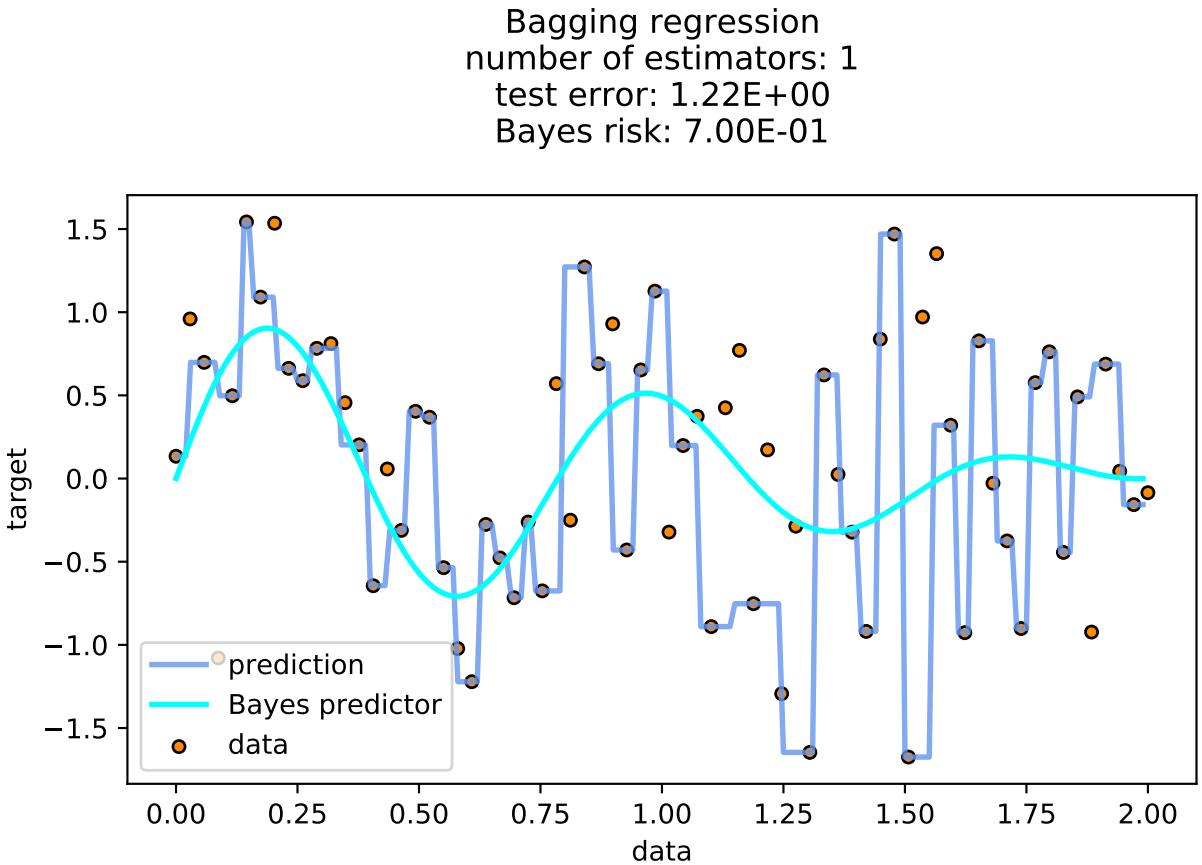
Fondamentaux théoriques du machine learning



Overview of lecture 8

Stochastic gradient descent (SGD) II

- Logistic regression
- Gradient descent
- Stochastic gradient descent (SGD)
- Stochastic gradient with averaging (SGA)

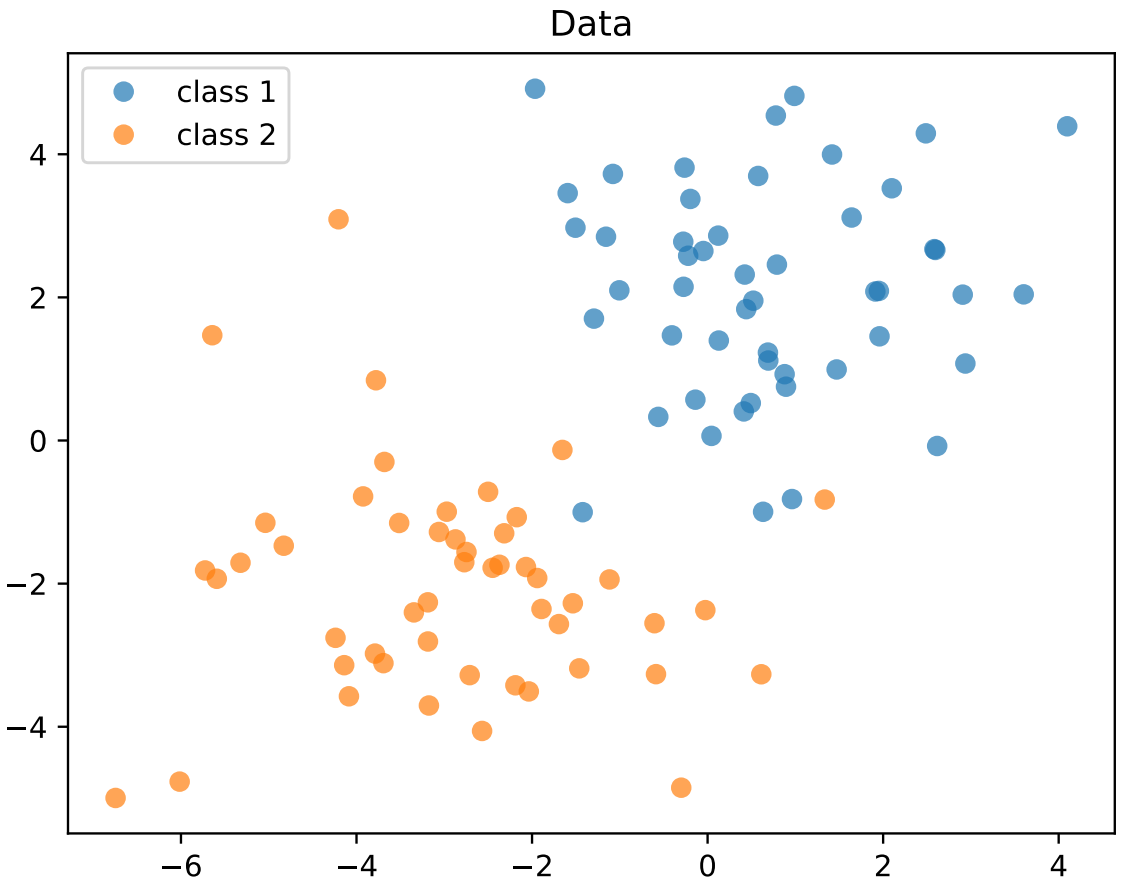
Decision trees

- Decision trees
- Construction of a decision tree
- Tree pruning

Ensemble learning

- Bagging
- Random forest
- Boosting

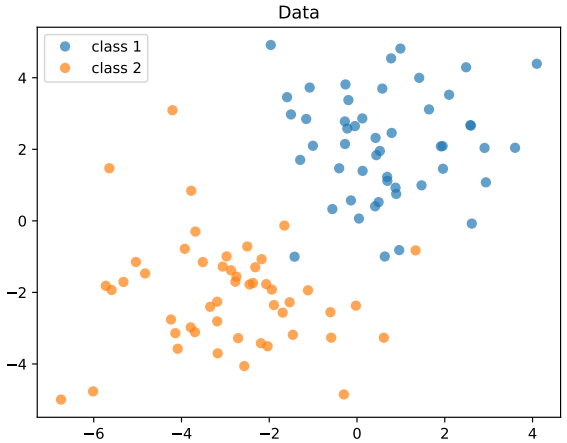
Logistic regression



Logistic regression

- $\mathcal{X} = \mathbb{R}^2$
- $\mathcal{Y} = \{-1, 1\}$
- $D_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
- Logistic loss :

$$l(\hat{y}, y) = \log(1 + e^{-\hat{y}y}) \tag{1}$$



Logistic regression

- $\mathcal{X} = \mathbb{R}^2$
- $\mathcal{Y} = \{-1, 1\}$
- Logistic loss :

$$l(\hat{y}, y) = \log(1 + e^{-\hat{y}y}) \tag{2}$$

$$\begin{aligned} \frac{\partial l}{\partial \hat{y}}(\hat{y}, y) &= \frac{-ye^{-\hat{y}y}}{1 + e^{-\hat{y}y}} \\ &= \frac{-ye^{-\hat{y}y}e^{\hat{y}y}}{(1 + e^{-\hat{y}y})e^{\hat{y}y}} \qquad (3) \\ &= \frac{-y}{1 + e^{\hat{y}y}} \\ &= -y\sigma(-\hat{y}y) \end{aligned}$$

Logistic regression

With the usual $L2$ regularization, the empirical risk writes :

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(x_i^T\theta, y_i) + \frac{\mu}{2}||\theta||^2 \tag{4}$$

We note $g_i(\theta) = l(x_i^T\theta, y_i) + \frac{\mu}{2}||\theta||^2$. Then

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} g_i(\theta) \tag{5}$$

SGD on sums of losses

With the usual $L2$ regularization, the empirical risk writes :

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(x_i^T\theta, y_i) + \frac{\mu}{2}||\theta||^2 \tag{6}$$

We note $g_i(\theta) = l(x_i^T\theta, y_i) + \frac{\mu}{2}||\theta||^2$. Then

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} g_i(\theta) \tag{7}$$

When performing SGD, we exploit the fact that $R_n(\theta)$ is a **sum**.

$$\nabla_\theta R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla g_i(\theta) \tag{8}$$

This is true for any setting where it is the case.

SGD for logistic regression

In this case,

$$\nabla_\theta g_i = -y_i \sigma(-x_i^T \theta y_i) x_i + \mu\theta \tag{9}$$

and

$$\nabla_\theta R_n = \frac{1}{n} \sum_{i=1}^{n} -y_i \sigma(-x_i^T \theta y_i) x_i + \mu\theta \tag{10}$$

GD and SGD

$R_n(\theta)$ is convex in θ.
We have seen that in for the optimization of a convex function f, two important properties carry information of the convergence garantees of GD and SGD :

- smoothness of f
- convexity or strong convexity of f

Smoothness

Definition

L-Lipschitz continuous gradients / smooth function

Let f be a differentiable function and $L > 0$. We say that f has L-Lipschitz continuous gradients or that f is L-smooth if $\forall x, y \in \mathbb{R}^d$,

$$||\nabla_x f - \nabla_y f|| \leq L||x - y||$$

Smoothness of the empirical risk

$$R_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} l(x_i^T\theta, y_i) + \frac{\mu}{2}||\theta||^2 \tag{11}$$

$$\nabla_\theta R_n = \frac{1}{n}\sum_{i=1}^{n} -y_i\sigma(-x_i^T\theta y_i)x_i + \mu\theta \tag{12}$$

Exercice 1 : Is $R_n(\theta)$ smooth ? at which condition ?

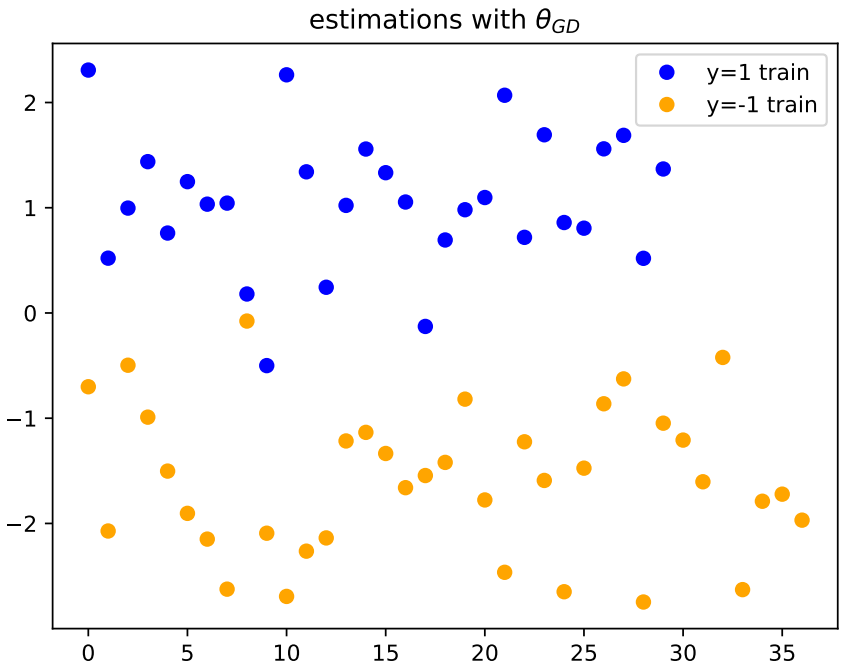
Strong convexity

Exercice 2 : Is $R_n(\theta)$ strongly convex ?

GD

Gradient descent :

$$\theta_{t+1} = \theta_t - \gamma \nabla_{\theta_t} f \tag{13}$$



Linear separation

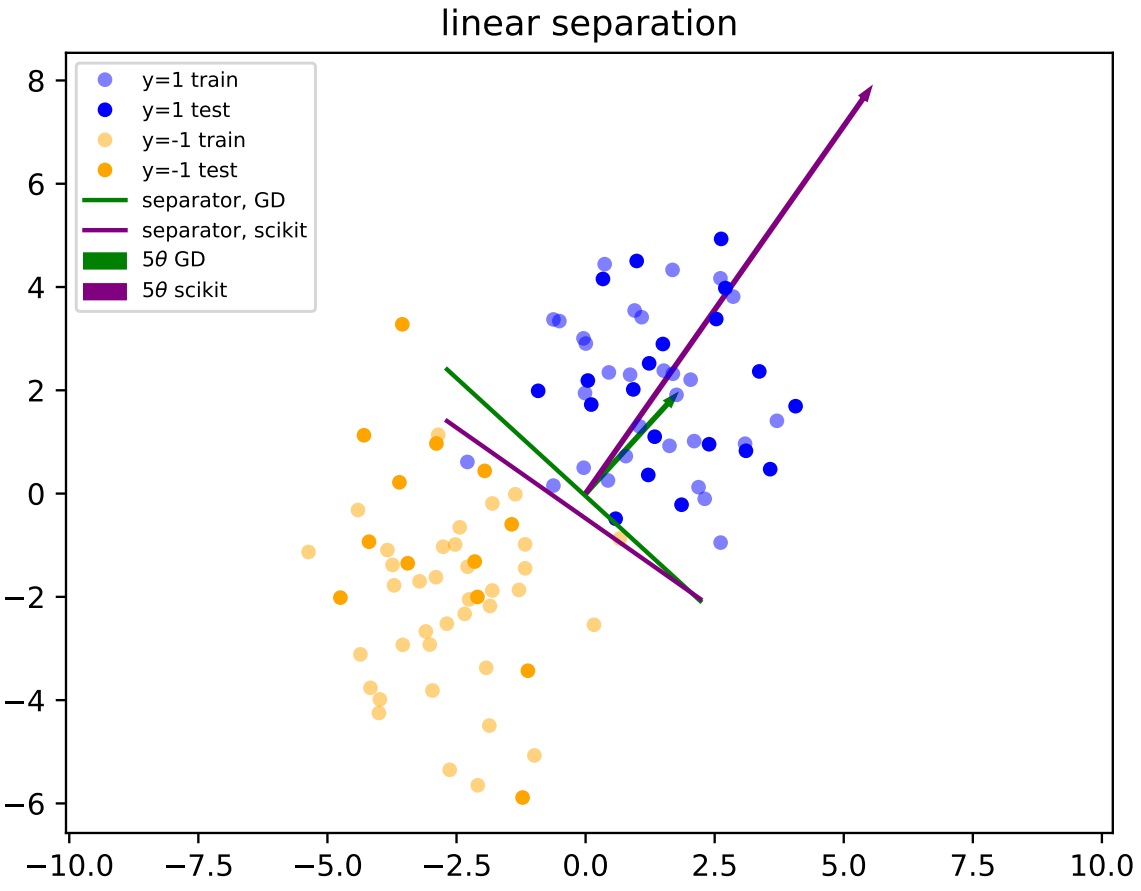


Figure – Linear separators returned by the algorithms. Note that if we

SGD

At each iteration t, sample $i(t)$ uniformly in $[1, n]$.

$$\begin{aligned} \theta_{t+1} &= \theta_t - \gamma \nabla_\theta g_i \\ &= \theta_t - \gamma(-y_i \sigma(-x_i^T \theta y_i) x_i + \mu\theta) \end{aligned} \tag{14}$$

Optimization time and computation time

- Optimization time : error as a funciton of t
- Computation time : error as a function of the computational complexity.
- One SGD update is n times less expensive than a GD update. If the optimization times are similar for both, the improvement can be significant.

Leaning rate schedules

- the learning rate schedule needs to be tuned carefully.
- the goal is to cancel the variance introduced by the randomness of SGD while keeping a good convergence speed.

Learning rate schedules

Example 1 :

$$\gamma_t = \frac{\gamma_0}{1 + \frac{t}{t_0}} \tag{15}$$

Example 2 :

$$\gamma_t = \frac{\gamma_0}{1 + \sqrt{\frac{t}{t_0}}} \tag{16}$$

Example 3 : constant.
Example 4 : piecewise constant.

