Ensemble Methods in Machine Learning

Master 2 - MALIA

Guillaume Metzler

Université Lumière Lyon 2
Laboratoire ERIC, UR 3083, Lyon

guillaume.metzler@univ-lyon2.fr

Fall 2024

How are we going to work ? I

This course will be divided as follows :

- 5 **to** 6 **classes of 3 hours** on Machine Learning and more precisely on Ensemble Methods. We are going to study how to combine several models together in order to build stronger ones and try to explain how/why it works. If we have time, I will also speak about one of the following topics : metric learning and imbalanced learning, transfer learning and domain adaptation.
- **1-2 practical sessions** in order to apply/use the presented methods and be sure that you are able to conduct experiments in a perfect manner.

How are we going to work ? II

Evaluations

You will have two different evaluations :

- One exam : it will be used to test what you have learned andunderstood from the courses. I am going to ask several questions and provide some exercises.
- One project : the idea will be to apply most of the methods I have presented during this course. You will have to apply them and compare them in order to write a report which will look like a research report.

How are we going to work ? III

Material

The material of this course will be available on my website, at the following address :

Ressources

You will find :

- the slides of this course,
- a document with the basics in Machine Learning and also the content of this course (some and more information than you have in the slides),
- some exercises in order to practise what have been seen during the course,
- your project for this course.

Fundamentals in Machine Learning I

What do you remember ?

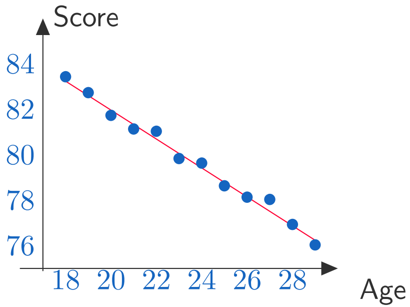
Tell me what you know about Machine Learning ?

Regression models

Regression Models I

Linear Regression

Linear regression are mainly used when we aim to predict a real value *e.g.* the price of an house or the score obtained at a given exam as illustrated after.



Regression Models II

in its simple formulation, the learned model takes the form of straight line. Given $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m \in \mathbb{R}^d \times \mathbb{R}$, we aim to find the best line which approximates the scatter plot, *i.e.* to learn a hypothesis h of the form :

$$h(\boldsymbol{\theta}, \mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d.$$

In the case of regression tasks we usually minimize the Mean Square Error (MSE).

Regression Models III

Proposition 2.1: Linear Regression

We consider the following probabilistic model for our data.

$$Y = \boldsymbol{\theta} X + \varepsilon,$$

where Y it the predicted variable and X is the set of variables that are used for the prediction and ε represents the error of the model.
We consider a hypothesis h of the form :

$$h(\boldsymbol{\theta}, \mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d.$$

Given a set S of m examples, $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_m)$ then the solution of Mean Square Error problem :

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{d+1}} \|\mathbf{y} - h(\boldsymbol{\theta}, \mathbf{X})\|_2^2 = \min_{\boldsymbol{\theta} \in \mathbb{R}^{d+1}} \sum_{i=1}^{m} (y_i - h(\boldsymbol{\theta}, \mathbf{x}_i))^2$$

is given by :

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}.$$

Regression Models IV

This first model is pretty simple and an analytical solution is available, meaning no training process is needed to learn the model.

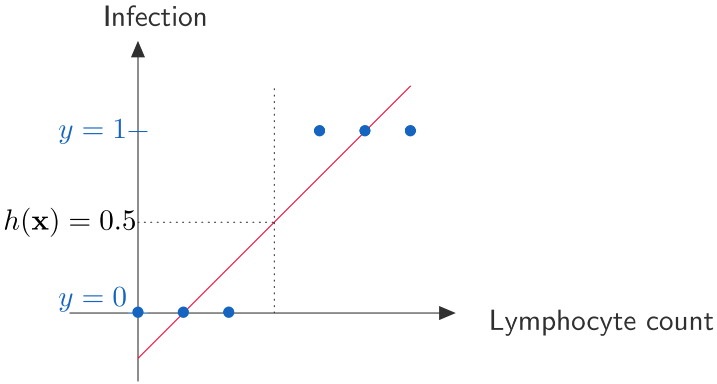
Let us place ourselves in a slightly different regression context now and consider the following example.

Regression Models V

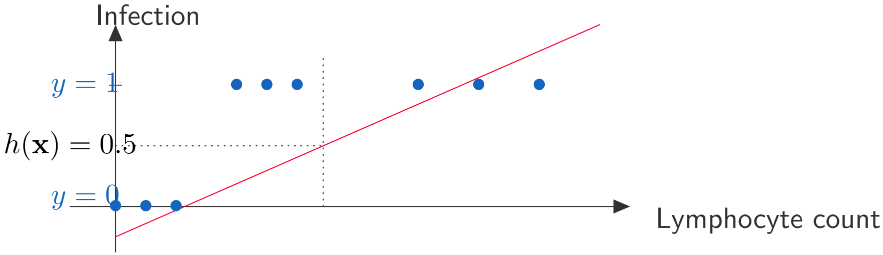
We are trying to build a regression model capable of determining whether or not an individual has an infection according to his lymphocyte count. The predicted variable can take two values : 1 if the person has an infection and 0 otherwise.

At first sight, nothing prevents us from learning a linear model to try to fit our new point cloud, as illustrated below.

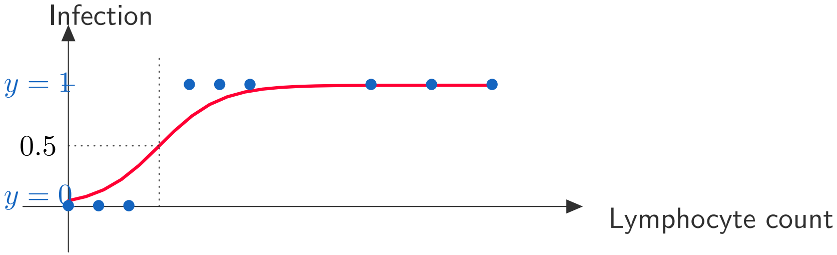
Regression Models VI



Regression Models VII



Regression Models VIII



Regression Models IX

Such a model takes its values in $[0, 1]$ and we can thus say that it estimates the probability of having an infection. To transform the values predicted by a linear regression model into probabilities, we use the logistic function, *i.e.* we compute :

$$\frac{1}{1 + \exp\left(-h(\mathbf{x})\right)}.$$

We talk about *linear logistic regression*.

Regression Models X

Linear Logistic Regression

The Logistic Regression model, also called the *logit model* has been introduced in the middle of the 20th century [Cox, 1958] but the use of *logit* models dates back to the end of the 19th century [Cramer, 2003].

This model is used to estimate the probability that an example belongs to a given class, for instance the positive class : $\eta = Pr(Y = 1 \mid X)$.

More precisely, the logistic regression aims to compute the logarithm of the *odds*, i.e. the ratio of the probabilities.
Then we estimate the log of this ratio using a linear model :

$$\ln\left(\frac{Pr(y=1 \mid \mathbf{x})}{Pr(y=0 \mid \mathbf{x})}\right) = h(\mathbf{w}, b, \mathbf{x}) = b + \langle \mathbf{x}, \mathbf{w} \rangle.$$

Regression Models XI

Thus, once the parameters of the model are learned, we can compute the probability of being in class 1 :

$$Pr(y = 1 \mid \mathbf{x}) = \frac{\exp(h(\mathbf{w}, b, \mathbf{x}))}{1 + \exp(h(\mathbf{w}, b, \mathbf{x}))} = \frac{1}{1 + \exp(-h(\mathbf{w}, b, \mathbf{x}))}.$$

Such function is called a *logistic function* and takes its values in $[0, 1]$.

An example $\mathbf{x}_i$ is (usually) predicted in class 1 if $Pr(y = 1 \mid x) > 0.5$, i.e. if $h(\mathbf{w}, b, \mathbf{x}) > 0$.

Regression Models XII

To estimate the parameters of the model, we maximize the likelihood of the data $\mathfrak{L}(\mathbf{w}, S)$, where S is a set of m examples.

$$\mathfrak{L}(\mathbf{w}, b, S) = \prod_{i=1}^{m} Pr(Y = y_i \mid X = \mathbf{x}_i),$$

$\downarrow$ separate $y_i = 0$ and $y_i = 1$

$$= \prod_{i=1, y_i=1}^{m} Pr(Y = y_i \mid X = \mathbf{x}_i) \times \prod_{i=1, y_i=0}^{m} Pr(Y = y_i \mid X = \mathbf{x}_i),$$

$\downarrow$ Use the fact that the underlying law is a Bernoulli law

$$= \prod_{i=1}^{m} \left(\frac{1}{1 + \exp(-h(\mathbf{w}, b, \mathbf{x}_i))} \right)^{y_i} \times \left(\frac{1}{1 + \exp(h(\mathbf{w}, b, \mathbf{x}_i))} \right)^{(1-y_i}$$

Regression Models XIII

Note that we usually prefer to minimize the negative log-likelihood of the data :

$$\begin{aligned}\ell(\mathbf{w}, b, S) = & -\ln\left(\mathfrak{L}(\mathbf{w}, b, S)\right), \\ = & -\sum_{i=1}^{m} y_i \ln\left(\frac{1}{1+\exp\left(-(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\right)}\right) \\ & + (1-y_i)\ln\left(1 - \frac{1}{1+\exp\left(-(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\right)}\right).\end{aligned}$$

Regression Models XIV

By doing so, we find the logistic loss function introduced before. In the following, for the sake of simplicity, we will set $g(\mathbf{w}, b, \mathbf{x}) = \frac{1}{1 + \exp\left(-(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\right)}$. Therefore, the optimization problem becomes :

$$\min_{\mathbf{w}, b \in \mathbb{R}^{d+1}} \quad -\frac{1}{m} \sum_{i=1}^{m} y_i \ln\left(g(\mathbf{w}, b, \mathbf{x}_i)\right) + (1 - y_i) \ln\left(1 - g(\mathbf{w}, b, \mathbf{x}_i)\right).$$

Regression Models XV

Some remarks :

- There is no analytical solution to this problem
- Use gradient descent algorithm to solve it : usually Newton's Method or an approximation such as BFGS
- Flexible with the use of the threshold (especially in imbalanced settings), if you want to tune specific measures.

Regression Models XVI

In order to avoid over-fitting, a regularization term of the form $\lambda\|\mathbf{w}\|$ is usually used in regression tasks. Thus, the optimization problem can be rewritten :

$$\min_{\mathbf{w},b\in\mathbb{R}^{d+1}} \quad -\frac{1}{m}\sum_{i=1}^{m} y_i \ln\left(g(\mathbf{w},b,\mathbf{x}_i)\right) + (1-y_i)\ln\left(1-g(\mathbf{w},b,\mathbf{x}_i)\right) + \lambda\|\mathbf{w}\|^2.$$

In the gaussian linear model, it can be written as :

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \|\mathbf{y}-h(\boldsymbol{\theta},\mathbf{X})\|_2^2 + \lambda\|\boldsymbol{\theta}\|^2 = \min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \sum_{i=1}^{m} \left(y_i - h(\boldsymbol{\theta},\mathbf{x}_i)\right)^2 + \lambda\|\boldsymbol{\theta}\|^2$$

Regression Models XVII

Multinomial Logistic Regression

Let us consider that the set of label is now $\{1, 2, \ldots, q\}$, where $q > 2$. For the sake of simplicity, we still write $\mathbf{w} = (w_0, w_1, \ldots, w_d)$ our vector of parameters.

In the binary case, we have built a linear model in order to estimate the log ratio of the probability of belonging in class 1 to the probability of belonging in class 0 (also called a *logit*).

