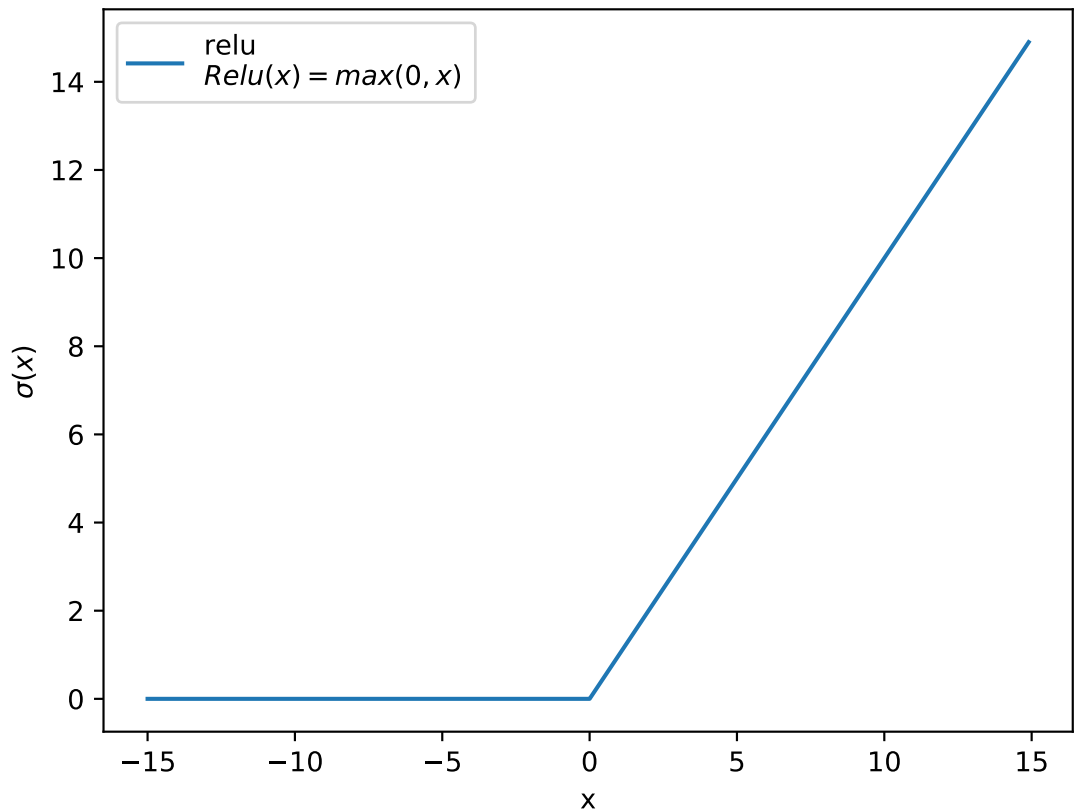
Fondamentaux théoriques du machine learning



Overview of lecture 7

Optimization of neural networks

- Difficulties of optimizing neural networks
- Specific methods for neural networks

Statistical learning

- Bounding the estimation error
- Interpolation regime and double descent

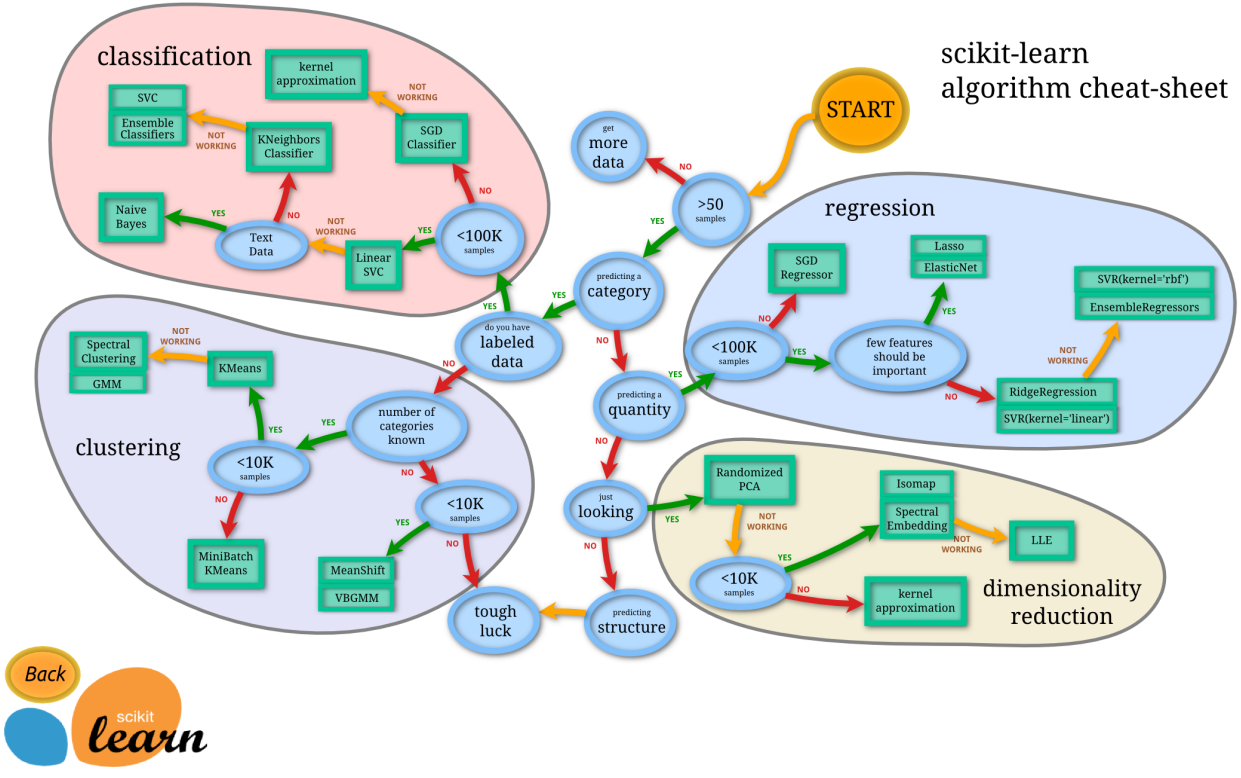
Local averaging methods

- Supervised learning
- Density estimation

Adaptivity

- No free lunch theorems
- Adaptivity

ML map



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

Neural networks

References / tools :

- https://www.deeplearningbook.org/
- https://d2l.ai/
- https://mlelarge.github.io/dataflowr-web/dldiy_ens.html
- https://playground.tensorflow.org/
- http://www.jzliu.net/blog/simple-python-library-visualize-neural-network/

Learning representations / features

- $\mathcal{X} = \mathbb{R}^d$.
- $\mathcal{Y} = \mathbb{R}$.

A neural network with scalar output learns a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^m$ **and** a linear regressor or classifier, $\theta \in \mathbb{R}^m$.

$$\forall x, f(x) = \langle \theta, \phi(x) \rangle \tag{1}$$

We can add a bias by adding a dimension to θ and adding a component with a 1 to each $\phi(x)$.

Learning representations / features

- $\mathcal{X} = \mathbb{R}^d$.
- $\mathcal{Y} = \mathbb{R}$.

A neural network learns a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^m$ **and** a linear regressor or classifier, $\theta \in \mathbb{R}^m$.

$$\forall x, f(x) = \langle \theta, \phi(x) \rangle \tag{2}$$

We can add a bias by adding a dimension to θ and adding a component with a 1 to each $\phi(x)$.
Remark : kernel methods use hardcoded features ϕ, that can be **implicit** (kernel trick).