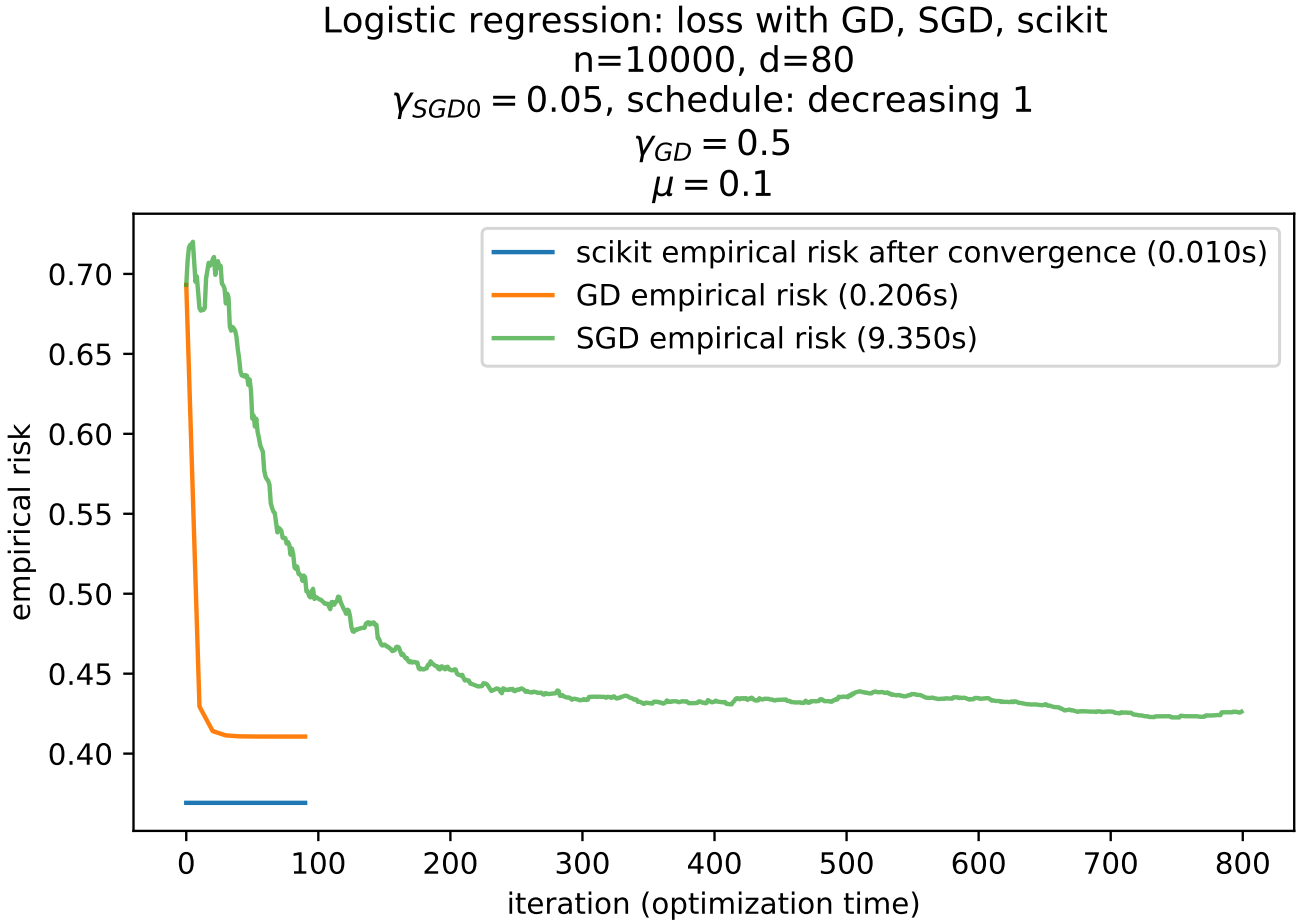
Conclusion

Many parameters are to be tuned :

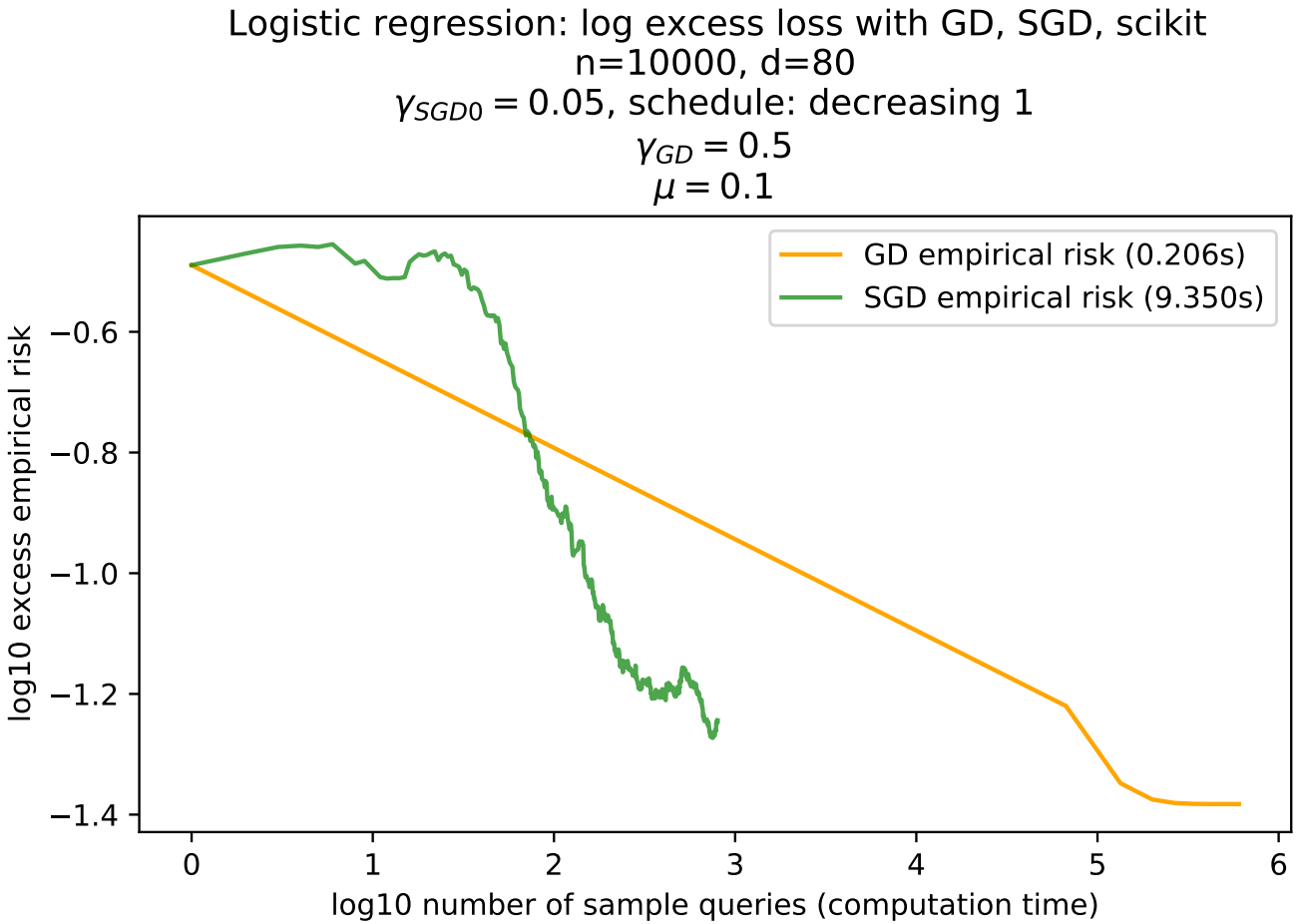
- choice of the regularization paraterer μ
- choice of γ_0, and t_0.
- choice of the learning rate schedule.

Randomness of SGD

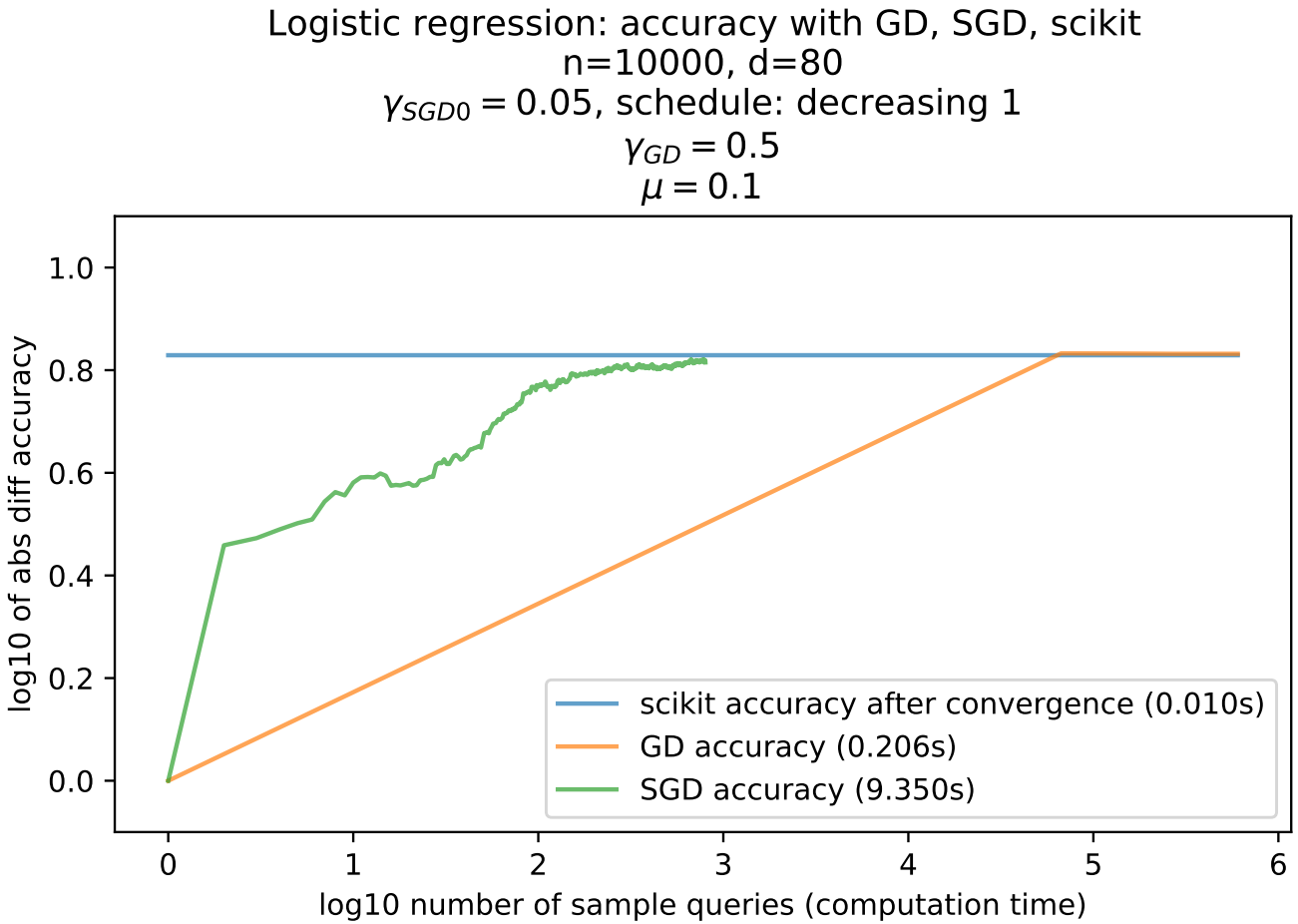
The loss can **increase** at a time step, leading some people to criticize the name Stochastic Gradient "Descent".



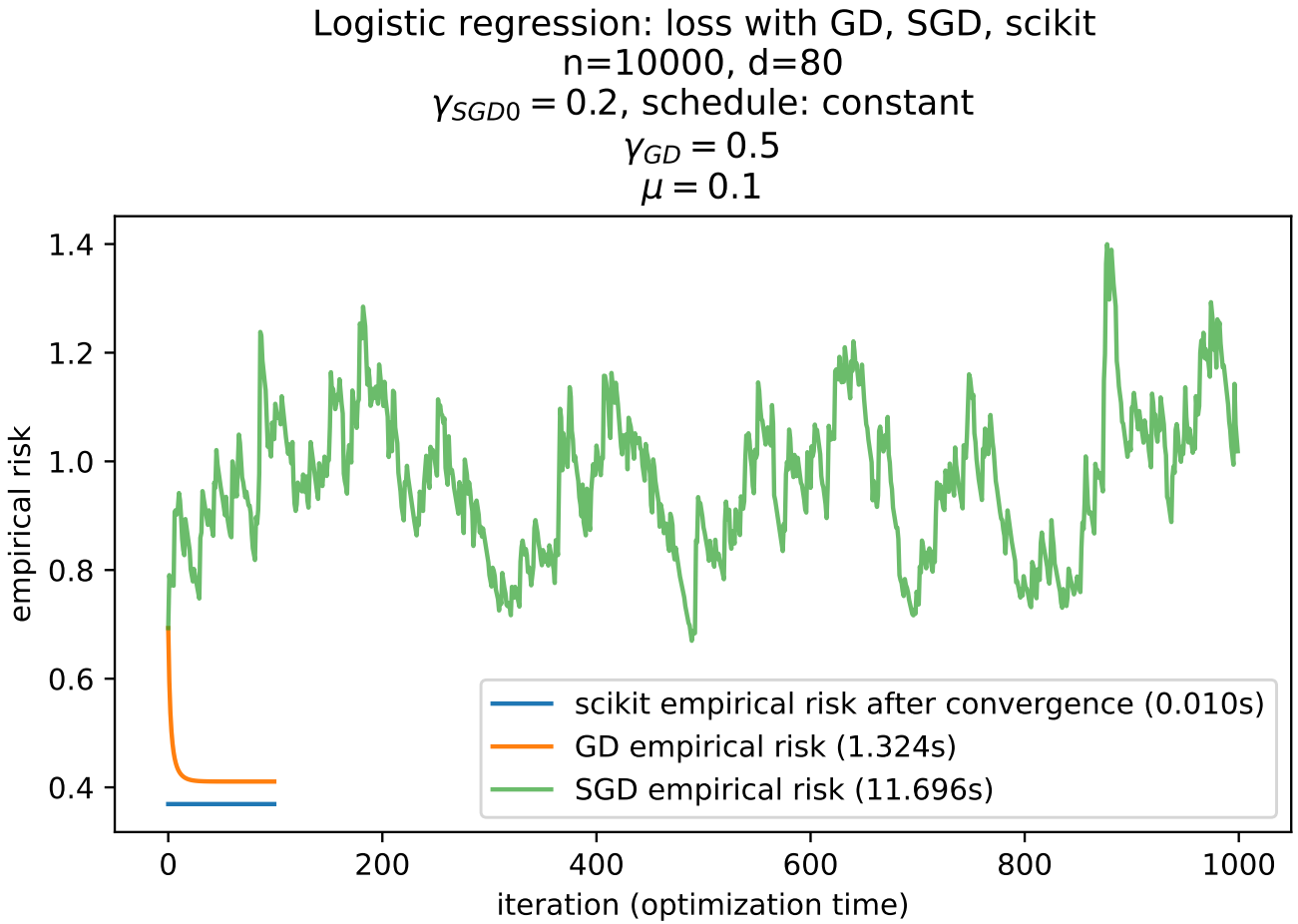
Speed of SGD and exponential convergence of GD



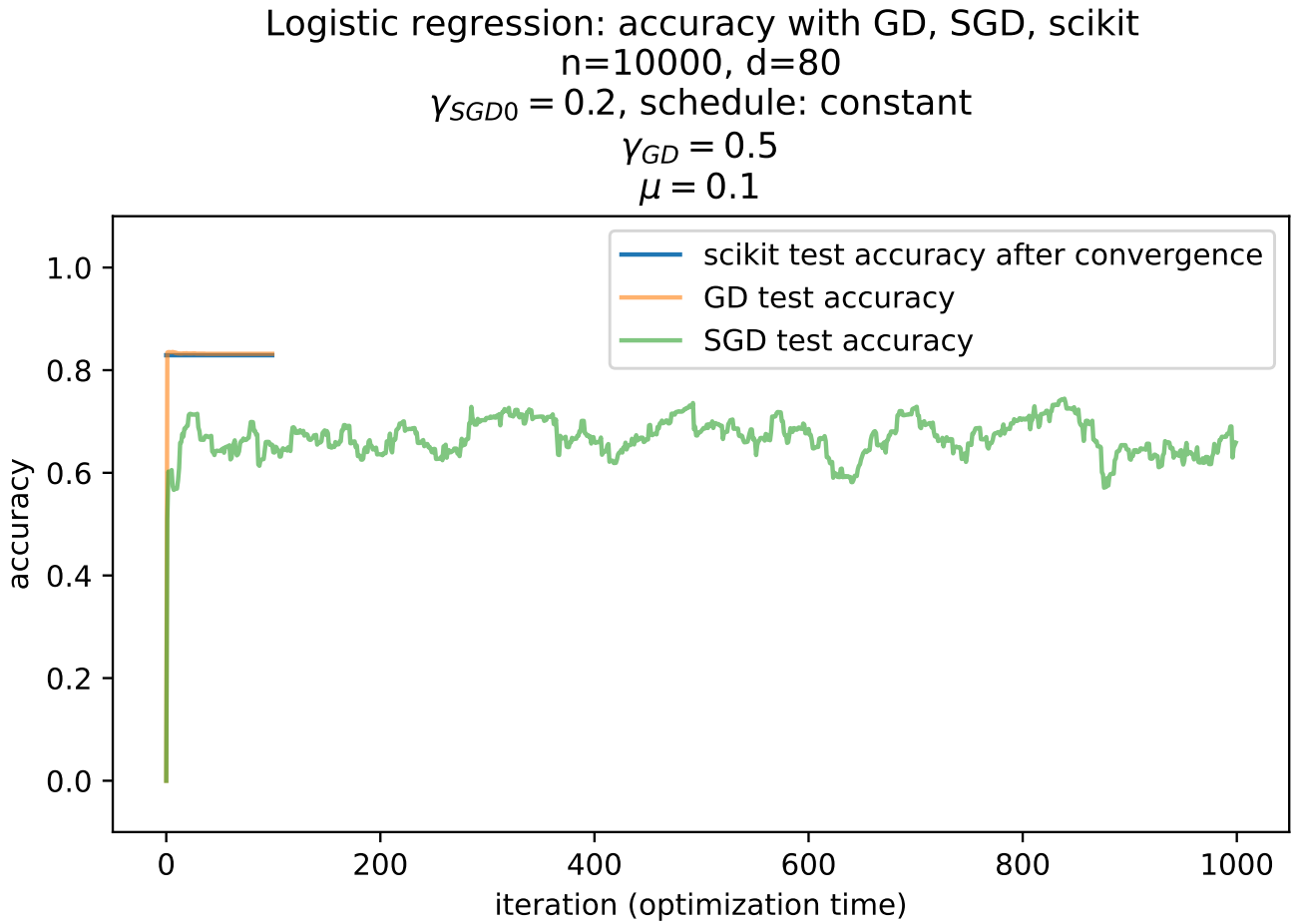
Accuracy


Logistic regression: accuracy with GD, SGD, scikit
n=10000, d=80
$\gamma_{SGD0} = 0.05$, schedule: decreasing 1
$\gamma_{GD} = 0.5$
$\mu = 0.1$
log10 of abs diff accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
log10 number of sample queries (computation time)
scikit accuracy after convergence (0.010s)
GD accuracy (0.206s)
SGD accuracy (9.350s)