Now, we have to do something similar but with more than two classes, we thus have to select one class as a reference class, let us say the class q, and we are going to build all the following models :

Regression Models XVIII

$$
\begin{aligned}
\ln\left(\frac{Pr(y=1 \mid \mathbf{x})}{Pr(y=q \mid \mathbf{x})}\right) &= \langle \mathbf{w}^{(1)}, \mathbf{x} \rangle, \\
\ln\left(\frac{Pr(y=2 \mid \mathbf{x})}{Pr(y=q \mid \mathbf{x})}\right) &= \langle \mathbf{w}^{(2)}, \mathbf{x} \rangle, \\
\cdots &= \cdots \\
\ln\left(\frac{Pr(y=q-2 \mid \mathbf{x})}{Pr(y=q \mid \mathbf{x})}\right) &= \langle \mathbf{w}^{(q-2)}, \mathbf{x} \rangle, \\
\ln\left(\frac{Pr(y=q-1 \mid \mathbf{x})}{Pr(y=q \mid \mathbf{x})}\right) &= \langle \mathbf{w}^{(q-1)}, \mathbf{x} \rangle.
\end{aligned}
$$

Regression Models XIX

Regarding this set of equations, it remains possible to provide the value of $Pr(y = k \mid \mathbf{x}), \forall k = 1, \ldots, q$. In fact, if we take the exponential and we sum all the equations and use the fact that $Pr(y = q \mid \mathbf{x}) = 1 - \sum_{k=1}^{q-1} Pr(y = k \mid \mathbf{x})$, then we have :

$$\frac{1 - Pr(y = q \mid \mathbf{x})}{Pr(y = q \mid \mathbf{x})} = \sum_{k=1}^{q-1} \exp\left(\langle \mathbf{w}^{(k)}, \mathbf{x} \rangle\right),$$

and thus

$$Pr(y = q \mid \mathbf{x}) = \frac{1}{1 + \sum_{k=1}^{q-1} \exp\left(\langle \mathbf{w}^{(k)}, \mathbf{x} \rangle\right)}.$$

Regression Models XX

Using this equality, we immediately have for all $k \in [\![1, q-1]\!]$,

$$Pr(y = k \mid \mathbf{x}) = \frac{\exp\left(\langle \mathbf{w}^{(k)}, \mathbf{x} \rangle\right)}{1 + \sum_{l=1}^{q-1} \exp\left(\langle \mathbf{w}^{(l)}, \mathbf{x} \rangle\right)}.$$

Several remarks about this model

- The defined probabilities sum to 1
- The number of parameters that we have to learn is equal to the number of parameters of a single binary logistic regression times number of classes minus one, *i.e.*, $(q-1) \times (d+1)$
- Note that the way model is built completely depends on the reference class, if we change it, the parameters have no reasons to be the same.

Regression Models XXI

Using this approach the learning process consists in learning $q-1$ binary logistic regression and then, at the prediction step, compute all the probabilities $Pr(y=k \mid \mathbf{x}), \forall k = 1, \ldots, q$. A new example $\mathbf{x}'$ is assigned to the class for which it has the highest probability of belonging, *i.e.*,

$$\hat{y} = \underset{k \in [\![1,q]\!]}{\arg\max}\, Pr(y=k \mid \mathbf{x}).$$

Regression Models XXII

To make our model independent from the reference class q, we can consider that each probabilities are given by :

$$Pr(y = k \mid \mathbf{x}) = \frac{\exp\left(\langle \mathbf{w}^{(k)}, \mathbf{x} \rangle\right)}{\sum_{l=1}^{q} \exp\left(\langle \mathbf{w}^{(l)}, \mathbf{x} \rangle\right)}, \quad \forall k = 1, \ldots, q.$$

The above function is know as the **Softmax** function (it can be seen as generalization of the logistic function) and we can notice that the sum of probabilities is still equal to one.

Regression Models XXIII

Estimation

Because we are dealing with several label, we have seen that we can assign a different probability for each example to belong in a given class. Thus the likelihood $\mathfrak{L}$ of our data will be based on the *Multinomial distribution* and defined by

$$\begin{aligned}\mathfrak{L}(\mathbf{W}, S) &= \prod_{k=1}^{q}\prod_{i=1}^{m} Pr(y = k \mid \mathbf{x}_i)^{\mathbb{1}_{\{y_i=k\}}}, \\ &= \prod_{k=1}^{q}\prod_{i=1}^{m} \left(\frac{\exp\left(\langle \mathbf{w}^{(k)}, \mathbf{x} \rangle\right)}{\sum_{l=1}^{q} \exp\left(\langle \mathbf{w}^{(l)}, \mathbf{x} \rangle\right)} \right)^{\mathbb{1}_{\{y_i=k\}}}.\end{aligned}$$

Regression Models XXIV

And the negative log-likelihood ℓ can be written as

$$\ell(\mathbf{W}, S) = -\sum_{k=1}^{q}\sum_{i=1}^{m} \mathbb{1}_{\{y_i=k\}} \ln\left(\frac{\exp\left(\langle \mathbf{w}^{(k)}, \mathbf{x}\rangle\right)}{\sum_{l=1}^{q} \exp\left(\langle \mathbf{w}^{(l)}, \mathbf{x}\rangle\right)}\right). \tag{1}$$

The estimation of the parameters works exactly the same as for the binary case, we usually use the Newton-Raphson algorithm.

Let us finish with a last type of regression model which is called *Kernel Regression or Non-Parametric Regression*.

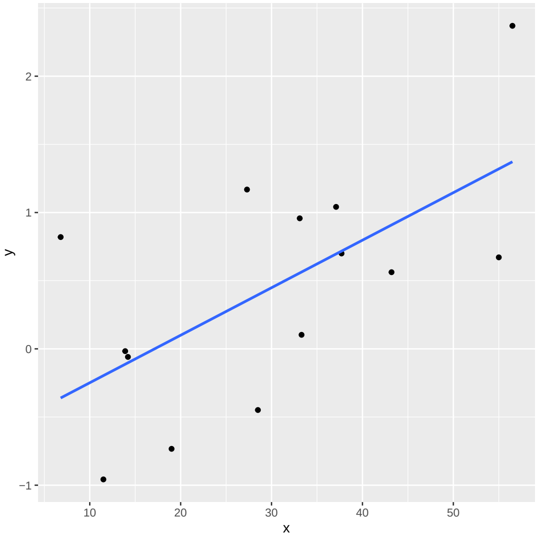
Regression Models XXV

Kernel Regression

The linear and logistic regression presented in the previous sections are both **parametric models**, *i.e.*, we need to learn or estimate the parameters (depending on shape of the model) in order to make predictions.

Non parametric models also exist where we do not need any parameter. The predictions for a new instance will only be based on the existed instances and, more precisely, on the characteristics of its most similar ones.

Regression Models XXVI



Regression Models XXVII

The use of linear model to estimate the values of y does not seem appropriate, but how a non parametric model will perform in this case ?

Let us introduce a first simple model. To estimate, the y associated to $\mathbf{x}$, *i.e.,* $y(\mathbf{x})$, we simply consider the mean values $y_i = y(\mathbf{x}_i)$ of the neighbors of $\mathbf{x}$. Thus, the estimator h is given by :

$$h(\mathbf{x}) = \frac{\sum_{k=1}^{n} K_\lambda(\mathbf{x}, \mathbf{x}_i) y_i}{\sum_{k=1}^{n} K_\lambda(\mathbf{x}, \mathbf{x}_i)},$$

where, in this case $K_\lambda(\mathbf{x}, \mathbf{x}_i) = 1$ if this distance $\|\mathbf{x} - \mathbf{x}_i\|$ between $\mathbf{x}$ and $\mathbf{x}_i$ is lower or equal to λ and 0 otherwise.

Regression Models XXVIII

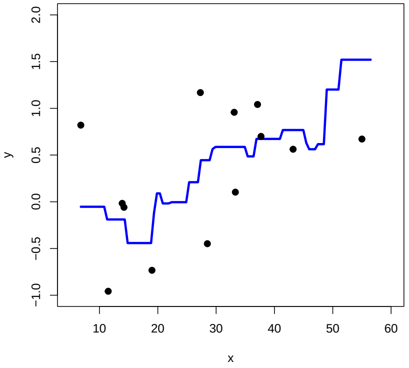


Illustration of the estimation made by a non parametric model when $K_\lambda(\mathbf{x}, \mathbf{x}_i) = \mathbb{1}_{\{\|\mathbf{x}-\mathbf{x}_i\| \leq \lambda\}}$ with $\lambda = 8$.

Regression Models XXIX

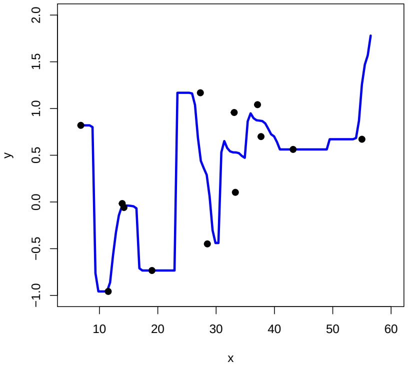


Illustration of the estimation made by a non parametric model when $K_\lambda(\mathbf{x}, \mathbf{x}_i) = \frac{1}{\lambda\sqrt{2\pi}} \exp\left(-\frac{1}{2\lambda^2}(\|\mathbf{x} - \mathbf{x}_i\|)\right)$ with $\lambda = 2$.

Classification and Regression Trees

Classification and Regression Trees I

Decision trees were introduced by [Quinlan, 1986] but the currently used version of Classification and Regression Trees algorithm was well introduced by [Breiman et al., 1984].

Decision trees consist of a series of rules that are successively applied to the dataset in order to separate the data into two or more groups. Here, we will only focus on binary decision trees, i.e. when a decision rule separates the data set in exactly two different sets.

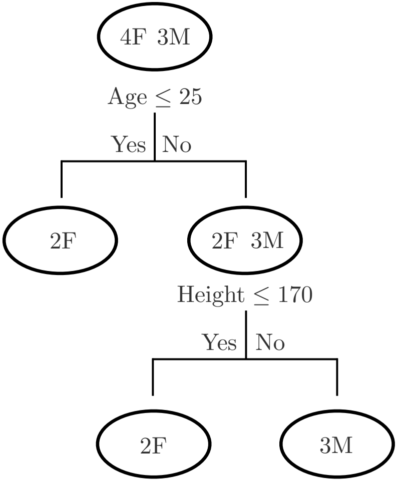
The nature of the tree depends on the output space $\mathcal{Y}$:

- when $\mathcal{Y} \subset \mathbb{R}$, we talk about regression tree,
- when $\mathcal{Y} = \{-1, 1\}$, we talk about classification tree.

Classification and Regression Trees II

Toy dataset

Age	Height	Sex
20	175	F
32	180	M
40	175	M
28	172	M
22	165	F
40	169	F
70	170	F



Classification and Regression Trees III

Such an algorithm is able to (non linearly) separate a complex dataset when using a large number of decision rules. To see how the decision rules are chosen, we define a criterion to optimize.

For this purpose, we need two tools, a *metric* which evaluates the quality of a node and a measure of improvement after a split, called the *gain* [Safavian and Landgrebe, 1991, Rokach and Maimon, 2005].

Classification and Regression Trees IV

The kind of used metrics depends on the type of tree we are dealing with. A list of such metrics are provided by [Rokach and Maimon, 2005] among which :

- the *Variance*, used for regression trees :

$$\frac{1}{n}\sum_{i=1}^{m_N}(y_i - \bar{y})^2,$$

  where m_N denotes the number of examples in the node N and $\bar{y}$ the average value of y_i in the node.

Classification and Regression Trees V

- the *Entropy* which is mainly used in physics in order to quantify the mess in a physical system, it is defined by $Ent_N = -\sum_{j=1}^{L} p_j \log_2(p_j)$, where p_j is the proportion of examples of class j in the node N.

- the *Gini impurity* used in classification tree. It measures the impurity of a node by computing the proportion of each classes present in the node. For instance, in binary classification, the Gini impurity G_N of a node N is defined by :

$$G_N = \sum_{j=1,-1} p_j(1-p_j) = 2p_1(1-p_1), \tag{2}$$

where p_j denotes the proportion of examples being in class j.

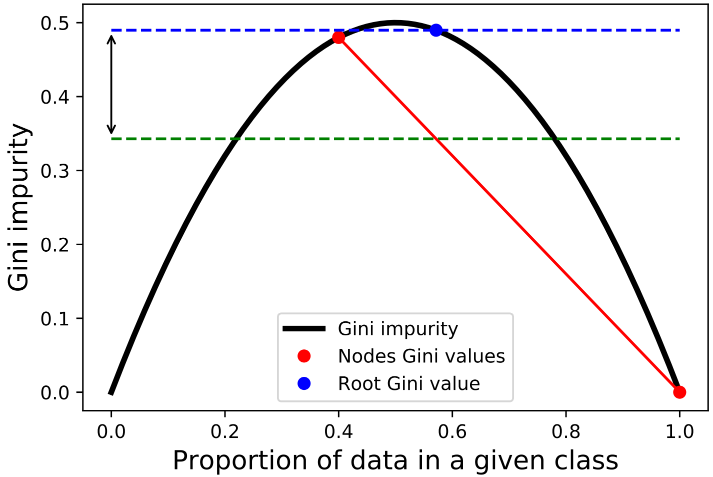
Classification and Regression Trees VI

The next step consists in choosing the optimal rule to split the dataset into two nodes. This rule is chosen in order to minimize the Gini impurity at the end of the tree. For this purpose, we define the *Gini gain* Γ as follows :

$$\Gamma = G_{\text{root}} - \left(\frac{|N_L|}{|N_L + N_R|} G_{N_L} + \frac{|N_R|}{|N_L + N_R|} G_{N_R} \right),$$

where G_{N_L} and G_{N_R} denote the Gini impurity of the node on the left, respectively on the right.

Classification and Regression Trees VII



Classification and Regression Trees VIII

The arrow between the two dashed lines represents the Gini gain Γ.

On this figure, we also see that the Gini function is concave.
This concavity ensures the positiveness of the gain by the Jensen Inequality [Jensen, 1906] so that each split leads to a lower classification error.

