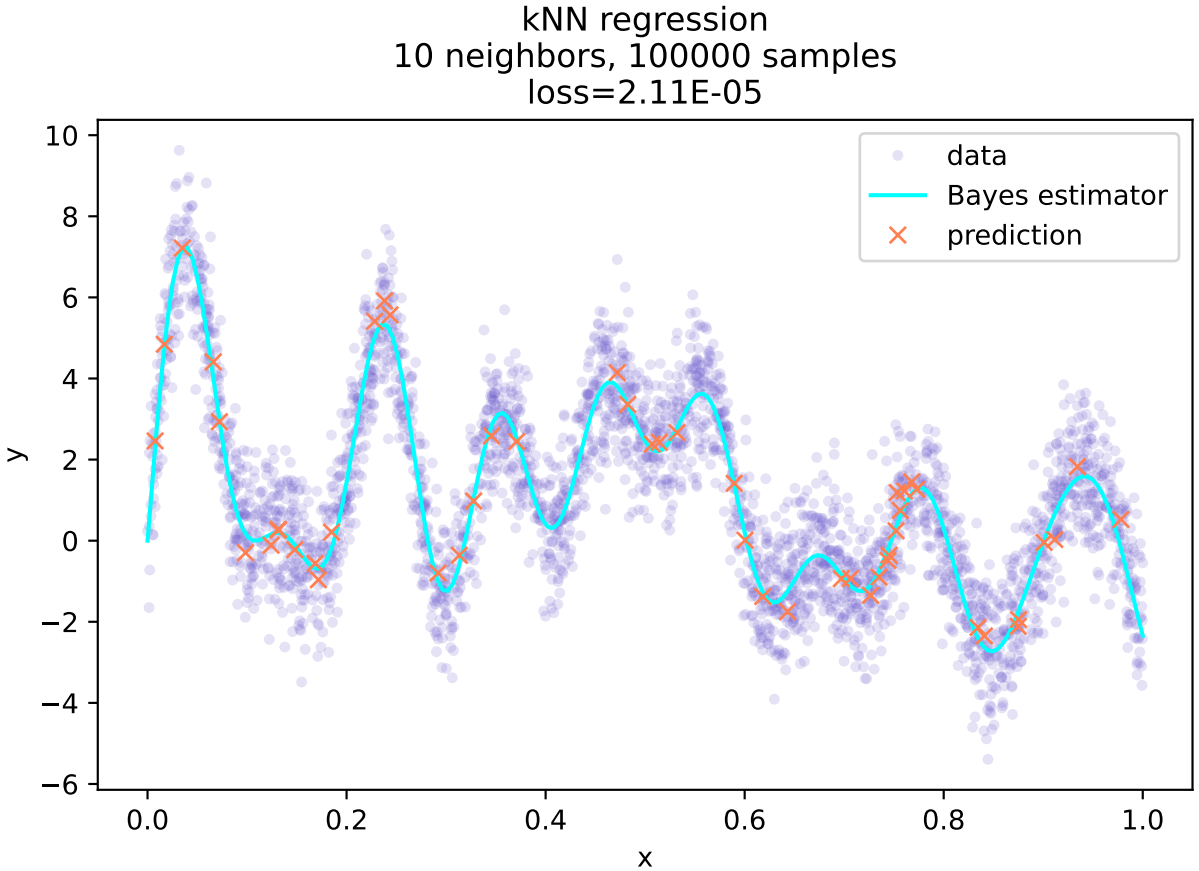
Fondamentaux théoriques du machine learning



Overview of lecture 11

Local averaging methods

- Supervised learning
- Density estimation

Metrics and representation for non-numerical data

- Categorical data
- Texts

Model selection and sparsity

- Model selection
- Lasso

Adaptivity

- No free lunch theorems
- Adaptivity

Local averaging methods

Local averaging methods : approximation **without** optimization of an empirical risk.

Setting

Let l be a loss. Generalization error :

$$R(f) = E_{(X,Y)\sim\rho}[l(Y, f(X))] \tag{1}$$

Bayes estimator :

$$f^* = \underset{f \text{ measurable}}{\arg\min} \; R(f) \tag{2}$$

The empirical risk is not considered.

Setting

Let l be a loss. Generalization error :

$$R(f) = E_{(X,Y)\sim\rho}[l(Y, f(X))] \tag{3}$$

Bayes estimator :

$$f^* = \underset{f \text{ measurable}}{\arg\min} \; R(f) \tag{4}$$

Bayes risk :

$$R^* = E_X\left[\inf_{y\in\mathcal{Y}} E_{Y\sim dP(Y|X)}\left[l(Y, y)|X\right]\right] \tag{5}$$

In the following, dP denotes the **distribution of probability**.
As always, $dP(X, Y)$ and $dP(Y|X = x)$, are unknown.

Classical case : regression with squared loss

$$f^*(x) = E\big[Y|X = x\big] \tag{6}$$

Assumption / example : $\forall x \in \mathcal{X}$, the random variable $Y|X = x$ has a continuous density, noted $p_{Y|X=x}$. Then

$$f^*(x) = \int_{y\in\mathbb{R}} y p_{Y|X=x}(y)dy \tag{7}$$

Actually, this assumption is not necessary with, the law of $Y|X = x$ need not have a density and we can write (Lebesgue integration)

$$f^*(x) = \int_{y\in\mathbb{R}} y dP(Y|X = x) \tag{8}$$

Classical case : binary classification with "0-1" loss

$$f^*(x) = \arg\max_{z \in \mathcal{Y}} P(Y = z | X = x) \tag{9}$$

Bayes estimator

In both previous cases, if we knew $dP(Y|X = x)$, we could compute the Bayes estimator directly.
If dP(Y|X=x) is known, learning is not necessary !

Bayes estimator

In both previous cases, if we knew $dP(Y|X = x)$, we could compute the Bayes estimator directly.
If dP(Y|X=x) is known, learning is not necessary !
However, $dP(Y|X = x)$ is not known.

Local averaging

- $D_n = \{(x_i, y_i), i \in [1, \dots, n]\}$
- $x_i \in \mathcal{X}$
- $y_i \in \mathbb{R}$ or $y_i \in \{0, 1\}$ (for instance)

Local averaging : based on the dataset D_n, compute an approximation $\hat{dP}(Y|X = x)$ of $dP(Y|X = x)$, without optimization of an empirical risk.
And then use it in the estimator.

Local averaging : regression

$\tilde{f}(x)$: local averaging estimator, in the case of regression, squared loss, we can use

$$\tilde{f}(x) = \int_{y \in \mathbb{R}} y d\hat{P}(Y|X = x) \tag{10}$$

Local averaging : classification

$\tilde{f}(x)$: local averaging estimator, in the case of binary classification, squared loss, we can use

$$\tilde{f}(x) = \arg\max_{z \in \mathcal{Y}} \hat{P}(Y = z | X = x) \tag{11}$$

Linear estimators

The question is then : how to choose the approximation $\hat{dP}(Y|X=x)$?

Linear estimators

$$\hat{dP}(Y|X=x) = \sum_{i=1}^{n} \hat{w}_i(x)\delta_{y_i}(y) \tag{12}$$

δ_{y_i} is the Dirac mass in y_i.

- $\forall i, \hat{w}_i(x) \geq 0$
- $\sum_{i=1}^{n} \hat{w}(x) = 1$

Linear estimators

Linear estimators

$$\hat{dP}(Y|X=x) = \sum_{i=1}^{n} \hat{w}_i(x)\delta_{y_i}(y) \tag{13}$$

δ_{y_i} is the Dirac mass in y_i.