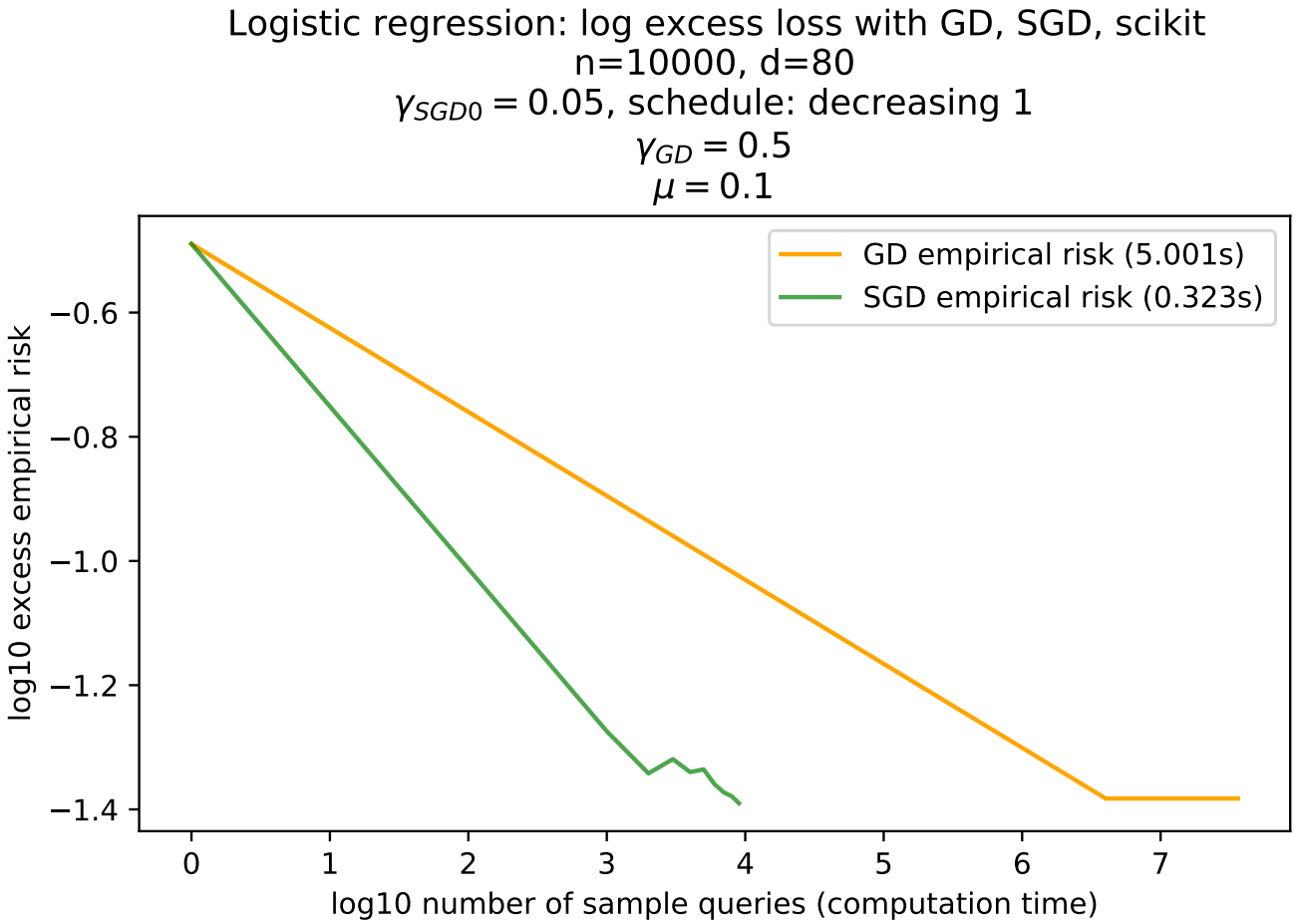
Non convergence of SGD



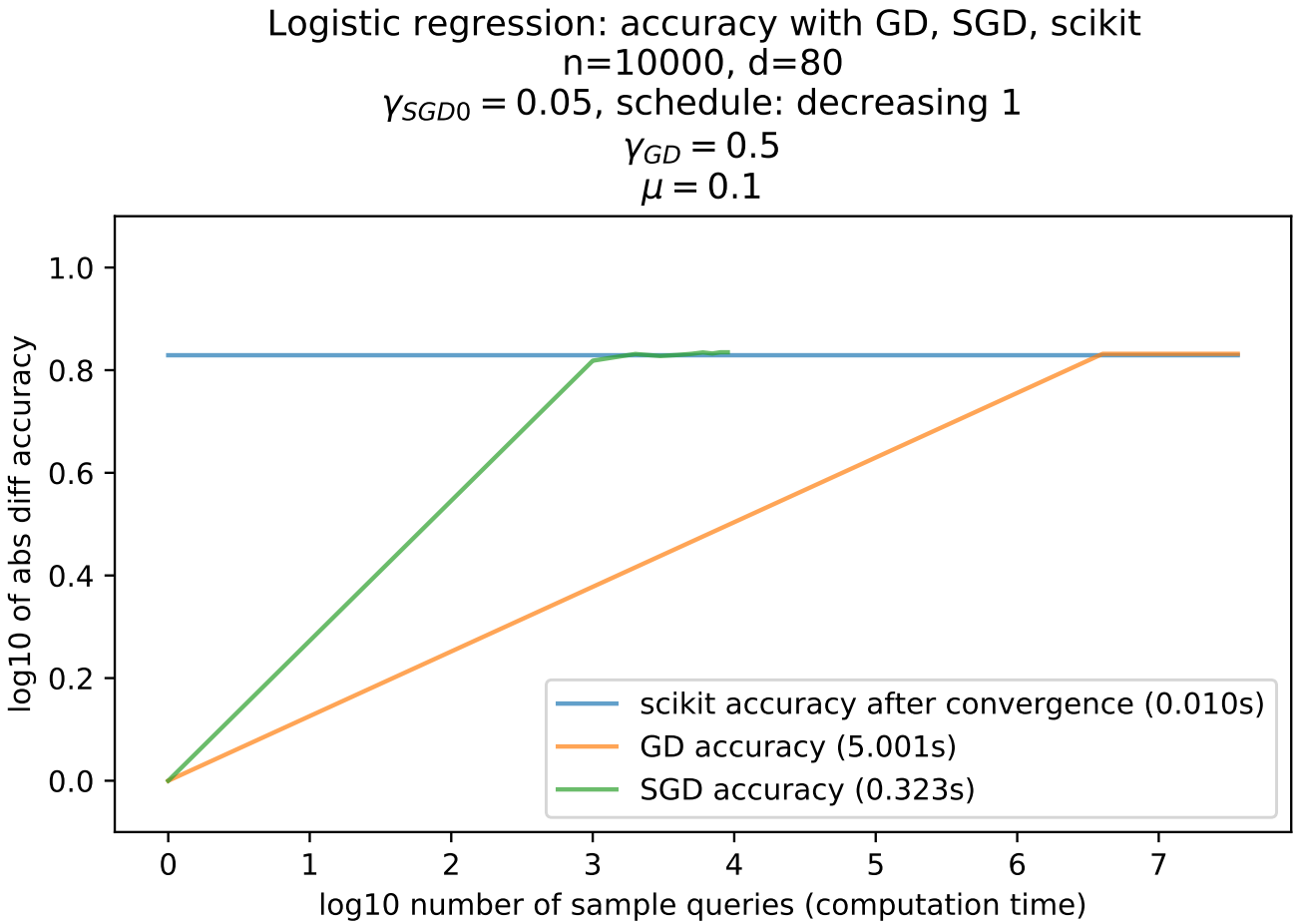
Non convergence of SGD



Speed of SGD


Logistic regression: log excess loss with GD, SGD, scikit
n=10000, d=80
$\gamma_{SGD0} = 0.05$, schedule: decreasing 1
$\gamma_{GD} = 0.5$
$\mu = 0.1$
GD empirical risk (5.001s)
SGD empirical risk (0.323s)
log10 excess empirical risk
−0.6
−0.8
−1.0
−1.2
−1.4
0
1
2
3
4
5
6
7
log10 number of sample queries (computation time)

Speed of SGD


Logistic regression: accuracy with GD, SGD, scikit
n=10000, d=80
$\gamma_{SGD0} = 0.05$, schedule: decreasing 1
$\gamma_{GD} = 0.5$
$\mu = 0.1$
log10 of abs diff accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
log10 number of sample queries (computation time)
scikit accuracy after convergence (0.010s)
GD accuracy (5.001s)
SGD accuracy (0.323s)

Averaging

In the most general setting, upper bounds on the convergence speeds are obtained for an average of the parameter θ at the end of optimization, such as

$$\hat{\theta} = \frac{1}{t} \sum_{i=1}^{t} \theta_k \tag{17}$$

Last-iterate

In some settings, we have convergence results **without averaging** [Varre et al., 2021]

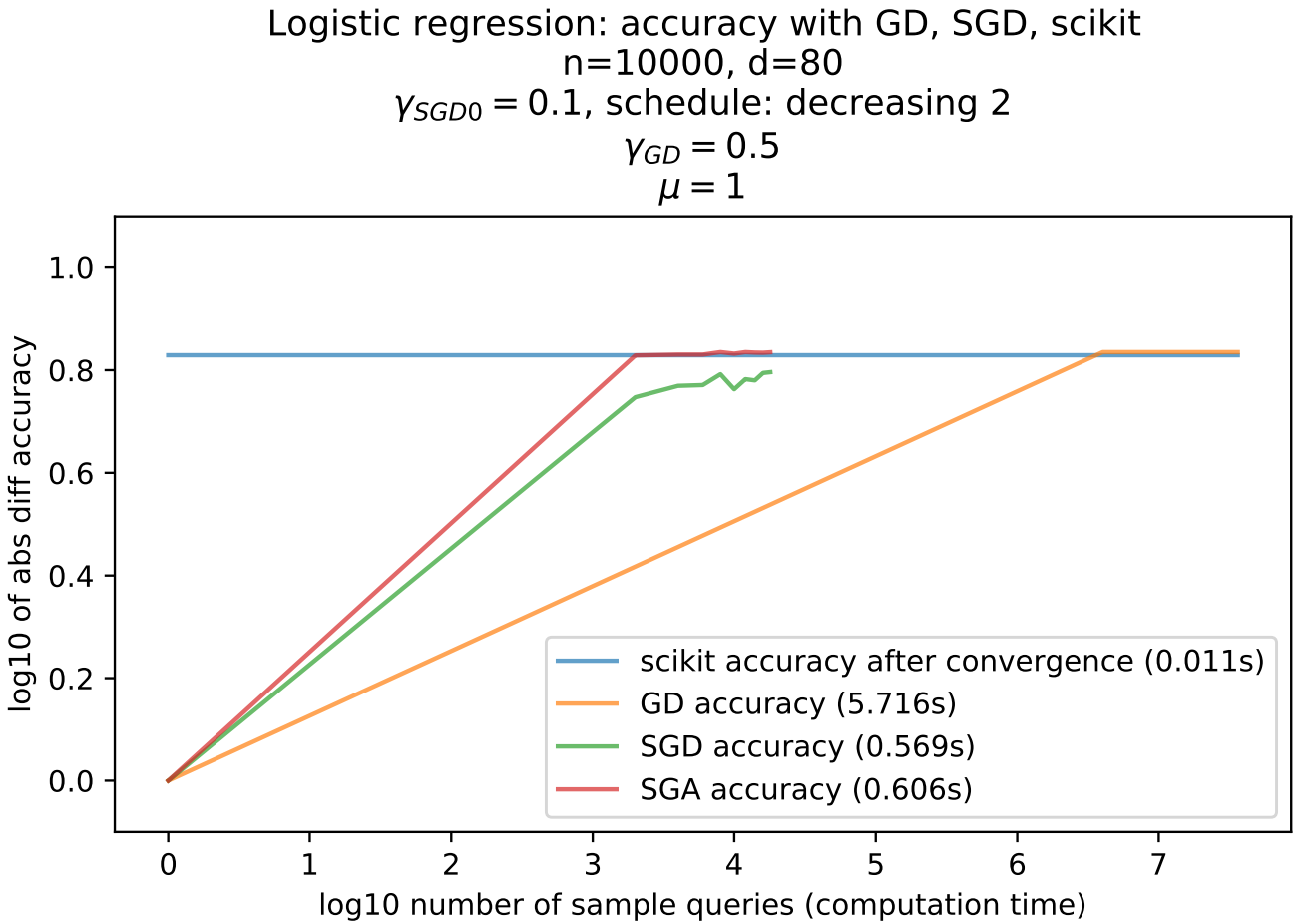
- loss function equal to 0 at optimum
- overparametrized setting
- interpolation regime
- well-specified model

Neural networks / Deep neural networks seem to belong to this setting for several problems (image recognition) and not fot others (Natural language processing).

Averaging strategy

- **Tail averaging** : only average on the latest half of the iterates
- more generally, **weighted averaging**.

Averaging

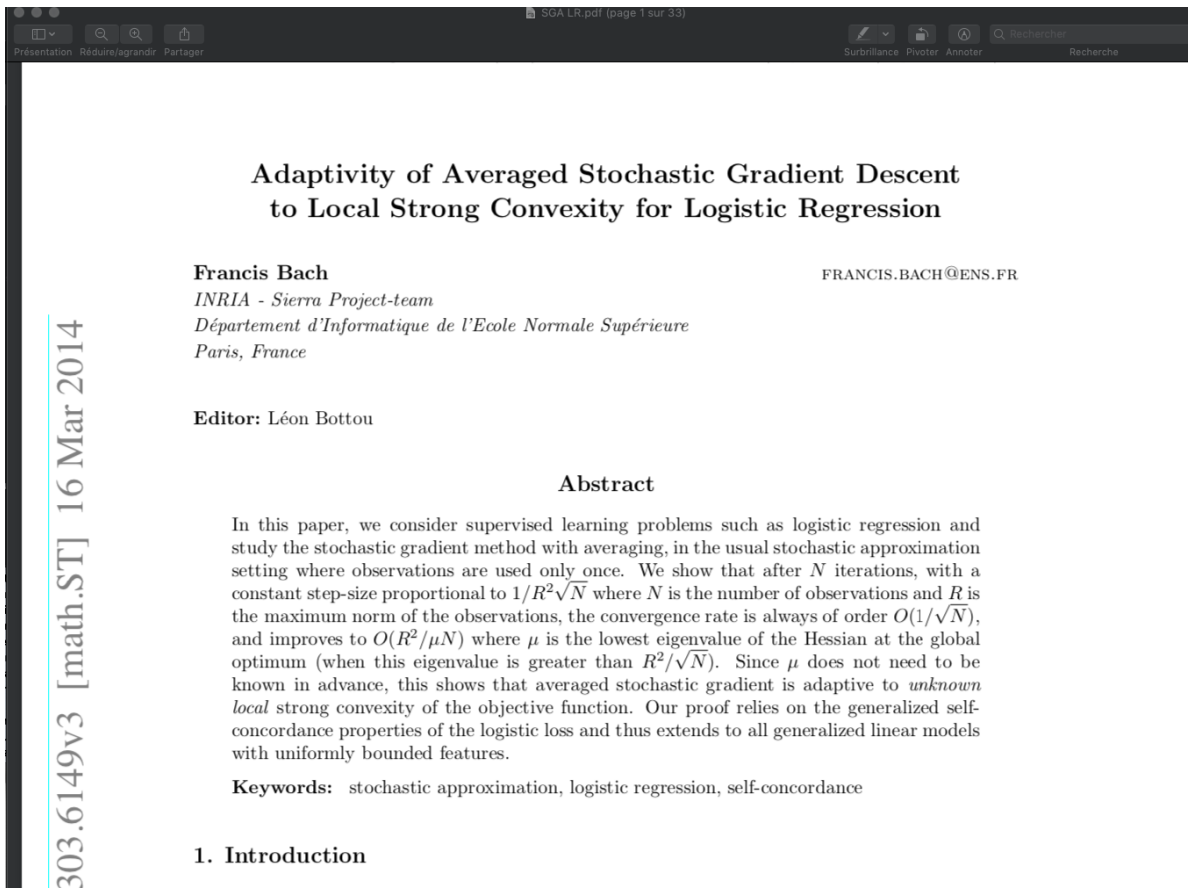

Logistic regression: accuracy with GD, SGD, scikit
n=10000, d=80
$\gamma_{SGD0} = 0.1$, schedule: decreasing 2
$\gamma_{GD} = 0.5$
$\mu = 1$
log10 of abs diff accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
log10 number of sample queries (computation time)
scikit accuracy after convergence (0.011s)
GD accuracy (5.716s)
SGD accuracy (0.569s)
SGA accuracy (0.606s)

Research

The tuning of SGD is still a researh topic.
The optimal tuning depends on the setting.

Averaging

In this article, Bach discusses the impact of averaging and the connection to strong convexity. [Bach, 2014]



SGA LR.pdf (page 1 sur 33)

Présentation Réduire/agrandir Partager Surbrillance Pivoter Annoter Rechercher Recherche

Adaptivity of Averaged Stochastic Gradient Descent to Local Strong Convexity for Logistic Regression

Francis Bach FRANCIS.BACH@ENS.FR
INRIA - Sierra Project-team
Département d'Informatique de l'Ecole Normale Supérieure
Paris, France

Editor: Léon Bottou

Abstract

In this paper, we consider supervised learning problems such as logistic regression and study the stochastic gradient method with averaging, in the usual stochastic approximation setting where observations are used only once. We show that after N iterations, with a constant step-size proportional to $1/R^2\sqrt{N}$ where N is the number of observations and R is the maximum norm of the observations, the convergence rate is always of order $O(1/\sqrt{N})$, and improves to $O(R^2/\mu N)$ where μ is the lowest eigenvalue of the Hessian at the global optimum (when this eigenvalue is greater than $R^2/\sqrt{N}$). Since μ does not need to be known in advance, this shows that averaged stochastic gradient is adaptive to *unknown local* strong convexity of the objective function. Our proof relies on the generalized self-concordance properties of the logistic loss and thus extends to all generalized linear models with uniformly bounded features.

Keywords: stochastic approximation, logistic regression, self-concordance

1. Introduction

SGD tuning

In this article, the authors discuss the combined effect of tail-averaging, using minibatches and multiple passes over the dataset. [Mücke et al., 2019]

Beating SGD Saturation with Tail-Averaging and Minibatching

Nicole Mücke* , Gergely Neu† and Lorenzo Rosasco‡

Abstract

While stochastic gradient descent (SGD) is a workhorse in machine learning, the learning properties of many practically used variants are hardly known. In this paper, we consider least squares learning and contribute filling this gap focusing on the effect and interplay of multiple passes, mini-batching and averaging, and in particular tail averaging. Our results show how these different flavors of SGD can be combined to achieve optimal learning errors, hence providing practical insights.

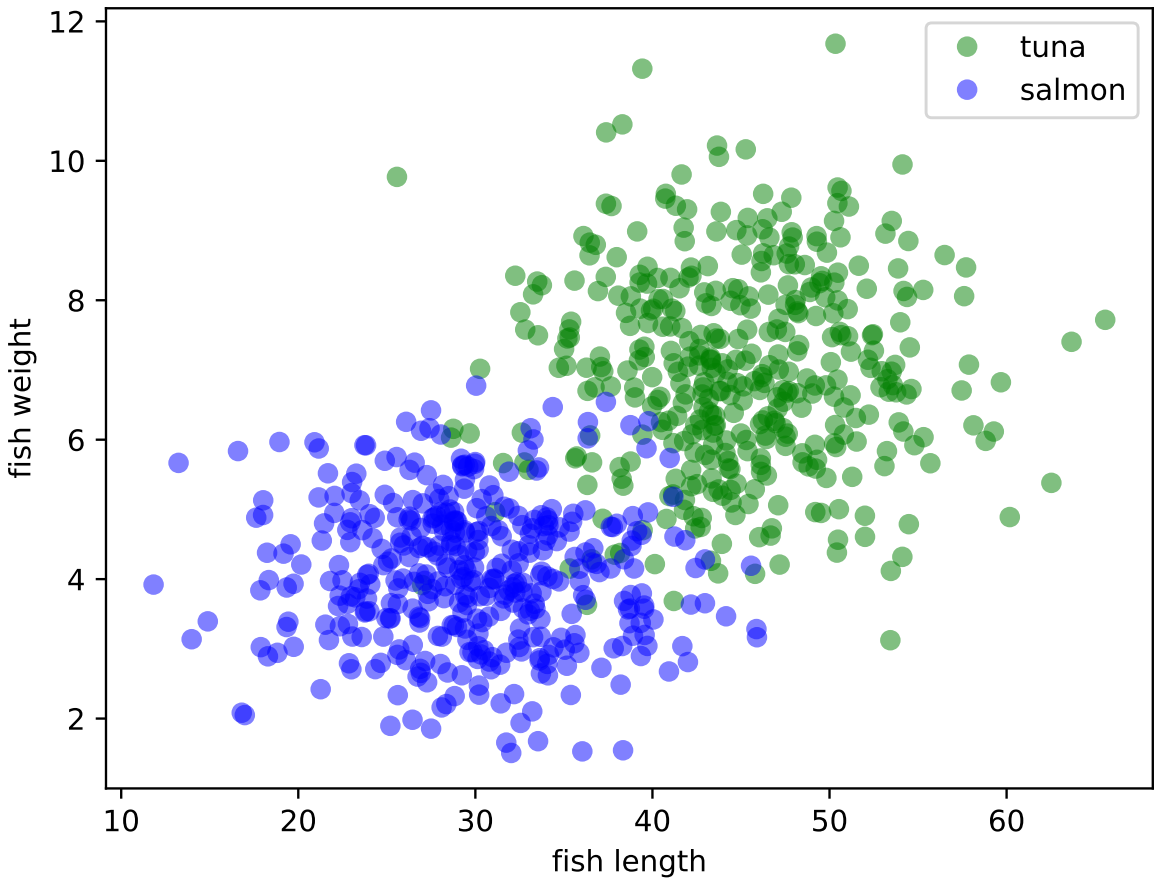
1 Introduction

Stochastic gradient descent (SGD) provides a simple and yet stunningly efficient way to solve a broad range of machine learning problems. Our starting observation is that, while a number of variants including multiple passes over the data, mini-batching and averaging are commonly used, their combination and learning properties are studied only partially. The literature on convergence properties of SGD is vast, but usually only one pass over the data is considered, see, e.g., [23]. In the context of nonparametric statistical learning, which we consider here, the study of one-pass SGD was probably first considered in [35] and then further developed in a number of papers (e.g., [25, 36, 37]). Another line of work derives statistical learning results for one pass SGD with averaging from a worst-case sequential prediction analysis [18, 28, 29]. The idea of using averaging also has a long history going back to at least the works of [32] and [27], see also [34] and references therein. More recently, averaging was shown to lead to larger, possibly constant, step-sizes, see [2, 10, 11]. A different take on the role of (weighted) averaging was given in [24], highlighting a connection with ridge regression, a.k.a. Tikhonov regularization. A different flavor of averaging called *tail averaging* for one-pass SGD was considered in [19]