Multi output

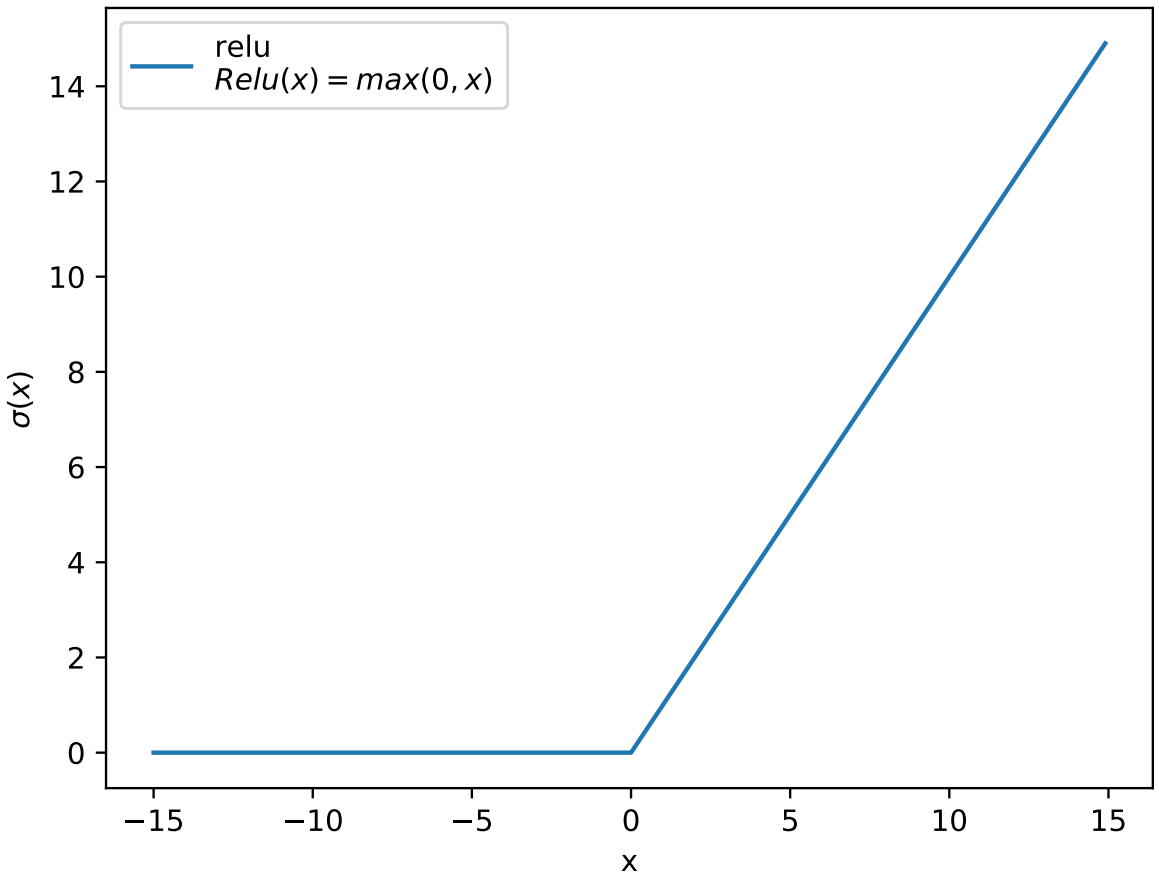
In order to learn a multidimensional output of dimension p ($\mathcal{Y} = \mathbb{R}^p$) , it is sufficient to lean a matrix $\theta \in \mathbb{R}^{m,p}$.

Single layer neural network

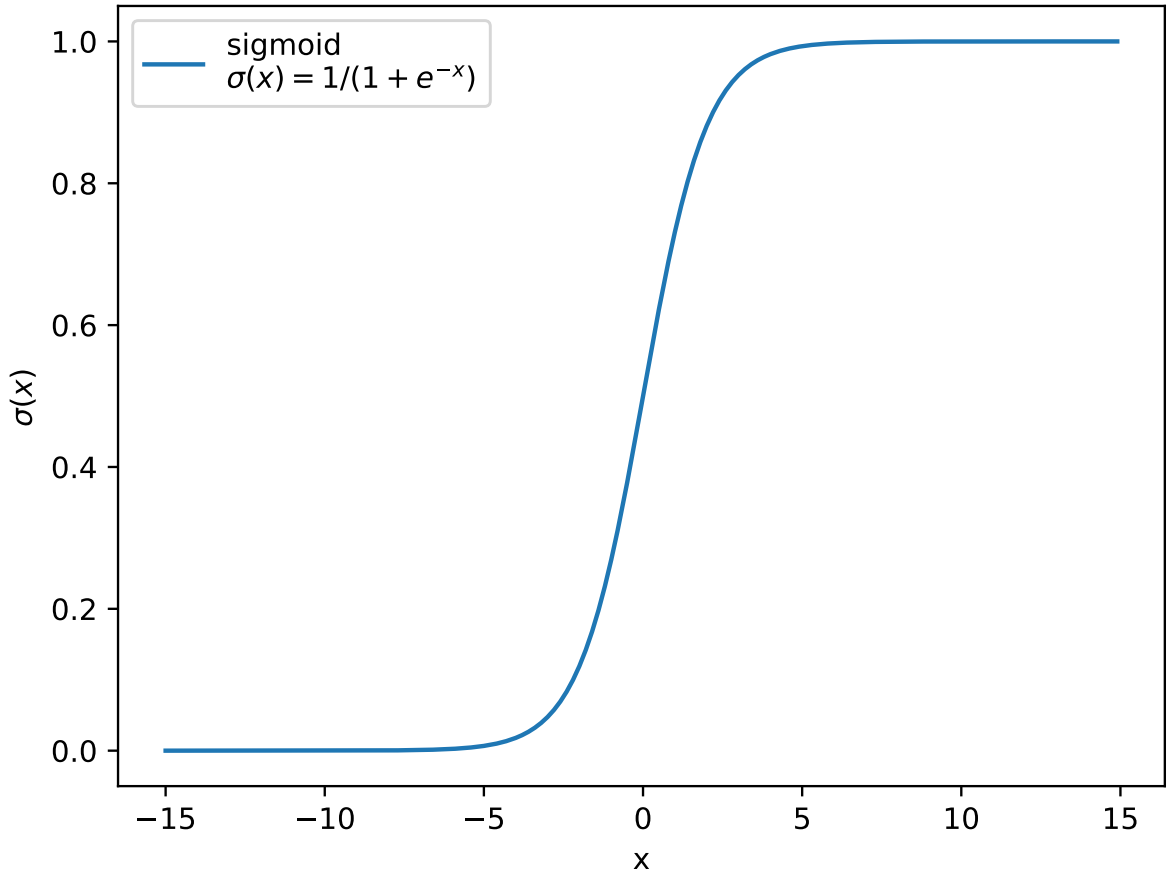
$$f(x) = \sum_{j=1}^{m} \theta_j \sigma(w_j^T x + b_j) \tag{3}$$

σ is an activation function (sigmoid, tanh, ReLu, etc).

