# Ensemble Methods

Fabrice Rossi

SAMM
Université Paris 1 Panthéon Sorbonne

2018

# Setting

## Standard supervised learning

- $\mathcal{X}$ the "input" space and $\mathcal{Y}$ the "output" space
- $D$ a fixed and unknown distribution on $\mathcal{X} \times \mathcal{Y}$
- a loss function $l$
- a data set $\mathcal{D} = ((\mathbf{X}_i, \mathbf{Y}_i))_{1 \leq i \leq N}$ with $\mathcal{D} \sim D^N$

## Combining models

- several models: $g_1, \ldots, g_K$, $K$ functions from $\mathcal{X}$ to $\mathcal{Y}$
- can one define $g$ from $g_1, \ldots, g_K$ in order to get better performances than with using only one of the $g_k$?

# Majority voting

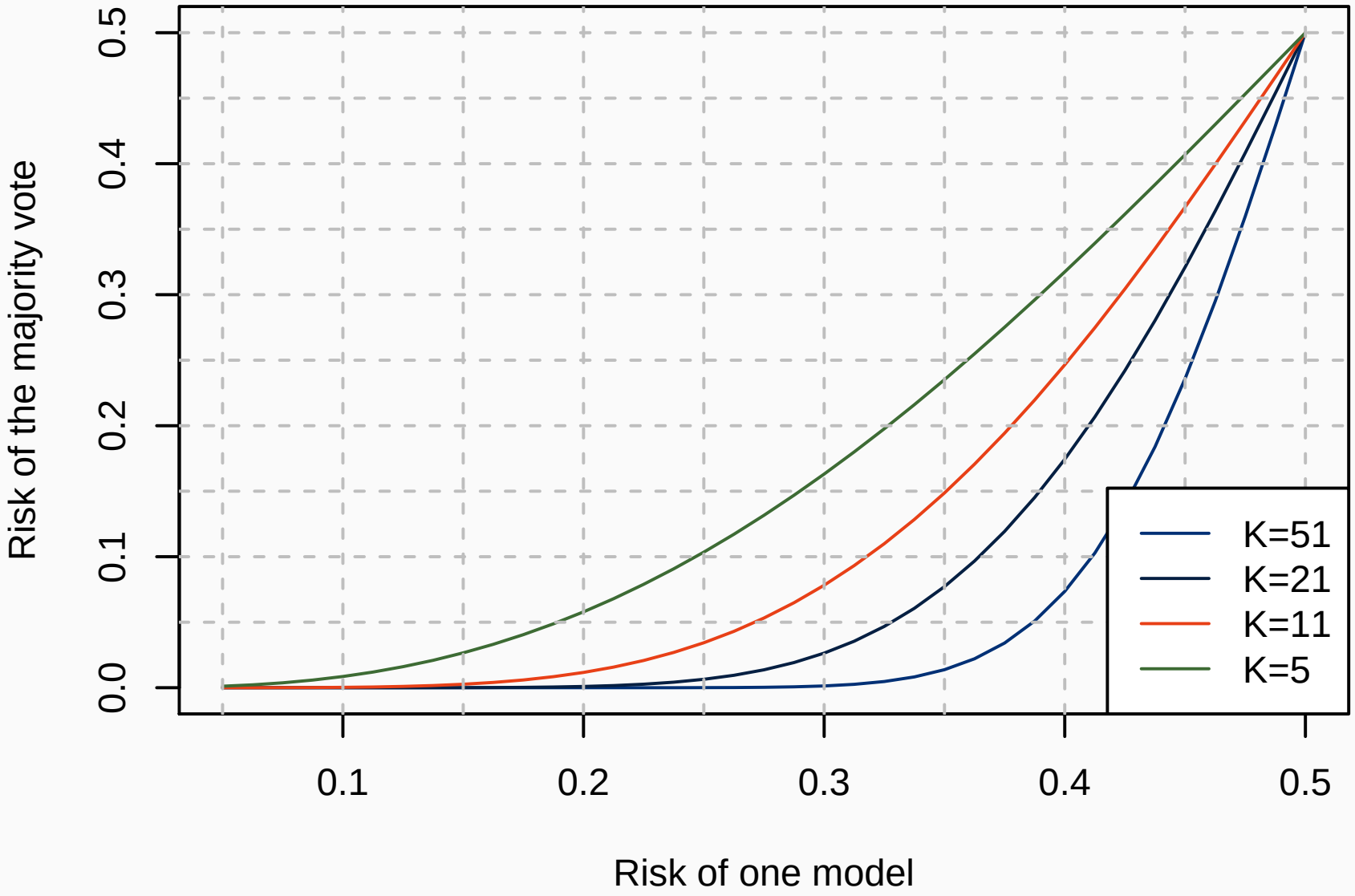
## A simple example

- $\mathcal{Y} = \{-1, 1\}$ and $l(p, t) = \mathbf{1}_{p \neq t}$
- $g(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} |\{1 \leq k \leq K \mid g_k(\mathbf{x}) = y\}|$ (majority voting)
- is $g$ better than the $g_k$?

## Analysis

- (partially unrealistic) assumptions:
  - $\forall k, R_l(g_k) = p < \frac{1}{2}$
  - the $g_k(\mathbf{x})$ are independent random variables (!)
- $\mathbb{P}(g(\mathbf{X}) \neq \mathbf{Y}) = \mathbb{P}\left(|\{1 \leq k \leq K \mid g_k(\mathbf{X}) \neq \mathbf{Y}\}| > \frac{K}{2}\right)$
- Binomial random variable with parameter $p$ and $K$
  - $p = 0.45$ and $K = 21$: $R_l(g) = 0.321$
  - $p = 0.3$ and $K = 21$: $R_l(g) = 0.0264$

# General behavior



# Main ideas

## To combine models

we need

- models with at least some predictive capabilities
- with independent errors

## Bias variance decomposition

- another argument for model combination
- data point of view rather than model point of view:
  - a unique learning algorithm
  - several data sets