Furthermore, at each step, we choose the feature and its corresponding value which maximizes the gain Γ. The decision rule is then applied and the node is separated into two different nodes until getting pure leaves.

Classification and Regression Trees IX

In practice, it is always possible to lead to such perfect leaves. However, building such trees might tend to over-fitting and bad performance in generalization. To overcome this issue, we usually use a pruning strategy which can be controlled by parameters :

- the *size/depth* of the tree,
- the size of a node : minimum number of examples required in the node to make a new split,
- the size of a leaf : minimum number of examples in both leaves after a split,
- a threshold on the gain : the minimum value of gain required to make a new split.

Classification and Regression Trees X

Let us take another example, using the same dataset where we aim is to predict the age of the person according to its height using our regression tree.
We first compute the variance at the root of our tree. It is equal to $Var_N = 245.71$. We now have to find the best binary split such that the gain, defined by :

$$\Gamma = Var_{\text{root}} - \left(\frac{|N_L|}{|N_L + N_R|} Var_{N_L} + \frac{|N_R|}{|N_L + N_R|} Var_{N_R} \right),$$

achieves the highest value.

Classification and Regression Trees XI

For this purpose, we are going to test each possible value of the variable *height* and compute the associated value of Γ. The results of such computations are provided in Table 1 and show that, among the three tested splits, the best one is obtained using a threshold equal to 170.

Threshold	$\lvert N_L \rvert$	$\lvert N_R \rvert$	Var_{N_L}	Var_{N_R}	Γ
169	2	5	81	297.6	10
170	3	4	392	52	**48**
172	4	3	342	67.56	21.33

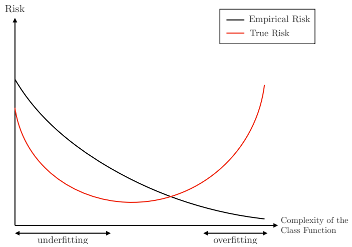
Table – Values of the gain Γ, in terms of variance, according to three different thresholds. An instance with a value lower or equal than the threshold will be put in left node, otherwise, in the right one.

Let us go a little bit further ! A study of the error in Machine Learning

Error Decomposition I

Performance and Complexity

We have seen the importance of the balance *bias* - *variance* in our learned model,the aim is to find a model with a good performance on the training set (a low bias) which has a low variance in order to perform well on unseed data.



Error Decomposition II

The example of the regression

We are going to explain where this bias-variance tradeoff comes from in Machine Learning and how it will be used in to order to build stronger models.

For thr sake of simplicity, we will consider the regression case and the following model :

$$y = f(\mathbf{x}) + \varepsilon,$$

where f is an unknow function that we aim to estimate using our data. These data can present some noise which is modeled by a random variable ε following a gaussian distribution in our model.

Error Decomposition III

Suppose that we have a dataset with which we have a learned a model h who tries to approximate f. We aim to estimate its generalization performances using the mean square error criterion, *i.e.*,

$$\mathbb{E}[(y - h(\mathbf{x}))^2].$$

The regression models assumption also imply that :

$$\mathbb{E}[y] = f(\mathbf{x}).$$

Error Decomposition IV

Proposition 3.1: Error Decomposition

We consider the following data generation model

$$y = f(\mathbf{x}) + \varepsilon,$$

Using a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ generated by the previous, we learn a hypothesis h which is an estimator of f. The generalization error of h according to mean squared error can be written :

$$\mathbb{E}[(y - h(\mathbf{x}))^2] = (\mathbb{E}[h(\mathbf{x})] - f(\mathbf{x}))^2 + \mathbb{E}\left[(\mathbb{E}[h(\mathbf{x})] - h(\mathbf{x}))^2\right] + \mathbb{E}[(y - f(\mathbf{x}))^2].$$

Error Decomposition V

Proof

We are going to use the *König-Huygens* which says that for all random variables which has a variance, we have :

$$\mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}[X^2] - \mathbb{E}[X]^2.$$

We develop the expression on the left :

$$\begin{aligned}\mathbb{E}[(y - h(\mathbf{x}))^2] &= \mathbb{E}[y^2 - 2h(\mathbf{x}) + h(\mathbf{x})^2], \\ &\quad \downarrow \text{ linearity of expectation} \\ &= \underbrace{\mathbb{E}[y^2]} - \mathbb{E}[2yh(\mathbf{x})] + \underbrace{\mathbb{E}[h(\mathbf{x})^2]}, \\ &\quad \downarrow \textit{ König-Huygens} \text{ formula}\end{aligned}$$

Error Decomposition VI

$$
\begin{aligned}
&= \mathbb{E}[y]^2 + \mathbb{E}\left[(y - \mathbb{E}[y])^2\right] - 2\,\mathbb{E}[y]\,\mathbb{E}[h(\mathbf{x})] \\
&\quad + \mathbb{E}[h(\mathbf{x})]^2 + \mathbb{E}\left[(h(\mathbf{x}) - \mathbb{E}[h(\mathbf{x})])^2\right],
\end{aligned}
$$

$\downarrow$ on utilise le modèle de génération des données

$$
\begin{aligned}
&= f(\mathbf{x})^2 + \mathbb{E}\left[(y - f(\mathbf{x}))^2\right] - 2f(\mathbf{x})\,\mathbb{E}[h(\mathbf{x})] \\
&\quad + \mathbb{E}[h(\mathbf{x})]^2 + \mathbb{E}\left[(h(\mathbf{x}) - \mathbb{E}[h(\mathbf{x})])^2\right],
\end{aligned}
$$

$\downarrow$ linearity of expectation

$$
\begin{aligned}
&= (\mathbb{E}[h(\mathbf{x})] - f(\mathbf{x}))^2 + \mathbb{E}\left[(\mathbb{E}[h(\mathbf{x})] - h(\mathbf{x}))^2\right] \\
&\quad + \mathbb{E}[(y - f(\mathbf{x}))^2].
\end{aligned}
$$

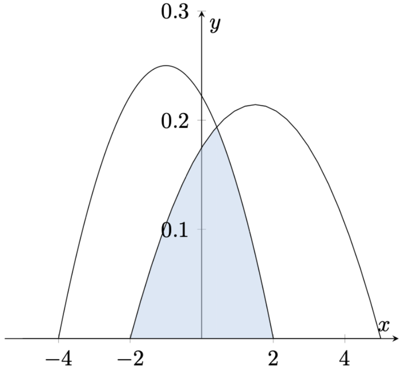
Error Decomposition VII

So more than a simple bias-variance tradeoff, we can see that our error can be divided in threee different parts :

- the square of the bias of the learner h : the difference of the mean value of h over all the data distribution and the f values.
- the second term represents the variance of the learner h, so how h values vary when the sample changes. In other words, it represents the stability or the sensivity of the model to the data.
- the thrid term is the bayes error, it does not depend on the data, nor on the model.

Error Decomposition VIII

Bayes Error



This is the blue area under the intersection of both curves. this quantity is hard to estimate in practise.

Error Decomposition IX

Exercise

Let us consider two densities d_1 and d_2 defined as below and for which the representations are given in the previous graph :

$$d_1 = \begin{cases} \frac{3}{2}x^2 + x & \text{when } 0 \leq x \leq 1, \\ 0 & \text{otherwise.} \end{cases} \quad \text{and} \quad d_2 = \begin{cases} 1 & \text{when } \frac{3}{4} \leq x \leq \frac{7}{4}, \\ 0 & \text{otherwise.} \end{cases}$$

1. Show that we effectively have two densities.
2. Determine the bayes error associated to this two densities.

Error Decomposition X

This error cannot be optimized and completely depends on the data distribution. Thus, in reality, what ware trying to minimize in Machine Learning, in order to define a good algorithm, is called, the excess of risk :

$$\mathbb{E}[(y - h(\mathbf{x}))^2] - \mathbb{E}[(y - f(\mathbf{x}))^2].$$

Let us focus on this term a little bit in order to draw the link with generalization bounds (we are going to provide an example in this presentation).

Error Decomposition XI

We now suppose more generally that the observations are sampled from a joint distribution $\mathcal{D} = \mathcal{X} \times \mathcal{Y}$ associated to a measure μ. We also consider a loss function ℓ

$$\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R},$$

which quantifies the cost of the error, of a hypothesis h, by predicting $h(\mathbf{x})$ when the true value is y. The function or hypothesis h we are looking for is the one that minimizes the expected error over $\mathcal{D}$, *i.e.*

$$\mathcal{R}(h) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} [\ell(y, h(\mathbf{x}))] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(y, h(\mathbf{x})) d\mu.$$

Error Decomposition XII

Keep in mind that, in practice, the hypothesis h is learned on a finite sample. We now assume that we minimize the risk over a function/hypothesis space $\mathcal{H}$. If we denote by $\mathcal{R}^\star$ the Bayes risk, we can decompose the **Bayes** regret as :

$$\mathcal{R}(h) - \mathcal{R}^\star = \left(\mathcal{R}(h) - \inf_{g \in \mathcal{H}} \mathcal{R}(g)\right) + \inf_{g \in \mathcal{H}} \mathcal{R}(g) - \mathcal{R}^\star.$$

- The first term is the excess of risk of h with respect to the best function in the hypothesis space $\mathcal{H}$.
- The second term is the **approximation error**, *i.e.* the smallest excess of risk we can achieve using a function in $\mathcal{H}$. This is bias term which does not depend on the data but only the hypothesis space $\mathcal{H}$.

Error Decomposition XIII

We will now draw the link with the generalization bounds and go little bit further by bounding the **excess of risk**.
If we denote by h the hypothesis obtained by minimizing the empirical risk over $\mathcal{H}$ using a sample S :

$$h \in \arg\min_{g \in \mathcal{H}} \mathcal{R}_S(g).$$

We will also denote by $h_{\mathcal{H}}$ the minimizer of the risk $\mathcal{R}$ over the hypotheses space $\mathcal{H}$, *i.e.*

$$h_{\mathcal{H}} = \arg\min_{g \in \mathcal{H}} \mathcal{R}(g).$$

The **excess of risk** : $\mathcal{R}(h) - \inf_{g \in \mathcal{H}} \mathcal{R}(g)$ can be rewritten as the sum of three terms :

Error Decomposition XIV

$$\mathcal{R}(h) - \mathcal{R}(h_{\mathcal{H}}) = (\mathcal{R}(h) - \mathcal{R}_S(h)) + (\mathcal{R}_S(h) - \mathcal{R}_S(h_{\mathcal{H}})) \\ + (\mathcal{R}_S(h_{\mathcal{H}}) - \mathcal{R}(h_{\mathcal{H}})),$$

where

- $(\mathcal{R}(h) - \mathcal{R}_S(h))$ is the difference between the true risk and the empirical risk of the hypothesis h. This quantity is the one we are interested in, when it comes to study the generalization of the algorithms.
- $(\mathcal{R}_S(h) - \mathcal{R}_S(h_{\mathcal{H}}))$ is a non positive term by construction.
- $(\mathcal{R}_S(h_{\mathcal{H}}) - \mathcal{R}(h_{\mathcal{H}}))$ is easier to control as it involves a deterministic function and the law of large numbers applies.

Error Decomposition XV

However, the first term can be bounded as follows :

$$\mathcal{R}(h) - \mathcal{R}_S(h) \leq \sup_{g \in \mathcal{H}} \left| \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} [\ell(g(\mathbf{x}), y)] - \frac{1}{m}\sum_{i=1}^{m} \ell(g(\mathbf{x}_i), y_i) \right|.$$

Since this quantity also bounds the third term, we immediately have :

$$\mathcal{R}(h) - \mathcal{R}_S(h) \leq 2 \sup_{g \in \mathcal{H}} \left| \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} [\ell(g(\mathbf{x}), y)] - \frac{1}{m}\sum_{i=1}^{m} \ell(g(\mathbf{x}_i), y_i) \right|.$$

This latest bound can be seen as variance term which increases with the size of $\mathcal{H}$. If the size of $\mathcal{H}$ is big we can achieve a low empirical risk, however, the expected risk has higher chance of being high.
A question can arise now :

Error Decomposition XVI

How can we define the size of hypotheses space $\mathcal{H}$?

When $\mathcal{H}$ contains a finite number of hypothesis, the answer is clear and the size is simply the number of hypothesis in the set. But when it is infinite, it exists several measures to evaluate $\mathcal{H}$ size as the *Rademacher complexity*.

The Rademacher complexity [Bartlett and Mendelson, 2003] has been introduced to measure the complexity of a set of hypotheses [Koltchinskii and Panchenko, 2000]. Informally, it measures how the set of hypotheses is able to fit noise in the dataset.