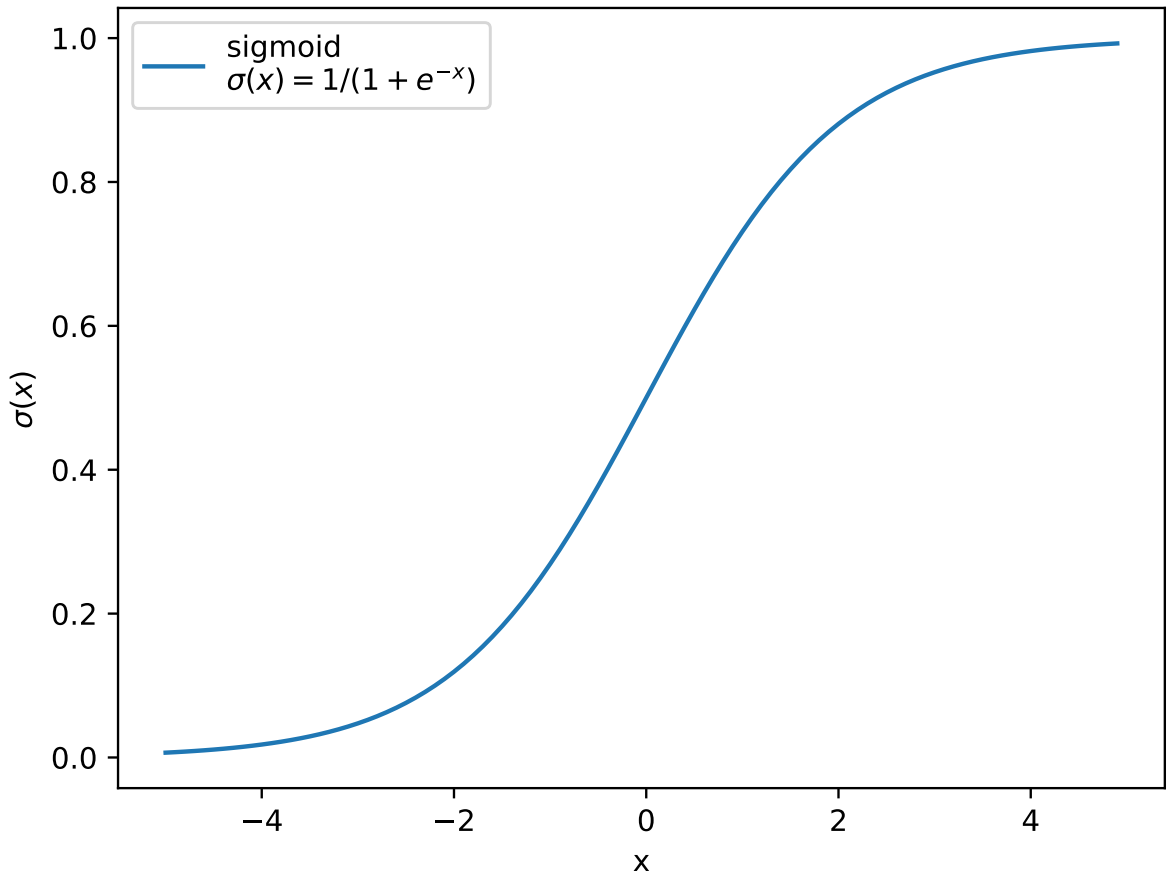
ReLU



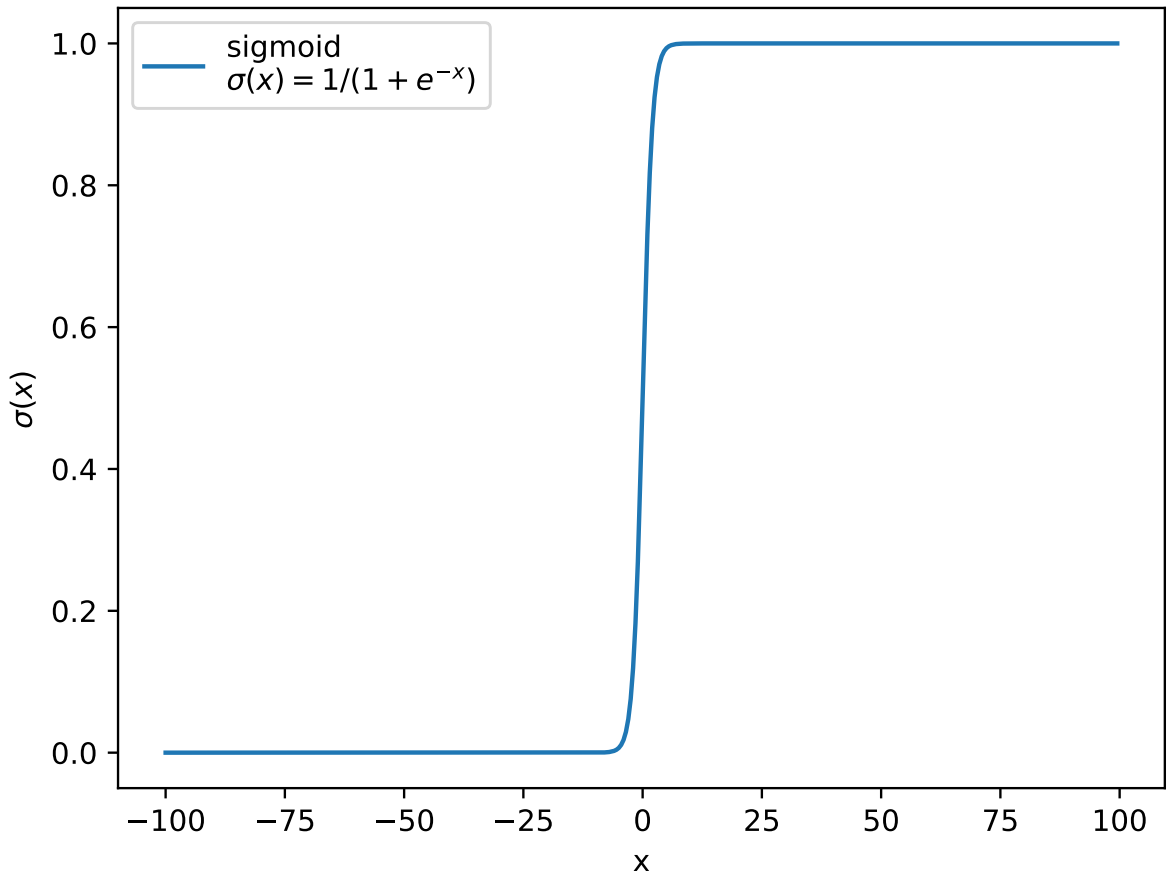
Sigmoid



Sigmoid



Sigmoid


sigmoid
$\sigma(x) = 1/(1 + e^{-x})$
1.0
0.8
0.6
0.4
0.2
0.0
$\sigma(x)$
−100
−75
−50
−25
0
25
50
75
100
x

Automatic differentiation

When working with neural networks, most used libraries implement automatic differentiation.

- tensorflow
- pytorch (autograd)

Optimizing neural networks

Optimizing neural networks comes with specific difficulties.

- the problem is non-convex
- there is often a large number of parameters (optimization in a high dimensional space)
- specific problems due to depth (vanishing gradients, see below)

`http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf`

Non-convexity

We know that

- If f is increasing and convex and g is convex, then $f \circ g$ is convex.
- Is f in convex and g is linear, then $f \circ g$ is convex.

With neural networks, we are in neither of these cases, as the activations σ are non linear.

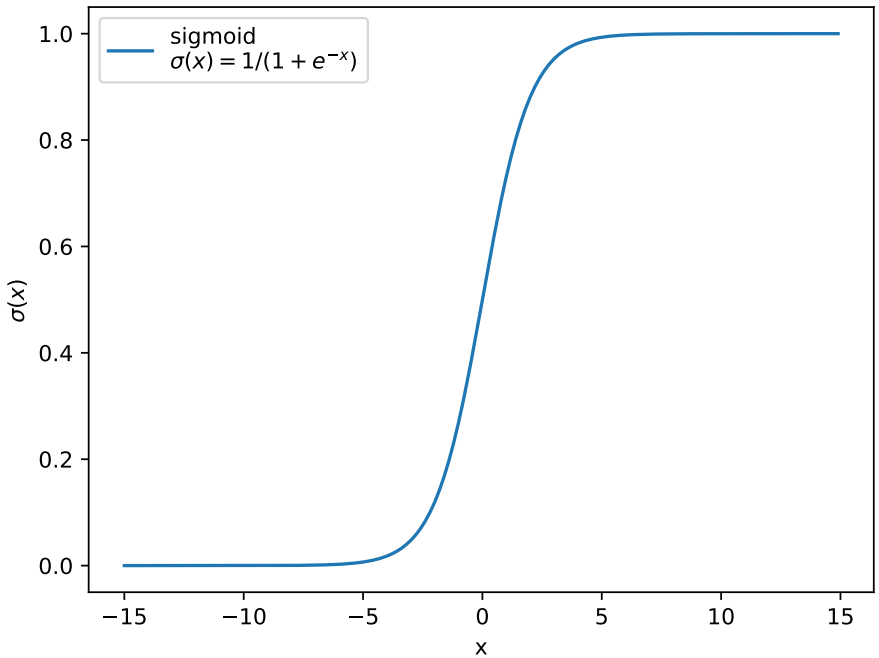
Non-convexity

We know that

- If f is increasing and convex and g is convex, then $f \circ g$ is convex.
- Is f in convex and g is linear, then $f \circ g$ is convex.

With neural networks, we are in neither of these cases, as the activations σ are non linear.
Hence **the objective function is non-convex**, and it remains difficult to understand why gradient based methods often perform well in practice

Vanishing gradients



Exercice 1 :

What is the maximum value of $|\sigma'(z)|$?

Exponentially decreasing gradients

- At each layer, the gradients are multiplied by a term of the form $\sigma'(u)$. Using a large number of layers leads to gradient norms that decrease rapidly when we move away from the output layer.
- This slows training down and caused deep learning to plateau for some years.
- Several initializations were necessary in order to obtain convergence, the result was unstable.

ReLU

The usage of ReLU solved this problem.
Other activation functions :
https://dashee87.github.io/deep%20learning/visualising-activation-functions-in-neural-networks/

SGD variants for neural networks

Several specific variations of SGD are commonly used for deep learning.
`https://pytorch.org/docs/stable/optim.html`

Specific methods for deep learning

Architectures :

- Convotutional networks
- Residual neural network (ResNet)

Optimization / regularization :

- dropout
- batch normalisation

Statistical learning

- We come back to the statistical analysis of supervised learning.
- More precisely, to that of empirical risk minimization.

Reminder on risks

Let l be a loss. Generalization error :

$$R(f) = E_{(X,Y)\sim\rho}[l(Y, f(X))] \tag{4}$$

The **empirical risk (ER)** of an estimator f writes

$$R_n(f) = \frac{1}{n}\sum_{i=1}^{n} l(y_i, f(x_i)) \tag{5}$$

Remember that the risks depends on the loss l.

Risk decomposition

- f^* : Bayes predictor
- F : Hypothesis space
- f_n : estimated predictor (hence in F).

$$E\Big[R(f_n)\Big] - R^* = \Big(E\Big[R(f_n)\Big] - \inf_{f \in F} R(f)\Big) + \Big(\inf_{f \in F} R(f) - R^*\Big) \quad (6)$$

Risk decomposition

- f^* : Bayes predictor
- F : Hypothesis space
- f_n : estimated predictor.

When doing empirical risk minimization, f_n is obtained by minimization of the empirical risk.
However :

- we have seen that in many cases, finding the exact minimizer of the empirical risk might be computationnaly hard.
- also, a natural question is whether it is sufficient to have an **approximate minimizer** of the empirical risk, as the empirical risk is an **approximation** of the generalization error.

Risk decomposition

Estimation error (variance term, fluctuation error, stochastic error) : depends on D_n, F, f_n.

$$E\Big[R(f_n)\Big] - \inf_{f \in F} R(f) \geq 0$$

Approximation error (bias term) : depends on f^* and F, not on f_n, D_n.

$$\inf_{f \in F} R(f) - R^* \geq 0$$

It is also possible to consider the **Optimization error** : depends on D_n, F, f_n.

$$E\Big[R(\hat{f}_n) - R(f_n)\Big] \tag{7}$$

where $\hat{f}_n$ is an approximate solution to the optimization problem.