- but a similar conclusion

## Ensemble methods

Techniques to combine models

# Outline

General principles

Bagging and Random Forests

Boosting

Other solutions

# Two main questions

## Building an ensemble method

1. given $K$ models, how to combine them efficiently?
2. given a data set and some learning strategies, how to produce $K$ models that have “independent” errors?

# Combining models

## Combination

- $g_1, \ldots, g_K$, $K$ functions from $\mathcal{X}$ to $\mathcal{Y}$
- how to build $g$ from $g_1, \ldots, g_K$?
- numerous solutions related to the nature of $\mathcal{Y}$ and to $l$

## Linear combination

- for $\mathcal{Y} = \mathbb{R}$ (or $\mathbb{R}^Q$)
- define

$$g(\mathbf{x}) = \sum_{k=1}^{K} w_k g_k(\mathbf{x})$$

- one *could* learn the $(w_k)_{1 \leq k \leq K}$ from the data but:
  - risk of overfitting
  - when there is no overfitting, the gain is small
- frequent solution $w_k = \frac{1}{K}$

# Discrete predictions

## Discrete case

- when $|\mathcal{Y}| < \infty$
- majority voting

$$g(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} |\{1 \leq k \leq K \mid g_k(\mathbf{x}) = y\}|$$

## Binary case with score

- when $\mathcal{Y} = \{-1, 1\}$ and $g_k(\mathbf{x}) = \text{sign}(f_k(\mathbf{x}))$
- linear combination

$$g(\mathbf{x}) = \text{sign}\left(\sum_{k=1}^{K} w_k f_k(\mathbf{x})\right)$$

# Probabilities and more

## Probabilities

- when $|\mathcal{Y}| < \infty$
- $g_k(\mathbf{y}, \mathbf{x})$ estimates $\mathbb{P}(\mathbf{Y} = \mathbf{y} | \mathbf{X} = \mathbf{x})$
- weighted product ($Z$ is a normalization constant)

$$g(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \prod_{k=1}^{K} g_k(\mathbf{y}, \mathbf{x})^{w_k}$$