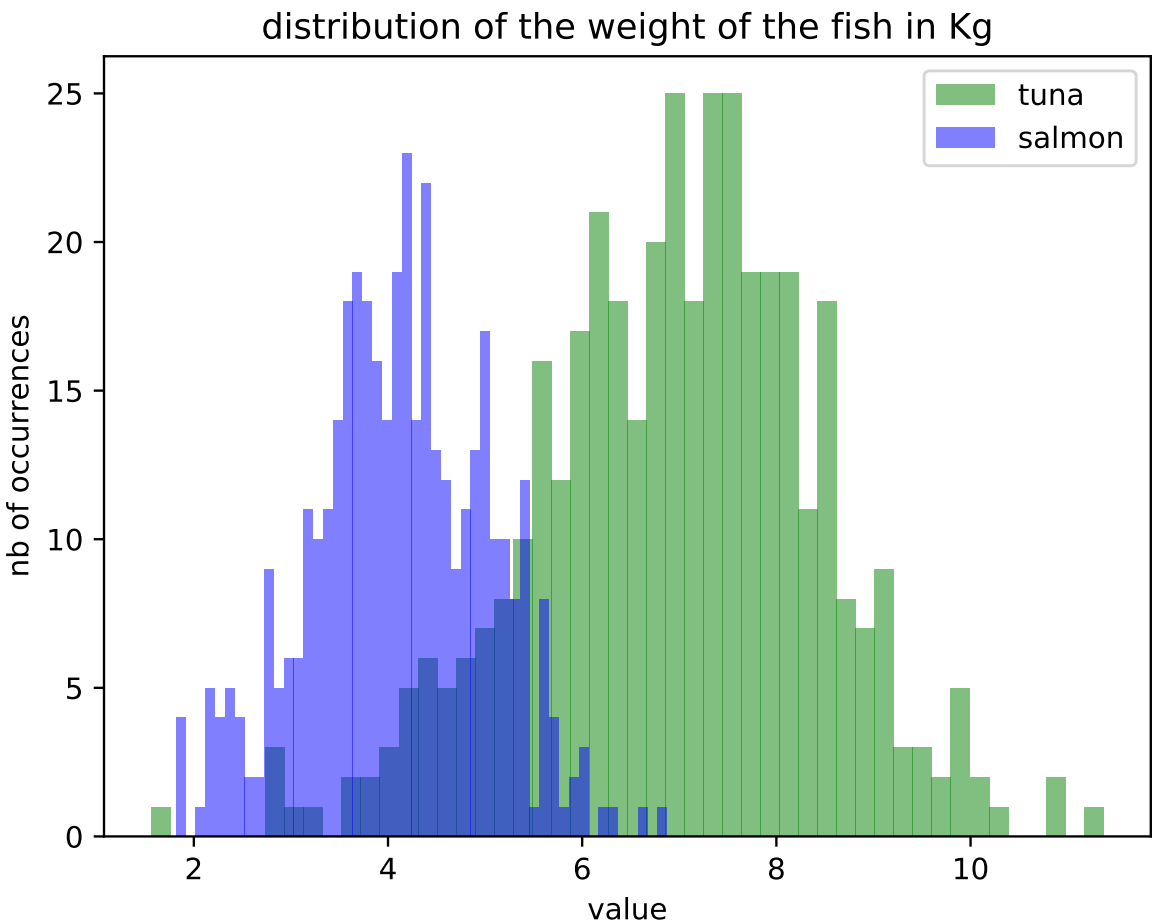
CART

- Segmentation method (partitionnement récursif), **binary splits.**
- First algorithm : **CART** (classification and regression tree, Breiman, 1984) [Breiman et al., 2017]

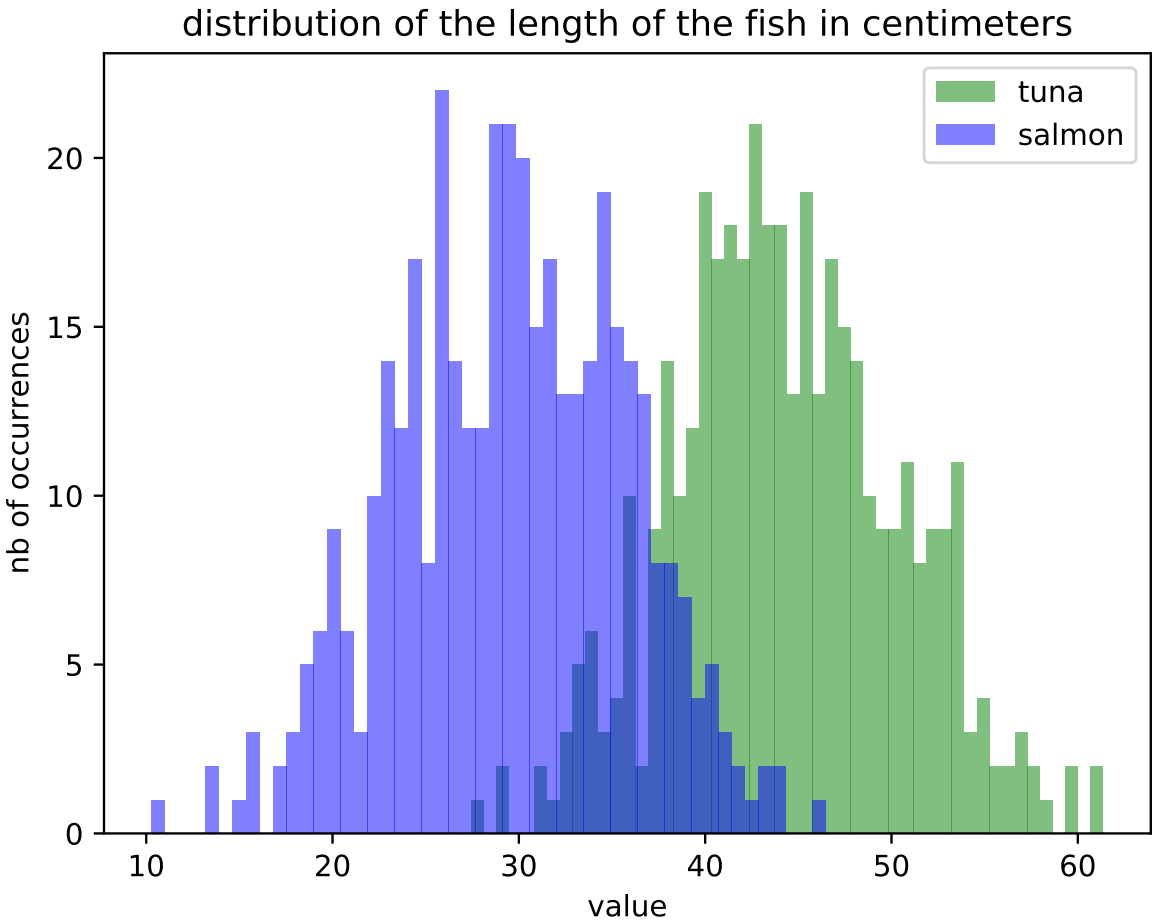
Example dataset : fishes



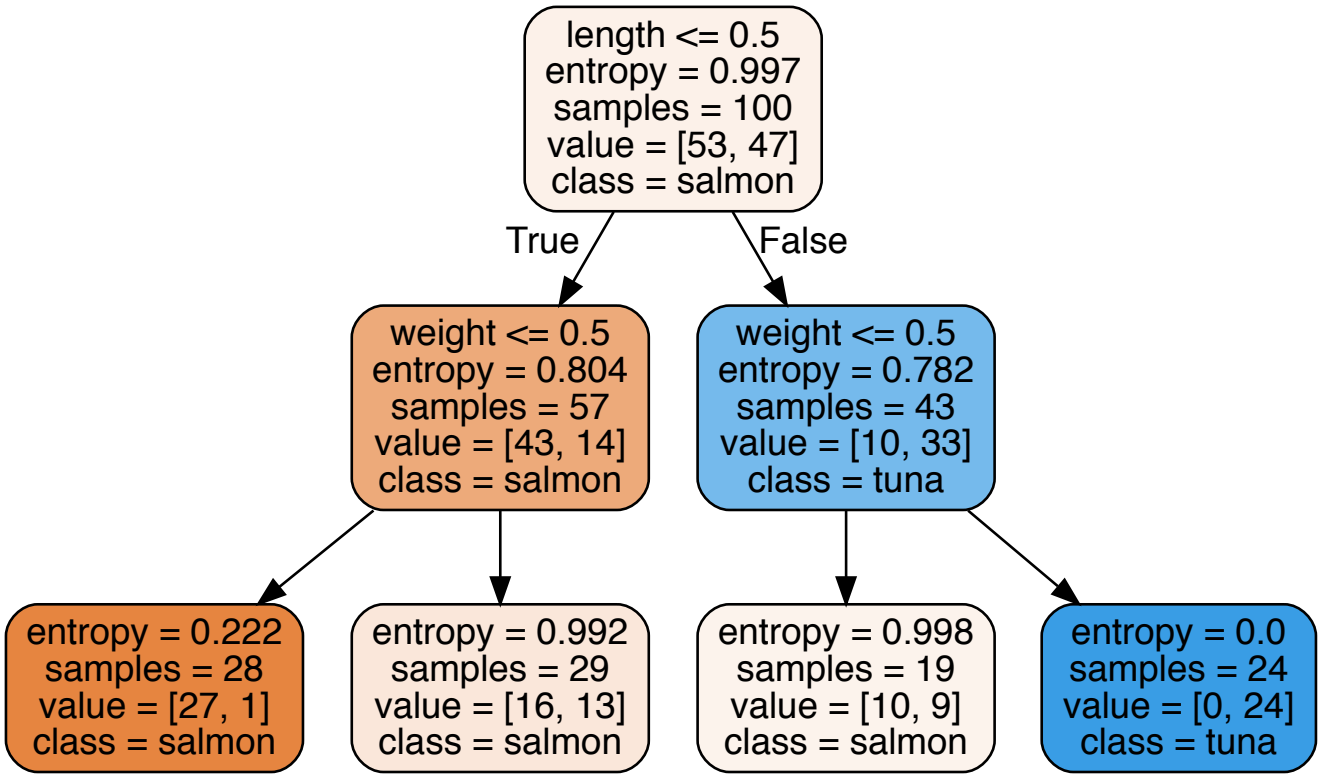
Fish dataset



Fish dataset

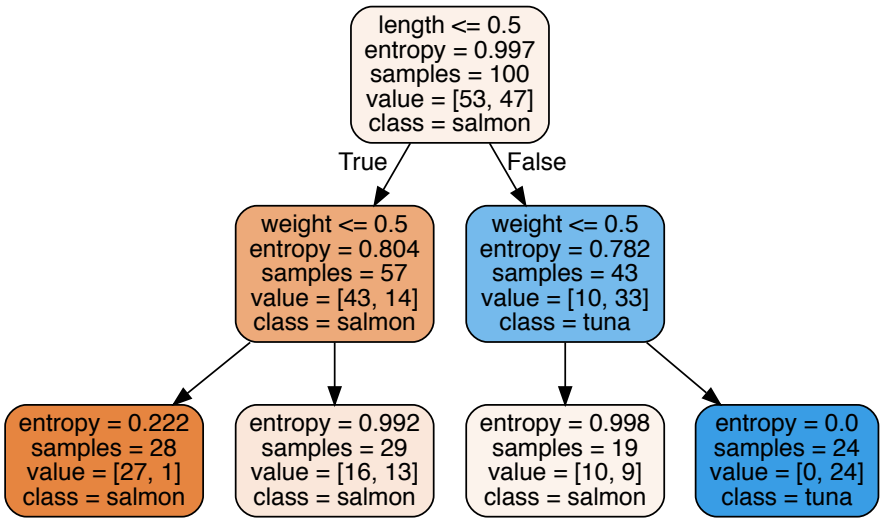


Example tree



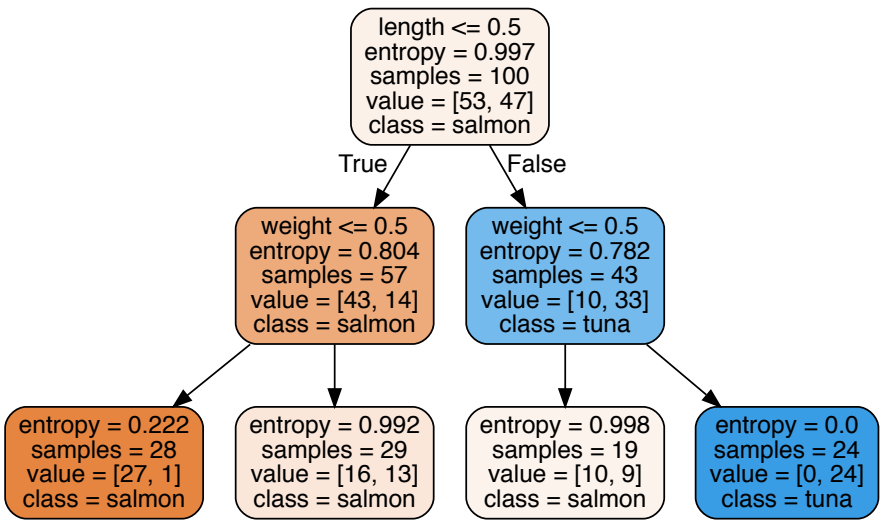
Splitting nodes

Each split should lead to more **homogeneous** nodes (in a sense that is to be formally defined)

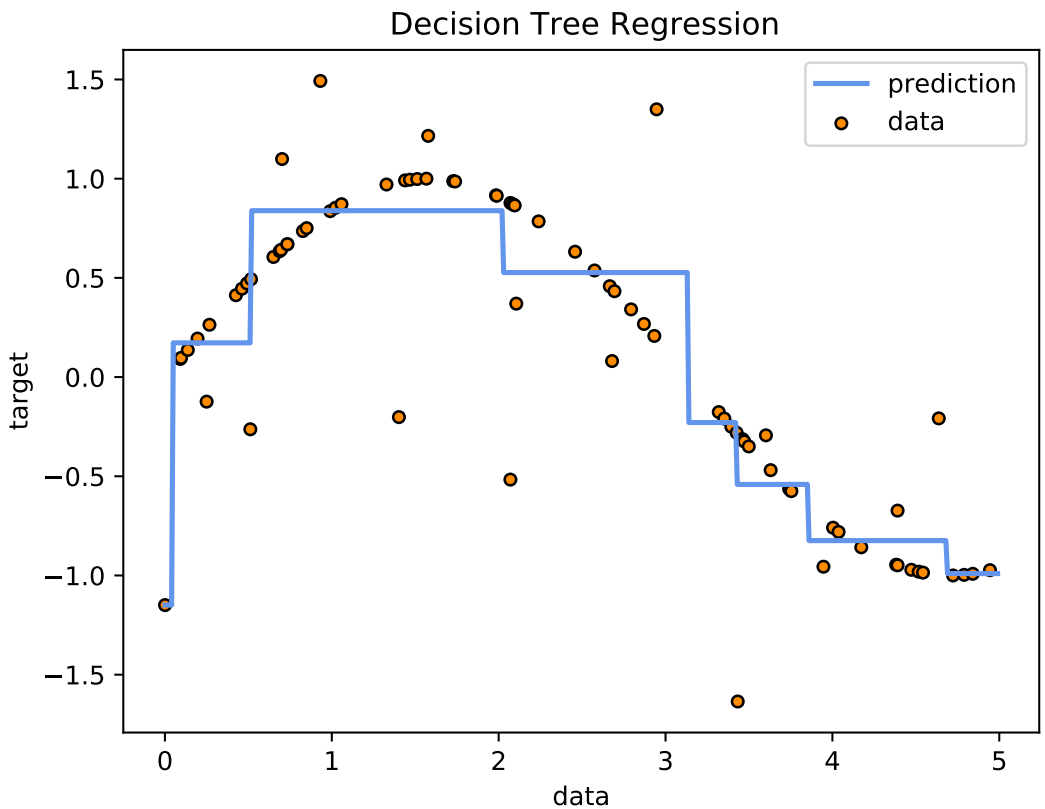


Splitting nodes

Leaf nodes correspond to a predicted value (class or real number for instance if $\mathcal{Y} = \mathbb{R}$)).



Decision trees can also be used for regression. In that case, the prediction is **piecewise constant**. It can be seen as a form of local averaging. Depending on the regularity of the value to predict, this implies that a decision tree might not be relevant.



Construction of a tree

A split node corresponds to :

- a segmentation variable V
- a segmentation value / threshold if V is quantitative. If V is a class, it is a two-fold partitionning of the set of classes.

The construction of a tree requires a critetion to :

- choose the segmentation variable and value
- decide when a node is terminal (leaf node)
- associate a prediction value to all leaf nodes

Homogeneity

We want the homogeneity in the child nodes to be greater than in the parent node. Equivalently, we can define a non-negative heterogeneity function H that describes each node and that must be :

- 0 if the node is homogeneous (only one class or one output value is represented in this node)
- maximal if the values of y are uniformly distirbuted within the node.
- if $H(L)$ it the value og H for the left child node (and $H(R)$ for the right one), then the segmentation should minimize

$$H(L) + H(R) \tag{18}$$

Stopping criterion

In order to prevent the tree from being too deep, a heuristic criterion is used to stop the segmentations, for instance it is not possible to divide a node if a number of samples smaller than 5.

Homogeneity criterion for regression

In the case of regression, $\mathcal{Y} = \mathbb{R}$ and the heterogeneity $H(n)$ of node n is its empirical variance :

$$H(n) = \frac{1}{|n|} \sum_{i \in n} (y_i - \bar{y}_n)^2 \tag{19}$$

Homogeneity criterion for classification

In the case of classification, $\mathcal{Y} = [1, \ldots, m]$ and several criteria exist.

Homogeneity criterion for classification : entropy (information gain)

- We use the convention that $0 \log(0) = 0$
- p_n^l is the proportion of class l in node n.

The **entropy** writes :

$$H(n) = -\sum_{l=1}^{n} p_n^l \log(p_n^l) \tag{20}$$

Exercice 3 : What are the maximum and minimum value of the entropy ?

Homogeneity criterion for classification : Gini impurity

The Gini impurity writes :

$$H(n) = \sum_{l=1}^{n} p_n^l (1 - p_n^l) \tag{21}$$

Homogeneity criterion for classification : Gini impurity

The Gini impurity writes :

$$H(n) = \sum_{l=1}^{n} p_n^l (1 - p_n^l) \tag{22}$$

Exercice 4 : Interpret the meaning of the Gini impurity in terms of probabilities.

- Y_{pred} is the random variable representing this prediction (proportional)
- Y is the random variable representing the class, in this node (empirical distribution)

$$
\begin{aligned}
P(Y_{pred} \neq Y) &= \sum_{l=1}^{m} P\big(Y_{pred} \neq Y | Y = l\big) P(Y = l) \\
&= \sum_{l=1}^{m} \Big(1 - P\big(Y_{pred} = Y | Y = l\big)\Big) P(Y = l) \quad (23) \\
&= \sum_{l=1}^{m} (1 - p_n^l) p_n^l
\end{aligned}
$$

Homogeneity criterion for classification : Gini impurity

$$H(n) = \sum_{l=1}^{n} p_n^l (1 - p_n^l) \tag{24}$$

If we predict the classes in node n according to the proportions of the labels in n, then the Gini impurity is the probability of making a mistake, given that we are in node n.

Homogeneity criterion for classification : misclassification probability

$$H(n) = 1 - \max_{l}(p_n^l) \tag{25}$$

If we predict the most represented class in n, then this is the misclassification probability.

