AI for healthcare

Lecture 2/4 – Flat vector spaces

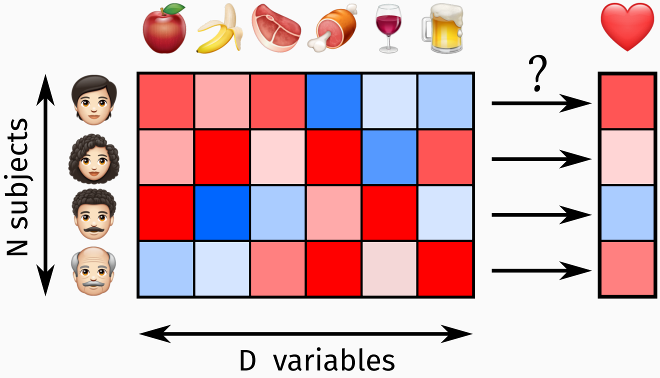
Jean Feydy
HeKA team, Inria Paris, Inserm, Université Paris-Cité

Thursday, 2pm–5pm – 4 lectures

Epita, rooms KB404 + SM15

Validation: team project + quizz

Remember this slide from lecture 1?



Supervised learning = **Regression**.

We look for a formula $F(x_1, \dots, x_D)$ of the D variables

that best approximates an important quantity (♡).

First thing you should do?

Working with **clients** < **colleagues** < **friends**.

Wake up: get out of the matrix!

Data **science** is never done in a vacuum.
Our (big) spreadsheets are **partial projections** of a complex reality.

What are we trying to achieve?
What type of information is available?
What do we already know?

To understand this **context**, you must break the ice with domain experts.

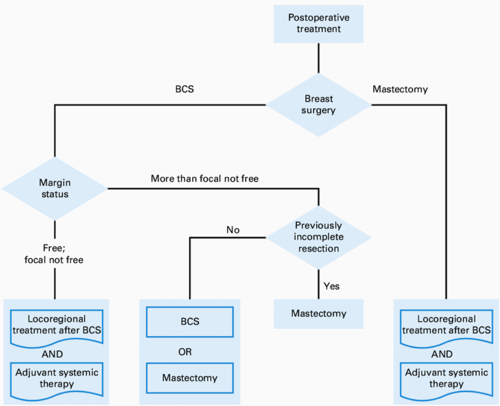
This is a **continuous**, **time-consuming** and **enjoyable** process.

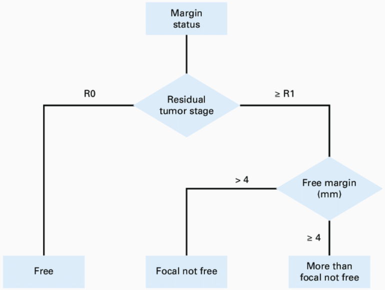
Today: well-rounded methods for high-quality features

1. **Decision trees – for heterogeneous data**
   - Greedy training and regularizations.
2. **K-Nearest Neighbors – a first isotropic method**
   - Euclidean metrics and normalization.
3. **Linear regression – to estimate global trends**
   - Linear, piecewise linear and polynomial regression.
4. **Kernel methods – specify a custom prior**
   - Smoothness, short- and long-range interactions.
   - Nadaraya–Watson–Shepard and Ridge regressions.

1. Decision trees

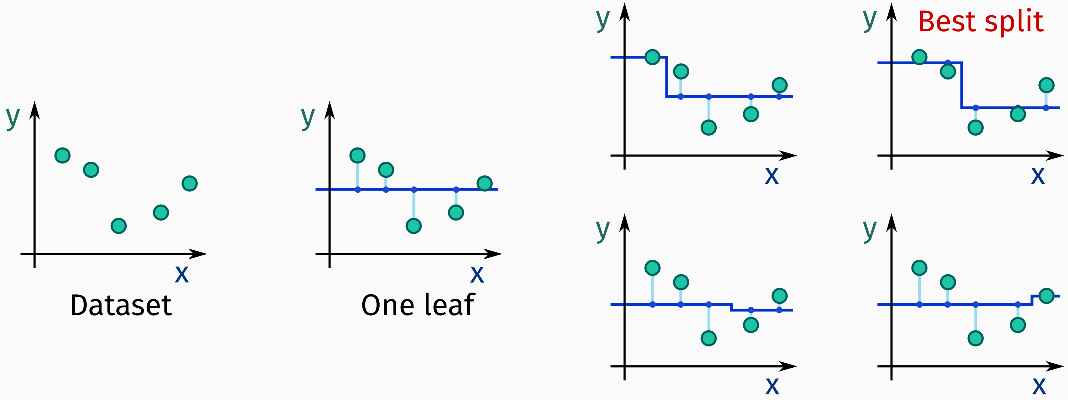
Expert knowledge is often distilled as a tree





Transformation of the national ***breast cancer guideline*** *into data-driven clinical decision trees*, Hendriks et al., 2019

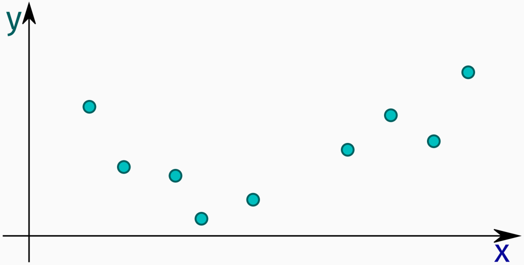
Tree models are easy to train with a greedy algorithm