## Numerous other solutions

- mostly based on the concept of generalized mean
- adapted to special cases such as ranking (or even unsupervised settings)
- but in general, the mean, the uniform product or the majority voting are sufficient

# Building models

## Diversity

- is mandatory!
- most learning algorithms are deterministic: same data $\Rightarrow$ same model
- two general solutions:
  1. introduce some randomness in the algorithm (or leverage the natural randomness of the algorithm)
  2. run the same algorithm on modified data (randomly or deterministically)

# Data level diversity

## Instance subsets

- each classifier is obtained on a “subset” of the data set
- *Bagging*: bootstrap sample
- *Cross-validated committee*: block decomposition

## Feature subsets

- each classifier is obtained using only a subset of the features
- *Random Subspace Method*: as is
- *Random Forests*: combined with *Bagging*

# Data level diversity

## Weighted instances

- each classifier is obtained using a weighted version of the data set
- *Boosting*: sequential training where weights are increased for badly predicted instances so far

# Algorithmic level

## Feature subsets

- when features are randomly selected repeatedly at different stages of the algorithm
- *Random Forests*

## Natural instability

- complex models with strong dependence to the random initialization
- typical example: *Neural networks*
- could be forced further:
  - ensemble training
  - penalty for correlated prediction
  - *Negative Correlation Learning*

# Outline

# Bagging

## Boostrap Aggregating

- Breiman (1996)
- simple and efficient
- consists simply in training models on bootstrap samples!
- adapted to models with high variance such as decision trees

## Bagging and imbalanced data

- a bootstrap sample contains roughly 63.2% of the original data: could induce serious balance issues
- stratified bootstrap must be used
- avoid balancing the data *before* the bagging

# Out-of-bag estimate

## Principle

- leverage the fact that a bootstrap sample contains only 63.2% of the original data
- compute the prediction for a data point $x$ using only the models for which $x$ was not in the bootstrap sample

## Formal definition

- $K$ bootstrap samples $\mathcal{D}_1, \ldots, \mathcal{D}_K$ with associated models $g_1, \ldots, g_K$
- $O_i$: the set of indices $k$ for each $\mathbf{X}_i \notin \mathcal{D}_k$
- risk estimate (averaging case):

$$\widehat{R}_l^{oob}(g_{bag}) = \frac{1}{N} \sum_{i=1}^{N} l \left( \frac{1}{|O_i|} \sum_{k \in O_i} g_k(\mathbf{X}_i), \mathbf{Y}_i \right)$$

# Increasing the diversity

## Feature subsets

- the bootstrap might not induce enough diversity
- adding random subspace might also be insufficient

## Random Forest

- bagging of trees
- local random feature subsets
- tree growing:
  - standard CART approach: chose the best variable among all the variables
  - random forests: chose the best variable among a random subset of the variables
- no pruning

# Random Forests

## In practice

- one of the best state-of-the-art solution for classical data
- leverage the out-of-bag estimate (all in one solution)
- parallel computation of the trees
- classical parameters:
  - at least 500 trees (a.k.a. bootstrap samples), depends on the complexity of the data
  - the “optimal” size of the subset of variables depends on several aspects:
    - highly correlated variables do not need a large subset (!)
    - Breiman recommended to use $P/3$ for regression problems and $\sqrt{P}$ for classification problems
  - the parameters can (should) be optimized

# Outline

General principles

Bagging and Random Forests

Boosting

Other solutions

# Boosting

## Principle

- Freund and Schapire (AdaBoost 1997)
- sequential learning of an ensemble of models
- weight based variability: each model is obtained from the same data set but with different weights
- weights emphasize badly predicted examples