Prediction for a leaf node

- **regression** : predict the average value in the node
- **classification** : several possibilities
  - most represented class in the node
  - most probable class *a posteriori* if the *a priori* probabilities are known (Bayesian probabilities)
  - class that costs the less in case of missclassification

Pruning

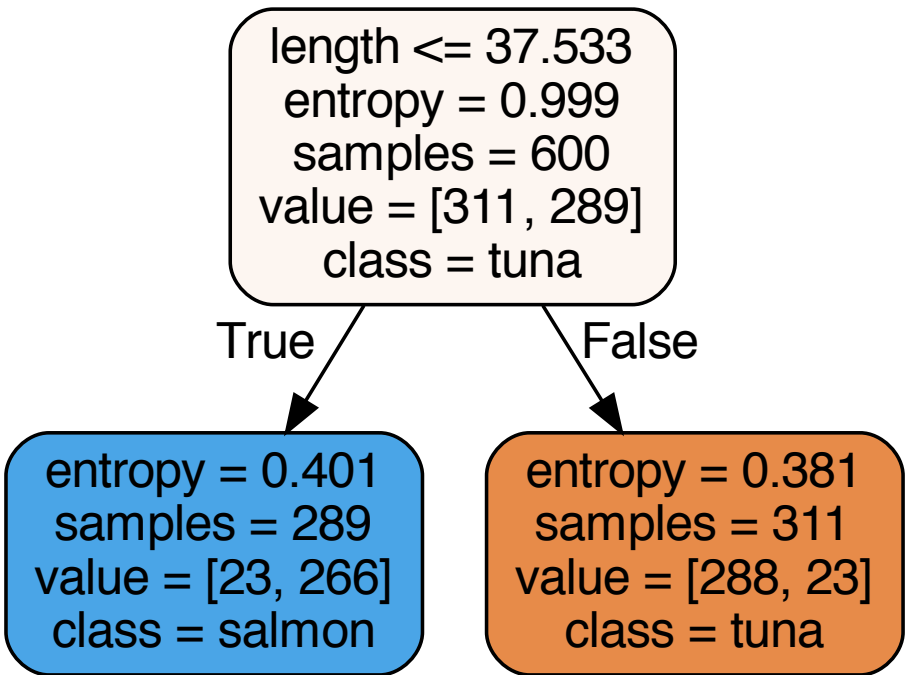
- If the segmentation is not restricted, the tree can fit the training dataset perfectly (overfitting).
- learning would then be highly dependent on the choice of the training dataset, leading to **instability.**
- To avoid it, **pruning** (élaguage) is used : it consists in restrincting the size of the tree (and can be seen as an equivalent to regularization when we perform ridge regression of logistic regression).

Pre-pruning by restricting divisions

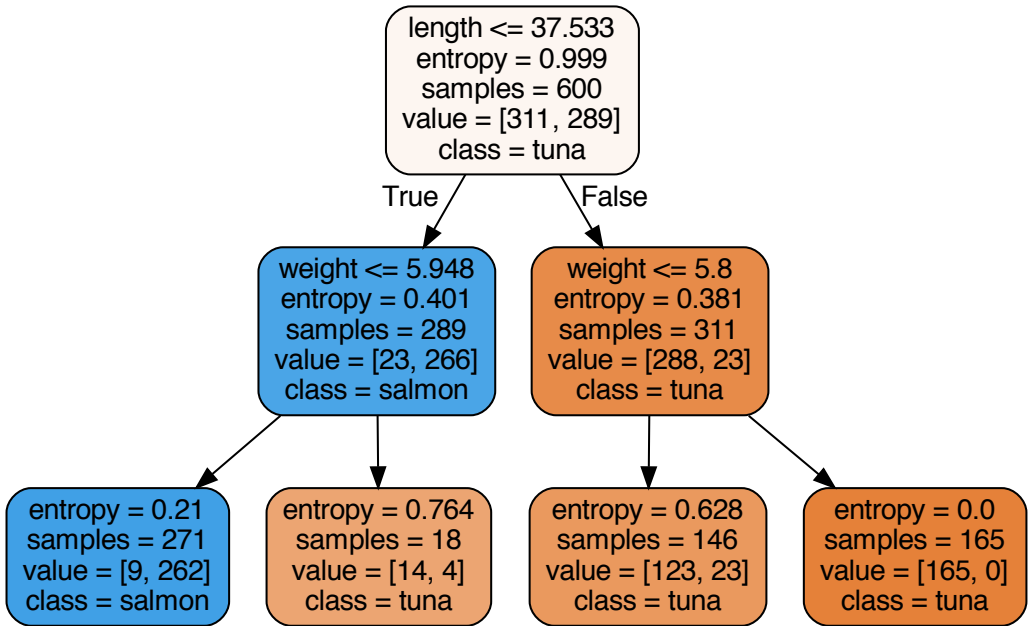
We can impose

- minium impurity decrease
- a minimum number of samples in a leaf node
- a minimum number of samples to authorize splitting a node

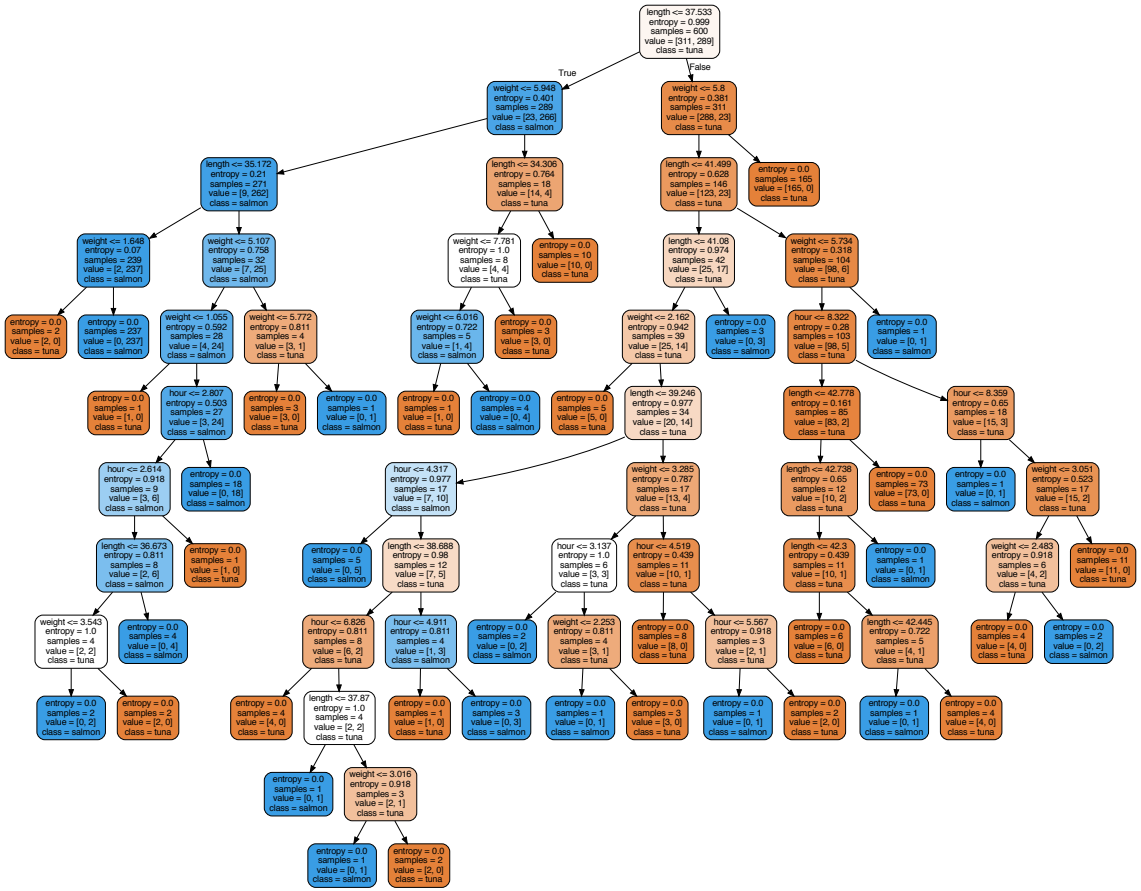
Simulation : fish dataset, max depth=1



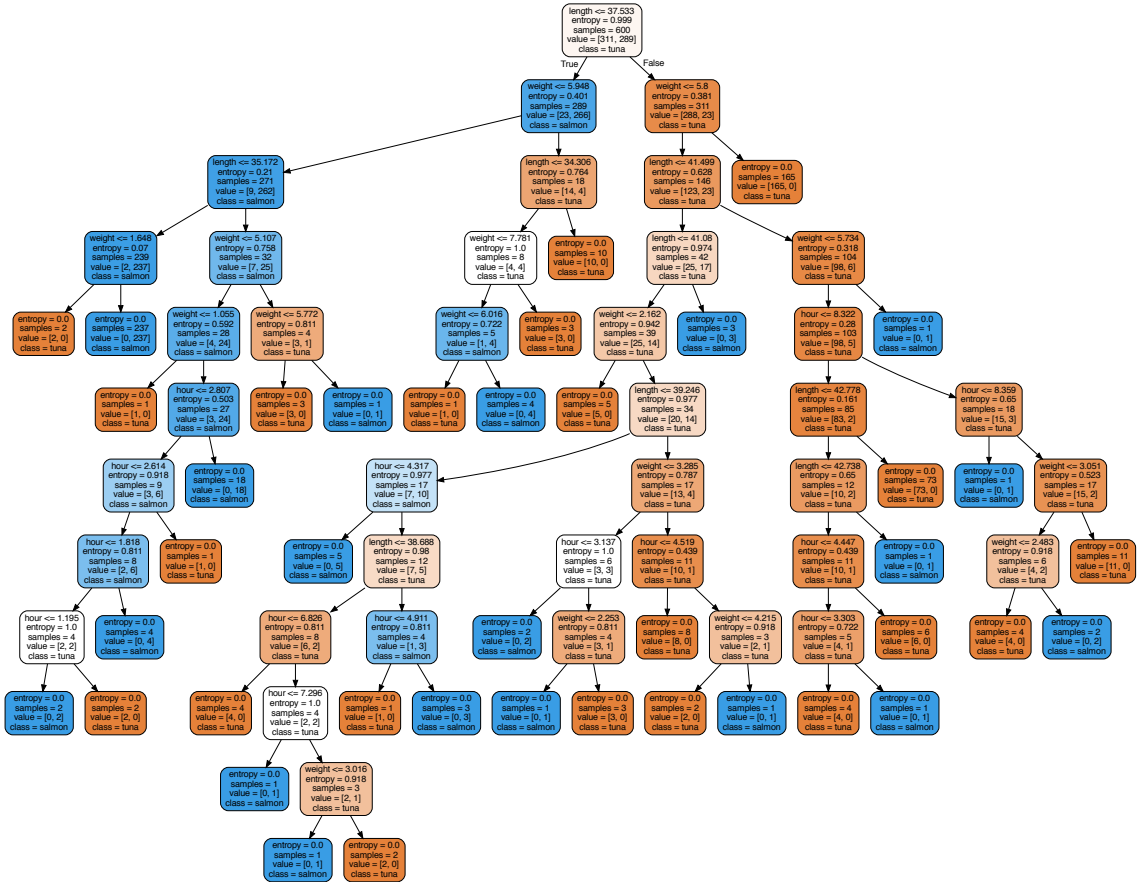
Simulation : fish dataset



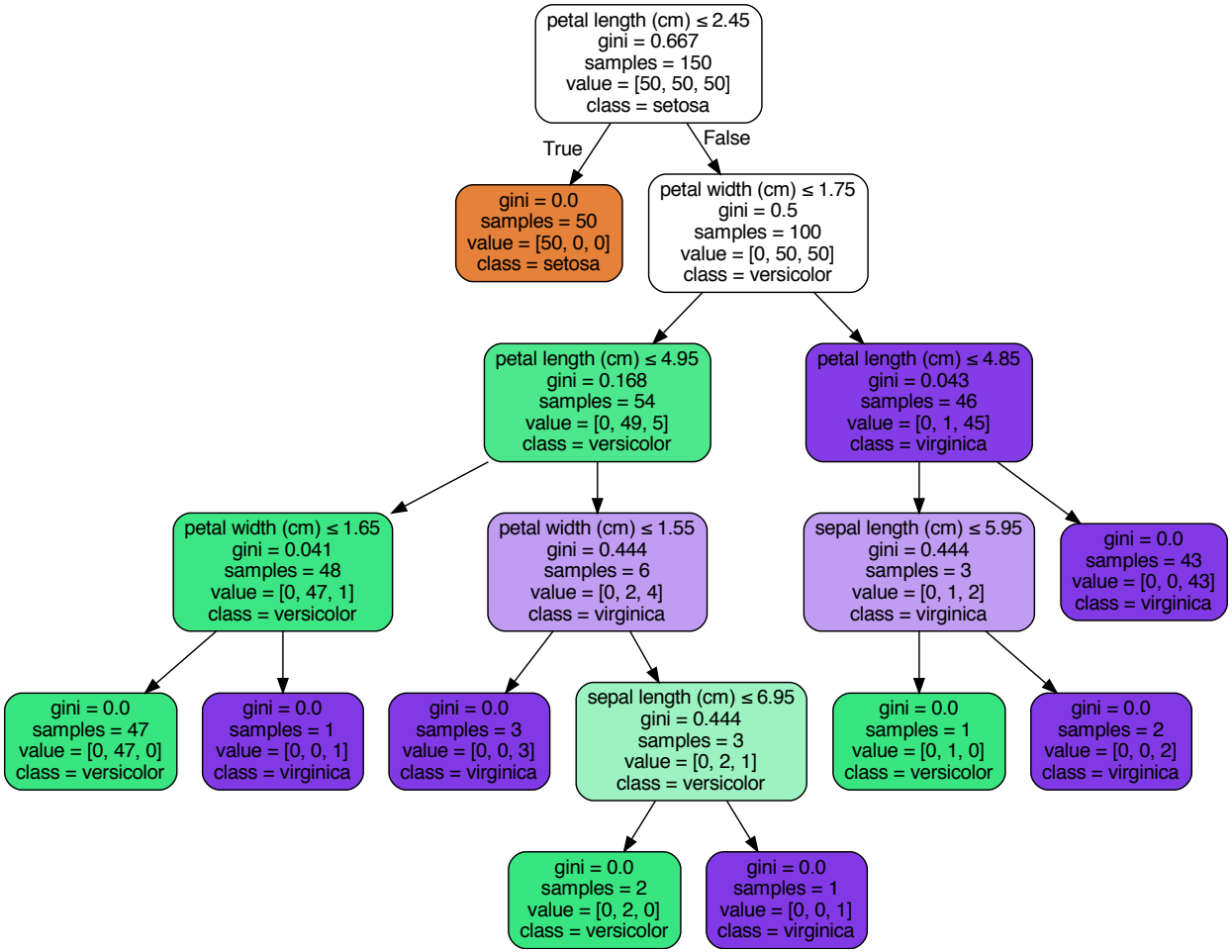
Simulation : fish dataset



Simulation : fish dataset



Overfitting : Iris dataset, max depth=34



Avoiding overfitting : Iris dataset

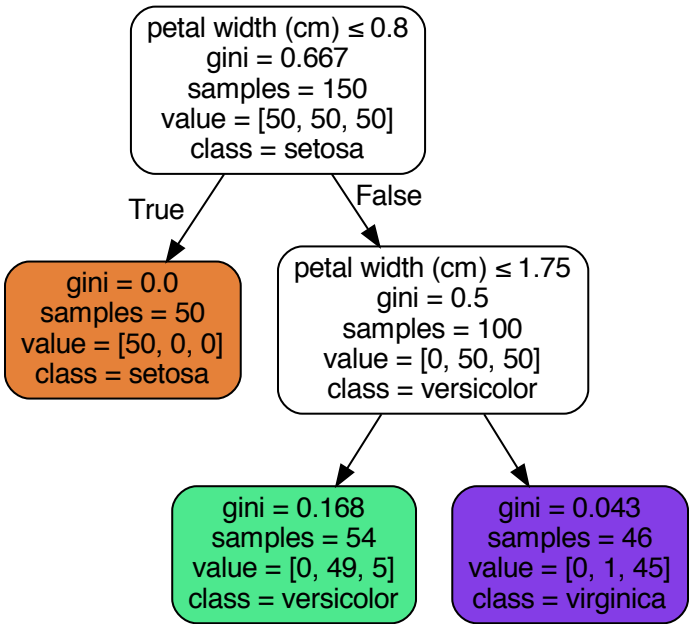


Figure – Avoid overfitting with pre-pruning. In this case, impose a minimum impurity decrease of 0.1

Post-pruning

It is also possible to prune after the construction of a large tree.

Remarks on decision trees I

- Decision trees do not require hypotheses on the distribution of de features.
- Feature selection is integrated in the algorithm, so they might be relevant in a situation with many features (variables explicatives)
- A segmentation is invariant to a monotonic change in a feature. It is thus robust to outliers or to very asymetrical distributions.
- Decision trees allow a straigtforward **interpretation** of the model.

Remarks on decision trees II

- Decision trees are geedy : they might find a local minimum
- they are hierarchical : a bad choice in a segmentation at the top of the tree propagates in the whole tree
- Hence their instability and sensibility to the choice of the training set.

Ensemble learning

- *Bagging* and *boosting* are methods that combine estimators (in parallel or sequentially)
- they are often applied to decision trees
- they reach state-of-the-art performance in several supervised learning tasks [Fernández-Delgado et al., 2014]
- they are an active area of research (both theoretically and for practical applications).

`https://scikit-learn.org/stable/modules/ensemble.html`

Aggregating

- usual setup : input space $\mathcal{X}$, output space $\mathcal{Y}$.
- we note z_b a dataset sampled from the unkwow distribution ρ. if we sample b different dataset, $b \in [1, B]$,

$$z_b = \{(x_{1b}, y_{1b}), \dots (x_{nb}, y_{nb})\} \tag{26}$$

- we note $\hat{f}_{z_b}$ the estimator obtained after learning with z_b.

Aggregating

- we note z_b a dataset sampled from the unkwow distribution ρ. if we sample b different dataset, $b \in [1, B]$,

$$z_b = \{(x_{1b}, y_{1b}), \dots (x_{nb}, y_{nb})\} \tag{27}$$

- we note $\hat{f_{z_b}}$ the estimator obtained after learning with z_b.

Aggregating consists in using as an estimator :

- for regression

$$\hat{f_B} = \frac{1}{B} \sum_{b=1}^{B} \hat{f_{z_b}} \tag{28}$$

- for classification

$$\hat{f_B}(x) = \arg\max_j |\{b, \hat{f_{z_b}}(x) = j\}| \tag{29}$$

Aggregating

We note $\hat{f}_{z_b}$ the estimator obtained after learning with z_b.
Aggregating consists in using as an estimator :

- for regression

$$\hat{f}_B = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_{z_b} \tag{30}$$

- for classification (voting)

$$\hat{f}_B(x) = \arg\max_j |\{b, \hat{f}_{z_b}(x) = j\}| \tag{31}$$

The goal is to reduce the variance of the estimator.

Bootstrapping

If B is large, it is not possible to sample B independent datasets with n samples, from a finite dataset.
Bootstrapping consists in sampling B times a sample dataset with n elements **with replacement**.

Bagging

Bagging is the combination of boostrapping and aggregating.

Out-of-bag error

For an observation (x_i, y_i), the **out-of-bag error** is the mean error on this observation among the estimators that were trained on a bootstrap dataset **not** containing it.

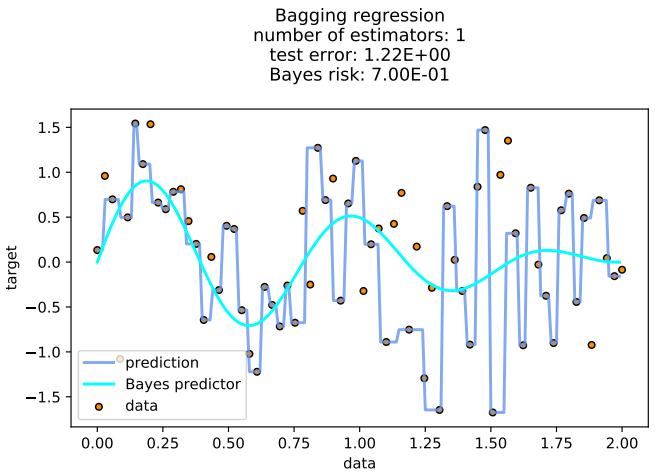
Trees and bagging

Bagging is most often used with decision trees. Several strategies are possible in order to build the base estimators $\hat{f}_{z_b}$ and to prune them.