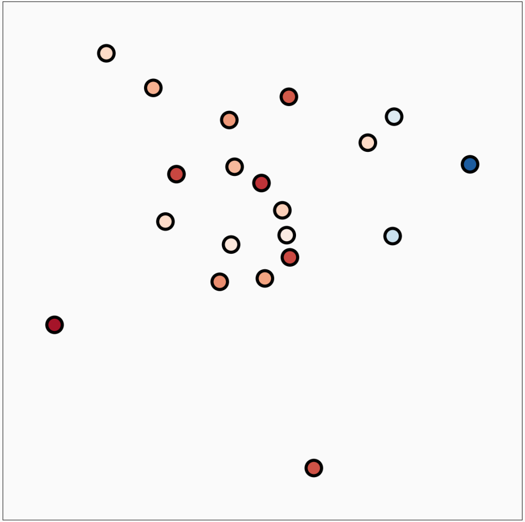


Recursive splits that stop if improvements < T $\iff$ **greedy** minimization of

$$\text{Fit}_{x,y}(F) + \text{Reg}(F) = \frac{1}{2}\sum_i \|F(x_i) - y_i\|^2 + T \cdot \#\text{Leaves}(F)\,.$$

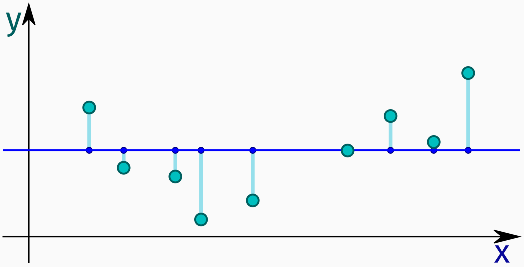
Two toy regression problems





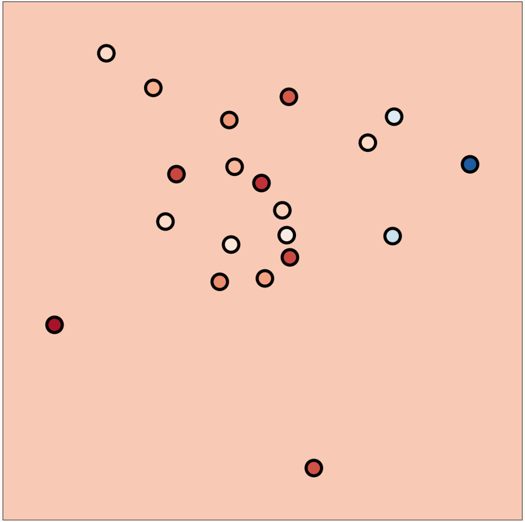
D=1 – 9 points **x** on the unit interval $[0, 1]$.
D=2 – 20 points **x** on the unit square $[0, 1]^2$.

In both cases: **scalar** output values **y**.

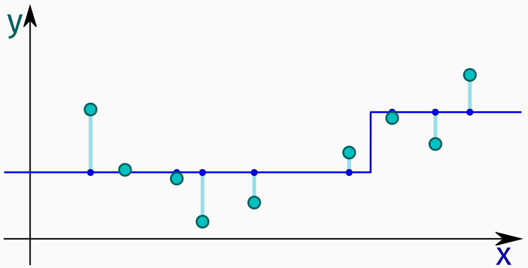


Depth 0:

1 constant value.



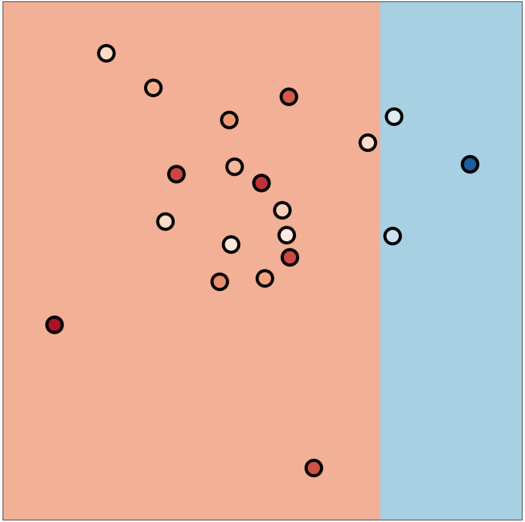
Decision trees



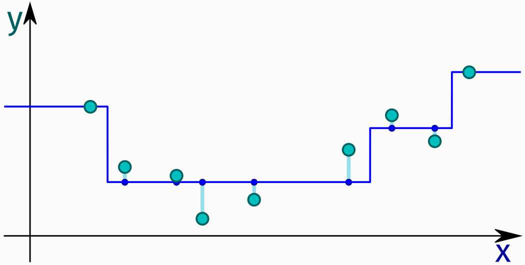
Depth 1:

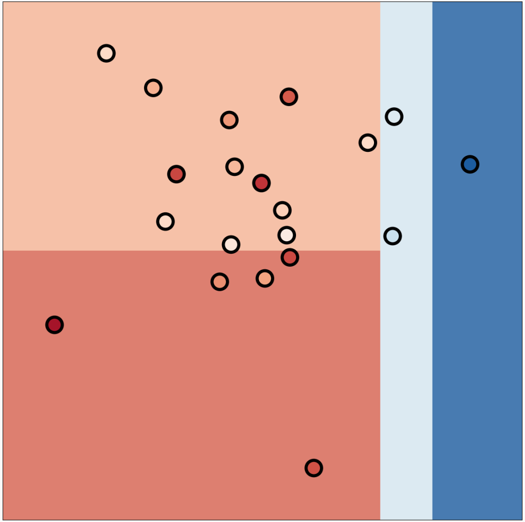
2 distinct values.

The model is **piecewise constant**.