Error Decomposition XVII

Définition 3.1: (Empirical) Rademacher complexity

Let $\mathcal{H}$ be a family of functions and $S = \{\mathbf{x}_i\}_{i=1}^m$ a fixed sample of size m. Then, the empirical Rademacher complexity of $\mathcal{H}$ with respect to S is defined as :

$$\mathfrak{R}_S(\mathcal{H}) = \mathop{\mathbb{E}}_{\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}} \frac{1}{m}\sum_{i=1}^{m}\sigma_i h(\mathbf{x}_i)\right],$$

where $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_m)$ is a vector of Rademacher random variables, i.e. random variables taking values in $\{-1, +1\}$ both with probability $1/2$.
The Rademacher complexity is the expectation of the above quantity on the distribution $\mathcal{D}$ of the data :

$$\mathfrak{R}_m(\mathcal{H}) = \mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}[\mathfrak{R}_S(\mathcal{H})].$$

Error Decomposition XVIII

This measure increases with the size of $\mathcal{H}$ and decreases with the sample size. Thus using this measure we have

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m} |\mathcal{R}(h) - \mathcal{R}_S(h)| \\
&\leq 2 \mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[\sup_{g\in\mathcal{H}} \left| \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} [\ell(g(\mathbf{x}), y)] - \frac{1}{m}\sum_{i=1}^{m} \ell(g(\mathbf{x}_i), y_i)\right|\right], \\
&\leq 2\mathfrak{R}_m(\mathcal{H}).
\end{aligned}
$$

Therefore,

$$
\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m} [\mathcal{R}(h) - \mathcal{R}^\star] \leq \inf_{g\in\mathcal{H}} \mathcal{R}(g) - \mathcal{R}^\star + 4\mathfrak{R}_m(\mathcal{H}).
$$

Error Decomposition XIX

This result illustrates a little more generally the bias-variance trade-off for risk minimization. It makes explicit the link between complexity and sample size.
It remains to explain where the following inequality comes from

$$\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[\sup_{g\in\mathcal{H}}\left|\mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(g(\mathbf{x}),y)]-\frac{1}{m}\sum_{i=1}^{m}\ell(g(\mathbf{x}_i),y_i)\right|\right]\leq 2\mathfrak{R}_m(\mathcal{H}).$$

We are going to show this inequality using the *symmetrization technique*. Concretely,

$$\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[\sup_{g\in\mathcal{H}}\left|\mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(g(\mathbf{x}),y)]-\frac{1}{m}\sum_{i=1}^{m}\ell(g(\mathbf{x}_i),y_i)\right|\right],$$

Error Decomposition XX

$$= \mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[\sup_{g\in\mathcal{H}}\left|\mathop{\mathbb{E}}_{S'\sim\mathcal{D}^m}\left[\frac{1}{m}\sum_{i=1}^{m}\ell(g(\mathbf{x}'_i),y'_i)\right]-\frac{1}{m}\sum_{i=1}^{m}\ell(g(\mathbf{x}_i),y_i)\right|\right],$$

$\downarrow$ the second term does not depend on S'

$$= \mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[\sup_{g\in\mathcal{H}}\left|\mathop{\mathbb{E}}_{S'\sim\mathcal{D}^m}\left[\frac{1}{m}\sum_{i=1}^{m}\ell(g(\mathbf{x}'_i),y'_i)-\frac{1}{m}\sum_{i=1}^{m}\ell(g(\mathbf{x}_i),y_i)\right]\right|\right],$$

$\downarrow$ linearity of the empirical mean

$$= \mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[\sup_{g\in\mathcal{H}}\left|\mathop{\mathbb{E}}_{S'\sim\mathcal{D}^m}\left[\frac{1}{m}\sum_{i=1}^{m}\ell(g(\mathbf{x}'_i),y'_i)-\ell(g(\mathbf{x}_i),y_i)\right]\right|\right],$$

$$\leq \mathop{\mathbb{E}}_{S,S'\sim\mathcal{D}^m}\left[\sup_{g\in\mathcal{H}}\left|\frac{1}{m}\sum_{i=1}^{m}\ell(g(\mathbf{x}'_i),y'_i)-\ell(g(\mathbf{x}_i),y_i)\right|\right],$$

Error Decomposition XXI

where the last inequality uses the fact the supremum of mean values is lower than the mean values of suprema. We now introduce our Rademacher variables $\sigma_i, \ i = 1, \ldots, m$ and notice that

$$\begin{aligned}\mathop{\mathbb{E}}_{S,S' \sim \mathcal{D}^m}&\left[\sup_{g \in \mathcal{H}} \left| \frac{1}{m} \sum_{i=1}^{m} \ell(g(\mathbf{x}_i'), y_i') - \ell(g(\mathbf{x}_i), y_i) \right|\right] \\ &\leq \mathop{\mathbb{E}}_{S,S' \sim \mathcal{D}^m, \boldsymbol{\sigma}} \left[\sup_{g \in \mathcal{H}} \left| \frac{1}{m} \sum_{i=1}^{m} \sigma_i \left(\ell(g(\mathbf{x}_i'), y_i') - \ell(g(\mathbf{x}_i), y_i) \right) \right|\right]\end{aligned}$$

This is due to the fact that the data are *i.i.d.*. Furthermore, this inequality holds for any choice of $\boldsymbol{\sigma}$.
Finally,

Error Decomposition XXII

$$
\begin{aligned}
&\mathop{\mathbb{E}}_{S,S'\sim\mathcal{D}^m,\boldsymbol{\sigma}}\left[\sup_{g\in\mathcal{H}}\left|\frac{1}{m}\sum_{i=1}^{m}\sigma_i\left(\ell(g(\mathbf{x}_i'),y_i')-\ell(g(\mathbf{x}_i),y_i)\right)\right|\right],\\
&\leq \mathop{\mathbb{E}}_{S\sim\mathcal{D}^m,\boldsymbol{\sigma}}\left[\sup_{g\in\mathcal{H}}\left|\frac{1}{m}\sum_{i=1}^{m}\sigma_i\ell(g(\mathbf{x}_i),y_i)\right|\right]\\
&\quad+\mathop{\mathbb{E}}_{S'\sim\mathcal{D}^m,\boldsymbol{\sigma}}\left[\sup_{g\in\mathcal{H}}\left|\frac{1}{m}\sum_{i=1}^{m}\sigma_i\ell(g(\mathbf{x}_i'),y_i')\right|\right],\\
&=2\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m,\boldsymbol{\sigma}}\left[\sup_{g\in\mathcal{H}}\left|\frac{1}{m}\sum_{i=1}^{m}\sigma_i\ell(g(\mathbf{x}_i),y_i)\right|\right],\\
&=2\mathfrak{R}_m(\mathcal{H})
\end{aligned}
$$

Error Decomposition XXIII

It exists other measures to express the capacity of the space of hypothesis $\mathcal{H}$ such as the *VC-dimension* [Vapnik and Chervonenkis, 1971] which are more easy to compute when we are dealing with linear classifiers.

But we are not going to deal with such a measure, we rather come back to generalization bounds and provide a first bound using this complexity measure.

Error Decomposition XXIV

Using the Definition 3.1 we can provide a first generalization bound using this complexity measure.

Théorème 3.1: Rademacher Generelization Bound

Let $\mathcal{H}$ our class of hypothesis associated to any loss function ℓ mapping from $\mathcal{X} \times \mathcal{Y}$ to $[0, 1]$. Then for any $\delta > 0$, with probability at least $1 - \delta$ over the draw of an *i.i.d.* sample S of size m, each of the following holds for all $h \in \mathcal{H}$:

$$\mathcal{R}(h) \leq \mathcal{R}_S(h) + 2\mathfrak{R}_m(\mathcal{H}) + \sqrt{\frac{\log(1/\delta)}{2m}}, \quad \text{and}$$

$$\mathcal{R}(h) \leq \mathcal{R}_S(h) + 2\mathfrak{R}_S(\mathcal{H}) + 3\sqrt{\frac{\log(1/\delta)}{2m}}.$$

Error Decomposition XXV

Proof

Let us consider a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ random drawn from $\mathcal{D}$. For any $h \in \mathcal{H}$, we denote by $\mathcal{R}_S(h) = \underset{(\mathbf{x},y)\sim S^m}{\mathbb{E}} [\ell(h(\mathbf{x}), y)]$ the empirical risk associated to a loss ℓ and hypothesis h.
The proof consists in applying McDiarmid's inequality to the function Φ defined on any sample S by

$$\Phi(S) = \sup_{h\in\mathcal{H}} \mathcal{R}(h) - \mathcal{R}_S(h).$$

If we now consider S and S' two samples differing by exactly one point : $\mathbf{z}_m = (\mathbf{x}_m, y_m) \in S$ and $\mathbf{z}'_m = (\mathbf{x}'_m, y'_m) \in S'$. Then, since the difference of suprema does not exceed the supremum of the difference, we have

Error Decomposition XXVI

$$\Phi(S) - \Phi(S') \leq \sup_{h \in \mathcal{H}} \left(\mathcal{R}_S(h) - \mathcal{R}_{S'}(h)\right) = \sup_{h \in \mathcal{H}} \frac{\ell(h(z_m)) - \ell(h(z'_m))}{m} \leq \frac{1}{m}.$$

Similarly, we can obtain $\Phi(S') - \Phi(S) \leq \dfrac{1}{m}$ and we can bound the absolute value of the difference $|\Phi(S') - \Phi(S)|$. Then, by McDiarmid's inequality, for any $\delta > 0$, with probability at least $1 - \delta/2$, the following holds :

$$\Phi(S) \leq \mathop{\mathbb{E}}_{S \sim \mathcal{D}^m} \Phi(S) + \sqrt{\frac{\log(2/\delta)}{2m}}.$$

Our next step consists in bounding the expectation on the right-hand side of the previous inequality

Error Decomposition XXVII

$$
\begin{aligned}
\mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}[\Phi(S)] &= \mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[\sup_{h\in\mathcal{H}}\left(\mathcal{R}(h)-\mathcal{R}_S(h)\right)\right],\\
&= \mathop{\mathbb{E}}_{S\sim\mathcal{D}^m}\left[\sup_{h\in\mathcal{H}}\left(\mathop{\mathbb{E}}_{S'\sim\mathcal{D}^m}\left[\mathcal{R}_{S'}(h)-\mathcal{R}_S(h)\right]\right)\right],\\
&\leq \mathop{\mathbb{E}}_{S,S'\sim\mathcal{D}^m}\left[\sup_{h\in\mathcal{H}}\left(\mathcal{R}_{S'}(h)-\mathcal{R}_S(h)\right)\right],\\
&= \mathop{\mathbb{E}}_{S,S'\sim\mathcal{D}^m}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{m}\sum_{i=1}^{m}\ell(\mathbf{z}_i)-\ell(\mathbf{z}'_i)\right)\right],\\
&= \mathop{\mathbb{E}}_{S,S'\sim\mathcal{D}^m,\boldsymbol{\sigma}}\left[\sup_{h\in\mathcal{H}}\left(\frac{1}{m}\sum_{i=1}^{m}\sigma_i(\ell(\mathbf{z}_i)-\ell(\mathbf{z}'_i))\right)\right],
\end{aligned}
$$

Error Decomposition XXVIII

$$
\begin{aligned}
&\leq \mathop{\mathbb{E}}_{S' \sim \mathcal{D}^m, \boldsymbol{\sigma}} \left[\sup_{h \in \mathcal{H}} \left(\frac{1}{m} \sum_{i=1}^{m} \sigma_i \ell(\mathbf{z}'_i) \right) \right] \\
&\qquad + \mathop{\mathbb{E}}_{S \sim \mathcal{D}^m, \boldsymbol{\sigma}} \left[\sup_{h \in \mathcal{H}} \left(\frac{1}{m} \sum_{i=1}^{m} \sigma_i \ell(\mathbf{z}_i) \right) \right], \\
&= 2 \mathop{\mathbb{E}}_{S' \sim \mathcal{D}^m, \boldsymbol{\sigma}} \left[\sup_{h \in \mathcal{H}} \left(\frac{1}{m} \sum_{i=1}^{m} \sigma_i \ell(\mathbf{z}'_i) \right) \right], \\
&= 2\mathfrak{R}_m(\mathcal{H}).
\end{aligned}
$$

The sketch of the proof is exactly the same as the one provided when sutyding the excess of risk.

Error Decomposition XXIX

Using again McDiarmid's inequality, we get

$$\mathfrak{R}_m(\mathcal{H}) \leq \mathfrak{R}_S(\mathcal{H}) + \sqrt{\frac{\log(2/\delta)}{2m}}.$$

Thus, with probability at least $1-\delta$ we have :

$$\Phi(S) \leq 2\mathfrak{R}_S(\mathcal{H}) + 3\sqrt{\frac{\log(2/\delta)}{2m}}.$$

And thus, using the defintion of

$$\mathcal{R}(h) \leq \mathcal{R}_S(h) + 2\mathfrak{R}_m(\mathcal{H}) + \sqrt{\frac{\log(1/\delta)}{2m}}.$$

Error Decomposition XXX

It exists other tools to develop such generalization bounds that are based on different domains of Machine Learning :

- for instance, a complexity measure called the *VC-dimension* can used instead of the Rademacher Complexity.
- we can also directly evaluate the complexity of the space of hypotheses using the hyper-parameter ahead the regularization term.
- the use of the **PAC-Bayesian** theory is also an interesting tool to develop generalization guarantees as it is well dedicated to the study of ensemble algorithms/methods.