Bound on the estimation error

We will now focus on the estimation error

$$E\Big[R(f_n)\Big] - \inf_{f \in F} R(f) \geq 0$$

f_n is the empirical risk minimizer
We consider the best estimator in hypothesis space

$$f_a = \arg\min_{h \in F} R(h)$$

We have seen that

$$R(f_n) - R(f_a) \leq 2 \sup_{h \in F} |R(h) - R_n(h)| \tag{8}$$

Deterministic bound on the estimation error

$$f_a = \arg\min_{h \in F} R(h) \tag{9}$$

$$f_n = \arg\min_{h \in F} R_n(h) \tag{10}$$

$$\begin{aligned} R(f_n) - R(f_a) = &\big(R(f_n) - R_n(f_n)\big) \\ &+ \big(R_n(f_n) - R_n(f_a)\big) \\ &+ \big(R_n(f_a) - R(f_a)\big) \end{aligned} \tag{11}$$

Deterministic bound on the estimation error

$$f_a = \arg\min_{h \in F} R(h)$$

$$f_n = \arg\min_{h \in F} R_n(h) \tag{12}$$

$$\begin{aligned} R(f_n) - R(f_a) = &\big(R(f_n) - R_n(f_n)\big) \\ &+ \big(R_n(f_n) - R_n(f_a)\big) \\ &+ \big(R_n(f_a) - R(f_a)\big) \end{aligned} \tag{13}$$

But by definition f_n minimizes R_n, so $\big(R_n(f_n) - R_n(f_a)\big) \leq 0$.

Deterministic bound on the estimation error

$$f_a = \arg\min_{h \in F} R(h)$$

$$f_n = \arg\min_{h \in F} R_n(h) \tag{14}$$

$$\begin{aligned} R(f_n) - R(f_a) = &\big(R(f_n) - R_n(f_n)\big) \\ &+ \big(R_n(f_n) - R_n(f_a)\big) \\ &+ \big(R_n(f_a) - R(f_a)\big) \end{aligned} \tag{15}$$

But by definition f_n minimizes R_n, so $\big(R_n(f_n) - R_n(f_a)\big) \leq 0$.
Finally :

$$R(f_n) - R(f_a) \leq 2 \sup_{h \in F} |R(h) - R_n(h)| \tag{16}$$

Bound on the estimation error

If we are able to bound $\sup_{h \in F} |R(h) - R_n(h)|$, then we have a bound on the estimation error.

Bound on the estimation error

Theorem

Weak law of large numbers
Let $(X_i)_{i\in\mathbb{N}}$ be a sequence of i.i.d. variables that have a moment of order 2. We note m their expected value. Then

$$\forall \epsilon > 0, \lim_{n\to+\infty} P\Big(|\frac{1}{n}\sum_{i=1}^{n} X_i - m| \geq \epsilon\Big) = 0 \tag{17}$$

We say that we have ***convergence in probability****.*

Bound on the estimation error

Theorem

Weak law of large numbers
Let $(X_i)_{i\in\mathbb{N}}$ be a sequence of i.i.d. variables that have a moment of order 2. We note m their expected value. Then

$$\forall \epsilon > 0, \lim_{n\to+\infty} P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - m\right| \geq \epsilon\right) = 0 \tag{18}$$

*We say that we have **convergence in probability**.*

However, this is only an asymptotical result : it is a limit for $n \to +\infty$.

Bound on the estimation error

If we are able to bound $\sup_{h \in F} |R(h) - R_n(h)|$, then we have a bound on the estimation error.
To do this, we will use some other mathematical results :

- Boole's inequality
- Hoeffding's inequality (non-asymptotical probabilistic bound)

Boole's inequality

Proposition

Let $A_1, A_2, \ldots,$ be acountable set of events of a probability space $\{\Omega, \mathcal{F}, P)$.
Then,

$$P\Big(\cup_{i \geq 1} A_i \Big) \leq \sum_{i \geq 1} P(A_i) \tag{19}$$

This set might be infinite.

Boole's inequality

Proposition

Let $A_1, A_2, \ldots,$ be acountable set of events of a probability space $\{\Omega, \mathcal{F}, P)$.
Then,

$$P\Big(\cup_{i\geq 1} A_i\Big) \leq \sum_{i\geq 1} P(A_i) \tag{20}$$

This set might be infinite.

Exercice 2 : Proove the proposition.

Hoeffding's inequality

Theorem

Hoeffding's inequality
Let $(X_i)_{1 \leq i \leq n}$ *be n i.i.d real random variables such that* $\forall i \in [1, n]$, $X_i \in [a, b]$ *and* $E(X_i) = \mu \in \mathbb{R}$. *Let* $\bar{\mu} = \frac{1}{n} \sum_{i=1}^{n} X_i$.
Then $\forall \epsilon > 0$,

$$P\Big(|\bar{\mu} - \mu| \geq \epsilon\Big) \leq 2 \exp\Big(-\frac{2n\epsilon^2}{(b-a)^2}\Big)$$

We admit this theorem.

Setting

- Supervised learning.
- Finite space of estimator F.
- The loss l is uniformly bounded : $l(\hat{y}, y) \in [a, b]$ with a and b real numbers.

Step 1

We have seen that

$$R(f_n) - R(f_a) \leq 2 \sup_{h \in F} |R(h) - R_n(h)| \tag{21}$$

As a consequence, for all $t \geq 0$:

$$P\Big(R(f_n) - R(f_a) \geq t\Big) \leq P\Big(2 \sup_{h \in F} |R(h) - R_n(h)| \geq t\Big) \tag{22}$$

Step 2

The fact that

$$2 \sup_{h \in F} |R(h) - R_n(h)| \geq t \tag{23}$$

is equivalent to :

$$\cup_{h \in F} \left(2|R(h) - R_n(h)| \geq t \right) \tag{24}$$

Step 3

Boole's inequality shows that :

$$P\Big(\cup_{h\in F} \Big(2|R(h) - R_n(h)| \geq t\Big)\Big) \leq \sum_{h\in F} P\big(2|R(h) - R_n(h)| \geq t\big) \tag{25}$$

Step 4

For each $h \in F$, we need to bound

$$P\big(2|R(h) - R_n(h)| \geq t\big) \tag{26}$$

Exercice 3 : What bound does Hoeffding's inequality give ?

Step 4

For each $h \in F$, we need to bound

$$P\big(2|R(h) - R_n(h)| \geq t\big) \tag{27}$$

With Hoeffding's inequality we get

$$P\big(2|R(h) - R_n(h)| \geq t\big) \leq 2\exp\Big(-\frac{nt^2}{2(b-a)^2}\Big) \tag{28}$$

Step 5

With Hoeffding's inequality we get

$$P\big(2|R(h) - R_n(h)| \geq t\big) \leq 2\exp\Big(-\frac{nt^2}{2(b-a)^2}\Big) \tag{29}$$

Finally, putting everything together :

$$\begin{aligned} P\Big(R(f_n) - R(f_a) \geq t\Big) &\leq \sum_{h\in F} P\big(2|R(h) - R_n(h)| \geq t\big) \\ &\leq \sum_{h\in F} 2\exp\Big(-\frac{nt^2}{2(b-a)^2}\Big) \qquad (30) \\ &= 2|F|\exp\Big(-\frac{nt^2}{2(b-a)^2}\Big) \end{aligned}$$

Conclusion

$$P\Big(R(f_n) - R(f_a) \geq t\Big) \leq 2|F| \exp\Big(-\frac{nt^2}{2(b-a)^2}\Big) \qquad (31)$$

Conclusion

We write

$$\delta = 2|F| \exp\left(- \frac{nt^2}{2(b-a)^2} \right) \tag{32}$$

We assume that $b - a = 1$. Then, with probability $1 - \delta$, we can compute and show that

$$R(f_n) \leq R(f_a) + 2\sqrt{\frac{\log(|F|) + \log(\frac{2}{\delta})}{2n}} \tag{33}$$

Generalization

It is possible to generalize to infinite sets :

- by sampling F
- by using Rademacher complexity and Vapnik Vapnik-Chervonenkis theory.

Interpolation regime

- If the number of parameters is sufficient, it is possible to have $R_n(f_n) = 0$.
- However, the statistical error does not seem to explode for some deep networks.

For instance, for Wide Resnet, $\frac{p}{n} = 179$ with

- p : number of parameters in the netwok
- n : number of samples

Double descent

[Belkin et al., 2019]

Local averaging methods

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

Setting

Let l be a loss. Generalization error :

$$R(f) = E_{(X,Y)\sim\rho}[l(Y, f(X))] \tag{34}$$

Bayes estimator :

$$f^* = \underset{f \text{ measurable}}{\arg\min} \; R(f) \tag{35}$$

The empirical risk is not considered.