Decision trees





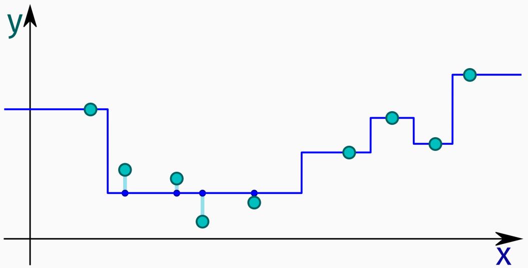
Depth 2:

up to 4 distinct values.

The model follows the **D axes**

of the feature space.

Decision trees

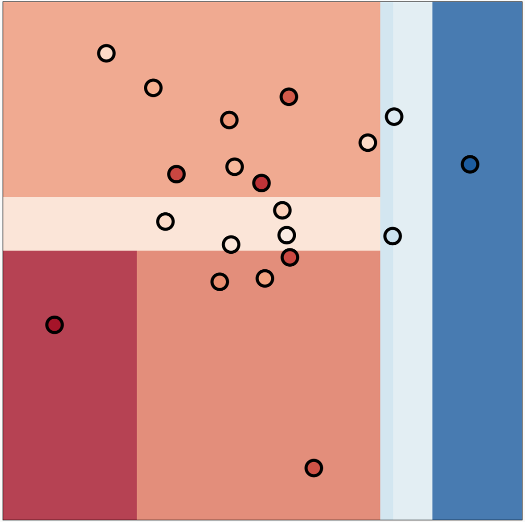


Depth 3:

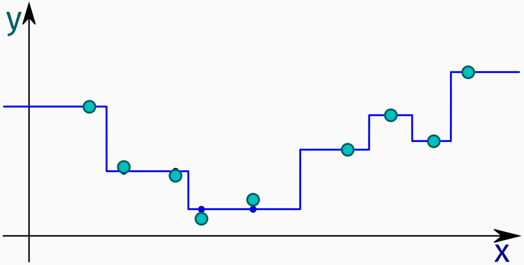
up to 8 distinct values.

We may choose not to use them all

to limit the **complexity** of the model.



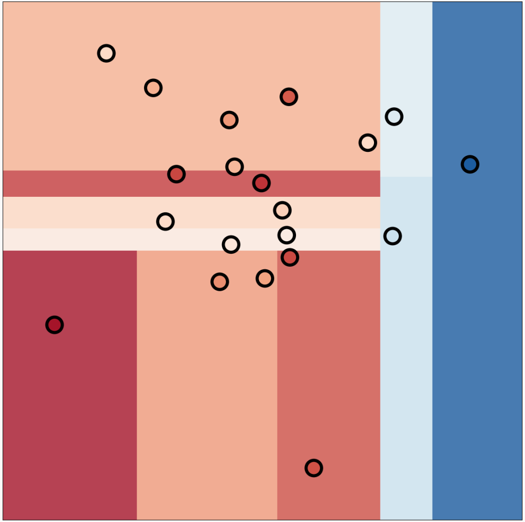
Decision trees



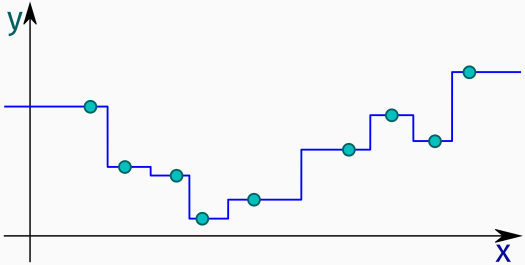
Depth 4:

up to 16 distinct values.

Starting to clearly **overfit**.



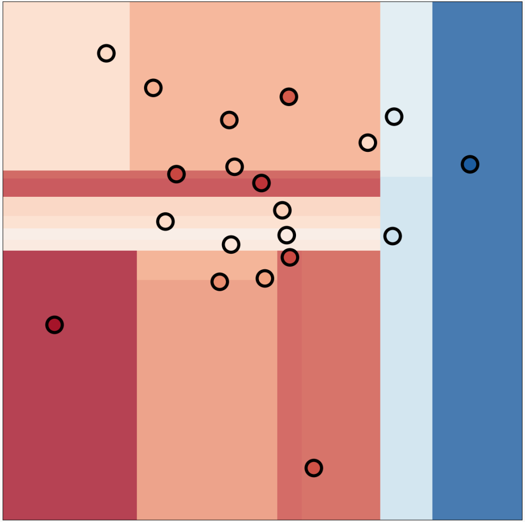
Decision trees



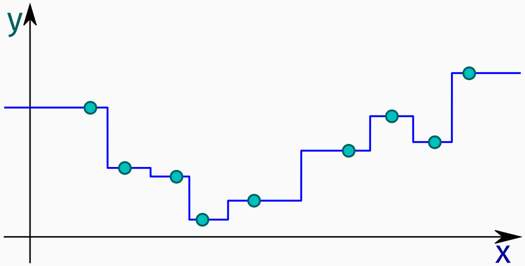
Depth 5:

up to 32 distinct values.

Starting to clearly **overfit**.



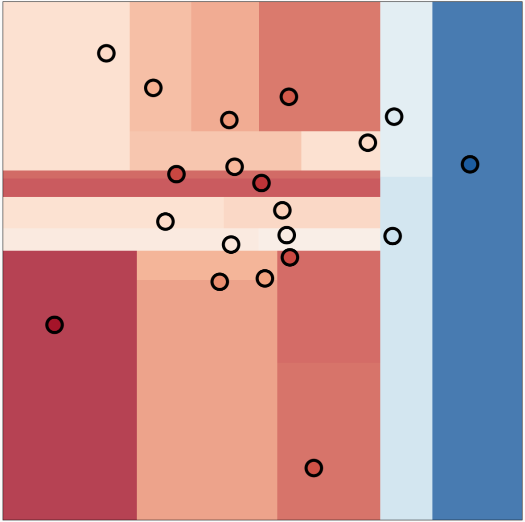
Decision trees



Depth 10:

up to 1,024 distinct values.