- $\forall i, \hat{w}_i(x) \geq 0$
- $\sum_{i=1}^{n} \hat{w}(x) = 1$

Application to regression :

$$\tilde{f}(x) = \sum_{i=1}^{n} \hat{w}_i(x) y_i \tag{14}$$

Linear estimators

Linear estimators

$$\hat{dP}(Y|X=x) = \sum_{i=1}^{n} \hat{w}_i(x)\delta_{y_i}(y) \tag{15}$$

δ_{y_i} is the Dirac mass in y_i.

- $\forall i, \hat{w}_i(x) \geq 0$
- $\sum_{i=1}^{n} \hat{w}(x) = 1$

Application to classification :

$$\tilde{f}(x) = \arg\max_{j \in \{0,1\}} \sum_{i=1}^{n} \hat{w}_i(x) 1_{y_i=j} \tag{16}$$

Choice of the weights

Linear estimators

$$\hat{dP}(Y|X=x) = \sum_{i=1}^{n} \hat{w}_i(x)\delta_{y_i}(y) \tag{17}$$

δ_{y_i} is the Dirac mass in y_i.

- $\forall i, \hat{w}_i(x) \geq 0$
- $\sum_{i=1}^{n} \hat{w}(x) = 1$

For any sample i, the weight function $\hat{w}_i(x)$ should be

- closer to 1 for training point x_i that are close to x.
- closer to 0 for training point x_i that are far from x.

Choice of the weights

Linear estimators

$$\hat{dP}(Y|X=x) = \sum_{i=1}^{n} \hat{w}_i(x)\delta_{y_i}(y) \tag{18}$$

δ_{y_i} is the Dirac mass in y_i.

- $\forall i, \hat{w}_i(x) \geq 0$
- $\sum_{i=1}^{n} \hat{w}(x) = 1$

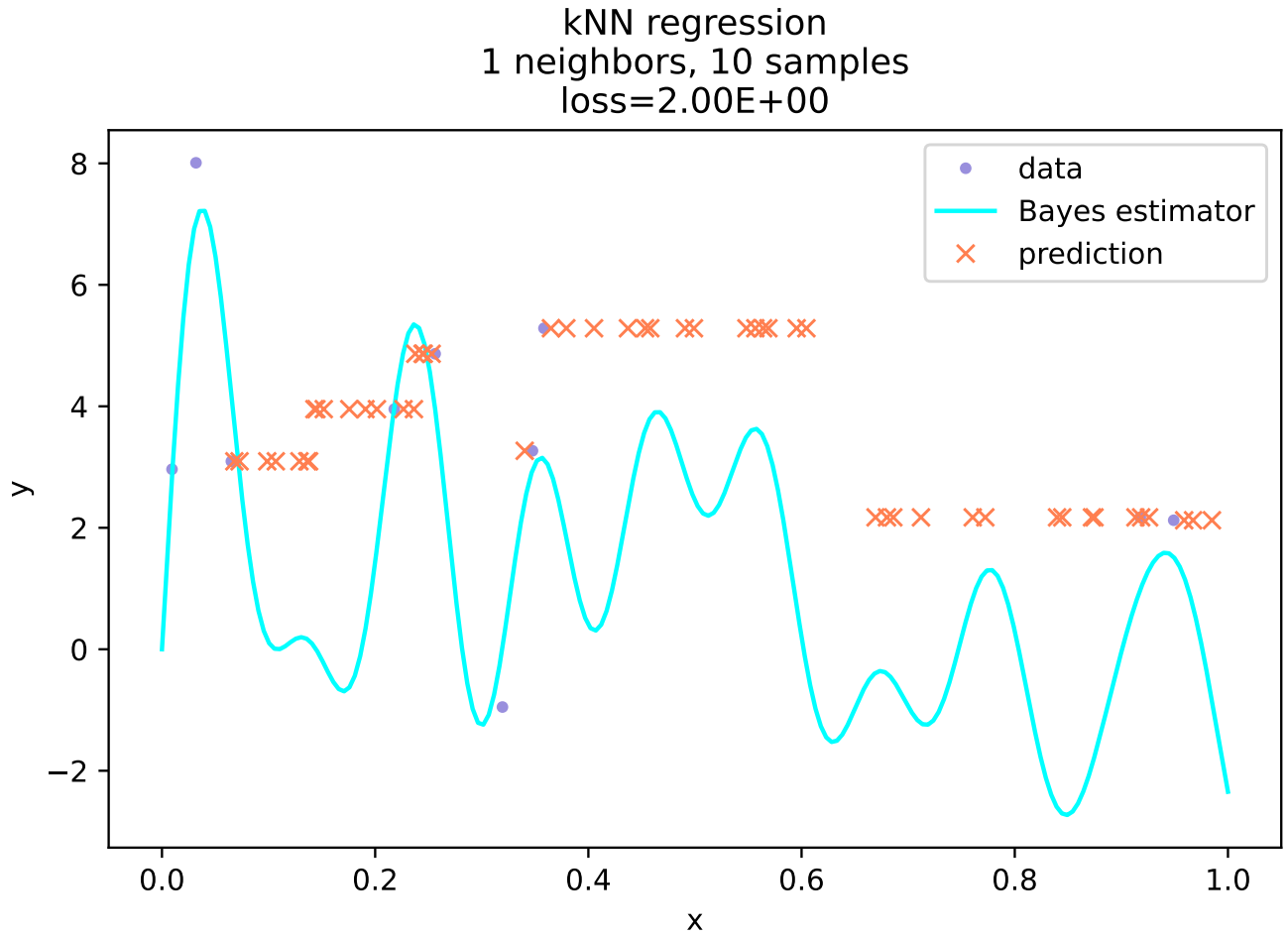
Three possibilities :

- partition estimators
- nearest neighbors
- Nadaraya-Watson (kernel regression)

