for binary classification is the risk based on the exponential loss:

$$\mathcal{R}_n(F) = \frac{1}{n}\sum_{i=1}^{n} e^{-y_i F(x_i)} \text{ with } F \in \mathbb{R}^{\mathcal{X}}.$$

- We're looking for an additive model of the form:

$$F_T(x) = \sum_{t=1}^{T} \alpha_t f_t(x) \text{ with } f_t \in \{-1,+1\}^{\mathcal{X}} \text{ and } \alpha_t \in \mathbb{R}.$$

- The corresponding optimization problem is therefore:

$$\min_{(\alpha_t, f_t)_{1\leq t\leq T}} \frac{1}{n}\sum_{i=1}^{n} e^{-y_i \sum_{t=1}^{T} \alpha_t f_t(x_i)}.$$

- Yet, this optimization problem is too complicated.

Forward stagewise additive modeling (cont.)

- A simple alternative is to search for an approximate solution by sequentially adding basis functions and associate weights:
  - define $F_0 = 0$;
  - define $F_t = F_{t-1} + \alpha_t f_t$;
  - solve sequentially the subproblems

$$\min_{\alpha, f} \frac{1}{n} \sum_{i=1}^{n} e^{-y_i (F_{t-1}(x_i) + \alpha f(x_i))}.$$

- This approach:
  - is reminiscent of gradient descent (with line search);
  - can be straightforwardly abstracted to any loss function L:

$$\min_{\alpha, f} \frac{1}{n} \sum_{i=1}^{n} L(y_i, F_{t-1}(x_i) + \alpha f(x_i)).$$

Forward stagewise additive modeling (cont.)

- At each iteration $t \geq 1$, we have to solve

$$(\alpha_t, f_t) = \underset{\alpha, f}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} e^{-y_i(F_{t-1}(x_i) + \alpha f(x_i))}.$$

- The solution is given by

$$f_t = \underset{f \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} w_i^t \mathbb{I}_{\{y_i \neq f(x_i)\}} \text{ with } w_i^t = \frac{e^{-y_i F_{t-1}(x_i)}}{\sum_{j=1}^{n} e^{-y_j F_{t-1}(x_j)}}$$

$$\text{and } \alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right) \text{ with } \epsilon_t = \sum_{i=1}^{n} w_i^t \mathbb{I}_{\{y_i \neq f_t(x_i)\}}.$$

- This is the AdaBoost Algorithm!

Bounding the empirical risk

- Define the edge $\gamma_t = \frac{1}{2} - \epsilon_t$, which measures how much better than random guessing (error rate of $\frac{1}{2}$) is the error rate of the t^{th} learned classifier f_t
- how good is the ensemble $H_T(x) = \text{sgn}(F_T(x))$ compared to a single weak learner?

Theorem

Write $\gamma_t = 1 - \epsilon_t$ the edge of the t^{th} classifier. The empirical risk of the combined classifier H_T produced by AdaBoost satisfies

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}_{\{y_i \neq H_T(x_i)\}} \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq e^{-2\sum_{t=1}^{T} \gamma_t^2}.$$

Rationale

- Consider the binary classification problem with a convex surrogate:
  - we're looking for a classifier $H(x) = \text{sgn}(F(x))$ with $F \in \mathbb{R}^{\mathcal{X}}$;
  - let $L(y, F(x))$ be a convex surrogate to the binary loss (for example, the exponential loss, $L(y, F(x)) = e^{-yF(x)}$);
  - we would like to minimize the empirical risk:

$$\min_{F \in \mathbb{R}^{\mathcal{X}}} \mathcal{R}_n(F) \text{ with } \mathcal{R}_n(F) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, F(x_i)).$$

- Convex optimization:
  - as a sum of convex function, $\mathcal{R}_n$ is convex in F;
  - a standard approach for minimizing a convex function is to perform a gradient descent:

$$F_{t+1} = F_t - \alpha_t \nabla_F \mathcal{R}_n(F_t).$$

- Problem: F is a function, what is ∇_F? Is it usable?

Functional gradient and related Hilbert space

- Assume that $\mathcal{X}$ is measurable and let μ be a probability measure.
- Define $L^2(\mathcal{X}, \mathbb{R}, \mu)$ the set of all equivalence classes of functions $F \in \mathbb{R}^{\mathcal{X}}$ such that the Lebesgue integral $\int_{\mathcal{X}} F(x)^2 d\mu(x)$ is finite.
- This Hilbert space has a natural inner product: $\langle F, G \rangle_\mu = \int_{\mathcal{X}} F(x)G(x)d\mu(x)$.
- For a functional $J : L^2(\mathcal{X}, \mathbb{R}, \mu) \to \mathbb{R}$, the Fréchet derivative is the linear operator $\nabla J(F)$ satisfying

$$lim_{G \to 0} \frac{J(F+G) - J(F) - \langle \nabla J(F), G \rangle_\mu}{\|G\|_\mu} = 0.$$

Back to risk minimization

- The probability measure we're interested in here is the discret measure that associates a probability $\frac{1}{n}$ to each input x_i. The associated inner product is

$$\langle F, G \rangle_n = \frac{1}{n} \sum_{i=1}^{n} F(x_i) G(x_i).$$

- We can compute the Fréchet derivative, so we can write the gradient descent:

$$F_{t+1} = F_t - \alpha_t \nabla_F \mathcal{R}_n(F_t).$$

- However, the Fréchet derivative is a set of function, known only in the datapoints x_i. It does not allows generalizing and it is not a practical object for computing. The idea is therefore to "restrict" this gradient to the hypothesis space $\mathcal{H}$ of interest, by looking for the function of $\mathcal{H}$ being the more collinear to the gradient (with a comparable norm)

$$f_t \in \underset{f \in \mathcal{H}}{\operatorname{argmax}} \frac{\langle \nabla_F \mathcal{R}_n(F_t), f \rangle_n}{\|f\|_n}.$$

- then, we replace the gradient by its approximation in the update rule:

$$F_t = F_{t-1} - \alpha_t f_t.$$

Application to the exponential loss

- Consider the risk based on the exponential loss:

$$\mathcal{R}_n(F) = \frac{1}{n}\sum_{i=1}^{n} e^{-y_i F(x_i)}.$$

- Restricting the gradient gives

$$\underset{f\in\mathcal{H}}{\operatorname{argmax}} \frac{\langle \nabla \mathcal{R}_n(F_t), f\rangle_n}{\|f\|_n} = \underset{f\in\mathcal{H}}{\operatorname{argmax}} \frac{1}{n}\sum_{i=1}^{n} e^{-y_i F_t(x_i)} \mathbb{I}_{\{y_i \neq f(x_i)\}} = -f_t$$

- The optimal learning rate can be obtained by performing a line search:

$$\alpha_t = \underset{\alpha>0}{\operatorname{argmin}}\, \mathcal{R}_n(F_t + \alpha f_t) = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

$$\text{with } \epsilon_t = \frac{\sum_{i=1}^{n} e^{-y_i F_t(x_i)} \mathbb{I}_{\{y_i \neq f_t(x_i)\}}}{\sum_{i=1}^{n} e^{-y_i F_t(x_i)}}.$$

- This is Adaboost!