Full **overfit** on both datasets.



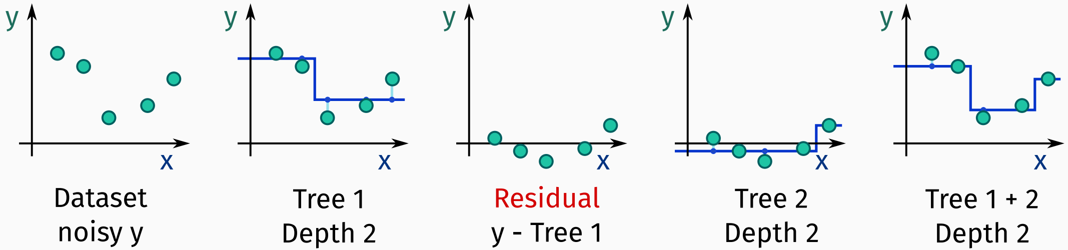
Decision trees - strengths and weaknesses

Decision trees are:

- **Interpretable.**
- **Easy** to train and deploy.
- **Fast** and CPU-friendly.
- **Robust**:
    - Only use a few columns at a time.
    - Work well with **heterogenous** information.
    - Only rely on the **ordering** of the features.

However, trees also **overfit** quickly and produce **blocky** results.
Regularization methods mitigate these issues, at the cost of **interpretability**.

1st regularization strategy: boosted sequence of trees

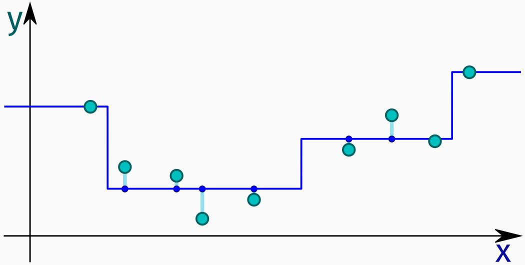


Iterative fits on the prediction residuals with shallow trees.

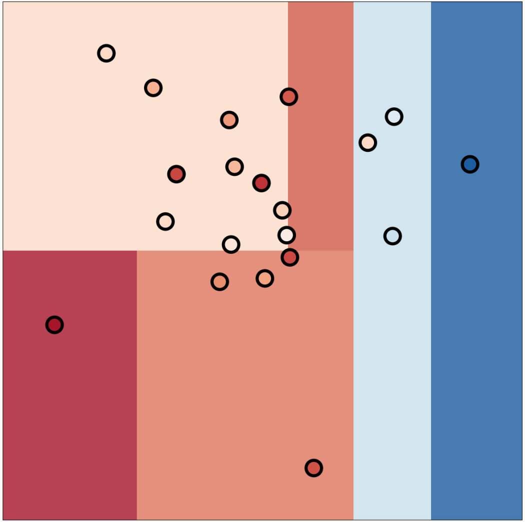
Use a **small** learning rate for better regularization:

$$\textbf{Residual}_{\mathbf{i}} = y_i - \mathbf{0.1} \cdot \sum_k \text{Tree}_k(x_i)\ .$$

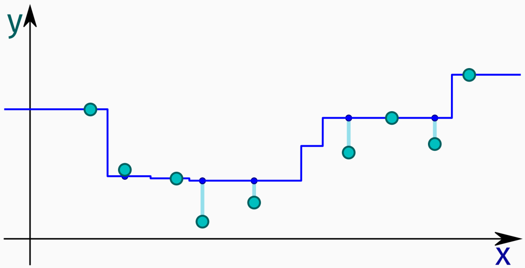
1st regularization strategy: boosted sequence of trees



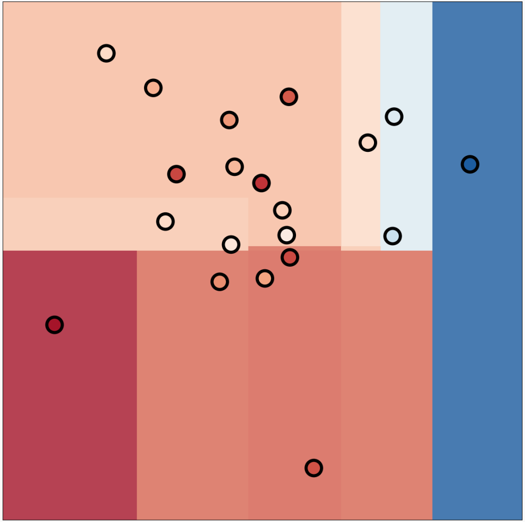
1 tree of depth 3:
a simple decision tree,
with moderate complexity.



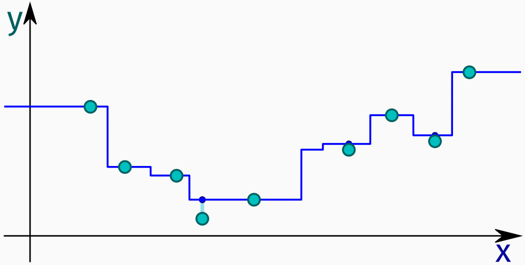
1st regularization strategy: boosted sequence of trees



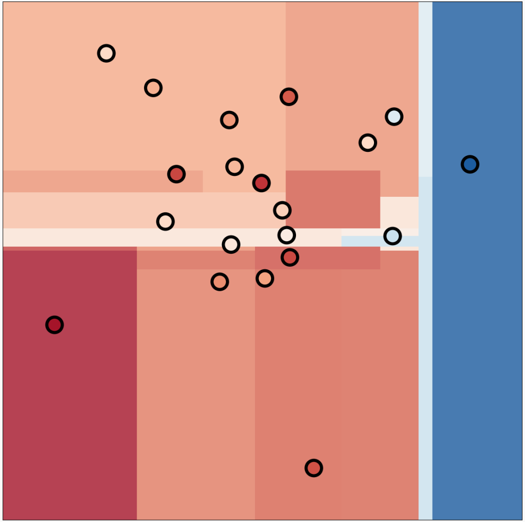
3 trees of depth 3:
sum of three simple decision trees,
fitted iteratively on **residuals**.