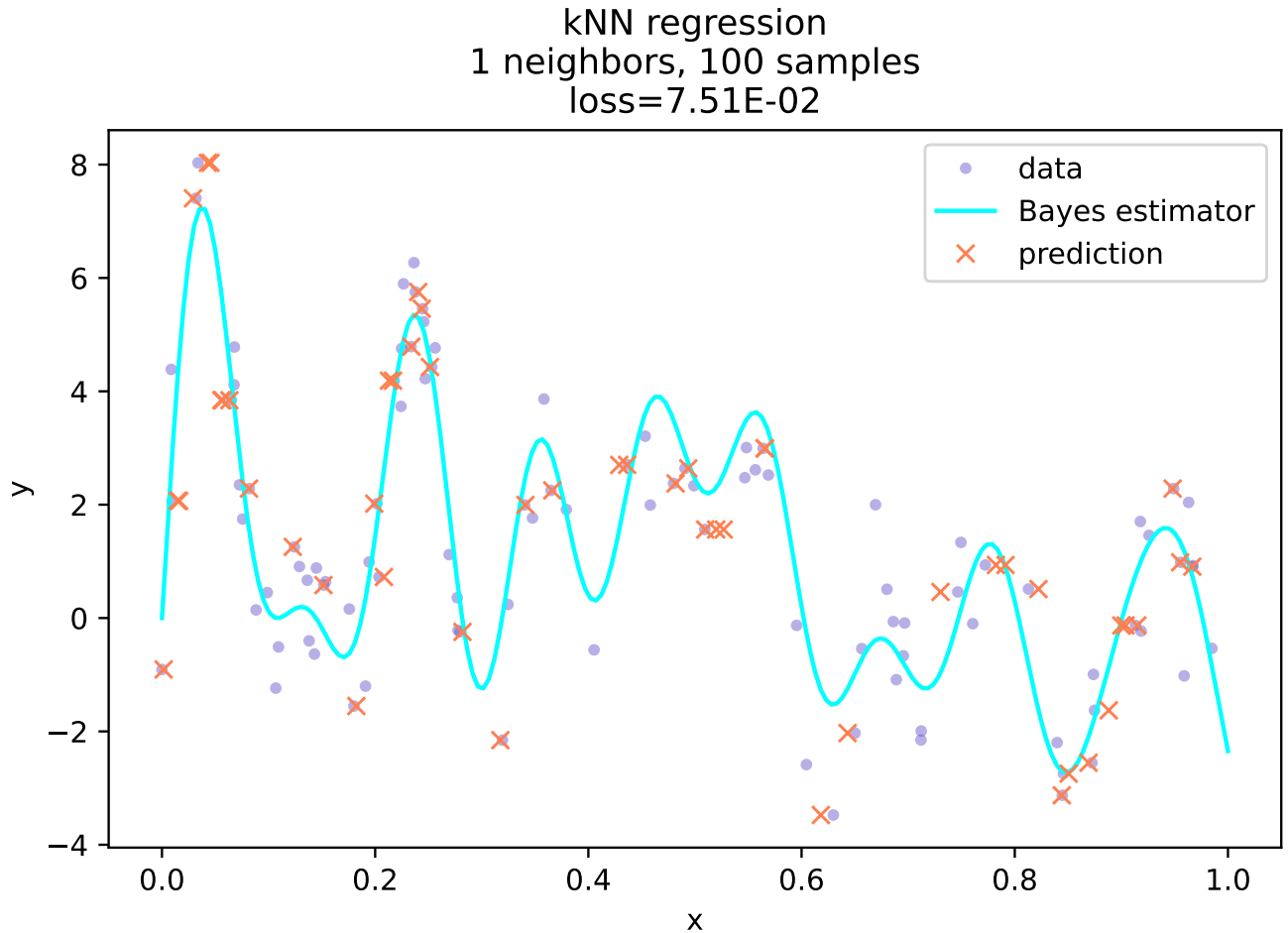
Nearest neighbors

Given $k \geq 1$, and a metric d on $\mathcal{X}$, average the predictions of the k nearest neighbors (for regression) or take the majority vote (for classification).