Setting

Let l be a loss. Generalization error :

$$R(f) = E_{(X,Y)\sim\rho}[l(Y, f(X))] \tag{36}$$

Bayes estimator :

$$f^* = \underset{f \text{ measurable}}{\arg\min} \; R(f) \tag{37}$$

Bayes risk :

$$R^* = E_X\left[\inf_{y\in\mathcal{Y}} E_{Y\sim dP(Y|X)}\left[l(Y, y)|X\right]\right] \tag{38}$$

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

$$f^*(x) = E\big[Y|X = x\big] \tag{39}$$

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

$$f^*(x) = \int_{y\in\mathbb{R}} y p_{Y|X=x}(y)dy \tag{40}$$

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

$$f^*(x) = \int_{y\in\mathbb{R}} y dP(Y|X = x) \tag{41}$$

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

$$f^*(x) = \arg\max_{z \in \mathcal{Y}} P(Y = z | X = x) \tag{42}$$

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

- $D_n = \{(x_i, y_i), i \in [1, \ldots, n]\}$
- $x_i \in \mathcal{X}$
- $y_i \in \mathbb{R}$ 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

$$\tilde{f}(x) = \int_{y \in \mathbb{R}} y d\hat{P}(Y|X = x) \tag{43}$$

Local averaging : classification

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

$$\tilde{f}(x) = \arg\max_{z \in \mathcal{Y}} \hat{P}(Y = z | X = x) \tag{44}$$

Linear estimators

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

Linear estimators

$$\hat{dP}(Y|X=x) = \sum_{i=1}^{n} \hat{w}_i(x)\delta_{y_i}(y) \tag{45}$$

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

$$\hat{dP}(Y|X=x) = \sum_{i=1}^{n} \hat{w}_i(x)\delta_{y_i}(y) \tag{46}$$

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 :

$$\tilde{f}(x) = \sum_{i=1}^{n} \hat{w}_i(x) y_i \tag{47}$$

Linear estimators

Linear estimators

$$\hat{dP}(Y|X=x) = \sum_{i=1}^{n} \hat{w}_i(x)\delta_{y_i}(y) \tag{48}$$

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 :

$$\tilde{f}(x) = \arg\max_{j \in \{0,1\}} \sum_{i=1}^{n} \hat{w}_i(x) 1_{y_i=j} \tag{49}$$

Choice of the weights

Linear estimators

$$\hat{dP}(Y|X=x) = \sum_{i=1}^{n} \hat{w}_i(x)\delta_{y_i}(y) \tag{50}$$

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

$$d\hat{P}(Y|X=x) = \sum_{i=1}^{n} \hat{w}_i(x)\delta_{y_i}(y) \tag{51}$$

δ_{y_i} is the Dirac mass in y_i.

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

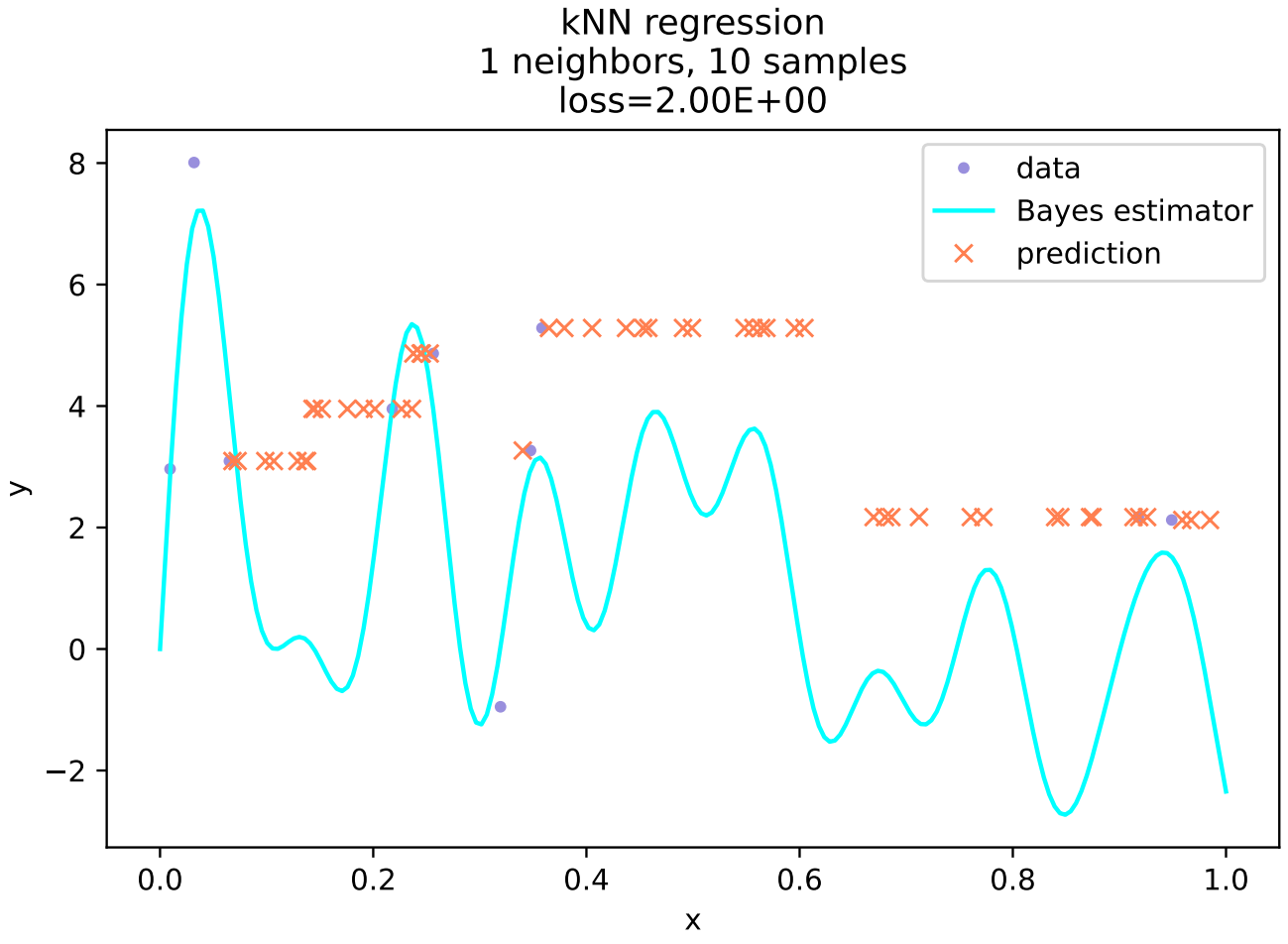
Three possibilities :

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

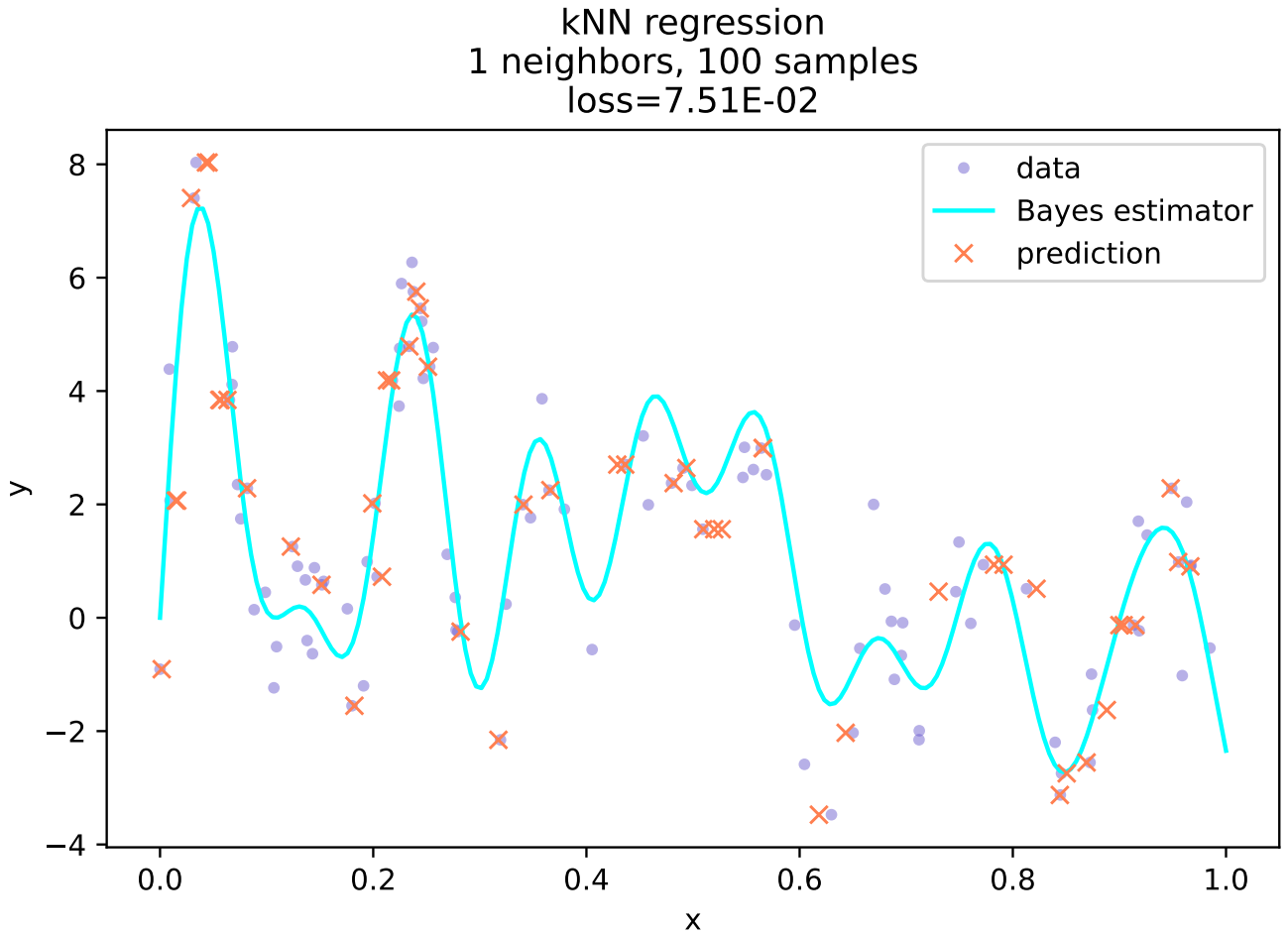
Nearest neighbors

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).