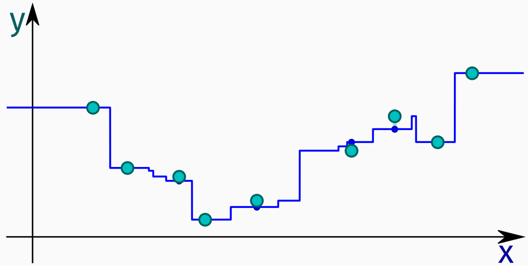
1st regularization strategy: boosted sequence of trees



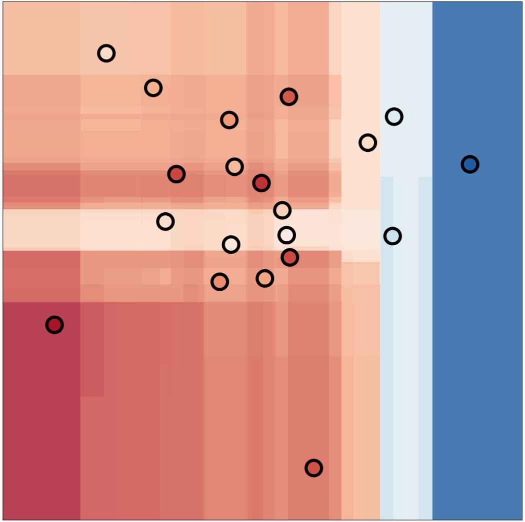
5 trees of depth 3:
sum of five simple decision trees,
fitted iteratively on **residuals**.



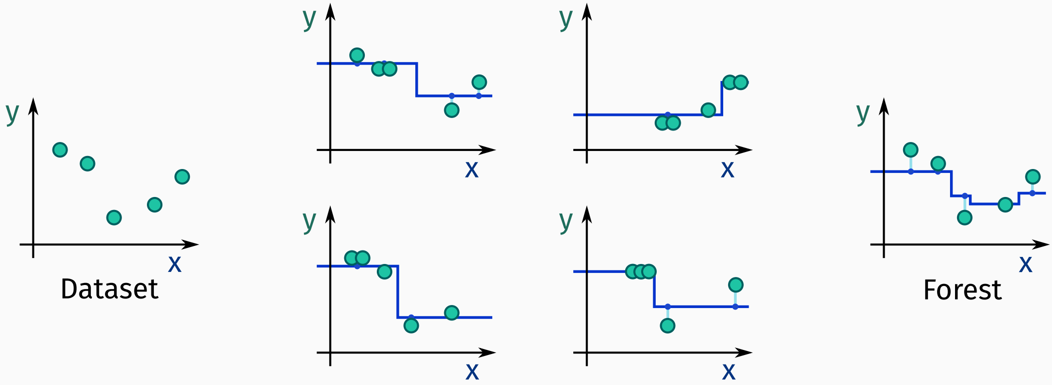
1st regularization strategy: boosted sequence of trees



100 trees of depth 3:
sum of a hundred simple decision trees.
We reach a high training accuracy
with a relatively smooth model.

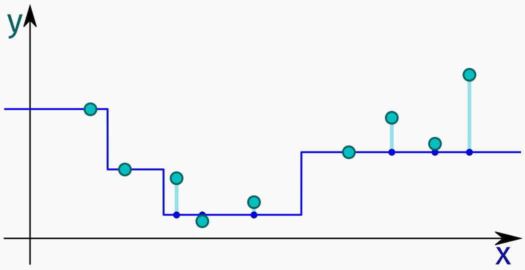


2nd regularization strategy: random forests

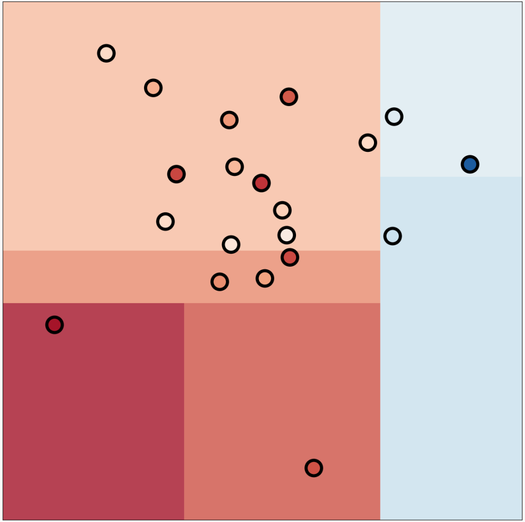


Parallel fits on **bootstrap** samples of the original dataset.
The final model is the **average** of a **forest** of independent trees.