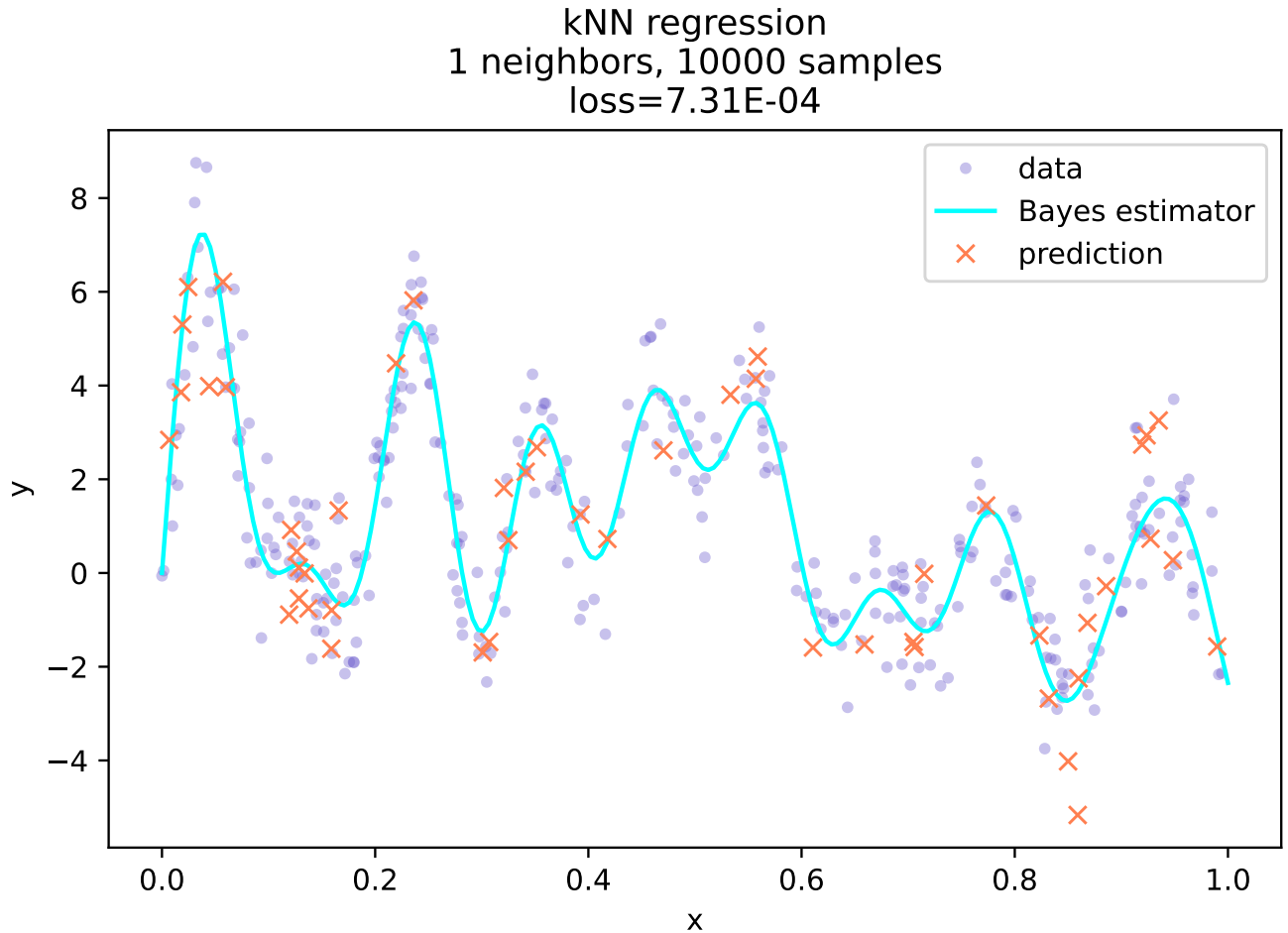
Nearest neighbors



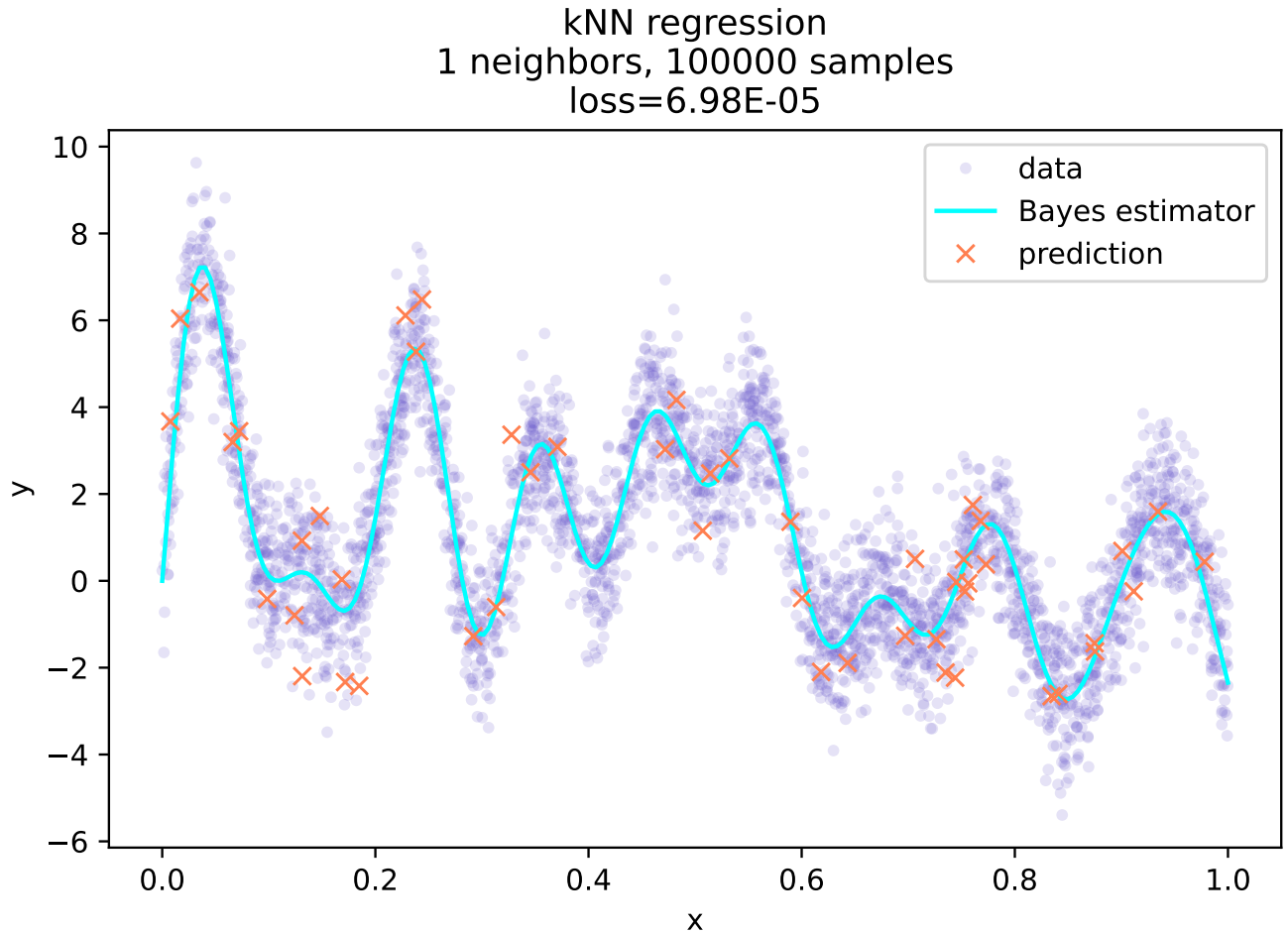
Nearest neighbors



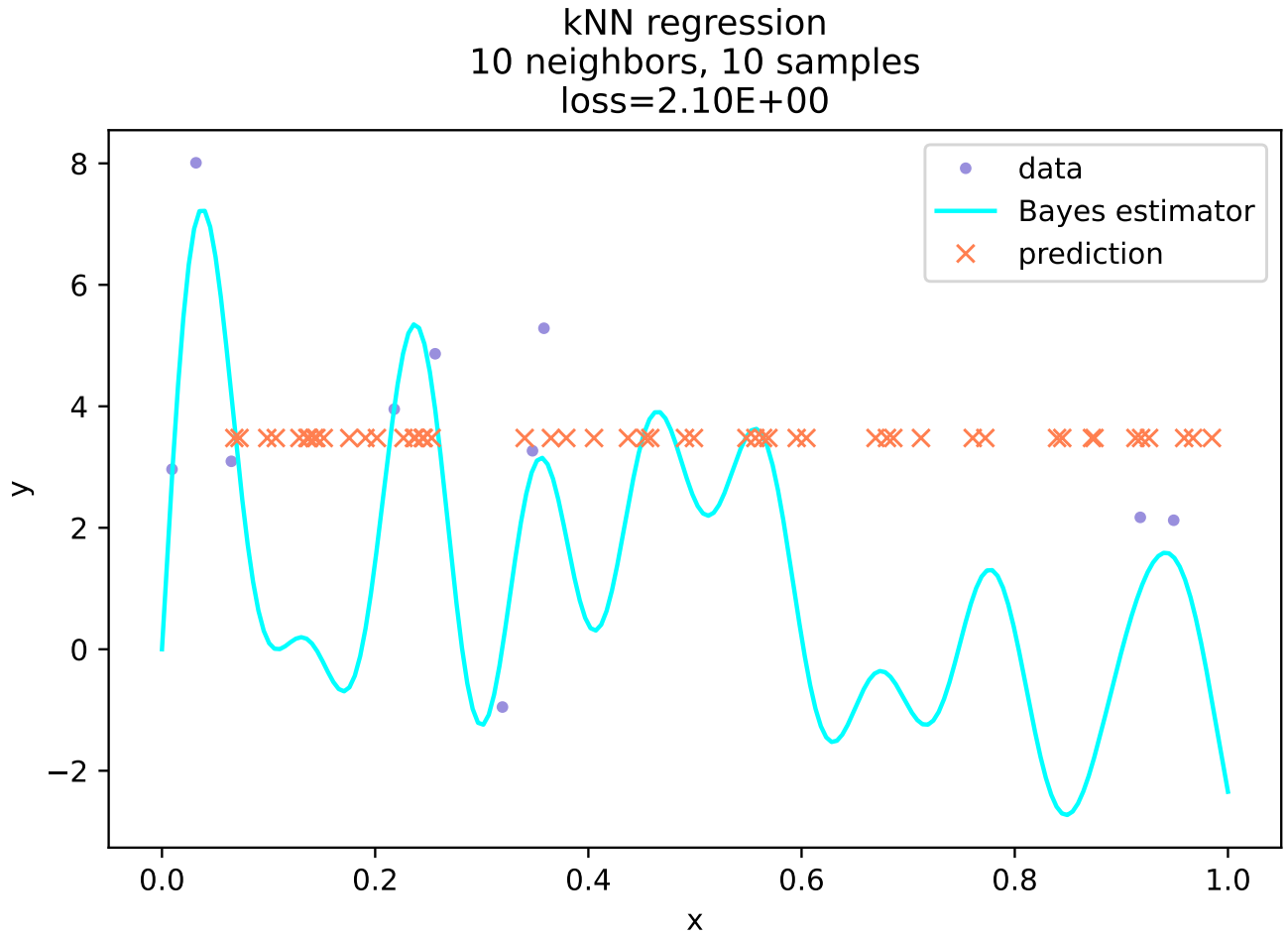
Nearest neighbors



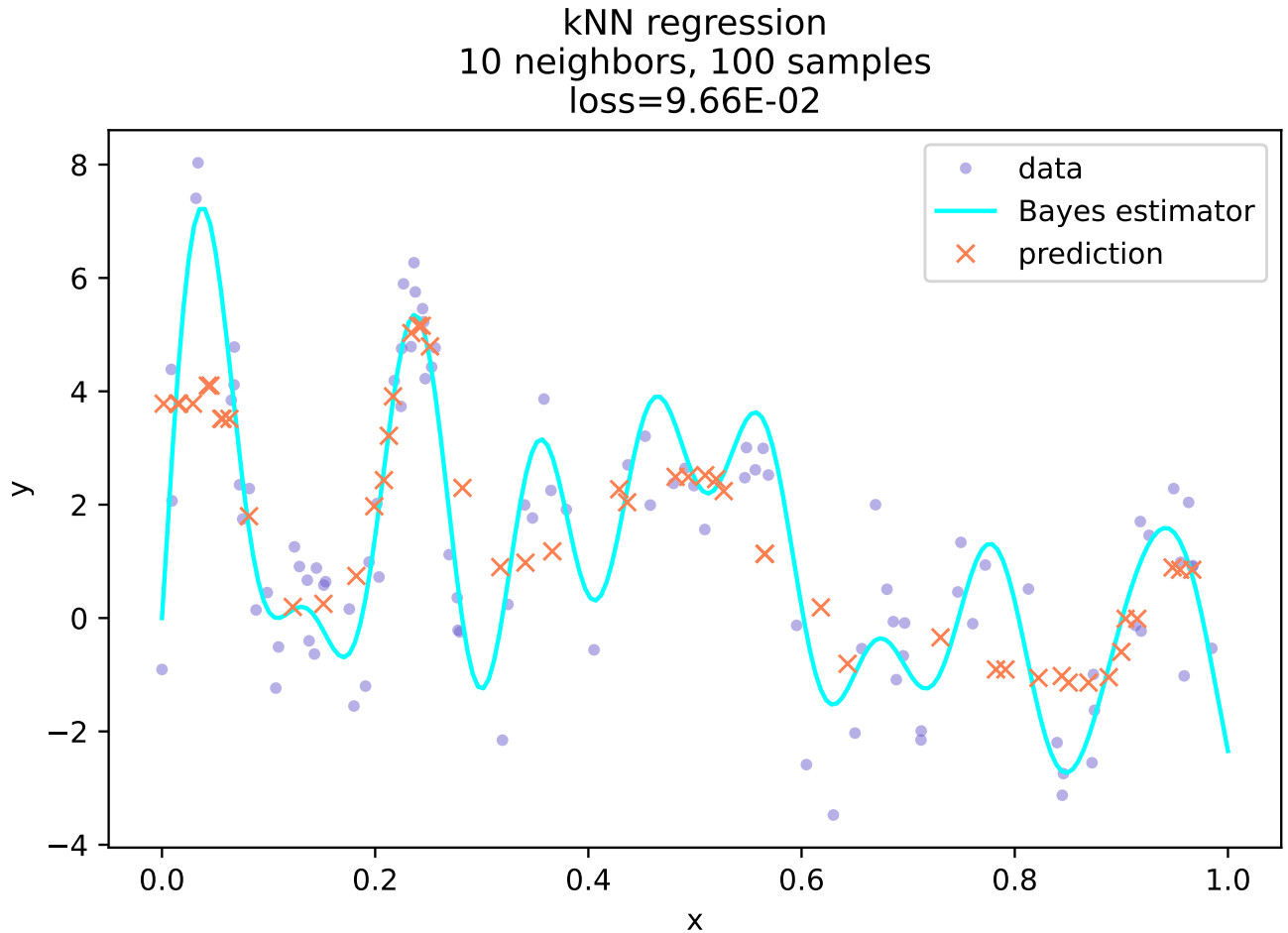
Nearest neighbors



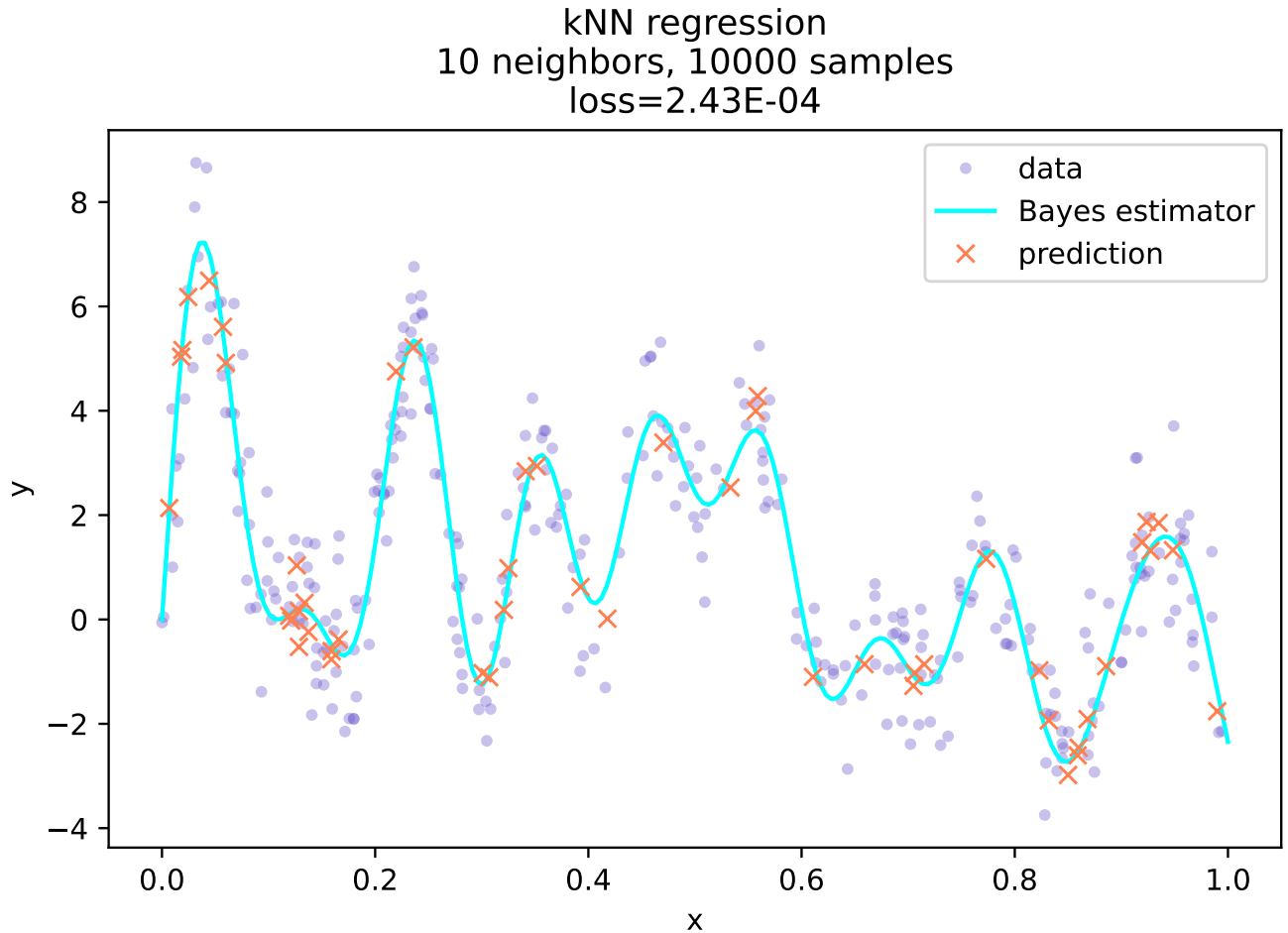
Nearest neighbors



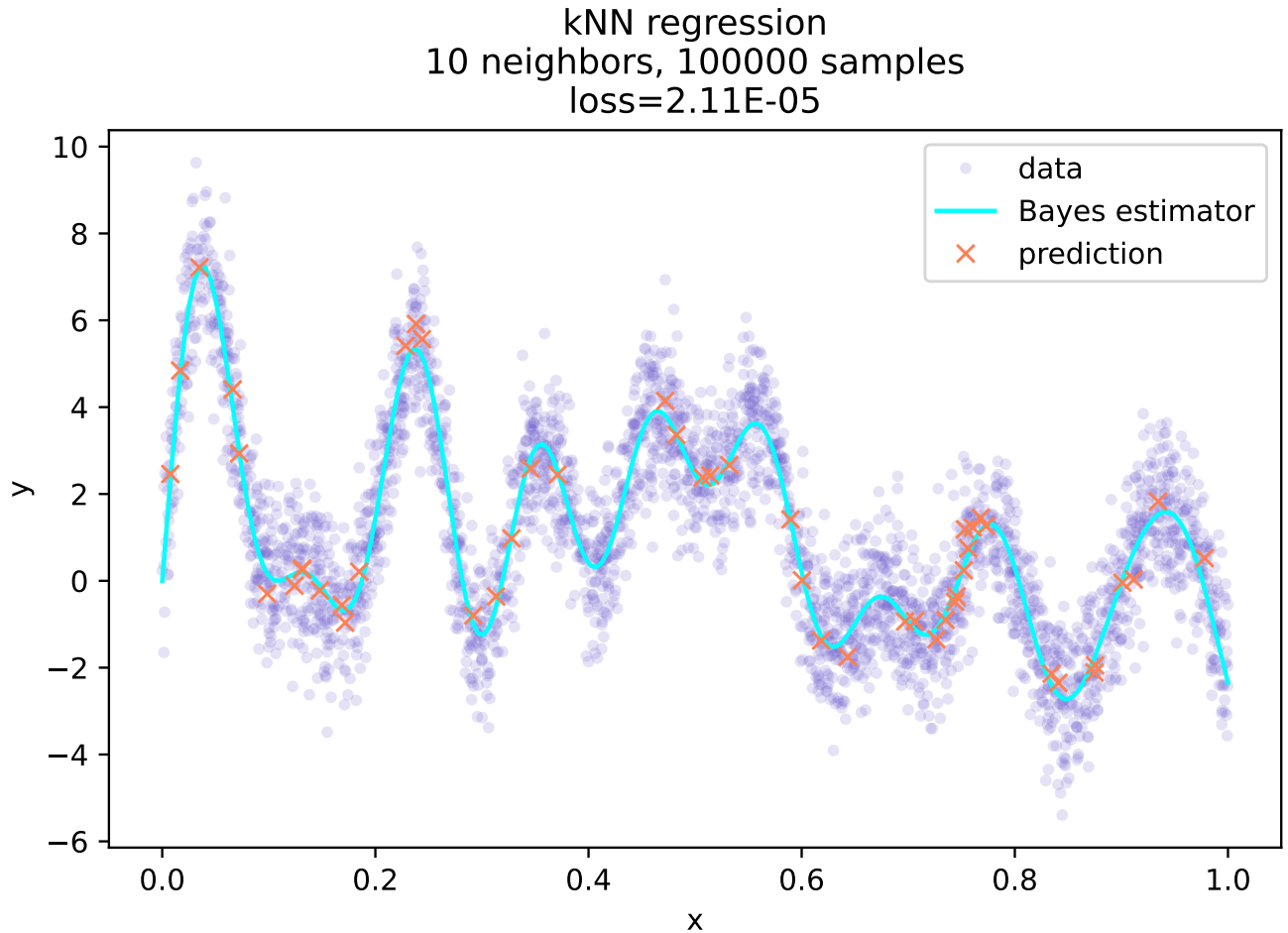
Nearest neighbors



Nearest neighbors



Nearest neighbors



Nearest neighbors

Given $k \geq 1$, and a metric d on $\mathcal{X}$, average the predictions of the k nearest neighbors (for regression) or take the majority vote (for classification).

Exercice 1 : What is $\hat{w}_i(x)$?

Nearest neighbors

$$\hat{w}_i(x) : \begin{cases} 1/k \text{ if i is in the closest neighbors} \\ 0 \text{ otherwise} \end{cases}$$

Nearest neighbors