- pre-pruning by enforcing a minimal number of samples per leaf node (e.g. 5) : good tradeoff between computation speed and quality of the prediction. The goal is to reduce high variance of the base estimators by averaging.
- pre-pruning by enforcing a maximal tree depth or maximal number of leaves.
- post-pruning with cross validation (more computationally expensive with less gains).

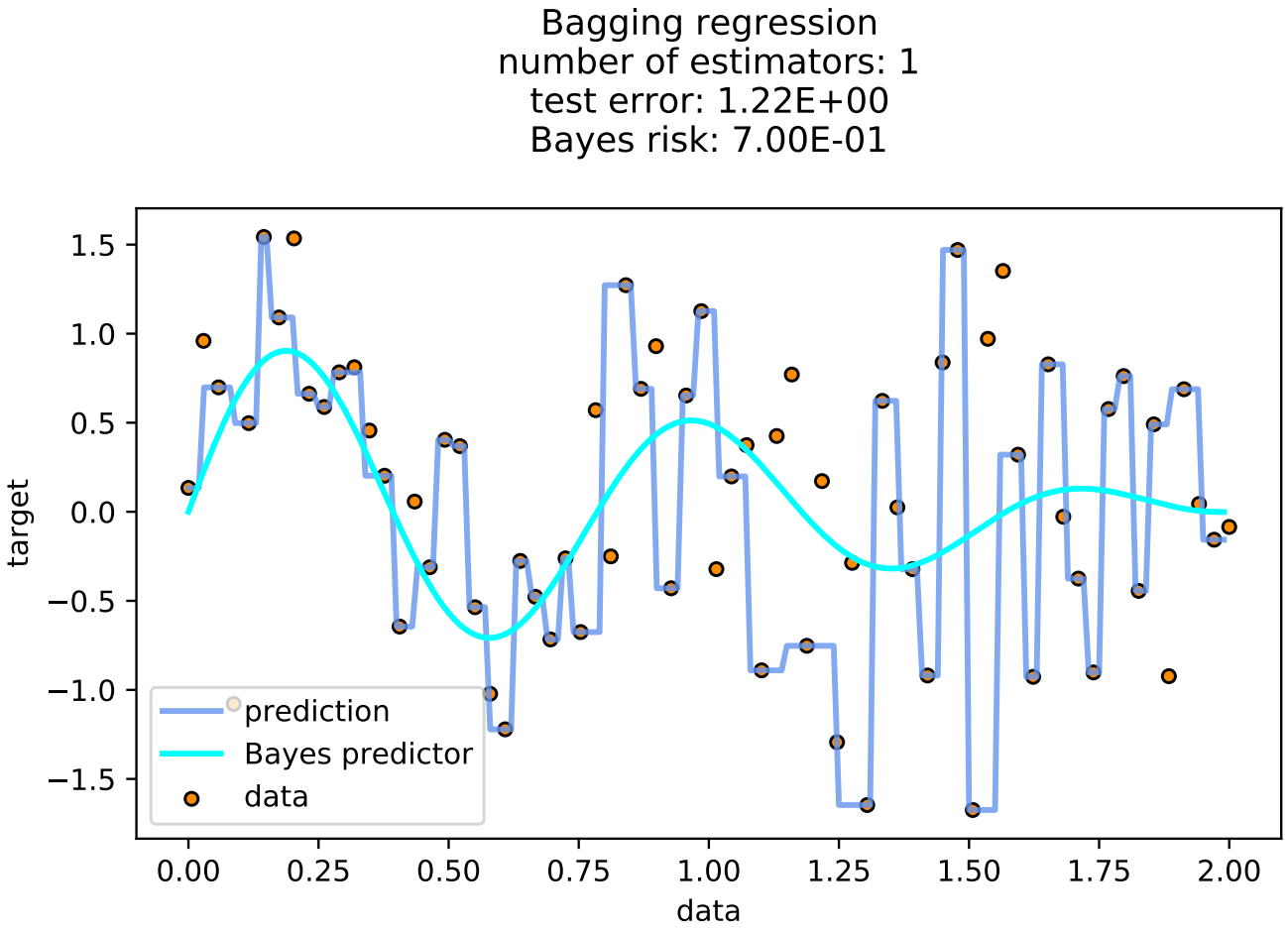
Simulation

- $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = \mathbb{R}$.
- $Y = f(X) + \epsilon$, where ϵ is a centered gaussian noise with variance σ^2 and f is a deterministic function.

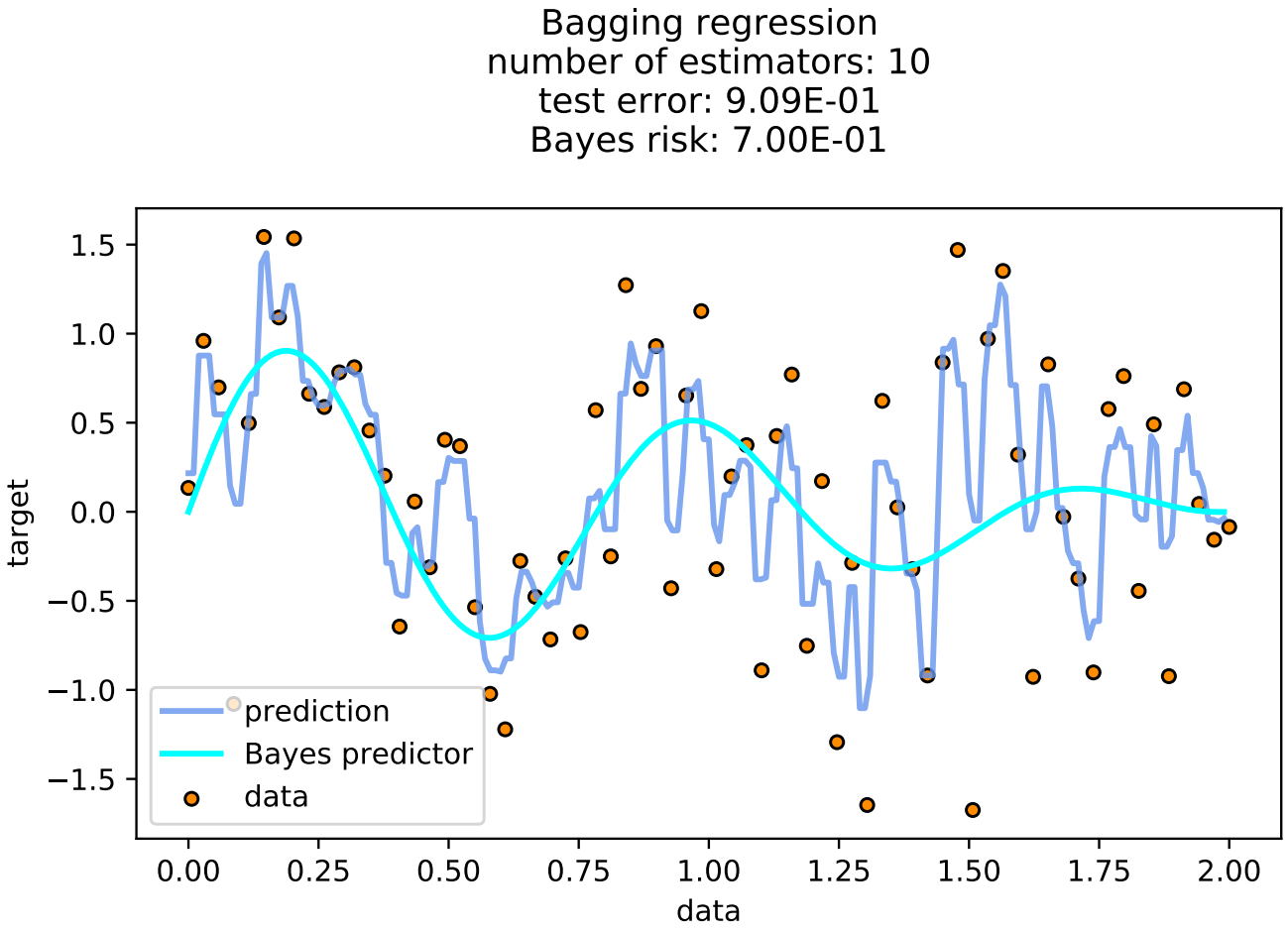


Exercice 5 : With the square loss, what is the Bayes estimator and the Bayes risk ?

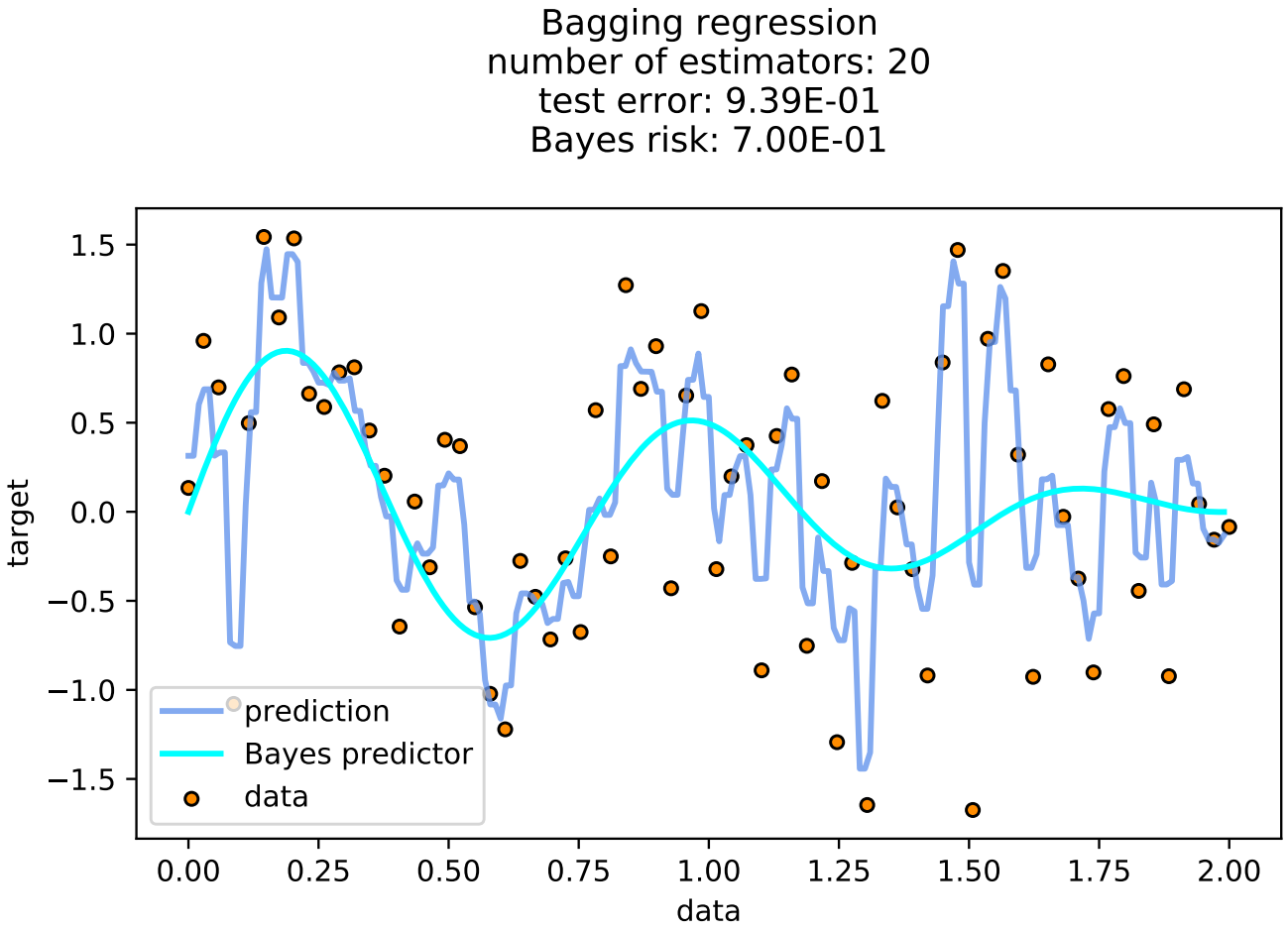
Simulation


Bagging regression
number of estimators: 1
test error: 1.22E+00
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

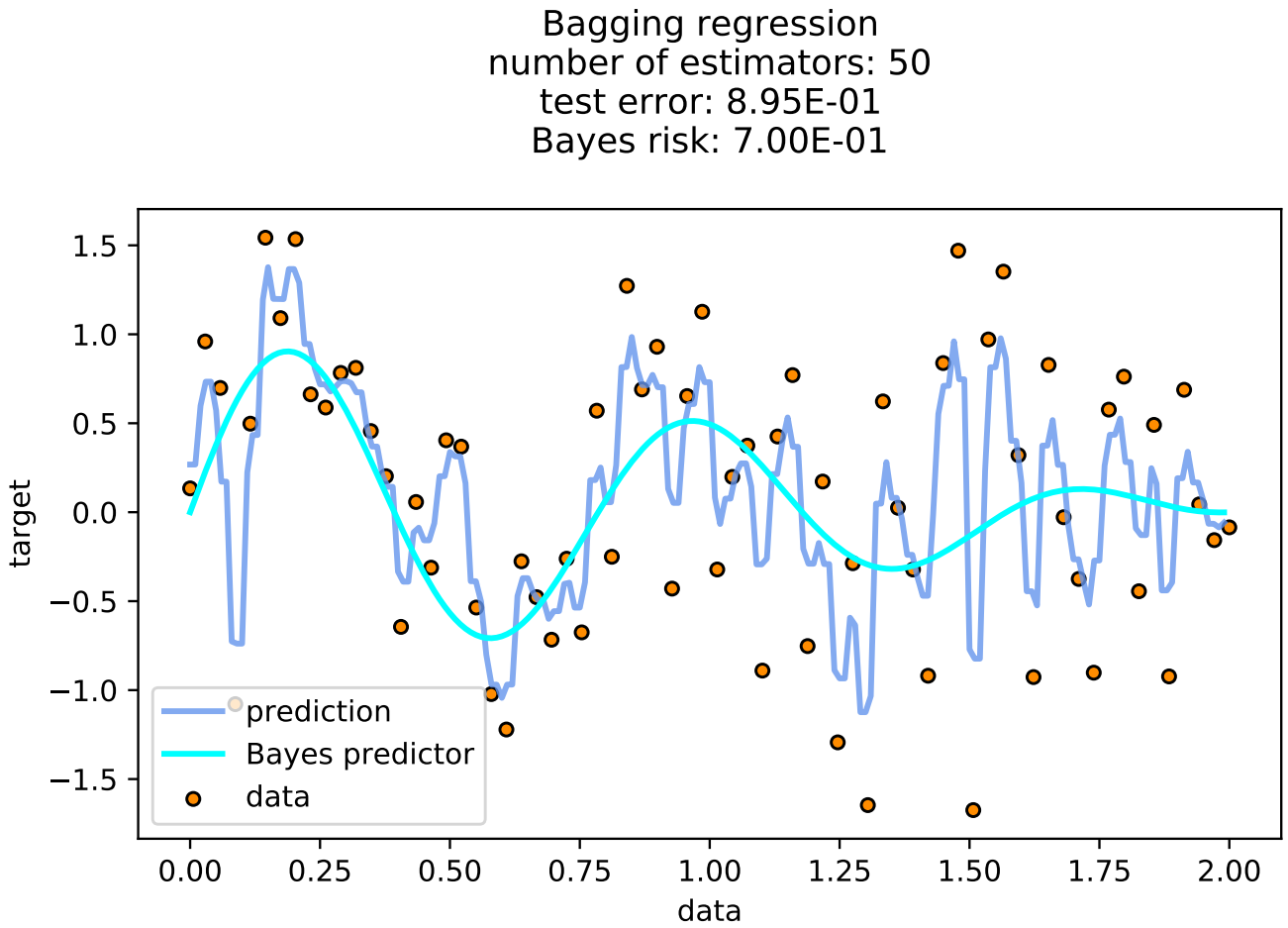
Simulation


Bagging regression
number of estimators: 10
test error: 9.09E-01
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

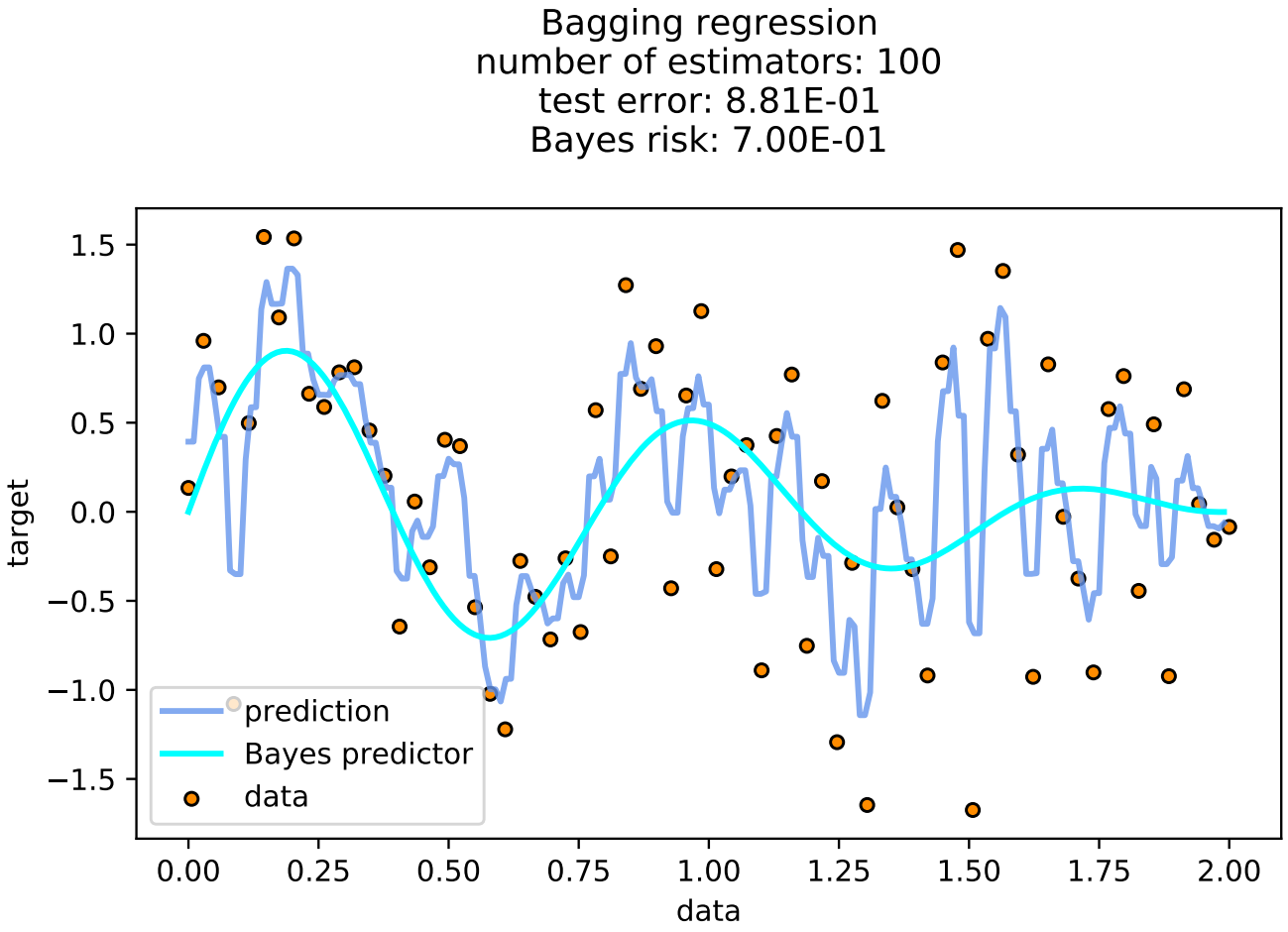
Simulation


Bagging regression
number of estimators: 20
test error: 9.39E-01
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Simulation


Bagging regression
number of estimators: 50
test error: 8.95E-01
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Simulation


Bagging regression
number of estimators: 100
test error: 8.81E-01
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Individual estimators