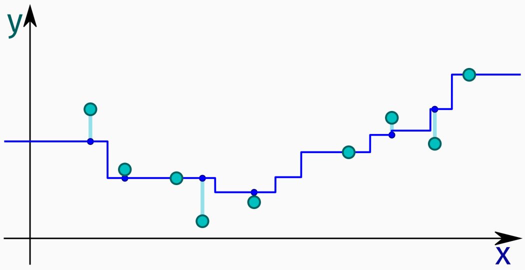
2nd regularization strategy: random forests



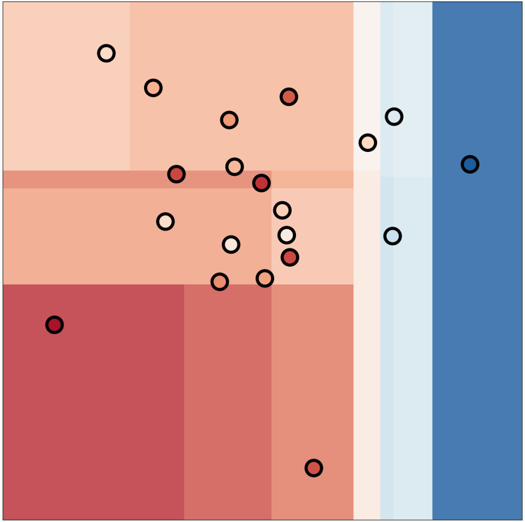
1 tree of depth 3:
a simple decision tree,
computed on a bootstrap
subset of the original sample.



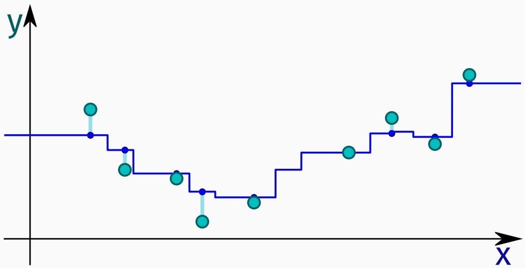
2nd regularization strategy: random forests

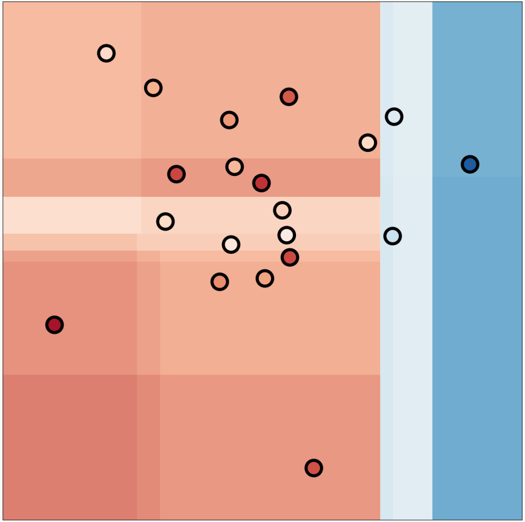


2 trees of depth 3:
average of two decision trees,
fitted on two independent
bootstrap samples.



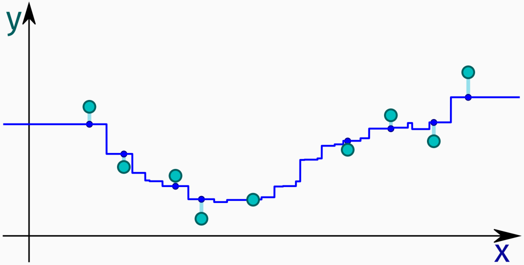
2nd regularization strategy: random forests

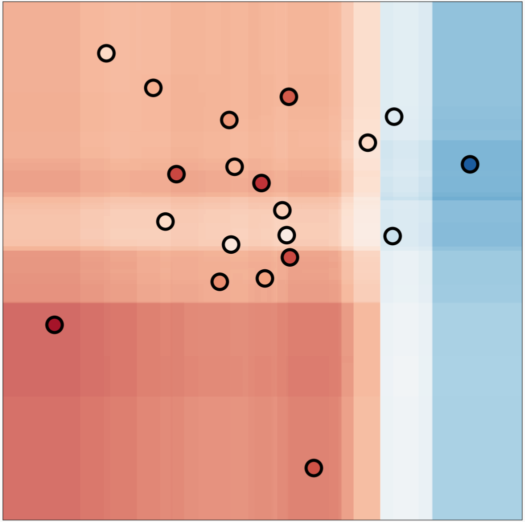




5 trees of depth 3:
average of five decision trees,
fitted on five independent
bootstrap samples.

2nd regularization strategy: random forests





100 trees of depth 3:
a regularized decision rule.
The model still follows the **axes**
of the feature space, but is **smoother**.

Some features may require more work – understand the context! [Wit]