## Original AdaBoost

- classification $\mathcal{Y} = \{-1, 1\}$ and $l(p, t) = \mathbf{1}_{p \neq t}$
- binary or score based models $g_k(\mathbf{x}) = \text{sign}(f_k(\mathbf{x}))$

# AdaBoost

set $w_{1,i} = \frac{1}{N}$ for all $i$
**for** $k = 1$ to $K$ **do**
    learn $g_k$ on $(\mathbf{X}_i, \mathbf{Y}_i)_{1 \leq i \leq N}$ using the weights $w_{k,i}$
    compute $\epsilon_k = \sum_i w_{k,i} \mathbf{1}_{g_k(\mathbf{X}_i) \neq \mathbf{Y}_i}$
    compute $\alpha_k = \frac{1}{2} \log \frac{1-\epsilon_k}{\epsilon_k}$
    compute $w_{k+1,i} = w_{k,i} \exp(-\alpha_k \mathbf{Y}_i g_k(\mathbf{X}_i))$
    normalize the weights $w_{k+1,i}$
**end for**
final model $g = \text{sign}\left(\sum_k \alpha_k g_k\right)$

# Interpretation

## Analysis

- $\epsilon_k = \sum_i w_{ki} \mathbf{1}_{g_k(\mathbf{X}_i) \neq \mathbf{Y}_i}$: weighted empirical risk
- $\exp(-\alpha_k \mathbf{Y}_i g_k(\mathbf{X}_i))$:
  - if $\mathbf{Y}_i = g_k(\mathbf{X}_i)$, the prediction is correct and the weight is divided by $\exp(\alpha_k)$
  - if $\mathbf{Y}_i \neq g_k(\mathbf{X}_i)$, the prediction is incorrect and the weight is multiplied by $\exp(\alpha_k)$
- $\exp(\alpha_k) = \sqrt{\frac{1-\epsilon_k}{\epsilon_k}}$:
  - no effect when $g_k$ is a bad model (i.e. when $\epsilon_k$ is close to $\frac{1}{2}$)
  - strong effect when $g_k$ is a good model

# Additive models

## Model

- an additive model has the form

$$g(\mathbf{x}) = \sum_{k=1}^{K} \beta_k f(\mathbf{x}, \gamma_k),$$

  where $f(., \gamma_k)$ is a general parametric model (e.g. a tree)
- classical risk minimisation is complex (doable with respect to $\beta_k$ complex in general with respect to $\gamma_k$)

## Sequential optimization

- $h_k(\mathbf{x}) = \sum_{m=1}^{k} \beta_m f(\mathbf{x}, \gamma_m)$ ($g = h_K$)
- $h_k(\mathbf{x}) = h_{k-1}(\mathbf{x}) + \beta_k f(\mathbf{x}, \gamma_k)$
- optimize $h_1$, fix its parameters, then optimize $h_2$ with respect to $\beta_2$ and $\gamma_2$ only, etc.

# Exponential loss

## Standard surrogate loss

- $l_e(g(\mathbf{x}), \mathbf{y}) = e^{-\mathbf{y}g(\mathbf{x})}$
- upper convex approximation of the binary loss

## Sequential optimization

- additive to multiplicative

$$\begin{aligned} l_e(h_k(\mathbf{x}), \mathbf{y}) &= e^{-\mathbf{y}h_{k-1}(\mathbf{x})} e^{-\beta_k \mathbf{y} f(\mathbf{x}, \gamma_k)} \\ &= l_e(h_{k-1}(\mathbf{x}), \mathbf{y}) e^{-\beta_k \mathbf{y} f(\mathbf{x}, \gamma_k)} \end{aligned}$$

- empirical loss, with $w_{k,i} = \frac{1}{N} l_e(h_{k-1}(\mathbf{X}_i), \mathbf{Y}_i)$ (and $w_{1,i} = \frac{1}{N}$):

$$\widehat{R}_{l_e}(h_k) = \sum_{i=1}^{N} w_{k,i} e^{-\beta_k \mathbf{Y}_i f(\mathbf{X}_i, \gamma_k)}$$