Ensemble Methods

Naïve Approaches I

From now on, all the presented algorithms we are considered as simple models because they were studied alone, they were more or less complex for our both regression or classification tasks.
But we can wonder :

- Why using only model or one learner/hypothesis ?
- Would it be interesting to combine several of them ?

Thinking about the linear models may provide a first part of the answers as they are to much simple to solve complex problems.

Naïve Approaches II

A naïve rule

Let us imagine we want to create K models using a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ of size m. We can train an hypothesis h_k for k from 1 to K using the same training sample S for each k.

After that, we can combine the different hypotheses into a single one, noted H_K, doing the average to take our final decision :

- for a regression task, the predicted value will correspond to the mean value over all prediction made by hypotheses h_k, *i.e.*

$$H_K(\mathbf{x}) = \frac{1}{K}\sum_{k=1}^{K} h_k(\mathbf{x}).$$

Naïve Approaches III

- for a classification task, we can apply the same rule. However, instead of taking the mean value as the output of the combined classifier, we rather take the sign of this mean value as the output, *i.e.*.

  $$H_K(\mathbf{x}) = sign\left(\frac{1}{K}\sum_{k=1}^{K} h_k(\mathbf{x})\right).$$

  In the case where the hypotheses h_k return a value that is -1 or 1, our hypothesis H_K can then be seen as an majority vote with equal weights.

  On the contrary, if the hypotheses h_k return real values, then we can imagine that it is a weighted majority vote.

Naïve Approaches IV

Although this rule is simple in practice, it is not very useful.

Indeed, recall that if all our problems are convex, there is a good chance that all the hypotheses h_k are similar if we use the same training set each time.

Naïve Approaches V

Based on Hyper-parameter

One could then be tempted to vary the hypotheses by imposing different hyper-parameter values for each hypothesis, *e.g.* one could impose K different hyper-parameter values in order to obtain K different models. Again this solution is not satisfactory :

- it may lead to hypotheses with low predictive power
- using such a process would call into question the cross-validation process presented earlier which allows to optimize the values of these hyper-parameters

Naïve Approaches VI

We are going to see that, even if our algorithm does not depend on hyper-parameter, it is possible to improve it using a single training set S. These two methods are known as **Bagging** and **Boosting**.

These two methods act differently on the performance of algorithms or more precisely on the different components of the error of an algorithm. We will also present a third one, more general which also consists in combining several algorithms. But before presenting these approaches, let us motivate the use of several models.

Naïve Approaches VII

A theoretical Analysis

We will now try to explain why it is interesting to combine several model To do this, let us consider data $(\mathbf{x}, y)$ from a distribution $\mathcal{D}$ where $\mathbf{x}$ s the feature vector and y is the response variable or the value to predict.

In a regression setting, we will then learn a hypothesis h which aims to predict the value y according to $\mathbf{x}$. For the sake of simplicity, we suppose there exists a *true function* r such that $r(\mathbf{x}) = y$ for all $(\mathbf{x}, y) \sim \mathcal{D} = \mathcal{X} \times \mathcal{Y}$. We also consider a training set S of size m.

Naïve Approaches VIII

Remember that we aim learn a set of hypothesis $h_t,\ t = 1, \ldots, T$.
Thus for any instance $\mathbf{x}$ and any hypothesis h_t

$$h_t(\mathbf{x}) = r(\mathbf{x}) + \varepsilon_t(\mathbf{x}),$$

where $\varepsilon_t(\mathbf{x})$ is the error between the predicted value and the true value of the function at $\mathbf{x}$.

Keep it mind that, in a regression setting, the error we consider is the MSE, *i.e.* the mean squared error.
Thus the generalization error of a single hypothesis h is defined by :

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[(h(\mathbf{x}) - r(\mathbf{x}))^2\right] = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[\varepsilon(\mathbf{x})^2\right].$$

Naïve Approaches IX

And the generalization error of our averaged classifier $H_K = \frac{1}{T}\sum_{t=1}^{T} h_t$ is

$$\underset{\mathbf{x}\sim\mathcal{X}}{\mathbb{E}}\left[(H_T(\mathbf{x}) - r(\mathbf{x}))^2\right] = \underset{\mathbf{x}\sim\mathcal{X}}{\mathbb{E}}\left[\left(\frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x}) - r(\mathbf{x})\right)^2\right],$$

$\downarrow$ applying definition

$$= \underset{\mathbf{x}\sim\mathcal{X}}{\mathbb{E}}\left[\left(\frac{1}{T}\sum_{t=1}^{T} \varepsilon_t(\mathbf{x})\right)^2\right].$$

Let us suppose now that our error ε_t are centered (mean value equal to 0) and uncorrelated. We can rewrite the previous expression as :

Naïve Approaches X

$$\underset{\mathbf{x}\sim\mathcal{X}}{\mathbb{E}}\left[\left(H_T(\mathbf{x})-r(\mathbf{x})\right)^2\right] = \underset{\mathbf{x}\sim\mathcal{X}}{\mathbb{E}}\left[\left(\frac{1}{T}\sum_{t=1}^{T}\varepsilon_t(\mathbf{x})\right)^2\right],$$

$\downarrow$ we develop the square term

$$= \underset{\mathbf{x}\sim\mathcal{X}}{\mathbb{E}}\left[\left(\frac{1}{T}\sum_{t_1=1}^{T}\varepsilon_{t_1}(\mathbf{x})\right)\left(\frac{1}{T}\sum_{t_2=1}^{T}\varepsilon_{t_2}(\mathbf{x})\right)\right],$$

$\downarrow$ we use the fact that $\underset{\mathbf{x}\sim\mathcal{X}}{\mathbb{E}}[\varepsilon_{t_1}(\mathbf{x})\varepsilon_{t_2}(\mathbf{x})] = 0$

$\downarrow$ for all $t_1 \neq t_2$.

$$= \frac{1}{T^2}\sum_{t=1}^{T}\underset{\mathbf{x}\sim\mathcal{X}}{\mathbb{E}}\left[\varepsilon_t(\mathbf{x})^2\right],$$

Naïve Approaches XI

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[\left(H_T(\mathbf{x})-r(\mathbf{x})\right)^2\right]=\frac{1}{T}\left(\frac{1}{T}\sum_{t=1}^{T}\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[\varepsilon_t(\mathbf{x})^2\right]\right).$$

This last equation shows that the generalization error of a combination of hypothesis is just the average error of the set of the T hypotheses divided by T.
Note that, in practice, the assumption of uncorrelated error is essentially wrong because several hypotheses h_t are learned using similar samples S_t.
However, we can still show that the error of bagging hypotheses is no more than average error of the set of hypotheses h_t.

Naïve Approaches XII

In fact, for any random random variable X, we have $\mathbb{E}[X^2] \geq \mathbb{E}[X]^2$ (this a consequence of Jensen' Inequality) Thus, taking $X = \varepsilon_t$, we immediately have :

$$\frac{1}{T}\sum_{t=1}^{T} \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[\varepsilon_t(\mathbf{x})^2\right] \geq \left(\frac{1}{T}\sum_{t=1}^{T} \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[\varepsilon_t(\mathbf{x})\right]\right)^2 .$$

Let us know go on with a first ensemble approach : the *bagging* procedure.

Bagging and Random Forests I

Bagging

It is a way to combine models that have good performances on the training set. We have previously seen that we can decompose our *Bayes Regret* into the sum of two terms

$$\mathcal{R}(h) - \mathcal{R}^\star \leq \inf_{g \in \mathcal{H}} \mathcal{R}(g) - \mathcal{R}^\star + \mathfrak{R}(\mathcal{H}).$$

What will interest us here is more precisely the variance term in the error decomposition. This value will show us how sensitive the algorithm is to the variation of the training set, if this value is low, our algorithm will be little sensitive to variations of the training set.

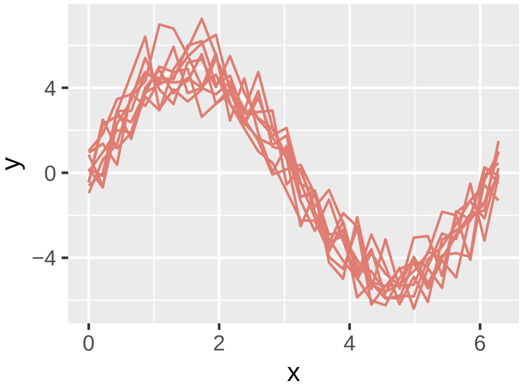
Bagging and Random Forests II

We will learn a polynomial regression model of degree 15 on different data from the same distribution. Each model is learned on a training set of size 30.

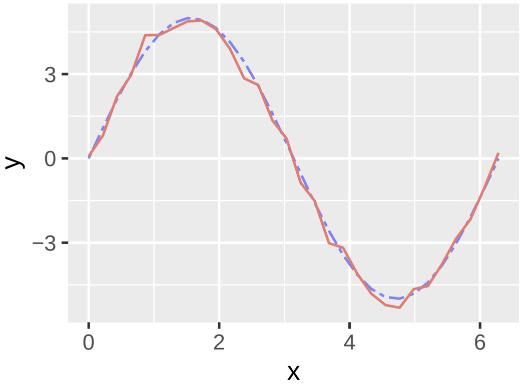
In total, we learn 10 different models which are represented on the graph on the left.

We note that the variance of the models is very important, i.e. for the same value on the abscissa, the models return very different values on the ordinate, so it is very sensitive to the training data.

Bagging and Random Forests III



Bagging and Random Forests IV



We notice that this graph shows much less variation around the true distribution of the data. We have therefore succeeded in reducing the variance by combining several models.

Bagging and Random Forests V

In this example the models learned are on different data each time.
But in practice we have only one training set S.
So we have to find a way to create several training sets S_k from this set S.
This can be done using the Bootstrap method, which is a sampling method based on a random draw.
How does it work in practice ?

Let us consider of training set S of size m. To create a bootstrap sample S_k of the same size m it will be necessary simply to carry out a draw with replacement of m examples in the set S, *i.e.* we have the same probability to draw each example in the set S.

Bagging and Random Forests VI

This sampling method will thus create diversity in the learned models. Indeed, the sets S_t being different, the hypotheses h_t will focus on different regions of the data space, because S_t contains different examples and, sometimes, several times the same example.

Rate of examples in S that we will find in S_t, on average ?

Bagging and Random Forests VII

As one can then guess after this reading, bagging means bootstrap aggregating : generate several samples and hypothesis and then you aggregate the results.
It is summarized as follows.

```
Input: Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, number of model $T$
Output: A model $H_T$
begin
    for $t = 1, ..., T$ do
        create a bootstrap sample $S_t$ of size $m$ using $S$.
        learn a hypothesis $h_t$ using $S_t$
    set $H_T = \frac{1}{T}\sum_{t=1}^{T} h_t$ return $H_T$
```

Bagging and Random Forests VIII

Theoretical Analysis

We can conduct a similar analysis as the one presented in the previous section.
Let us also denote $H_T = \frac{1}{T}\sum_{t=1}^{T} h_t$ our bagging classifier used to estimate (for instance) the output of a regression function y, *i.e.*, $H_T(\mathbf{x}) \simeq y$.
Then, the average error made by the set of classifiers is equal to :

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}, y\sim\mathcal{Y}}\left[\frac{1}{T}\sum_{t=1}^{T}(h_t(\mathbf{x}) - y)^2\right] \\
= &\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}, y\sim\mathcal{Y}}\left[y^2 - 2y\frac{1}{T}\sum_{t=1}^{T}h_t(\mathbf{x}) + \frac{1}{T}\sum_{t=1}^{T}h_t(\mathbf{x})^2\right].
\end{aligned}$$

Bagging and Random Forests IX

Using again Jensen' Inequality, we have $\left(\frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x})\right)^2 \leq \frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x})^2$, so that

$$\begin{aligned}
& \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X},y\sim\mathcal{Y}}\left[\frac{1}{T}\sum_{t=1}^{T}(h_t(\mathbf{x})-y)^2\right] \\
\geq\ & \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X},y\sim\mathcal{Y}}\left[y^2 - 2y\frac{1}{T}\sum_{t=1}^{T}h_t(\mathbf{x}) + \left(\frac{1}{T}\sum_{t=1}^{T}h_t(\mathbf{x})\right)^2\right], \\
=\ & \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X},y\sim\mathcal{Y}}\left[\left(y-\frac{1}{T}\sum_{t=1}^{T}h_t(\mathbf{x})\right)^2\right], \\
=\ & \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X},y\sim\mathcal{Y}}\left[(y-H_T(\mathbf{x}))^2\right].
\end{aligned}$$

Bagging and Random Forests X

This last inequality show that the error made, on average, by the bagging classifier is always lower than the error made by a single classifier.