k is a hyperparameter, hence it must be tuned, for instance with cross validation.

- too small k : underfitting
- too large k : overfitting

Nearest neighbors search

The search for nearest neighbors is a problem itsself !

https://scikit-learn.org/stable/modules/neighbors.html

https://en.wikipedia.org/wiki/K-d_tree

https://en.wikipedia.org/wiki/Ball_tree

Partition estimators

$\mathcal{X} = \cup_{j \in J} A_j$.

A_1	A_2	A_3	A_4	A_5
A_6	A_7	A_8	A_9	A_{10}
A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{16}	A_{17}	A_{18}	A_{19}	A_{20}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}

For each x, average the predictions of the samples that are in the same A_j as x. We can note it $A(x)$.

Partition estimators

$\mathcal{X} = \cup_{j \in J} A_j$.

A_1	A_2	A_3	A_4	A_5
A_6	A_7	A_8	A_9	A_{10}
A_{11}	A_{12}	A_{13}	A_{14}	A_{15}
A_{16}	A_{17}	A_{18}	A_{19}	A_{20}
A_{21}	A_{22}	A_{23}	A_{24}	A_{25}

For each x, average the predictions of the samples that are in the same A_j as x. We can note it $A(x)$.
Exercice 2 : What is $\hat{w}_i(x)$?

Partition estimator

$$\hat{w}_i(x) = \frac{1_{x_i \in A(x)}}{\sum_{k=1}^{n} 1_{x_k \in A(x)}} \tag{19}$$

Partition estimator

Exercice 3 : We have seen in previous classes one example of partition estimator. What is it ?

Kernel regression (Nadaraya-Watson)

We consider a non-negative kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ and

$$\hat{w}_i(x) = \frac{k(x, x_i)}{\sum_{i=1}^{n} k(x, x_i)} \tag{20}$$

Non-negative kenrels

Often

$$k(x, x') = \frac{1}{h^d} q(\frac{x - x'}{h}) \tag{21}$$

with d the dimension, h a bandwidth parameter.

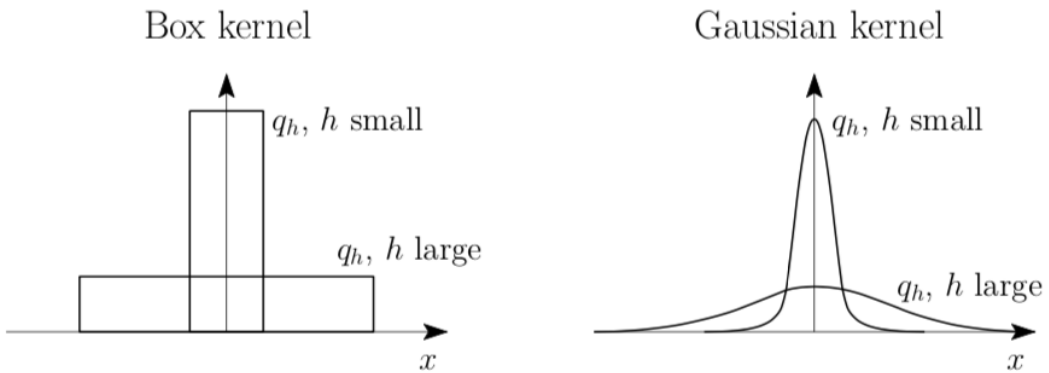


Image from [Bach, 2021].