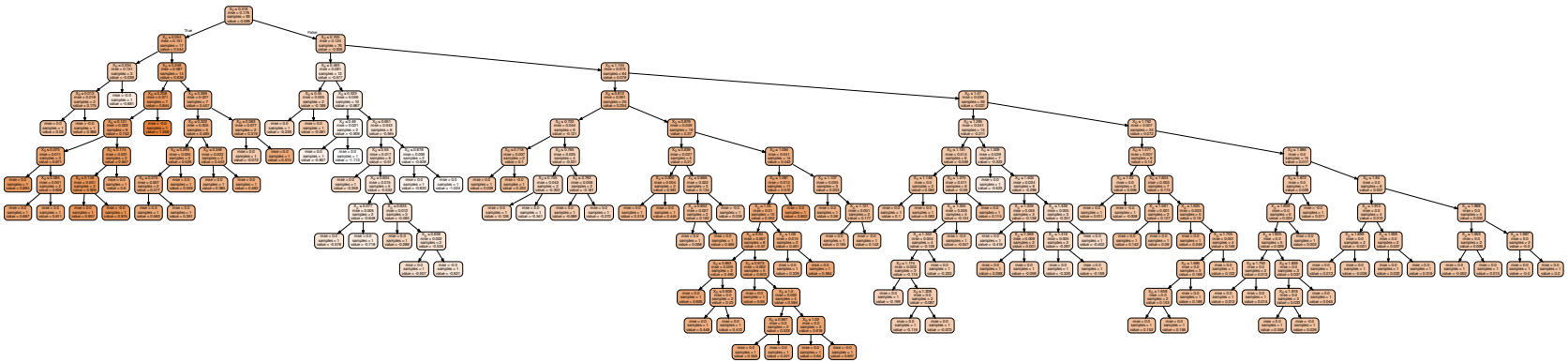


Figure – Estimator used in the averaging

Individual estimators

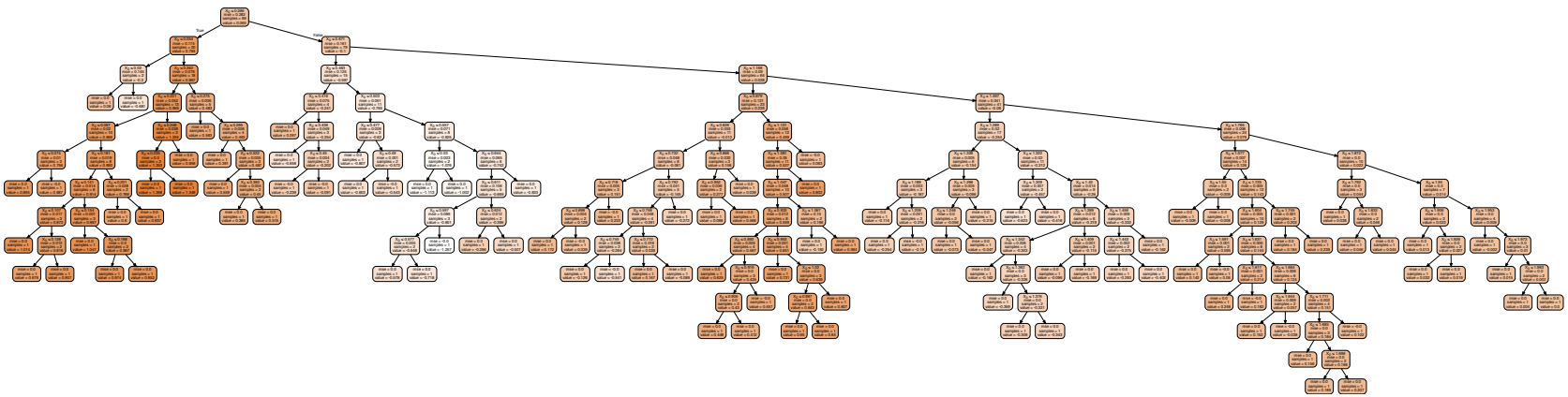


Figure – Other estimator used in the averaging

Possible issues

- number of trees to compute (B) in order to have a stable out-of-bag error.
- need for storing all the base estimators
- the final estimator is a *black-box* model.

Random forest

- Random forest [Breiman, 2001] is a bagging method with binary CART estimators, with additional randomness added to the choice of segmentation variables.
- the goal is to increase the **independence** between the base trees.
- although the theoretical properties of random forest are not yet fully understood, it is an important benchmark in supervised learning (but for instance not when the problem can be solved in a linear way).

Variance of an average : independent variables

Recall that if $(X_k)_{k\in\mathbb{N}}$ is a sequence of i.i.d. real variables that have a moment of order 2, and hence a variance σ^2, and an expected value of m. Then if $S_B = \frac{1}{B}\sum_{i=1}^{B} X_i$

$$\begin{aligned} Var(S_B) &= \sum_{i=1}^{B} Var(\frac{1}{B}X_i) \\ &= \frac{1}{B^2}\sum_{i=1}^{B} Var(X_i) \\ &= \frac{\sigma^2}{B} \end{aligned} \tag{32}$$

Variance of the average : correlated variables

However, if the X_i are identically distributed but correlated with correlation c, then we admit that

$$Vac(S_B) = c\sigma^2 + \frac{1-c}{B}\sigma^2 \tag{33}$$

Random forest diminishes r by adding some randomness in the choice of the segmentation variables.

Random forest

Let d be the number of features of each x_i. Let $m \leq d$ be an integer.

Result: Random forest estimator

for $b \in [1, B]$ **do**

Sample a bootsrap dataset z_b;

Estimate $\hat{f}_{z_b}$ with **randomization** of the variables. The search of the optimal segmentation is done on m randomly sampled variables.;

end

$$\hat{f}_B = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_{z_b} \tag{34}$$

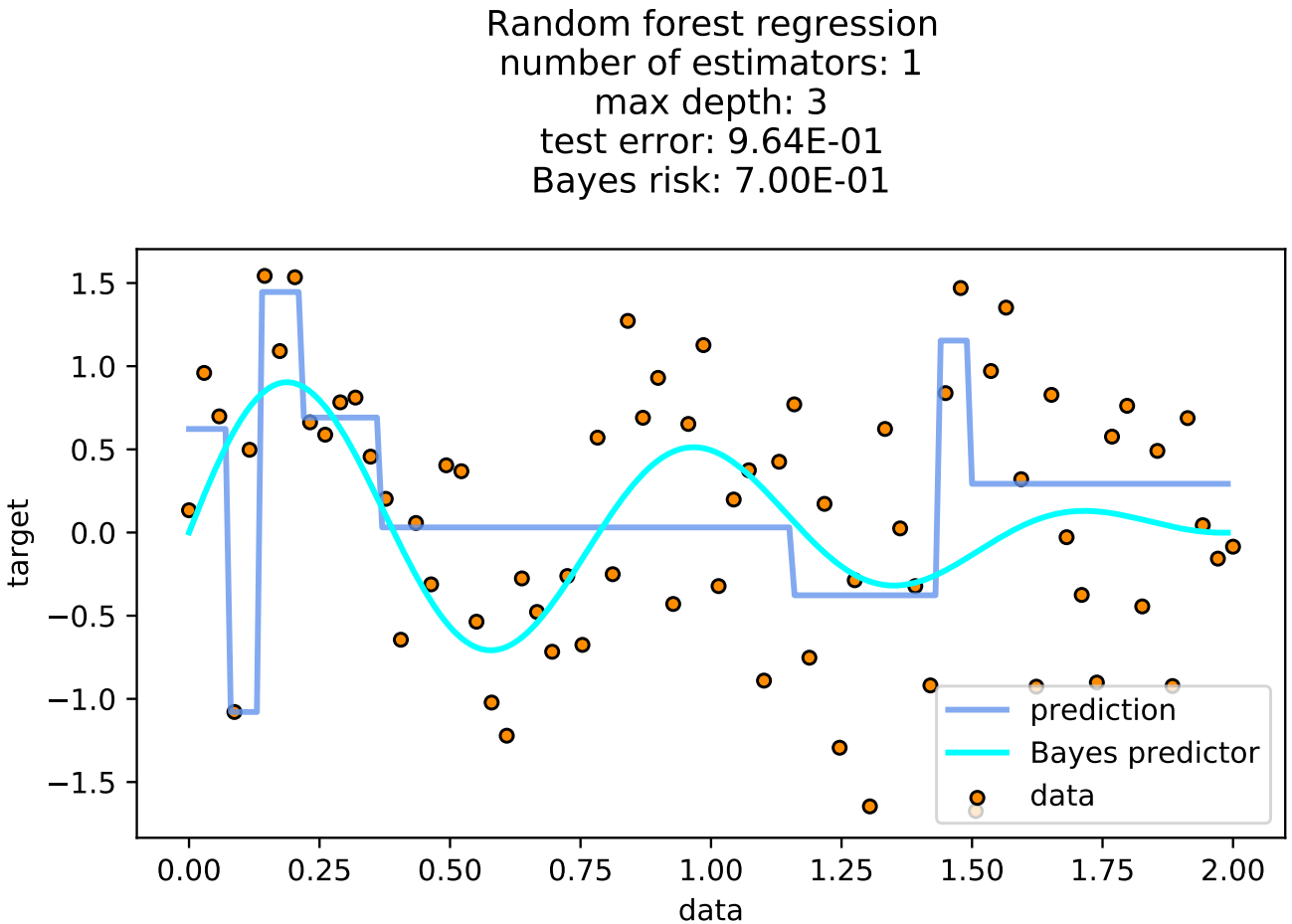
return $\hat{f}_B$

Algorithm 1: Random forest

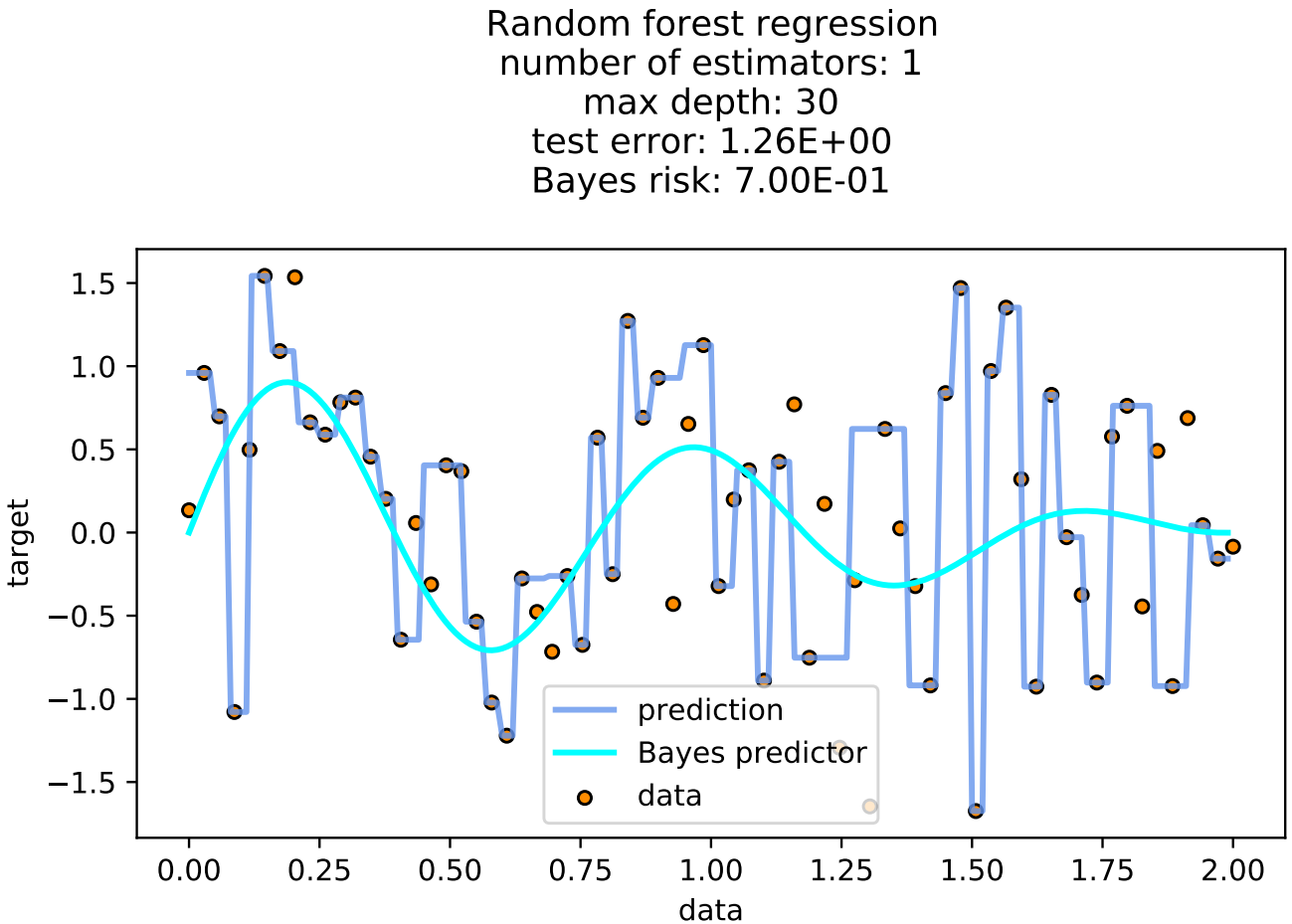
Random forest II

- With random forest, it is possible to use a more brutal pruning (for instance using only trees of depth 2)
- To tune m, heuristics are used. Common ones include :
  - $m = \sqrt{d}$ for classification
  - $m = p/3$ for regression
  - cross validation.

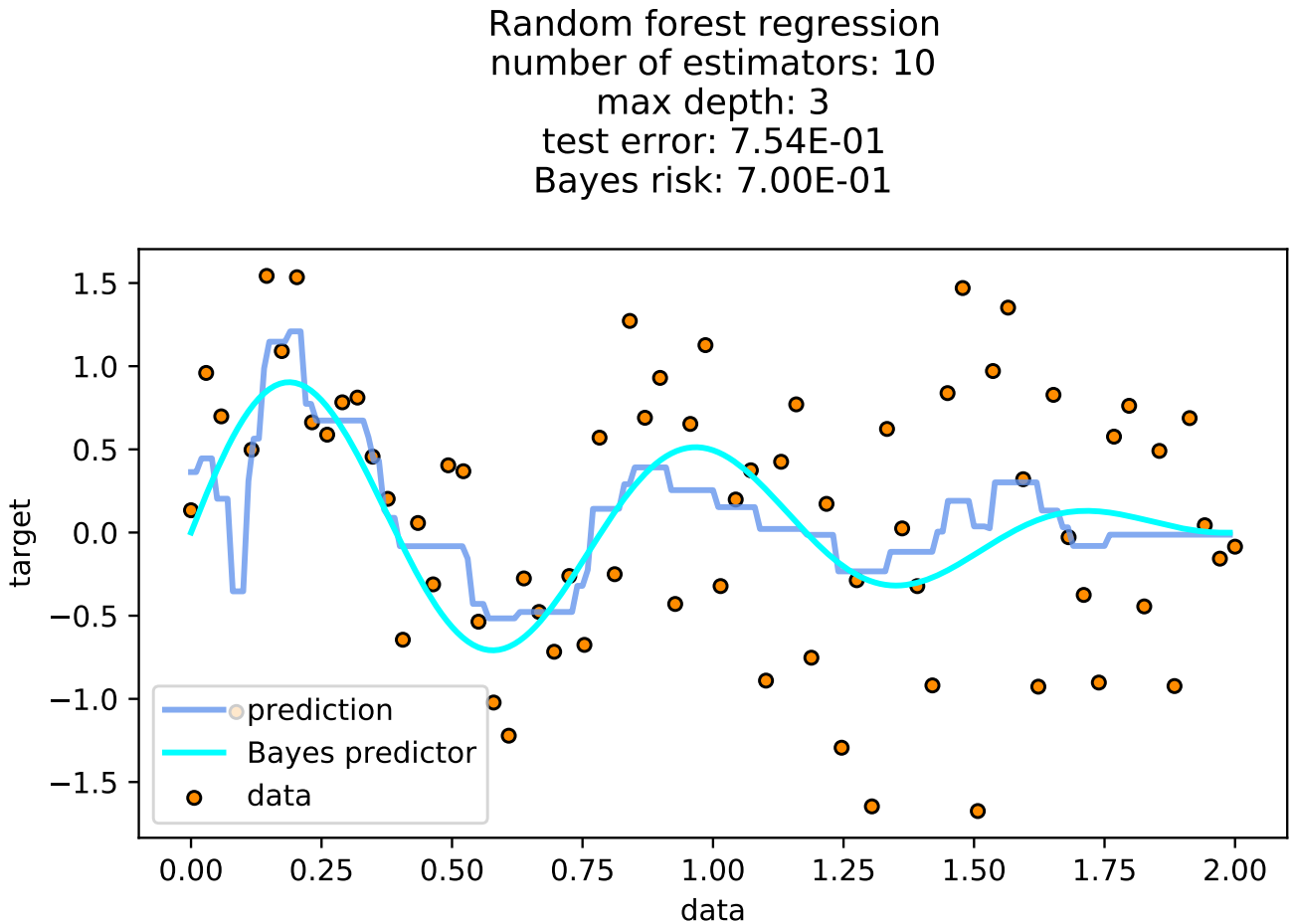
Example in 1D (which is not the most common application of random forest)



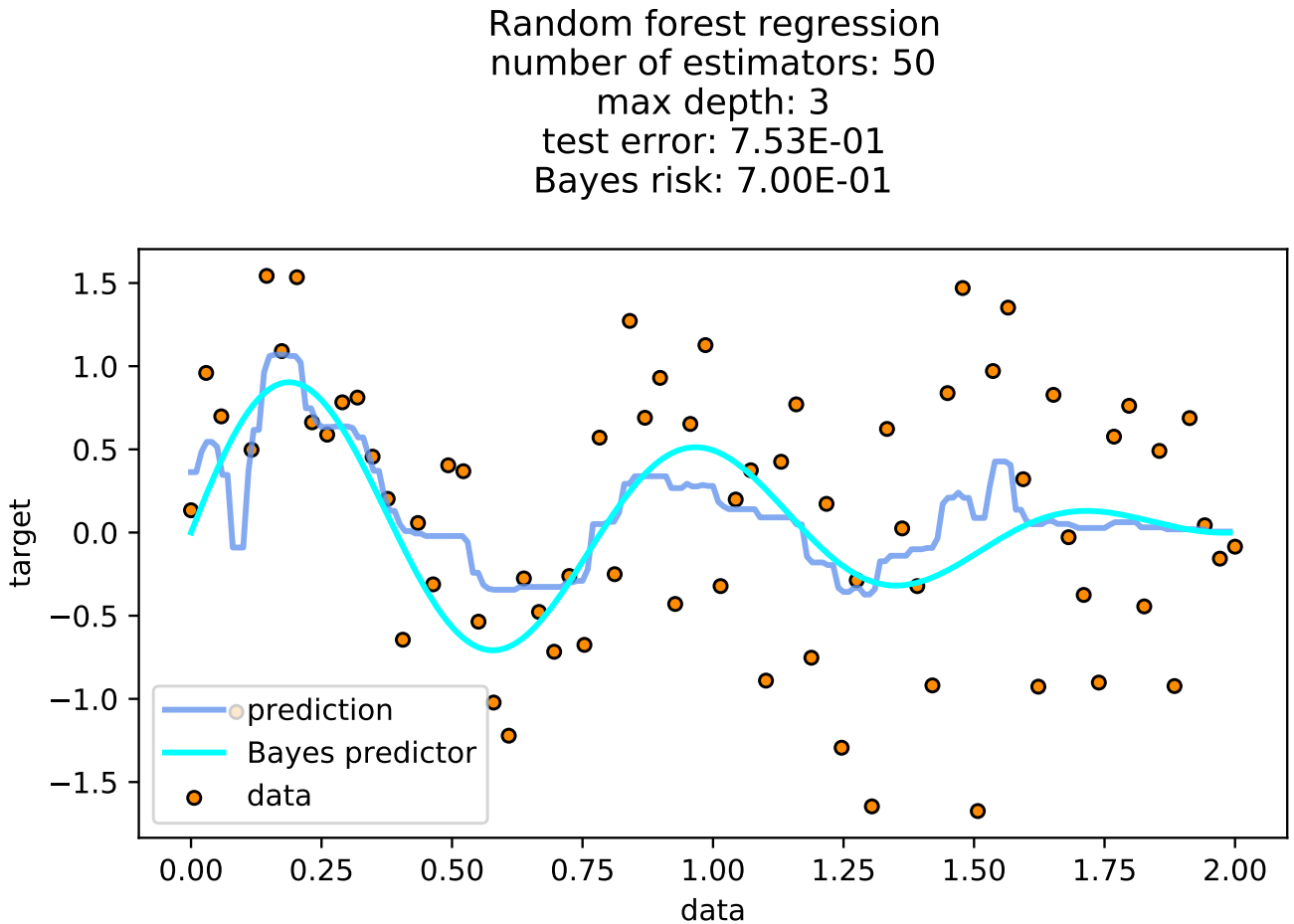
Example in 1D (which is not the most common application of random forest)