We can go a little bit further by studying the potential gain of the bagging classifier. We can see that this gain depends on the difference of the two following elements :

$$\left(\frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x})\right)^2 \leq \frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x})^2,$$

which can be seen as the variance term associated to our set of classifiers h_t.

Bagging and Random Forests XI

If the variance is important, meaning that the difference $\frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x})^2 - \left(\frac{1}{T}\sum_{t=1}^{T} h_t(\mathbf{x})\right)^2$ is big, then the gain will be important.

It shows that, it is more interesting to use a set of base classifiers that have a high variance but a low bias in order to achieve a good a bagging classifier, which is exactly the case of **Decision Trees**. This is why it is interesting to combine them using this procedure to lead us to the **Random Forest** algorithm.

Bagging and Random Forests XII

Random Forests

When we introduced decision trees, we said that the size of the tree, i.e. its depth, depends on the data. The tree will thus grow until we obtain pure leaves.

If we go back to our error decomposition story, the (deep) decision trees thus form hypotheses with a low bias (an error rate that decreases with depth) but with a high variance. They are very sensitive to the data and the structure can vary greatly from one training set to another.

In fact, we find ourselves in exactly the same case as in our regression example. We will therefore proceed to a combination of tree models in order to reduce the variance of the trees while maintaining their predictive capacities using bagging. This combination of trees by bagging gives rise

Bagging and Random Forests XIII

to the random forest algorithm founded in the early 21^{th} century [Breiman, 2001].

The idea is then to build several trees based on different training sets S_t that are drawn randomly from S with replacement, *i.e.* using the bootstrap procedure and to combine the results of the different trees. But the random forest algorithm (presented after is) in fact more sophisticated than that. It is based on the principle of **double sampling :** sampling both on the examples and on the variables.

Bagging and Random Forests XIV

```
Input: Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, number of trees $T$, a sample of
       size $m'$ and a number of features $p'$
Output: A hypothesis $H_T$
begin
    for $t = 1, ..., T$ do
        Create a bootstrap sample $S_t$ of size $m'$ using $S$.
        Build a decision tree $h_t$ where at each split, a random
         subsample of $p'$ features are used to split the node.
    Set $H_T = \frac{1}{T} \sum_{t=1}^{T} h_t$ return $H_T$
```

Bagging and Random Forests XV

This double sampling will make it possible to create a diversity at the sample level and features level (can be interesting for those who wants to study the *Multiview Learning*)

Compared to the standard bagging algorithm, note the boostrap sample is of size $m' \leq m$ and that the number of used features at each node of a given tree is less than the dimension d of the data. Note that it is not mandatory to have $m' < m$, it only give the possibility to have a faster learning procedure.

From an algorithmic point of view, it allows to learn the different trees faster : you have less examples and the splitting procedure is applied to a less number of features.

Bagging and Random Forests XVI

The presented algorithms is the simplest one. It is not rare to give different weights to the trees according to their performance, remember, this our weighted majority vote.

Decision Trees and Random Forests are used in many applications such as finance, security or social sciences in general. These algorithms have the advantage to be easy to build and their decision is easy to understand (explainable AI), you just to follow the road of the data in the tree.

Bagging and Random Forests XVII

Our of Bag Error (OOB)

The bootstrap procedure means that we don't use all the data to learn a hypothesis. More precisely, given a sample S of size m and if we consider that we draw any example $\mathbf{x} \in S$ with the same probability $1/m$, then the probability of not selecting an example in m draws is equal to $(1 - 1/m)^m$. If the sample size m is high enough, this figure tends towards $e^{-1} \simeq 0.37$.

It means that the bagging procedure do not use approximately 37% of the data when a learning a base classifier, thus approximately 63% of the data are used.

Bagging and Random Forests XVIII

The remaining examples are called **Out Of Bag** data, they are not used to train the classifier but rather to tune the hyper-parameters of the model by estimating their generalization capacities.

The Bagging is then a procedure which is a mainly interesting to

- achieve an ensemble classifier with less variance,
- train a stronger model with higher generalization capacities with fewer resources (from a an experimental point of view).

As for simple decision trees, random forests can be used for both regression and classification as the main difference lies in the way the different trees are built, *i.e.*, the used loss function.

Bagging and Random Forests XIX

Measuring the importance of the variables

The way to measure the feature importance as been introduced at the same time as the Random forest algorithm [Breiman, 2001].

Remember that, to assess the quality of a split in decision trees according to a feature x^j, we measure the difference of entropy at a node N, ΔEnt_N after and before the split due to the variable X^j (we explain it for binary decision tree for the sake of simplicity) :

$$\Delta Ent_N(X^j) = Ent(\mathbf{p}_N) - Ent(\mathbf{p}_N, X^j),$$

where,

$$Ent(\mathbf{p}_N) = -\sum_{k=1}^{C} p_{N,k} \log_2(p_{N,k})$$

Bagging and Random Forests XX

and

$$Ent(\mathbf{p}_N, X^j) = -\frac{|N_L|}{|N|}\sum_{k=1}^{C} p_{N_L,k}\log_2(p_{N_L,k}) - \frac{|N_R|}{|N|}\sum_{k=1}^{C} p_{N_R,k}\log_2(p_{N_L,k})$$

In the previous expression, we use N_L and N_R to denote the two leaves (left and right) obtained after the split of the node N.
To measure the importance Imp^j of the variable X^j, we measure the mean value of $\Delta Ent_N(X^j)$ over the nodes N where X^j is used to split the node

$$Imp^j = \frac{1}{T}\sum_{t=1}^{T}\sum_{N=1}^{N^t}\frac{|N|}{N_t}\Delta Ent_N(X^j)\mathbb{1}_{\{X_j \text{is used for split}\}},$$

Bagging and Random Forests XXI

where N_t denotes the total number of nodes for t-th tree and T denotes the total of trees.

Boosting I

Setting

Let us introduce to definitions first of *strong* and *weak* learnability that are given in the context of classification, where the boosting is mainly used.

Boosting II

Définition 4.1: Strong Learnability [Mohri et al., 2012]

A concept class $\mathcal{C}$ is said to be (strongly) PAC learnable if there exists an algorithm $\mathcal{A}$ and polynomial function $poly$ such that for any $\varepsilon > 0$ and $\delta > 0$, for all distributions $\mathcal{X}$ and for any target concept $c \in \mathcal{C}$, the following holds for any sample size $m \geq poly(1/\varepsilon, 1/\delta, d, size(c))$:

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

where h_S is the hypothesis returned by $\mathcal{A}$ when trained on S. If $\mathcal{A}$ further runs in $poly(1/\varepsilon, 1/\delta, d, size(c))$, then $\mathcal{C}$ is said to be efficiently PAC-learnable.

Boosting III

In this definition, a concept c is a function from $\mathcal{X}$ to $\mathcal{Y}$ that reach a specific target. As an example, a concept may be the set of points inside a triangle.
A concept class is a set of concepts we may wish to learn and is denoted by $\mathcal{C}$. This could, for example, be the set of all triangles in the plane.

According to the previous definition, a concept class $\mathcal{C}$ is thus PAC-learnable if the hypothesis returned by the algorithm after observing a number of points polynomial in $1/\varepsilon$ and $1/\delta$ is approximately correct (error at most ε) with high probability (at least $1 - \delta$).

Boosting IV

Définition 4.2: Weak Learnability [Mohri et al., 2012]

A concept class $\mathcal{C}$ is said to be weakly PAC learnable if there exists an algorithm $\mathcal{A}, \gamma > 0$ and polynomial function $poly$ such that for any $\delta > 0$, for all distributions $\mathcal{X}$ and for any target concept $c \in \mathcal{C}$, the following holds for any sample size $m \geq poly(1/\delta, d, size(c))$:

$$\underset{S \sim \mathcal{D}^m}{\mathbb{P}} \left[\mathcal{R}(h_S) \leq \frac{1}{2} - \gamma \right] \geq 1 - \delta,$$

where h_S is the hypothesis returned by $\mathcal{A}$ when trained on S. when such an algorithm exists, it is called a weak learning algorithm, a weak learner or a base classifier.

Boosting V

Note that the difference between the two definitions is based on the error of the hypothesis h_S.

In the first one, we require this classifier to achieve an error of at most ε for a sufficiently large sample $m(\varepsilon)$.

In the second one we just want the same classifier to be slightly better (of a parameter $\gamma > 0$) than the random classifier.

Boosting VI

Strong classifiers :

- Decision trees
- Over parameterized neural networks

Weak classifiers :

- Small decision trees with a depth of one or two
- Linear classifiers as linear SVM when the problem is non linearly separable

Boosting VII

If bagging has worked with strong classifiers, boosting will work with weak ones. It will try to combine them in order to build a strong classifier, but the way it works is different from the bagging procedure. Instead of building bootstrap samples, we will modify the data distribution. This is the aim of the **Adaboost** algorithm.

Boosting VIII

Presentation of Boosting

- We are going to learn a sequence of $(h_t)_{t\in\mathbb{N}}$ as it has been done previously, but the concept of "different" has not the same definition.
- Assumptions are no more independant, *i.e.* there is no more possibility to learn them in parallel because they will be learned iteratively.
- The aim is to learn a strong model using a sequence of weaker ones where the aim the weak learner h_{t+1} is to focus on the examples missclassified by h_t.
- This will be done by reweighting the examples at each iteration.

Adaboost [Freund and Schapire, 1999] which iteratively focuses on examples difficult to classify.

Boosting IX

Adaboost

```
Input: A learning sample $S$ of size m $m$,
$T$ models
Output: A model $H_T = \sum_{t=0}^{T} \alpha_t h_t$
begin
    Uniform distribution $w_i^{(0)} = \frac{1}{m}$
    for $t = 1, ..., T$ do
        Learn a classifier $h_t$ from an algorithm $\mathcal{A}$
        compute the error $\varepsilon_t$ of the algorithm.
        if $\varepsilon_t > 1/2$ then
            Stop
        else
            Compute $\alpha_{(t)} = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$
            $w_i^{(t)} = w_i^{(t-1)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$
    Set $H_T = \sum_{t=0}^{T} \alpha_t h_t$
    return $H_T$
```

Boosting X

Details

At a round t, the i-th training samples has the weight $w_i^{(t)}$. For $t = 1$, all the training samples have the same weights equal to $1/m$. A hypothesis h_t is learned and we can compute its classification error ε_t

$$\varepsilon_t = \sum_{i=1}^{m} w_i^{(t)} \mathbb{1}_{\{h_t(\mathbf{x}_i) y_i < 0\}}$$

Using this value, we can compute the *weight* of the learn classifier α_t :

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

Boosting XI

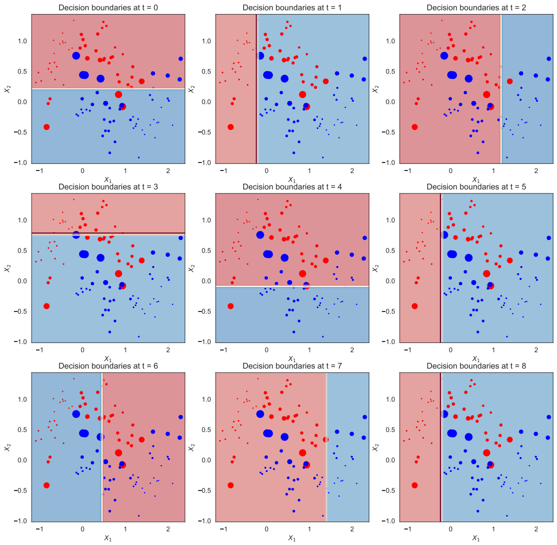
Keep in mind that the idea is to learn a hypothesis H_T that be expressed as a linear combination of weak learners h_t. The better the weak learner, the greater the weight.

The remaining of the procedure consist in finding a good reweighting function of the training samples such that, during the next round, the new classifier h_{t+1} is able to correct the mistakes done by the classifier h_t. This is done using the following update rule :

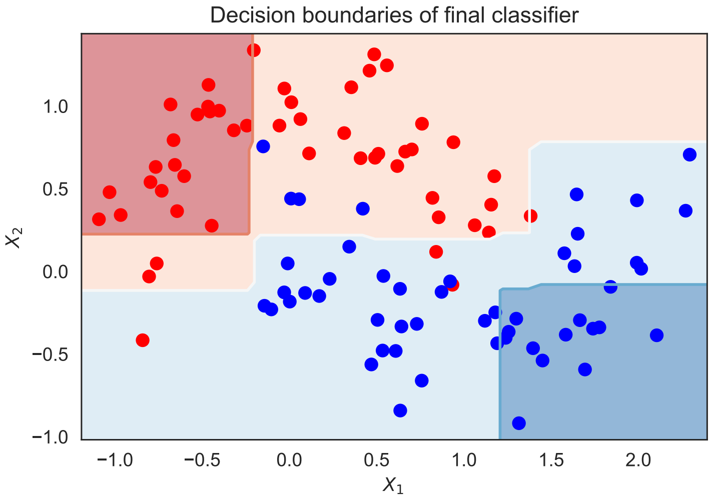
$$w_i^{(t+1)} = w_i^{(t)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t},$$

where Z_t is normalization factor which ensure that the sum of the weights is equal to 1.