Non-negative kenrels

$$k(x, x') = \frac{1}{h^d} q(\frac{x - x'}{h}) \tag{22}$$

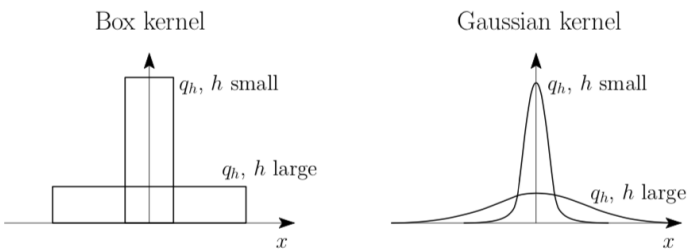


Image from [Bach, 2021].

- Box kernel : $q(x) = 1_{||x|| \leq 1}$
- Gaussian kernel : $q(x) = e^{-\frac{||x||^2}{2}}$

Remark

These kernels are not exactly the same as the ones we mentioned earlier (positive-definite kernels).
These kernels are more simply non-negative (less specific).
Estimator :

$$\tilde{f(x)} = \frac{\sum_{i=1}^{n} k(x, x_i) y_i}{\sum_{i=1}^{n} k(x, x_i)} \tag{23}$$

Curse of dimensionality

It is posible to show, that under some simple regularity assumptions on the target, the convergence rate of the error of these estimators, as a function of n, is $\mathcal{O}(n^{-\frac{2}{d+2}})$, where d is the underlying dimension.
In order to have an error smaller than ϵ, we need to have

$$n \geq \left(\frac{1}{\epsilon}\right)^{\frac{d+2}{2}} \tag{24}$$

- It is not easy to exploit a higher regularity of the target function (**no adaptivity to the regularity**)
- It is not possible to learn with these methods in high dimension.

Kernel density estimation

It is possible to use similar ideas to perform Kernel density estimation (KDE). (Again, here it is juste a non-negative kernel)
https://francisbach.com/cursed-kernels/
https://seaborn.pydata.org/generated/seaborn.jointplot.html
https://fr.wikipedia.org/wiki/Estimation_par_noyau
https://en.wikipedia.org/wiki/Kernel_density_estimation

Metrics

We have mostly considered metrics on vector spaces (e.d. euclidean distance).
We have also mentioned similarities, that are slightly more general than distances (e.g. for graphs). Example : gaussian similarity, derived from a distance.

Categorical data

Categorical data (e.g. names, nationality) are sometimes encountered in machine learning problems.
They need to be encoded in a numerical way, in order to be used by an algorithm.

Categorical data : one-hot encoding

Categorical data (e.g. names, nationality) are sometimes encountered in machine learning problems.
They need to be encoded in a numerical way, in order to be used by an algorithm.
Most of the time, assigning an integer to a category might not be a good idea, as it introduces an artificial information in the dataset (through the induced rankings).
Instead, **one-hot encoding** is often used.

Texts

We introduce the **cosine similarity** that allows to compare texts inside a corpus (**bag of words representation**).

- Text A represented by the vector u_A
- Text B represented by the vector u_B

$$S_C(\text{text A}, \text{text B}) = \frac{(u_A|u_B)}{||u_A||||u_B||} \tag{25}$$

Demo.

Example

- If $d >> n$ and we want to learn a linear model $x \mapsto \langle \theta, x \rangle$, we have seen that this raises statistical issues (high variance, overfitting).
- However, if we know in advance that θ only has $s < d$ non-zero coordinates (sparse θ), we can reformulate to an easier problem.
- But most of the time this is not the case, s is not known, so we need to test several subsets of non-zero coordinates.

Example

We could write the following regularized optimization problem

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^d}{\arg\min} \left(||Y - X\theta|| + \lambda ||\theta||_0 \right) \tag{26}$$

- $y \in \mathbb{R}^n$ (labels)
- $X \in \mathbb{R}^{n,d}$ (design matrix)
- $||\theta||_0$: number of non-zero components of θ

Example

We could write the following regularized optimization problem

$$\hat{\theta} = \underset{\theta \in \mathbb{R}^d}{\arg\min} \left(||Y - X\theta|| + \lambda ||\theta||_0 \right) \tag{27}$$

- $y \in \mathbb{R}^n$ (labels)
- $X \in \mathbb{R}^{n,d}$ (design matrix)
- $||\theta||_0$: number of non-zero components of θ

However,

- optimization issue (not convex)
- computationally prohibitive to test all subsets of $[1, d]$.

Lasso

Le Lasso replaces $||\theta||_0$ by $||\theta||_1$.

$$||\theta_1|| = \sum_{i=1}^{d} |\theta_i| \tag{28}$$

Lasso estimator :

$$\tilde{\theta}_\lambda \in \underset{\theta \in \mathbb{R}^d}{\arg\min} \{||Y - X\theta||^2 + \lambda ||\theta||_1\} \tag{29}$$

For subtle reasons, the optimization with the lasso leads to sparser solutions.

Lasso

Lasso estimator :

$$\tilde{\theta}_\lambda \in \underset{\theta \in \mathbb{R}^d}{\arg\min} \{ ||Y - X\theta||^2 + \lambda ||\theta||_1 \} \tag{30}$$

For subtle reasons, the optimization with the lasso leads to sparser solutions. Frequently used optimization algorithm :

- coordinate descent (algorithm used in scikit)
- Fista
- LARS

`https://en.wikipedia.org/wiki/Coordinate_descent`

Lasso regularization path

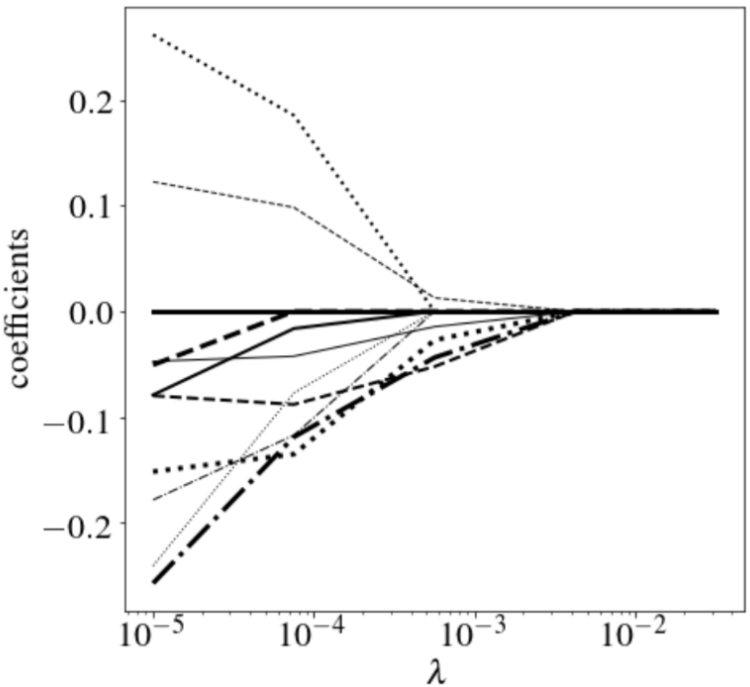


Figure – Regularization path with a Lasso optimization of a problem with $d = 12$.

Each line represents the evolution of a θ_i when λ increases. Image from [Azencott, 2022].