# Link with AdaBoost

## Binary models

- $f(\mathbf{x}, \gamma_k) \in \{-1, 1\}$
- therefore

$$e^{-\beta_k \mathbf{Y}_i f(\mathbf{X}_i, \gamma_k)} = \begin{cases} e^{-\beta_k} & \text{if } f(\mathbf{X}_i, \gamma_k) = \mathbf{Y}_i \\ e^{\beta_k} & \text{if } f(\mathbf{X}_i, \gamma_k) \neq \mathbf{Y}_i \end{cases}$$

- empirical loss

$$\begin{aligned} \widehat{R}_{l_e}(h_k) &= e^{-\beta_k} \sum_{f(\mathbf{X}_i, \gamma_k) = \mathbf{Y}_i}^{N} w_{k,i} + e^{\beta_k} \sum_{f(\mathbf{X}_i, \gamma_k) \neq \mathbf{Y}_i}^{N} w_{k,i} \\ &= (e^{\beta_k} - e^{-\beta_k}) \sum_{i=1}^{N} w_{k,i} l_b(f(\mathbf{X}_i, \gamma_k), \mathbf{Y}_i) + e^{-\beta_k} \sum_{i=1}^{N} w_{k,i} \end{aligned}$$

where $l_b$ is the binary loss

# Link with AdaBoost

## Solving for $\gamma_k$

- both the weights $w_{k,i}$ and $\beta_k$ are held constant
- then we have a standard (weighted) empirical risk minimization:

$$\arg\min_{\gamma_k} \widehat{R}_{l_e}(h_k) = \arg\min_{\gamma_k} \frac{1}{N} \sum_{i=1}^{N} w_{k,i} l_b(f(\mathbf{X}_i, \gamma_k), \mathbf{Y}_i)$$

## Solving for $\beta_k$

- both the weights $w_{k,i}$ and $\gamma_k$ are held constant
- if we assume $\sum_i w_{k,i} = 1$, then

$$\arg\min_{\beta_k} \widehat{R}_{l_e}(h_k) = \frac{1}{2} \log \frac{1 - \epsilon_k}{\epsilon_k} = \alpha_k,$$

with $\epsilon_k = \sum_{i=1}^{N} w_{k,i} l_b(f(\mathbf{X}_i, \gamma_k), \mathbf{Y}_i)$

# Link with AdaBoost

## Weights

- $w_{1,i} = \frac{1}{N}$: initial set
- update

$$\begin{aligned} w_{k+1,i} &= l_e(h_k(\mathbf{X}_i), \mathbf{Y}_i), \\ &= w_{k,i} e^{-\beta_k \mathbf{Y}_i f(\mathbf{X}_i, \gamma_k)}, \\ &= w_{k,i} e^{-\alpha_k \mathbf{Y}_i f(\mathbf{X}_i, \gamma_k)}, \end{aligned}$$

  using the optimal value for $\beta_k$

## In summary

- AdaBoost is a form of sequential learning for $g(\mathbf{x}) = \sum_{k=1}^{K} \beta_k f(\mathbf{x}, \gamma_k)$ with the exponential loss
- with independent optimisation of $\beta_k$ and $\gamma_k$

# Gradient Boosting

## Generalization

- sequential learning of additive models
- arbitrary loss function

## Steepest descent

- iterative minimization of $\widehat{R}_l(g)$:
  1. start with some initial model $g_0$
  2. update the model following the steepest descent:
     $g_{k+1} = g_k - \rho \nabla \widehat{R}_l(g_k)$
- leads to an additive model if $\nabla \widehat{R}_l(g_k)$ is a function of the form $f(., \gamma_k)$
- *gradient boosting*: approximate $\nabla \widehat{R}_l(g_k)$ with $f(., \gamma_k)$