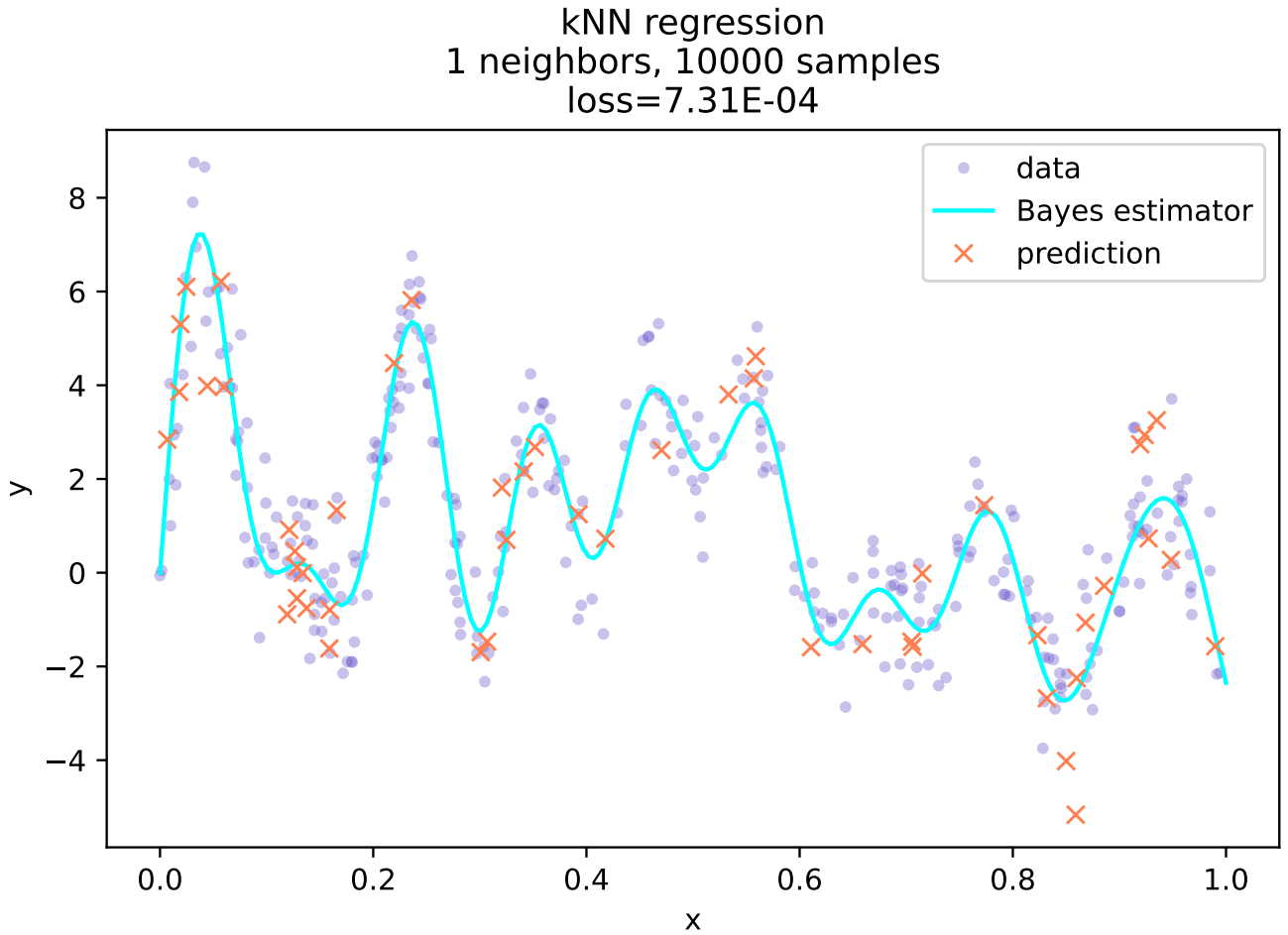
Nearest neighbors



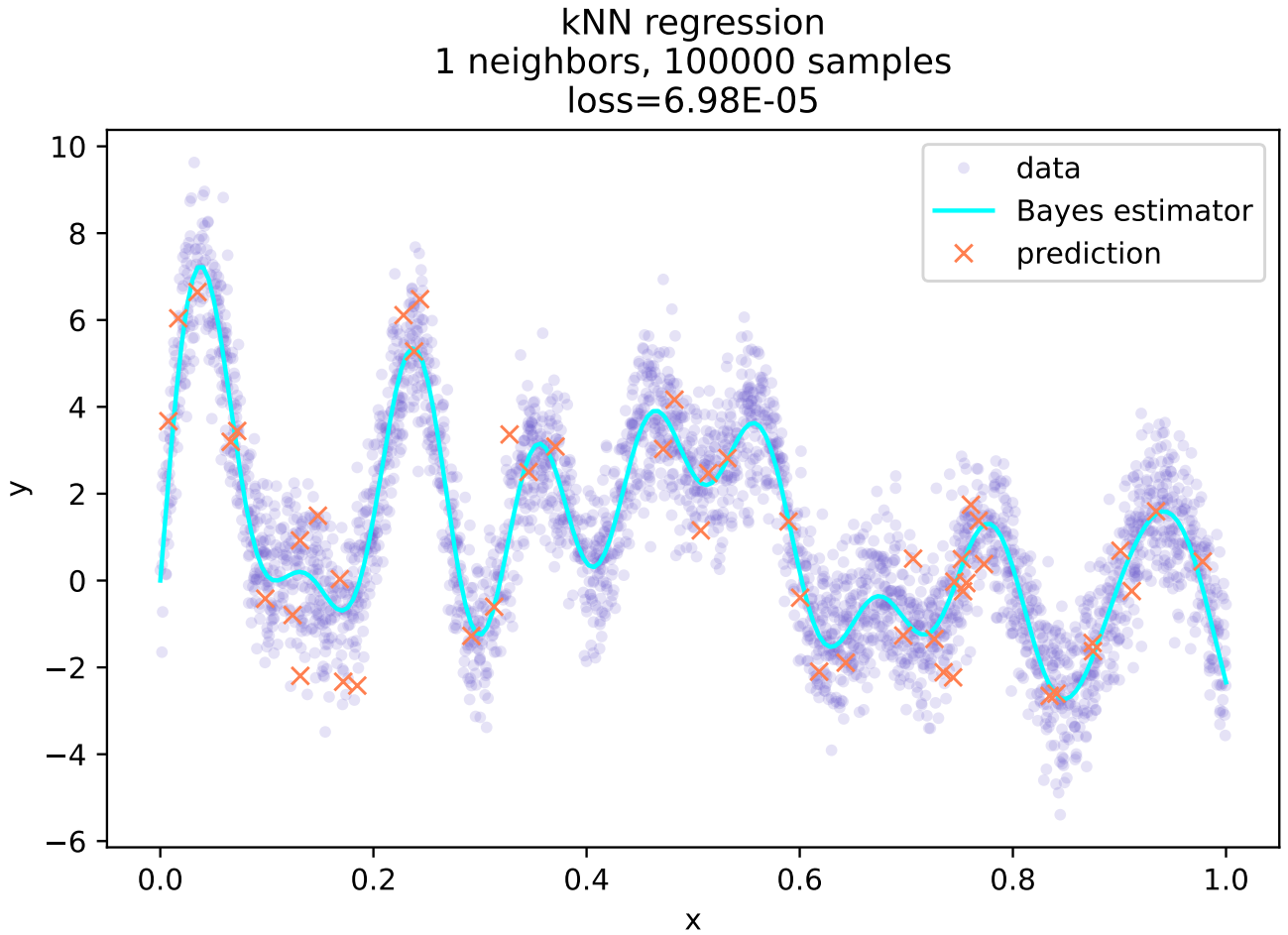
Nearest neighbors



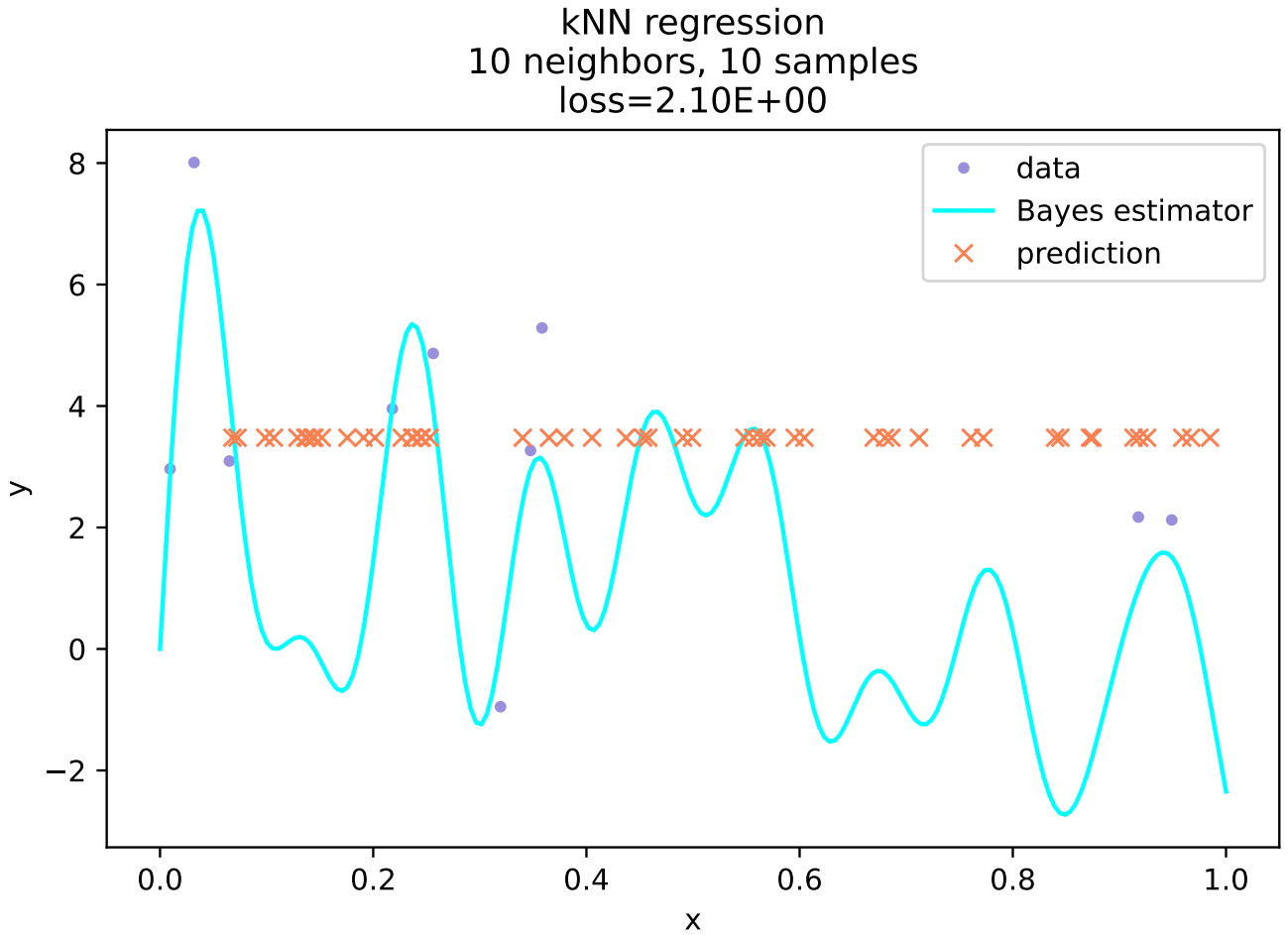
Nearest neighbors



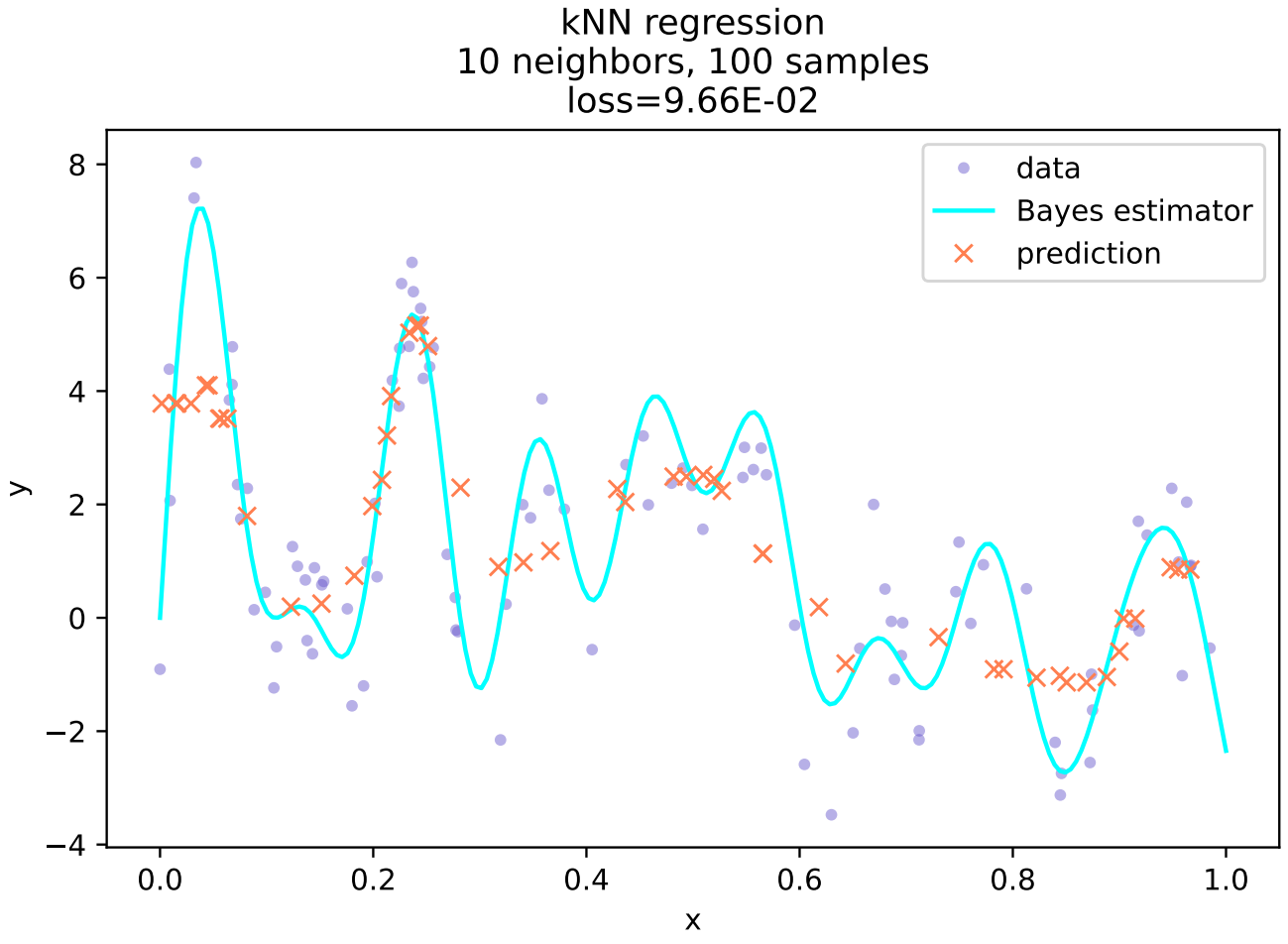
Nearest neighbors



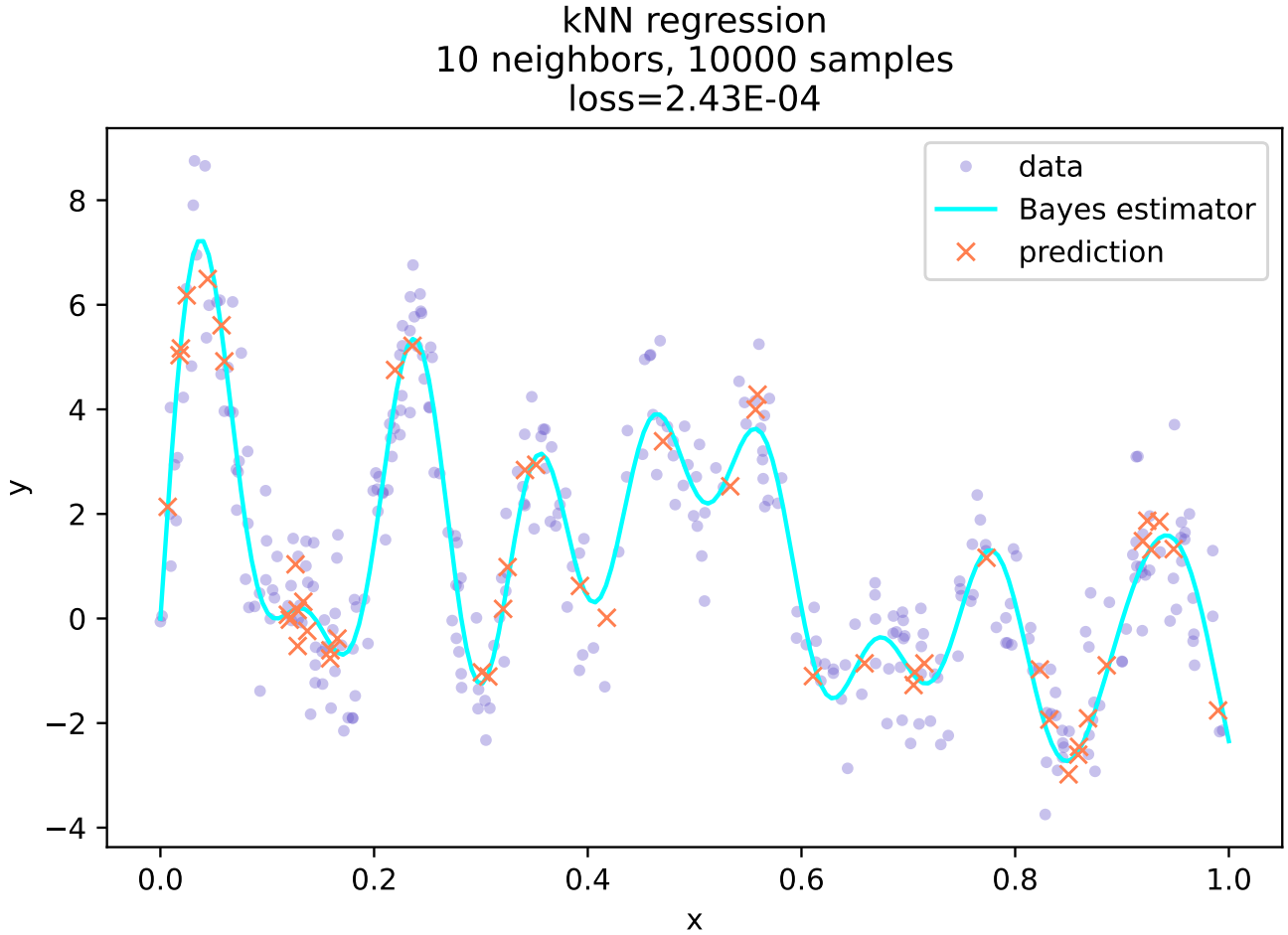
Nearest neighbors



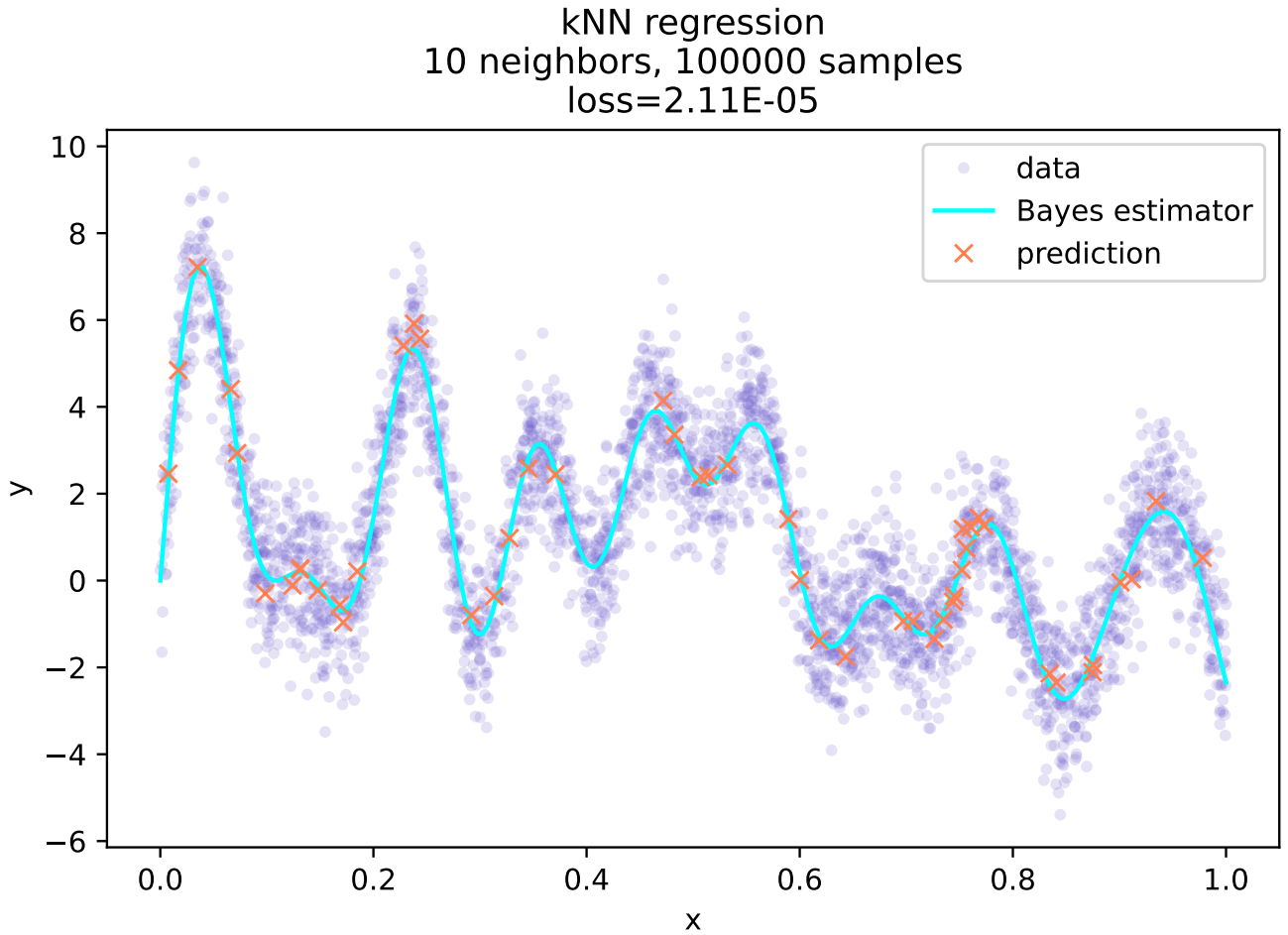
Nearest neighbors



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 4 : 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 5 : What is $\hat{w}_i(x)$?

Partition estimator

$$\hat{w}_i(x) = \frac{1_{x_i \in A(x)}}{\sum_{k=1}^{n} 1_{x_k \in A(x)}} \tag{52}$$

Partition estimator

Exercice 6 : 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