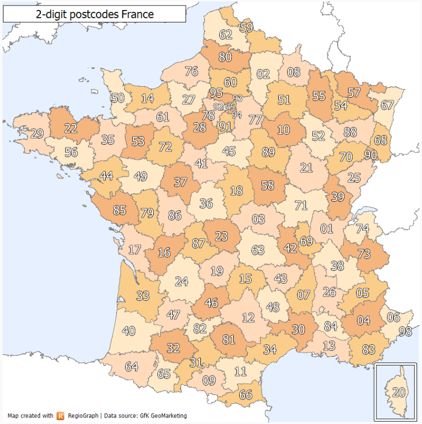
WEIGHT

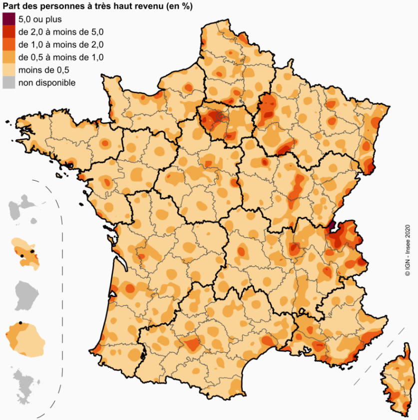
kgs / cm (HEIGHT)	41	45	50	54	59	64	68	73	77	82	86	91	95	100	104	109	113	118	122	127	132
142.2	20	22	25	27	29	31	34	36	38	40	43	45	47	49	52	54	56	58	61	63	65
144.7	19	22	24	16	28	30	32	35	37	39	41	43	45	48	50	52	54	56	58	61	63
147.3	19	21	19	25	27	29	31	33	36	38	40	42	44	46	48	50	52	54	56	59	61
149.8	18	20	22	24	26	28	30	32	34	26	38	40	42	44	46	48	51	53	55	57	59
152.4	18	20	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49	51	53	55	57
154.9	17	19	21	22	25	26	28	30	32	34	36	39	40	42	43	45	47	49	51	53	55
157.4	16	18	20	22	24	26	27	29	31	33	35	37	38	20	42	44	46	48	49	51	53
160.0	16	18	19	21	23	25	27	28	30	32	34	35	37	39	41	43	44	46	48	50	51
162.5	15	17	19	21	22	24	26	27	29	31	33	34	36	27	39	41	43	45	46	48	50
165.1	15	17	18	20	22	23	25	27	28	30	32	33	35	27	38	40	42	43	45	47	49
167.6	15	16	18	19	21	23	24	26	27	29	31	32	34	26	37	39	40	42	44	45	47
170.1	14	16	17	19	20	22	24	25	27	29	30	31	33	25	36	38	39	41	42	44	45
172.7	14	15	17	18	20	21	23	24	26	27	29	30	32	24	35	37	38	40	41	43	44
175.2	13	15	16	18	19	21	22	24	25	27	28	30	31	24	34	35	37	39	40	41	43
177.8	13	14	16	17	19	20	22	23	24	26	27	29	30	23	33	34	36	37	39	40	42
180.3	13	14	15	14	18	20	21	22	24	25	27	28	29	22	32	33	35	36	38	39	40
182.8	12	14	15	16	18	16	20	22	23	24	26	27	28	22	31	33	34	35	37	38	39
185.4	12	13	15	16	17	18	20	21	22	24	25	26	28	21	30	32	33	34	36	37	39
187.9	12	13	14	15	17	18	19	21	22	23	24	26	27	21	30	31	32	33	35	36	37
190.5	11	13	14	15	16	17	19	20	21	23	24	25	26	20	29	30	31	32	34	35	36
193.0	11	12	13	15	16	17	18	19	21	22	23	24	26	19	28	29	30	32	33	34	35
195.5	11	12	13	14	15	17	18	19	20	21	23	24	25	19	27	28	30	31	32	33	34
198.1	10	12	13	14	15	16	17	18	20	21	22	23	24	18	27	28	29	30	31	32	34
200.6	10	11	12	14	15	16	17	18	19	20	21	23	24	18	26	27	28	29	30	32	33
203.2	10	11	12	13	14	15	16	18	19	20	21	22	23	18	25	26	27	28	30	31	32
205.7	10	11	12	13	14	15	16	17	18	19	20	21	23	17	25	25	27	28	29	30	31
208.2	9	10	12	13	14	14	16	17	18	19	20	21	22	17	24	25	26	27	28	29	30
210.8	9	10	11	12	13	14	15	16	17	18	19	20	21	16	23	25	25	27	28	29	30

The **Body Mass Index** $=$ weight / height^2

is a good indicator for many health problems.

Some features may require more work – understand the context!





Applying thresholds on **postal codes** is mostly useless.

Other statistics may be much more informative.

Some features may require more work – understand the context!

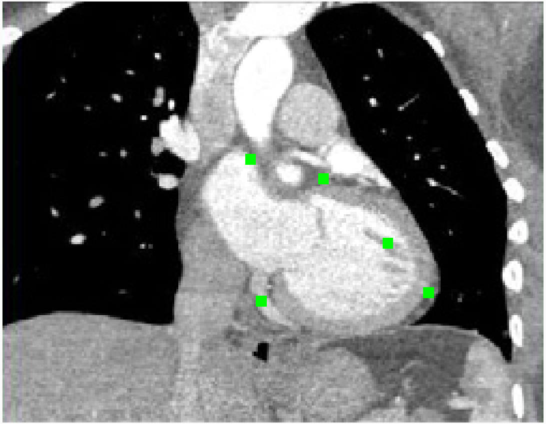
1665491486

SECONDS SINCE JAN 01 1970. (UTC)

2:31:28 PM

Applying thresholds on **UNIX timestamps** is mostly useless.
We must first apply periodic transforms to get hours-days-months.

Sometimes, the input features are just not good enough [EPW11]



Tree models cannot process **raw pixel values**.
Standard radiomic features only take you so far.

Trees and forests – conclusion

Tree-based models are:

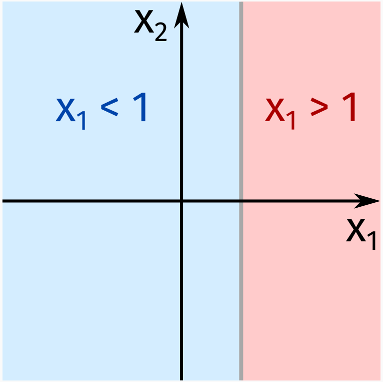
- Highly **interpretable**.
- Well suited to **high-quality heterogeneous** features.
- **Easy** to use: XGBoost, LightGBM, scikit-learn…

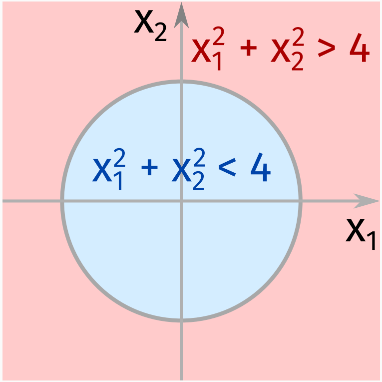
On the other hand, they produce **non-smooth** results
and are biased along the **axes** of the feature space.