# Gradient Boosting

In the following algorithm, $l'$ is the derivative of the loss function $l$ with respect to the prediction

set $g_0$ to the best constant function: $g_0(\mathbf{x}) = \arg\min_{\mathbf{y}} \sum_i l(\mathbf{y}, \mathbf{Y}_i)$
**for** $k = 1$ to $K$ **do**
compute $r_{k,i} = -l'(g_{k-1}(\mathbf{X}_i), \mathbf{Y}_i)$
learn $f_k$ on $(\mathbf{X}_i, r_{k,i})_{1 \leq i \leq N}$
compute

$$\rho_k = \arg\min_{\rho} \sum_i l(g_{k-1}(\mathbf{X}_i) + \rho f_k(\mathbf{X}_i), \mathbf{Y}_i)$$

set $g_k = g_{k-1} + \rho_k f_k$
**end for**
final model $g_K$

# Regularization

## The case of trees

- one could use arbitrary trees for $f_k$
- tends to overfit and to slow down learning
- common practice: limit the complexity of the trees
- typical choices: between 4 and 10 leaves

## General regularization

- size of the ensemble: $K$
- shrinkage: replace $g_k = g_{k-1} + \rho_k f_k$ by $g_k = g_{k-1} + \nu \rho_k f_k$ (with $0 < \nu < 1$)
- standard regularization terms (e.g. ridge like)

## In practice

- one of the best state-of-the-art solution for classical data
- very fast implementations such as XGBoost
- numerous practical tricks (e.g. including local random feature subsets in tree growing)
- parameters:
  - main one: size of the ensemble ($K$), must be tuned
  - other parameters can be optimized (e.g., number of leaves) but with less impact on the performances

# Outline

General principles

Bagging and Random Forests

Boosting

Other solutions

# Stacking

## Main idea

- learning to combine models
- Stacked generalization (Wolpert, 1992):
  - learn $K$ models $g_1, \ldots, g_K$
  - learn a new model $g$ using as data set $((g_1(\mathbf{X}_i), \ldots, g_K(\mathbf{X}_i)), \mathbf{Y}_i)_{1 \leq i \leq N}$
  - the final model is the composition of $g_1, \ldots, g_K$ with $g$

## Practical aspects

- to avoid massive overfitting the procedure leverages $L$ fold cross validation:
  - $((g_1(\mathbf{X}_i), \ldots, g_K(\mathbf{X}_i)), \mathbf{Y}_i)$ is computed by models learned on the blocks that do not contain $(\mathbf{X}_i, \mathbf{Y}_i)$
  - the $g_1, \ldots, g_K$ are "relearned" on the full data set once $g$ has been built
- works best with continuous outputs (e.g. scores)

# Mixture of experts

## Adaptive combination

- standard linear combination: $g(\mathbf{x}) = \sum_{k=1}^{K} w_k g_k(\mathbf{x})$
- data dependent combination: $g(\mathbf{x}) = \sum_{k=1}^{K} w_k(\mathbf{x}) g_k(\mathbf{x})$

## In practice

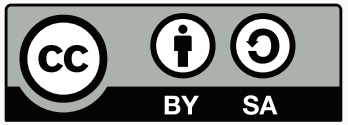
- $(w_1(\mathbf{x}), \ldots, w_K(\mathbf{x}))$ is a *gating* function: positive values, sum to one
- global learning:
  - emphasize task decomposition
  - simple models and complex gating
- related to ensemble approaches but rather different philosophy

# In summary

## Ensemble methods

- Random Forests
- Boosting
- pros and cons
  - \+ state of the art performances
  - \+ straightforward parallel implementation
  - \+ efficient large scale implementations
  - \+ adapted to mixed data
  - \+ handle missing data
  - \- black box models
  - \- high runtime compared to many other models
  - \- high storage cost compared to many other models

# Licence



# Version

Last git commit: 2018-06-06
By: Fabrice Rossi (Fabrice.Rossi@apiacoa.org)
Git hash: 1b39c1bacfc1b07f96d689db230b2586549a62d4