$$\hat{w}_i(x) = \frac{k(x, x_i)}{\sum_{i=1}^{n} k(x, x_i)} \tag{53}$$

Non-negative kenrels

Often

$$k(x, x') = \frac{1}{h^d} q(\frac{x - x'}{h}) \tag{54}$$

with d the dimension, h a bandwidth parameter.

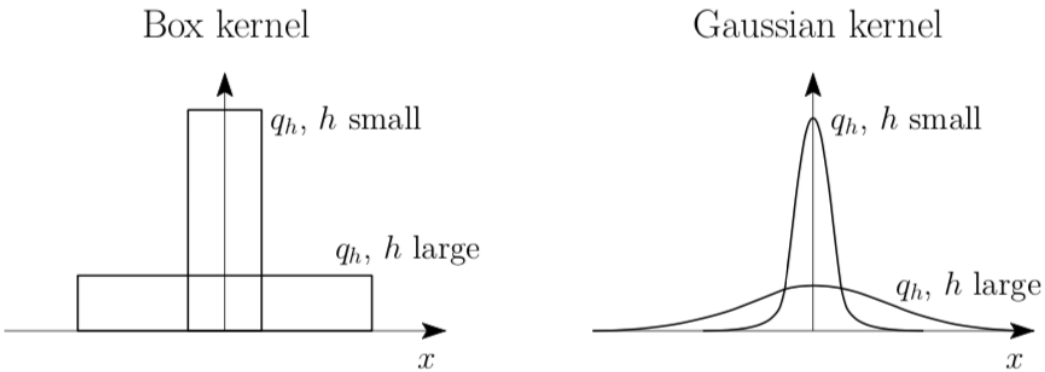


Image from [Bach, 2021].

Non-negative kenrels

$$k(x, x') = \frac{1}{h^d} q(\frac{x - x'}{h}) \tag{55}$$

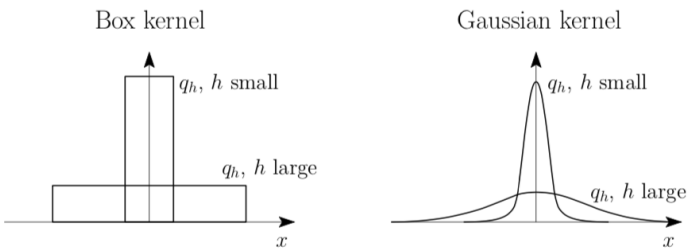


Image from [Bach, 2021].

- 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 :

$$\tilde{f(x)} = \frac{\sum_{i=1}^{n} k(x, x_i) y_i}{\sum_{i=1}^{n} k(x, x_i)} \tag{56}$$

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

$$n \geq \left(\frac{1}{\epsilon}\right)^{\frac{d+2}{2}} \tag{57}$$

- 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

No free lunch theorems

$\mathcal{A}$: learning rule. Takes the dataset D_n as input and outputs an estimator 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 $\mathcal{A}$

$$\sup_{dp \in \mathcal{P}} E\Big[R_{dp}\big(\mathcal{A}(D_n(dp))\big)\Big] - R^*_{dp} \geq \frac{1}{2} \tag{58}$$

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/`

References I


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


Belkin, M., Hsu, D., Ma, S., and Mandal, S. (2019).
Reconciling modern machine-learning practice and the classical bias–variance trade-off.
Proceedings of the National Academy of Sciences of the United States of America, 116(32) :15849–15854.