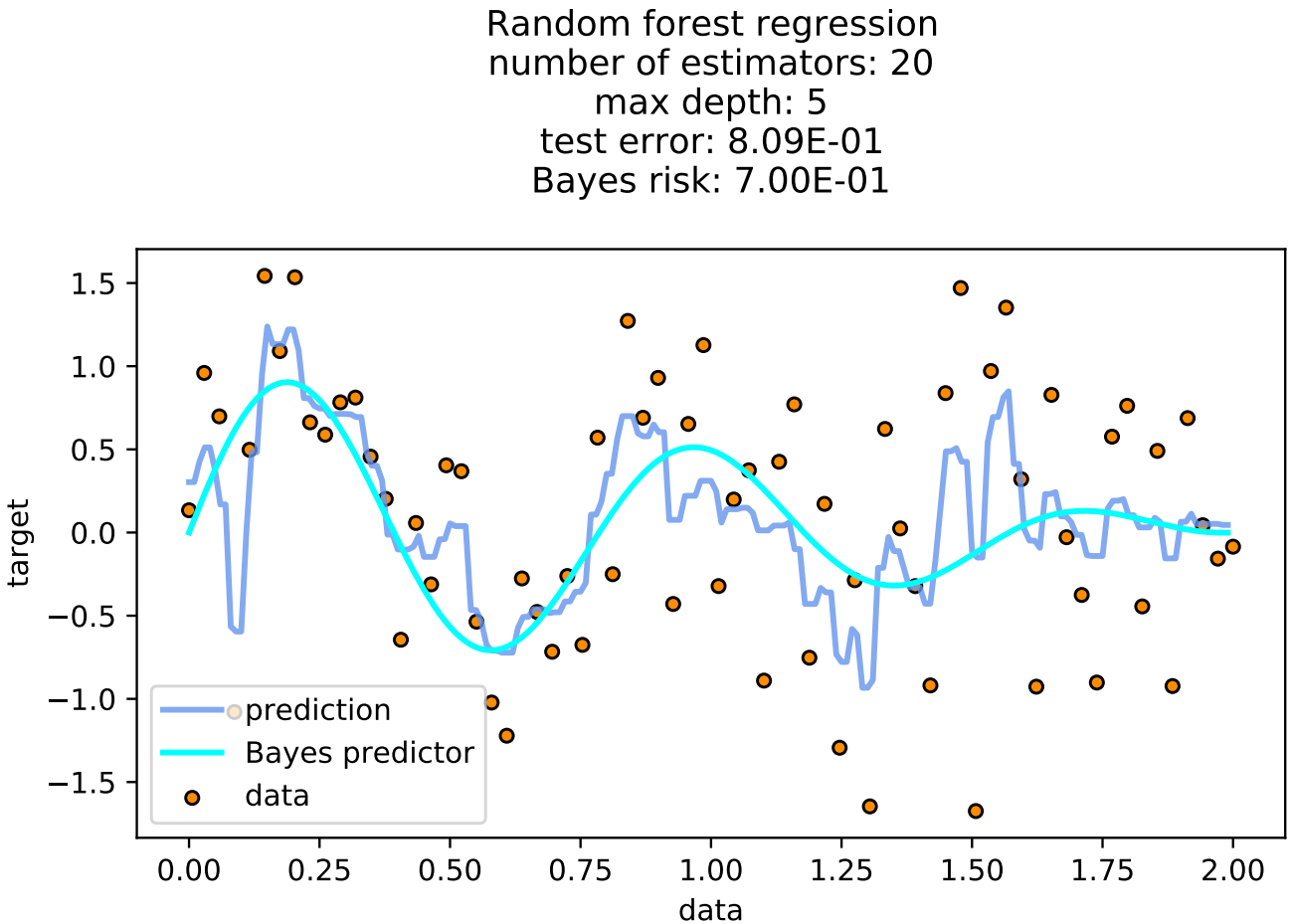
Example in 1D (which is not the most common application of random forest)



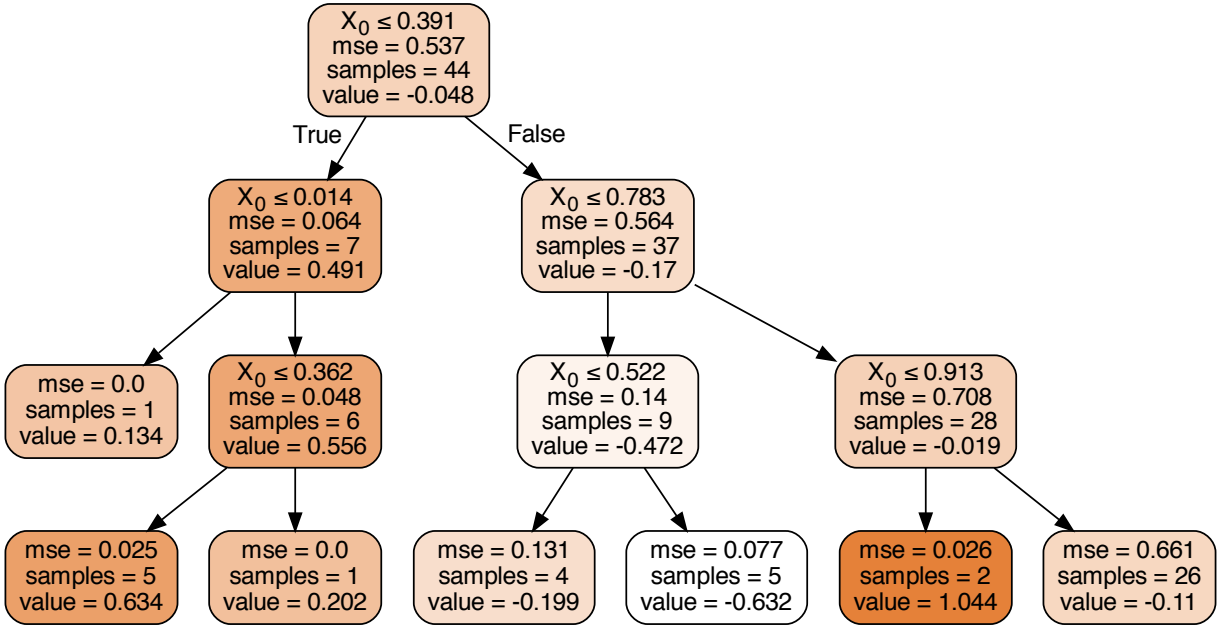
Example in 1D (which is not the most common application of random forest)



Example in 1D (which is not the most common application of random forest)



Estimators


$X_0 \leq 0.391$
mse = 0.537
samples = 44
value = -0.048
True
False
$X_0 \leq 0.014$
mse = 0.064
samples = 7
value = 0.491
$X_0 \leq 0.783$
mse = 0.564
samples = 37
value = -0.17
mse = 0.0
samples = 1
value = 0.134
$X_0 \leq 0.362$
mse = 0.048
samples = 6
value = 0.556
$X_0 \leq 0.522$
mse = 0.14
samples = 9
value = -0.472
$X_0 \leq 0.913$
mse = 0.708
samples = 28
value = -0.019
mse = 0.025
samples = 5
value = 0.634
mse = 0.0
samples = 1
value = 0.202
mse = 0.131
samples = 4
value = -0.199
mse = 0.077
samples = 5
value = -0.632
mse = 0.026
samples = 2
value = 1.044
mse = 0.661
samples = 26
value = -0.11

Importance of the variables

Although the obtained estimator is hard to interpret, we can still study the importance of variables.

Mean decrease accuracy

Consider a variable j. For a given tree b

- compute the out-of-bag error
- shuffle the values of j in the out-of-bag sample.
- compute the new out-of-bag error
- store the decrease in prediction quality, D_j^b.

Average the D_j^b on b in order to measure the imporance of the variable j.
See also "Mean decrease Gini".

Other parameters

- **subsampling** : sampling bootstrap datasets of size $s < n$. Smaller variance because of more independence between the trees, but higher bias.
- number of bins in each variable (for instance, impose a maximum number of bins).

Boosting

- While bagging is a random method, **boosting** is an adaptive method.
- Boosting builds on *weak classifiers*, and like bagging, aggreagates them, for instance with a **weighted sum**.
- However, the weak classifiers are built **sequentially**. The classifier $b + 1$ adapts the classifier b by focusing on the correct prediction of incorrectly classified training samples.
- Several variants exists (in the weighting, loss function, aggregating method, etc).

Adaboost

- Original boosting algorithm [Freund and Schapire, 1996]
- $\mathcal{Y} = \{-1, 1\}$ (but can also be adapted to a regression problem)
- Given the sample dataset $z = \{(x_1, y_1), \dots (x_n, y_n)\}$. We learn a **sequence of estimators** $(\delta_m)_{m \in [1..B]}$ (often CART).

Initialize the weights $w = \{w_i = \frac{1}{n}, i = 1..n\}$;
for $m = 1..B$ **do**

Estimate δ_m on the weighted dataset. Compute the error rate

$$\hat{\epsilon_m} = \frac{\sum_{i=1}^{n} w_i 1(\delta_m(x_i) \neq y_i)}{\sum_{i=1}^{n} w_i} \tag{35}$$

Compute the logit of $\hat{\epsilon_m}$

$$c_m = \log\left(\frac{1 - \hat{\epsilon_m}}{\hat{\epsilon_m}}\right) \tag{36}$$

Update the weigths

$$w_i \leftarrow w_i \exp\Big(c_m 1(\delta_m(x_i) \neq y_i)\Big) \tag{37}$$

end

$$\hat{\delta_B} = \text{sign}\Big(\sum_{b=1}^{B} \delta_{z_b}\Big) \tag{38}$$

return $\hat{f_B}$

Algorithm 2: Adaboost for binary classification (adaptive boosting)

Remarks

We need to enforce that $c_m \geq 0$. It is the case if $\hat{\epsilon_m} \geq \frac{1}{2}$.

$$c_m = \log\left(\frac{1 - \hat{\epsilon_m}}{\hat{\epsilon_m}}\right) \tag{39}$$

Adaptive resampling

It is also possible to bootstrap a sample dataset by weighting the probabilities of sampling each point. For instance if we perform regression (see next slide)

Initialize the probabilities $p = \{p_i = \frac{1}{n}, i = 1..n\}$;
for $m = 1..B$ **do**
    Sample with replaceement z_m according to p.
    Estimate f_m on z_m.
    Compute the loss

$$l_m(i) = l(y_i, \hat{f_m}(x_i)), i \in [1, n] \tag{40}$$

$$\epsilon_m(i) = \sum_{i=1}^{n} p_i l_m(i) \tag{41}$$

$$w_i = g(l_m(i))p_i \tag{42}$$

$$p_i = \frac{w_i}{\sum_{i=1}^{n} w_i} \tag{43}$$

end
return $\hat{f_B}$ as the weighted average or median of the $\hat{f_m}$ by the coefficients $\log(\frac{1}{\beta_m})$.

Algorithm 3: Adaboost for regression

Remarks

- l is a loss, for example the quadratic loss, absolute loss, etc.
- we define $L_m = \sup_{i \in [1,n]} l_m(i)$ the maximum error.
- $$\beta_m = \frac{\hat{\epsilon_m}}{L_m - \hat{\epsilon_m}} \tag{44}$$

  If we enforce $\epsilon_m \geq L_m/2$, then $0 \leq \beta_m \leq 1$.
- $$g(l_m(i)) = \beta_m^{1 - \frac{l_m(i)}{L_m}} \tag{45}$$

  This way, $x \mapsto g(x)$ is increasing.

Adaboost for other problems

- It is experimentally shown that if the weak classifiers are *stump trees* (2 leaves), then adaBoost performs better than one given tree with a number of leaves equal to the number of iterations of adaBoost (hence a comparable computation time).
- A number of leaves q between 4 and 8 for the weak classifiers is often recommended.
- Adaboost can be adapted to multiclass classification and regression [Schapire, 2003].

Gradient tree boosting

- differentiate with respect to the predictions of the tree (compute a gradient)
- approximate the gradient by a regression tree
- update the tree following this gradient.

Gradient tree boosting

Initialize $f_0 = \arg\min_z \sum_{i=1}^{n} l(y_i, z)$;
for $m = 1..B$ **do**
  Compute $r_{m_i} = -\frac{\partial l(y_i, f(x_i))}{\partial f(x_i)}(f = f_{m-1}), i \in [1, n]$
  Train a decision tree δ_m on $(x_i, r_{m_i})_{i \in [1,n]}$
  Compute γ_m minizing

$$\sum_{i=1}^{n} l(y_i, f_{m-1}(x)) + \gamma \delta_m(x_i) \tag{46}$$

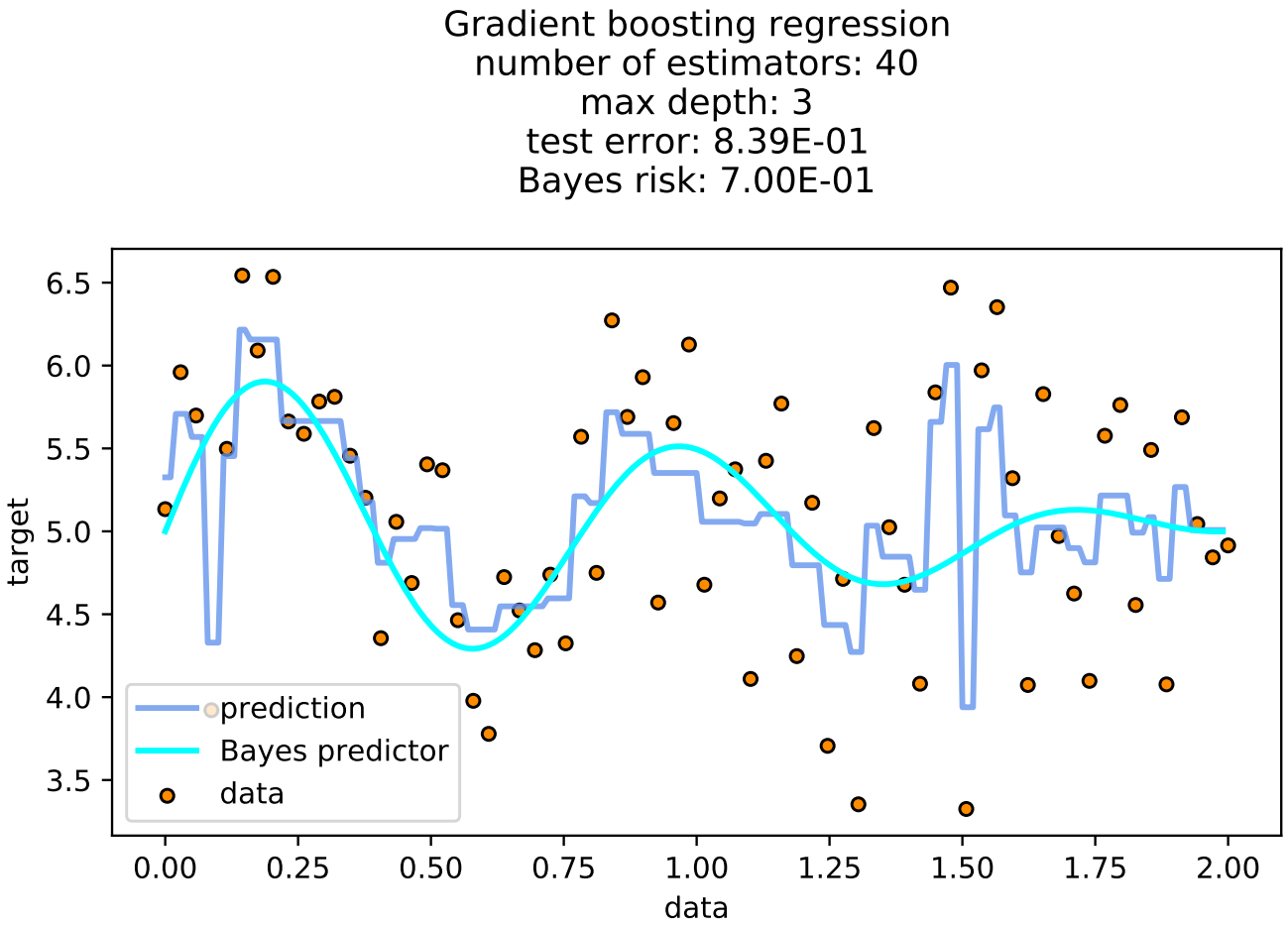
  Update the estimator

$$\hat{f_m} = \hat{f_{m-1}} + \gamma_m \delta_m \tag{47}$$

end
return $\hat{f_B}$

Algorithm 4: Adaboost for regression

Simulation


Gradient boosting regression
number of estimators: 40
max depth: 3
test error: 8.39E-01
Bayes risk: 7.00E-01
target
data
prediction
Bayes predictor
data
6.5
6.0
5.5
5.0
4.5
4.0
3.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Gradient tree boosting

Many parameter can be tuned :

- maximum depth of the estimators
- shrinkage η ($\hat{f_m} = \hat{f_{m-1}} + \eta\gamma_m\delta_m$, with $0 \leq \eta \leq 1$). If η is low, e.g. ≤ 0.1, it can lead to a slower convergence but might prevent overfitting.
- number of trees learned B

`https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-pytho`

It is necessary to experiment, read documentations, cross validate, use heuristics, etc.

Extrem gradient boosting

- XGBoost : variant of gradient boosting, very efficient on several benchmarks [Chen and Guestrin, 2016]
- can exploit parallelization
- `https://xgboost.readthedocs.io/en/latest/parameter.html`
- `https://github.com/dmlc/xgboost`

Catboost

- See also : Catboost
  `https://catboost.ai/`

Super learners

References I

Bach, F. (2014).
Adaptivity of averaged stochastic gradient descent to local strong convexity for logistic regression.
Journal of Machine Learning Research, 15 :595–627.

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

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (2017).
Classification And Regression Trees.
Routledge.

References II

Chen, T. and Guestrin, C. (2016).
XGBoost : A scalable tree boosting system.
Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 13-17-Augu :785–794.

Fernández-Delgado, M., Cernadas, E., Barro, S., and Amorim, D. (2014).
Do we need hundreds of classifiers to solve real world classification problems ?
Journal of Machine Learning Research, 15 :3133–3181.

Freund, Y. and Schapire, R. E. (1996).
Experiments with a New Boosting Algorithm.
Proceedings of the 13th International Conference on Machine Learning, pages 148–156.

References III


Mücke, N., Neu, G., and Rosasco, L. (2019).
Beating SGD saturation with tail-averaging and minibatching.
Advances in Neural Information Processing Systems, 32 :1–37.


Schapire, R. E. (2003).
The Boosting Approach to Machine Learning : An Overview
BT - Nonlinear Estimation and Classification.
Nonlinear Estimation and Classification, 171(Chapter 9) :149–171.


Varre, A., Pillaud-Vivien, L., and Flammarion, N. (2021).
Last iterate convergence of SGD for Least-Squares in the Interpolation regime.
CoRR, abs/2102.0.