Boosting XII



Boosting XIII



Boosting XIV

Theoretical Analysis

Proposition 4.1: Theoretical bound Adaboost

The empirical error of the classifier returned by Adaboost verifies :

$$\mathcal{R}_S(H_T) \leq \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2}-\varepsilon\right)^2\right].$$

Furthermore, if for all $t \in [\![1, T]\!]$, $\gamma \leq \left(\frac{1}{2}-\varepsilon_t\right)$, then :

$$\mathcal{R}_S(H_T) \leq \exp\big(-2\gamma^2 T\big).$$

Boosting XV

Proof

First, we recall that the exponential function is an upper bound of the indicator function, *i.e.*

$$\forall(\mathbf{x}, y)\ \mathbb{1}_{\{yh(\mathbf{x})<0\}} \leq \exp(-yh(\mathbf{x})).$$

We can then upper bound the empirical risk $\mathcal{R}_S(H_T)$:

$$\begin{aligned}\mathcal{R}_S(H_T) &= \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}_{\{y_i h(\mathbf{x}_i)<0\}}, \\ &\leq \frac{1}{m}\sum_{i=1}^{m} \exp(-y_i H_T(\mathbf{x}_i)).\end{aligned}$$

Boosting XVI

We can now express this last sum using the normalization factor Z_t and the weights $w_i^{(t+1)}$. Indeed, we have :

$$w_i^{(t+1)} = w_i^{(t)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t},$$

$\downarrow$ by recursion

$$= \frac{1}{m} \times \frac{\exp\left(-\sum_{s=1}^{t} \alpha_s h_s(x_i)\right)}{\prod_{s=1}^{t} Z_s}$$

Thus, the empirical risk can be upper bounded as :

$$\mathcal{R}_S(H_T) \leq \frac{1}{m} \sum_{i=1}^{m} \exp(-y_i H_T(\mathbf{x}_i)),$$

Boosting XVII

$\downarrow$ using the definition of H_T and the previous recursion

$$\leq \frac{1}{m}\sum_{i=1}^{m}\left(m\prod_{t=1}^{T} Z_t\right) w_i^{(T+1)},$$
$$\leq \prod_{t=1}^{T} Z_t.$$

We will now focus on the normalization factor and see how we can write it as a function of the classification error ε_t for all $t \in [\![1, T]\!]$.

$$Z_t = \sum_{i=1}^{m} w_i^{(t)}) \exp(-\alpha_t y_i h_t(\mathbf{x}_i)),$$
$$= \sum_{i:y_i h_t(\mathbf{x}_i)=1} w_i^{(t)} \exp(-\alpha_t) + \sum_{i:y_i h_t(\mathbf{x}_i)=-1} w_i^{(t)} \exp(\alpha_t),$$

Boosting XVIII

$$
\begin{aligned}
&= (1-\varepsilon_t)\exp(-\alpha_t) + \varepsilon_t \exp(\alpha_t), \\
&= (1-\varepsilon_t)\sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}} + \varepsilon_t\sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}, \\
&= 2\sqrt{\varepsilon_t(1-\varepsilon_t)}.
\end{aligned}
$$

As the empirical risk $\mathcal{R}_S(H_T)$ is upper bounded by the product of the normalization factors, we directly have :

$$
\begin{aligned}
\mathcal{R}_S(H_T) &\leq \prod_{t=1}^{T} Z_t, \\
&\leq \prod_{t=1}^{T} 2\sqrt{\varepsilon_t(1-\varepsilon_t)},
\end{aligned}
$$

Boosting XIX

$$\downarrow \text{ using } 4\varepsilon_t(1-\varepsilon_t) = 1 - 4\left(\frac{1}{2} - \varepsilon_t\right)^2$$

$$\leq \prod_{t=1}^{T} \sqrt{1 - 4\left(\frac{1}{2} - \varepsilon_t\right)^2},$$

$$\downarrow \text{ for all } x \in \mathbb{R}\ 1 - x \leq \exp(-x)$$

$$\leq \prod_{t=1}^{T} \exp\left[-2\left(\frac{1}{2} - \varepsilon_t\right)^2\right],$$

$$\downarrow \text{ property of the exponential}$$

$$\leq \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2} - \varepsilon_t\right)^2\right].$$

This ends the first part of the proof.

Boosting XX

Furthermore, if for all t, $\gamma \leq \left(\frac{1}{2} - \varepsilon_t\right)$:

$$\mathcal{R}_S(H_T) \leq \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2} - \varepsilon_t\right)^2\right],$$

$\downarrow$ using the assumption

$$\leq \exp\left[-2\sum_{t=1}^{T}\gamma^2\right],$$

$$\leq \exp\left[-2T\gamma^2\right].$$

Boosting XXI

We did not explain previously where the expression of α_t comes from, but the answer is in the proof. Indeed, it is chosen to minimize the upper bound of the empirical error. Thus it is chosen to minimize the function :

$$\varphi : \alpha \mapsto (1-\varepsilon_t)\exp(-\alpha) + \varepsilon_t \exp(\alpha).$$

This function is convex as a convex combination of two convex functions. So, it reaches its minimum for a single value α_t.

$$\begin{aligned} \nabla\varphi(\alpha_t) &= 0, \\ -(1-\varepsilon_t)\exp(-\alpha_t) + \varepsilon_t \exp(\alpha_t) &= 0, \\ (1-\varepsilon_t)\exp(-\alpha_t) &= \varepsilon_t \exp(\alpha_t), \end{aligned}$$

Boosting XXII

$$\begin{aligned} \frac{1-\varepsilon_t}{\varepsilon_t} &= \varepsilon_t \exp(2\alpha_t), \\ \alpha_t &= \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\varepsilon_t\right). \end{aligned}$$

Note also that the proof is available not only for a binary output h but also for type of hypothesis for which the output is in the range $[-1,+1]$.

Boosting XXIII

To finish

- A simple algorithm to use with good guarantees in terms of convergence.
- Possibility to extend its study and measure its generalization capacities.
- We can draw a link with a gradient descent algorithm (coordinate gradient descent)

Before going on with other boosting strategies, we will present a last way to combine models called **stacking**.

Stacking I

Stacked Generalization or
Stacking [Wolpert, 1992, Džeroski and Ženko, 2004] is another ensemble methods which works rather differently from bagging and boosting. Although the idea is always to combine models in order to optimize performance, stacking is a kind of meta-model that will be learned with the help of several sub-models.

The differences between the two previous procedures are as follow :

(i) all the models can be different (for instance we can combine an SVM algorithm with decision tree) and they are learned using the same dataset ;

(ii) the weight of each model is not learned iteratively.

Stacking II

All the models are learned independently and a meta model is then used to learn the optimal weight to assign to each base learner.
Its architecture involves at least two different learner to be used and we thus have two different steps in the procedure :

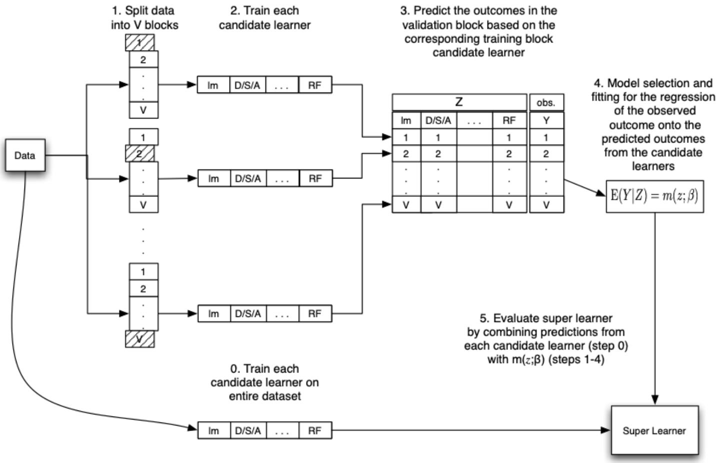
1. learn a set of base learner on a training data,
2. using the predictions made by each model as new features for the meta-model which will then learn a good combination of these predictions.

If each sub-model h_t is working with the initial features $\mathbf{x}$, the meta-model is working with the predictions of each sub-model, *i.e.*, the features of the metal model are $(h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_T(\mathbf{x}))$. Note that we dot specify the output of h_t and it can be a real value, a binary output, probability, *etc*.

Stacking III

We can go a little bit further and learn what we call a **Super Learner** [Van der Laan et al., 2007] which can be seen as a generalization of stacking with k-fold cross validation procedure as depicted in the next Figure[1].

Stacking IV


1. Split data into V blocks
2. Train each candidate learner
3. Predict the outcomes in the validation block based on the corresponding training block candidate learner
4. Model selection and fitting for the regression of the observed outcome onto the predicted outcomes from the candidate learners
$E(Y|Z) = m(z;\beta)$
5. Evaluate super learner by combining predictions from each candidate learner (step 0) with m(z;β) (steps 1-4)
0. Train each candidate learner on entire dataset
Data
lm
D/S/A
RF
Z
obs.
Y
Super Learner

Stacking V

The procedure can be described as follows :

1. Split the training data into k-folds (as in a k-fold cross validation procedure)
2. Train each base learner on $k - 1$ folds and use the remaining by computing their outputs through each base learner.
3. Repeat the previous point by changing the *validation* fold until all folds have been used as the *validation* one.
4. Use the new representation of the data to learn the parameters of your meta-model.
5. Evaluate its performance on the test data.

1. This image was extracted from the Super Learner course, slide 17.

Gradient Boosting I

Generalities

The Adaboost algorithm is based on the *exponential* loss, however, such a loss is not suited for all settings and it is sometimes better to use other losses depending on the model you want to learn or for a specific application.

This motivation leads us to the presentation of a more general boosting algorithm, the *Gradient Boosting* [Friedman, 2000].

Gradient Boosting II

Unlike the well-known Adaboost algorithm [Freund and Schapire, 1999], gradient boosting performs an optimization in the *function* space rather than in the *parameter* space.
At each iteration, a weak learner h_t is learned using the *residuals* (or the errors) obtained by the linear combination of the previous models.
The linear combination H_t at time t is defined as follows :

$$H_t = H_{t-1} + \alpha_t h_t \tag{3}$$

where H_{t-1} is the linear combination of the first $t-1$ models and α_t is the weight given to the t-th weak learner.

Gradient Boosting III

The weak learners are trained on the *pseudo-residuals* r_i of the current model. These pseudo-residual are given by the negative gradient, $-g_t$, of the used loss function ℓ with respect to the current prediction $H_{t-1}(\mathbf{x}_i)$:

$$r_i = -g_t(\mathbf{x}_i) = -\left[\frac{\partial \ell(y_i, H_{t-1}(\mathbf{x}_i))}{\partial H_{t-1}(\mathbf{x}_i)}\right].$$

Once the pseudo-residual r_i are computed, the following optimization problem is solved :

$$(h_t, \alpha_t) = \arg\min_{\alpha, h} \sum_{i=1}^{m} (r_i - \alpha h(\mathbf{x}_i))^2.$$

Finally, the update rule (3) is applied.

Gradient Boosting IV

This algorithm has been first developed for classification and regression trees, and most of the work and libraries such as **XGBoost** [Chen and Guestrin, 2016] are using decision trees as weak learners. The procedure is summarized as follows

```
**Input:** Initial hypothesis
    $H_0 \quad H^0(\mathbf{x}_i) = \underset{\rho \in \mathbb{R}}{\arg\min} \sum_{i=1}^m \ell(y_i, \rho) \; \forall i = 1, \dots, m$
**begin**
    **for** $t = 1, ..., T$ **do**
        Compute pseudo-residuals :
        $\tilde{y}_i = -\frac{\partial \ell(y_i, H_{t-1}(\mathbf{x}_i))}{\partial H_{t-1}(\mathbf{x}_i)}, \; \forall i = 1, ..., m$
        Fit the residuals : $a_t = \underset{a \in \mathbb{R}^d}{\arg\min} \; \sum_{i=1}^m (\tilde{y}_i - h_a(\mathbf{x}_i))^2$ to learn
         the new hypothesis
        Learn the weight of the classifier $h_{a^t}$ :
        $\alpha^t = \underset{a \in \mathbb{R}^+}{\arg\min} \; \sum_{i=1}^m \ell(y_i, H_{t-1}(\mathbf{x}_i) + \alpha h_{a^t}(\mathbf{x}_i))$
        Update $H_t(\mathbf{x}_i) = H_{t-1}(\mathbf{x}_i) + \alpha^t h_{a^t}(\mathbf{x}_i)$
    **return** $H_T$
```

Gradient Boosting V

Examples

Let us show what are the pseudo residuals for two different losses ℓ : the *square loss* for a regression task and the *logistic loss* for classification task.