Ridge regularization path

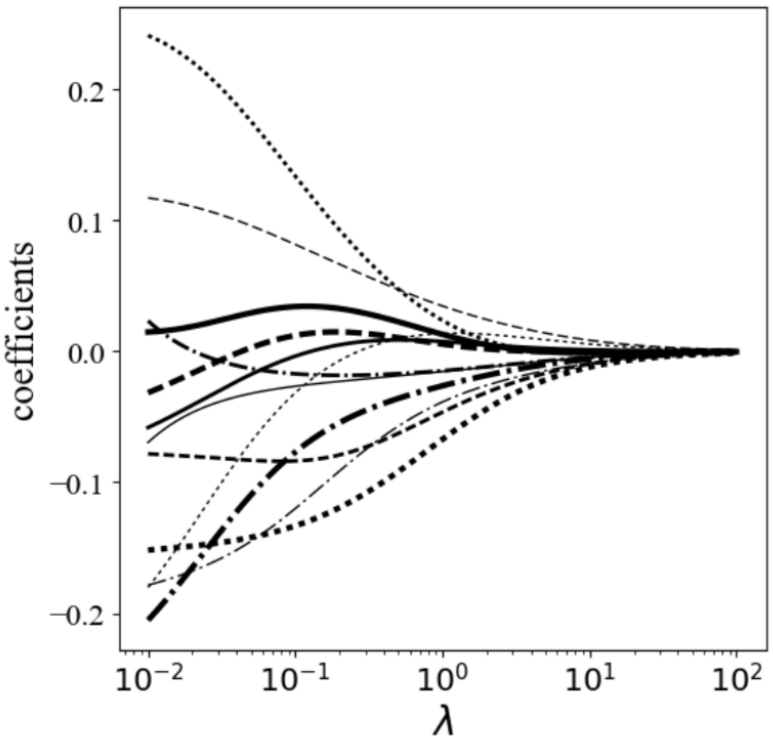


Figure – Regularization path with a Ridge optimization of a problem with $d = 12$.

Each line represents the evolution of a θ_i when λ increases. Image

Elastic net

Combination of $L1$ and $L2$ regularization.
Elastic-net estimator :

$$\tilde{\theta}_\lambda \in \arg\min_{\theta \in \mathbb{R}^d} \{||Y - X\theta||^2 + \lambda_1 ||\theta||_1 + \lambda_2 ||\theta||_2\} \qquad (31)$$

To choose λ_1 and λ_2 : cross validation.

No free lunch theorems

$\mathcal{A}$: learning rule. Takes the dataset D_n as input and outputs an estimator $\tilde{f}_n$ (for instance based on empirical risk minimization, local averaging, etc).
There are several no free lunch theorems.

No free lunch theorems

Theorem

No free lunch - fixed n
We consider a binary classification task with "0-1"-loss, and $\mathcal{X}$ infinite.
We note $\mathcal{P}$ the set of all probability distributions on $\mathcal{X} \times \{0, 1\}$.
For any $n > 0$ and any learning rule $\mathcal{A}$

$$\sup_{dp \in \mathcal{P}} E\Big[R_{dp}\big(\mathcal{A}(D_n(dp))\big)\Big] - R^*_{dp} \geq \frac{1}{2} \tag{32}$$

We write $D_n(dp)$ in order to emphasize that the dataset is sampled randomly from the distribution dp.

No free lunch

- For any learning rule, there exists a distribution for which this learning rule performs badly.
- No method is universal and can have a good convergence rate on all problems.

However, considering **all** problems is probably not relevant for machine learning.

Adaptivity

If the learning rule improves (faster convergence rate) when we add a property on the problem (for instance, regularity of the target function), we say that we have adaptivity to this property.
For instance : gradient descent is adaptive to the strong convexity of the target function, since with a proper choice of the learning rate γ, the convergence rate is exponential, with a rate that involves the strong convexity constant μ.
There are several forms of adaptivity.

Most general case

The target is just Lipshitz-continuous, no extra-hypothesis. In this case the optimal rate is of the form $\mathcal{O}(n^{-\frac{1}{d}})$ (curse of dimensionality) for all learning rules.

Adaptivity to the input space

If the input data lie on a submanifold (e.g. a subspace) of $\mathbb{R}^d$ of lower dimension than d, most methods adapt to this property.

Adaptivity to the regularity of the target function

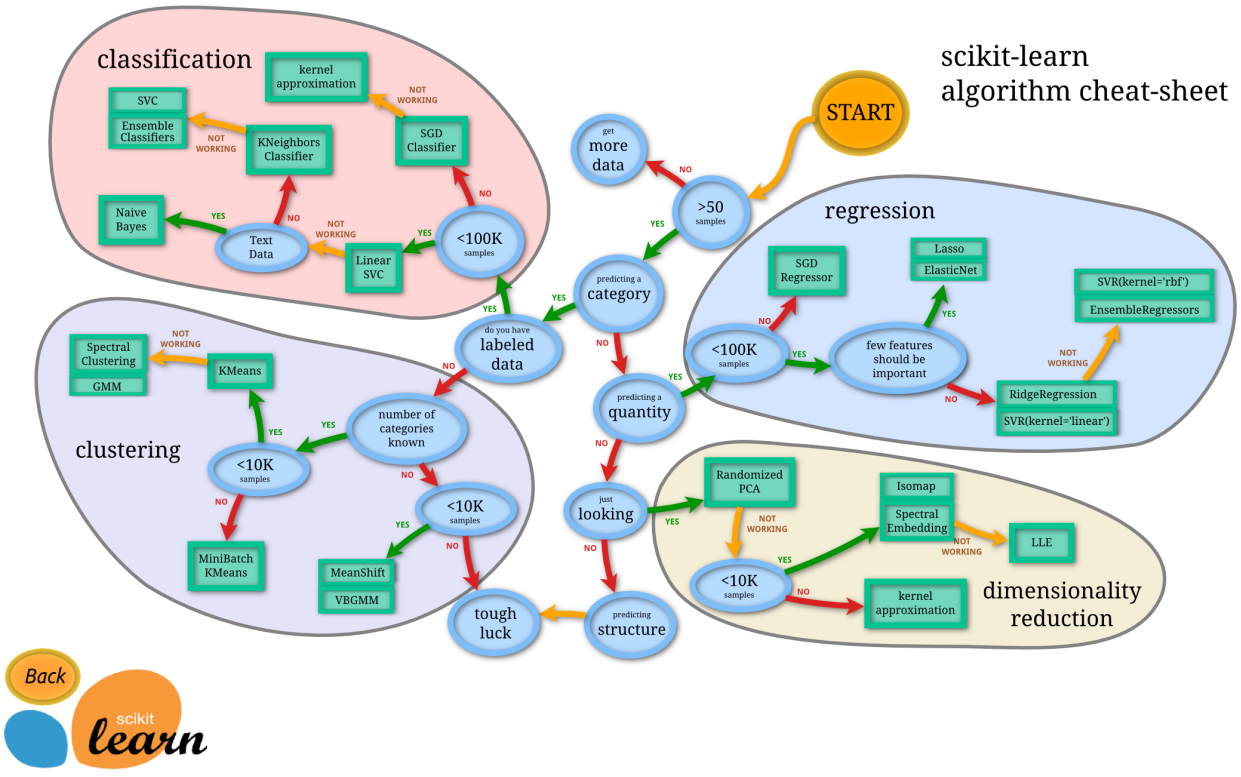
If the target is smoother (meaning that all derivatives up to order m are bounded), kernel methods (here, positive-definite kernels) and neural network adapt, if well optimized and regularized.
The rate can become $\mathcal{O}(n^{-\frac{m}{d}})$.

Adaptivity to latent variables

If the target function depends only on a k dimensional linear projection of the data, neural networks adapt, if well optimized. The rate can become $O(n^{-\frac{m}{k}})$.

`https://francisbach.com/quest-for-adaptivity/`

ML map



https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

References I


Azencott, C.-A. (2022).
Introduction au Machine Learning - 2e éd.


Bach, F. (2021).
Learning Theory from First Principles Draft.
Book Draft, page 229.