- Using the square loss ℓ, the pseudo residuals are defined by :

$$\begin{aligned}\tilde{y} &= -\frac{\partial \ell(y, H_{t-1}(\mathbf{x}))}{\partial H_{t-1}(\mathbf{x})}, \\ &= -\frac{\partial (y - H_{t-1}(\mathbf{x}))^2}{\partial H_{t-1}(\mathbf{x})}, \\ &= 2(y - H_{t-1}(\mathbf{x})).\end{aligned}$$

Gradient Boosting VI

The pseudo residual is just twice the difference between the true value and the predicted value (remember that this loss is mainly used in the regression setting).

- Using the logistic loss ℓ (for classification), the pseudo residuals are defined by :

$$\begin{aligned}\tilde{y} = & -\frac{\partial \ell(y, H_{t-1}(\mathbf{x}))}{\partial H_{t-1}(\mathbf{x})}, \\ = & -\frac{\partial \ln\left(1+\exp\left(-2yH_{t-1}(\mathbf{x})\right)\right)}{\partial H_{t-1}(\mathbf{x})}, \\ = & \frac{2y}{\left(1+\exp\left(-2yH_{t-1}(\mathbf{x})\right)\right)}\end{aligned}$$

Gradient Boosting VII

This is not the commonly used version of this loss that is used. Because $H_{t-1}(\mathbf{x})$ s a real value, we rather pass this output in the logistic function in order to get probabilities to belong in a given class. In such a case, the pseudo residuals are given by :

$$\begin{aligned}
\tilde{y} &= -\frac{\partial \ell(y, H_{t-1}(\mathbf{x}))}{\partial H_{t-1}(\mathbf{x})}, \\
&= -\frac{\partial - y \ln\left(\sigma(H_{t-1}(\mathbf{x}))\right) - (1-y)\ln\left(1 - \ln\left(\sigma(H_{t-1}(\mathbf{x}))\right)\right)}{\partial H_{t-1}(\mathbf{x})}, \\
&\downarrow \text{ similar development as the presentation of logistic regression} \\
&= y - \sigma(H_{t-1}(\mathbf{x})),
\end{aligned}$$

where σ denotes the logistic function $\sigma(x) = (1 + e^{-x})^{-1}$.

Gradient Boosting VIII

We can also draw a parallel between the gradient boosting algorithm and a gradient descent algorithm.

This is more natural since the algorithm involves the gradient.

Here, the parallel can be drawn with the gradient descent with optimal steepest descent

Gradient Boosting IX

Presentation of XGBoost

We consider a loss ℓ (we suppose the function is twice differentiable) and the following optimization problem :

$$\min \sum_{i=1}^{m} \ell(y_i, \hat{y}_i) + \beta\mathcal{L} + \frac{\lambda}{2}\sum_{j=1}^{\mathcal{L}} (f_j^{(t)})^2. \tag{4}$$

where $\beta\mathcal{L}$ and $\frac{\lambda}{2}\sum_{j=1}^{\mathcal{L}}(h_t^{(j)})^2$ are two regularization terms used to control the number of leaves and the weight of each leaf $f_t^{(j)}$ for the learned tree at iteration t.

Gradient Boosting X

We recall that the models are learned in an additive manner, so let us denote $\hat{y}^{(t-1)}$, the predicted value by the first $t-1$ functions h_k, i.e. $\hat{y}_i^{(t-1)} = \sum_{k=1}^{t-1} h_t(\mathbf{x}_i) = H_{t-1}(\mathbf{x}_i)$.

Let us now study how the next model is learned. For this purpose, we rewrite the quantity (4) to minimize as follows :

$$\sum_{i=1}^{m} \ell(y_i, \hat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)) + \beta\mathcal{L} + \frac{\lambda}{2}\sum_{j=1}^{\mathcal{L}}(h_t^{(j)})^2. \tag{5}$$

We only use the additive definition of the model.

Gradient Boosting XI

In practice, [Chen and Guestrin, 2016] only consider a second order approximation of the function they aim to optimize. This second order approximation is done with respect to the predicted value at the previous iteration, i.e. $\hat{y}_i^{(t-1)}$.
We will denote by respectively g and f the first and second order derivatives of the function ℓ with respect to $\hat{y}^{(t-1)}$. We can rewrite (5) as follows :

$$\sum_{i=1}^{m}\left[\ell(y_i,\hat{y}_i^{(t-1)}) + h_t(\mathbf{x}_i)g(\mathbf{x}_i) + \frac{1}{2}h_t^2(\mathbf{x}_i)f(\mathbf{x}_i)\right] + \beta\mathcal{L} + \frac{\lambda}{2}\sum_{j=1}^{\mathcal{L}}(f_t^{(j)})^2. \tag{6}$$

Gradient Boosting XII

Remember that we aim to learn the function $\mathbf{f}_t = (h_t^{(j)})_{j=1,\dots,\mathcal{L}}$.
So let us consider a leaf j and denote by I_j the set of index i such that $\mathbf{x}_i$ falls in the leaf l_j. Thus, using (6), the function $h_t^{(j)}$ shall minimize the following quantity V_j for a given index j :

$$V_j \ = \ \sum_{i \in I_j} \left[g(\mathbf{x}_i) h_t^{(j)}(\mathbf{x}_i) + \frac{1}{2} \left(\lambda + f(\mathbf{x}_i) \right) \left(h_t^{(j)}(\mathbf{x}_i) \right)^2 \right]. \qquad (7)$$

Gradient Boosting XIII

This function is convex and the minimum is given by the solution of *Euler's equation*, *i.e.* the function $f^{(t)}$ for which the gradient vanishes. This solution is given by :

$$h_t^{(j)} = -\frac{\sum_{i \in I_j} g(\mathbf{x}_i)}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}. \tag{8}$$

Gradient Boosting XIV

Example

Let us come back to our previous example where we have considered the *square loss* and the *logistic loss* and let us see what the weights of the leaves are when we use these two losses.

For the square loss $\ell(\hat{y}^{(t-1)}) = \frac{1}{2}(y - \hat{y}^{(t-1)})^2$. The gradient g with respect to the prediction is

$$g(\hat{y}^{(t-1)}) = \frac{\partial \ell}{\partial \hat{y}^{(t-1)}}(\hat{y}^{(t-1)}) = (\hat{y}^{(t-1)} - y).$$

And the second order derivative f with respect to the prediction is

$$f(\hat{y}^{(t-1)}) = \frac{\partial^2 \ell}{\partial (\hat{y}^{(t-1)})^2}(\hat{y}^{(t-1)}) = 1.$$

Gradient Boosting XV

So the optimal value of a leaf $h_t^{(j)}$ is :

$$h_t^{(j)} = -\frac{\sum_{i \in I_j} g(\mathbf{x}_i)}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda} = \frac{\sum_{i \in I_j} (y_i - \hat{y}^{(t-1)})}{\lambda + |I_j|}.$$

Thus, when learning the first tree, with the assumption $\hat{y}_i^{(0)} = 0$ for all i, we find that the optimal score in a leaf is equal to the average of the instances values in the leaf. For subsequent iterations, the optimal score of each leaf becomes the average of the pseudo-residuals.

Gradient Boosting XVI

Let us now focus on the logistic loss

$$\ell(\hat{y}^{(t-1)}) = -(y \ln\Big(p(\hat{y}^{(t-1)})\Big) + (1-y)\ln\Big(1 - p(\hat{y}^{(t-1)})\Big),$$

where p is the logistic function.

The gradient g with respect to the prediction is

$$g(\hat{y}^{(t-1)}) = \frac{\partial \ell}{\partial \hat{y}^{(t-1)}}(\hat{y}^{(t-1)}) = (p(\hat{y}^{(t-1)}) - y).$$

And the second order derivative f with respect to the prediction is

$$f(\hat{y}^{(t-1)}) = \frac{\partial^2 \ell}{\partial (\hat{y}^{(t-1)})^2}(\hat{y}^{(t-1)}) = p(\hat{y}^{(t-1)})(1 - p(\hat{y}^{(t-1)})).$$

Gradient Boosting XVII

So the optimal value of a leaf $h_t^{(j)}$ is :

$$h_t^{(j)} = -\frac{\sum_{i\in I_j} g(\mathbf{x}_i)}{\sum_{i\in I_j} f(\mathbf{x}_i) + \lambda} = \frac{\sum_{i\in I_j} (p(\hat{y}^{(t-1)}) - y)}{\lambda + \sum_{i\in I_j} p(\hat{y}^{(t-1)})(1 - p(\hat{y}^{(t-1)})}.$$

Gradient Boosting XVIII

Back to XGBoost

Let us now focus on the splitting criterion. Optimal weights are found (8), compute the optimal value $V_j^\star$ of the loss by using (7) :

$$
\begin{aligned}
V_j^\star &= \sum_{i \in I_j} \left[\underbrace{- g(\mathbf{x}_i) \frac{\sum_{i \in I_j} g(\mathbf{x}_i)}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}} + \underbrace{\frac{1}{2} (\lambda + f(\mathbf{x}_i)) \left(- \frac{\sum_{i \in I_j} g(\mathbf{x}_i)}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda} \right)} \right. \\
&= - \frac{\left(\sum_{i \in I_j} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda} + \frac{1}{2} \frac{\left(\sum_{i \in I_j} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}, \\
V_j^\star &= - \frac{1}{2} \frac{\left(\sum_{i \in I_j} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}.
\end{aligned}
$$

Gradient Boosting XIX

This formula is used to measure the quality of a leaf. It can be seen as a generalized formula for Gini Impurity for any loss function. Using this new measure, they define their splitting criterion, i.e. the gain associated to a split, as follows :

$$\frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g(\mathbf{x}_i)\right)^2}{\sum_{i\in I_L} f(\mathbf{x}_i)+\lambda}+\frac{\left(\sum_{i\in I_R} g(\mathbf{x}_i)\right)^2}{\sum_{i\in I_R} f(\mathbf{x}_i)+\lambda}-\frac{\left(\sum_{i\in I} g(\mathbf{x}_i)\right)^2}{\sum_{i\in I} f(\mathbf{x}_i)+\lambda}\right]-\beta,$$

where $I = I_L \cup I_R$ for a binary tree and the parameter β is used to control the number of leaves.

References I

Bartlett, P. L. and Mendelson, S. (2003).
Rademacher and gaussian complexities : Risk bounds and structural results.
Journal of Machine Learning Research, 3 :463–482.

Breiman, L. (2001).
Random forests.
Machine Learning, 45(1) :5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984).
Classification and regression trees.
The Wadsworth statistics/probability series. Wadsworth and Brooks/Cole Advanced Books and Software, Monterey, CA.

References II

Chen, T. and Guestrin, C. (2016).
Xgboost : A scalable tree boosting system.
In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794. ACM.

Cox, D. R. (1958).
The regression analysis of binary sequences (with discussion).
Journal of the Royal Statistical Society, 20 :215–242.

Cramer, J. (2003).
The origins of logistic regression.
SSRN Electronic Journal.

References III

Džeroski, S. and Ženko, B. (2004).
Is combining classifiers with stacking better than selecting the best one ?
Machine learning, 54 :255–273.

Freund, Y. and Schapire, R. E. (1999).
A short introduction to boosting.
In *In Proceedings of the Sixteenth IJCAI*, pages 1401–1406. Morgan Kaufmann.

Friedman, J. H. (2000).
Greedy function approximation : A gradient boosting machine.
Annals of Statistics, 29 :1189–1232.

References IV

Jensen, J. L. W. V. (1906).
Sur les fonctions convexes et les inégalités entre les valeurs moyennes.
Acta Mathematica, 30 :175–193.

Koltchinskii, V. and Panchenko, D. (2000).
Rademacher processes and bounding the risk of function learning.
In Giné, E., Mason, D. M., and Wellner, J. A., editors, *High Dimensional Probability II*, pages 443–457. Birkhäuser Boston.

Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012).
Foundations of Machine Learning.
The MIT Press.

Quinlan, J. R. (1986).
Induction of decision trees.
Mach. Learn., 1(1) :81–106.

References V

Rokach, L. and Maimon, O. (2005).
Top-down induction of decision trees classifiers - a survey.
IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 35(4) :476–487.

Safavian, S. R. and Landgrebe, D. (1991).
A survey of decision tree classifier methodology.
IEEE Transactions on Systems, Man, and Cybernetics, 21(3) :660–674.

Van der Laan, M. J., Polley, E. C., and Hubbard, A. E. (2007).
Super learner.
Statistical applications in genetics and molecular biology, 6(1).

References VI

Vapnik, V. and Chervonenkis, A. (1971).
On the uniform convergence of relative frequencies of events to their probabilities.
Theory of Probability & Its Applications, 16(2) :264–280.

Wolpert, D. H. (1992).
Stacked generalization.
Neural networks, 5(2) :241–259.