This is a major limitation if you work with **homogeneous** features:
the 3D xyz coordinates, pixel values, audio signals…

K-Nearest Neighbors

The Euclidean metric is isotropic

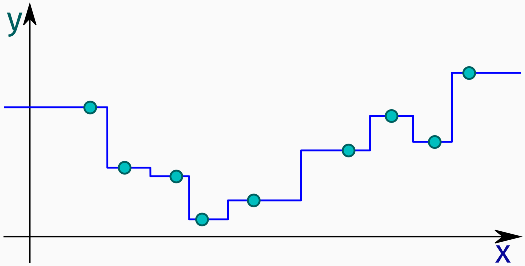




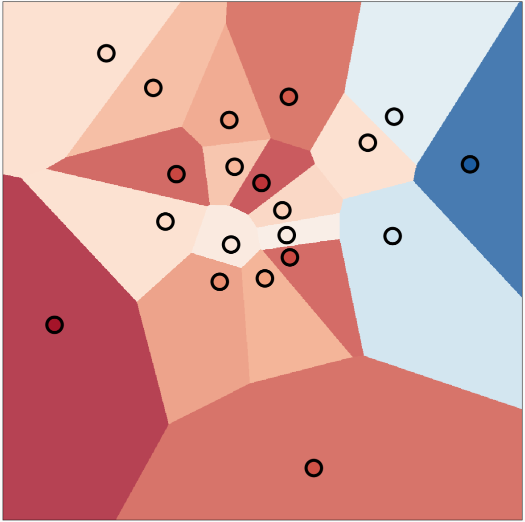
Thresholding features promotes decisions along the **axes** of the feature space.

The squared **Euclidean** metric $\|(x_1, \ldots, x_D)\|^2 = x_1^2 + \cdots + x_D^2$ is invariant to **rotations**.

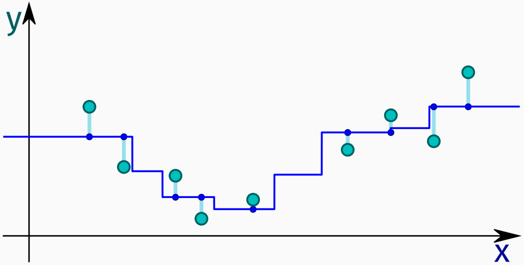
Average value among the K-Nearest Neighbors



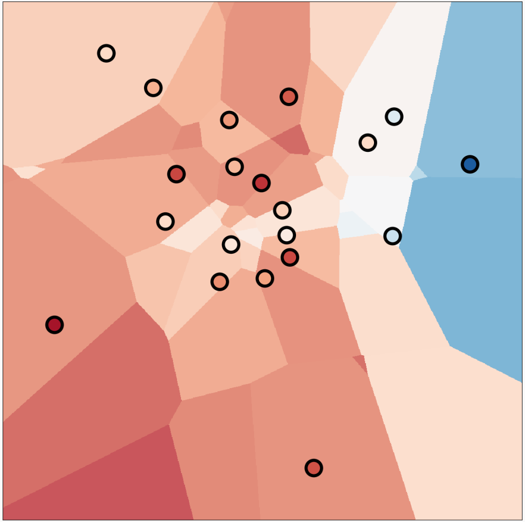
With **K = 1** neighbor, we retrieve
a simple nearest neighbor interpolation.
This model is piecewise constant
on the **Voronoi diagram** of x.



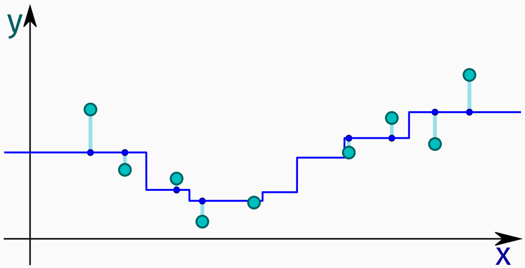
Average value among the K-Nearest Neighbors



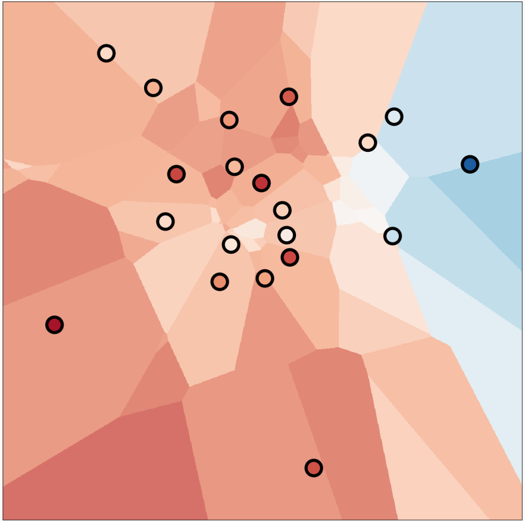
With **K = 2** neighbors,
the cells of the diagram
become smaller.



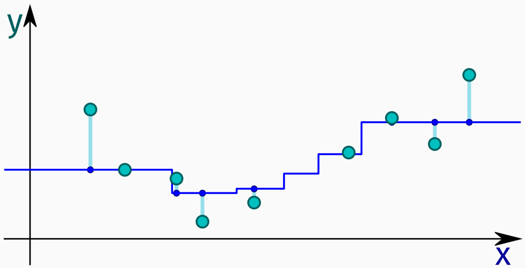
Average value among the K-Nearest Neighbors



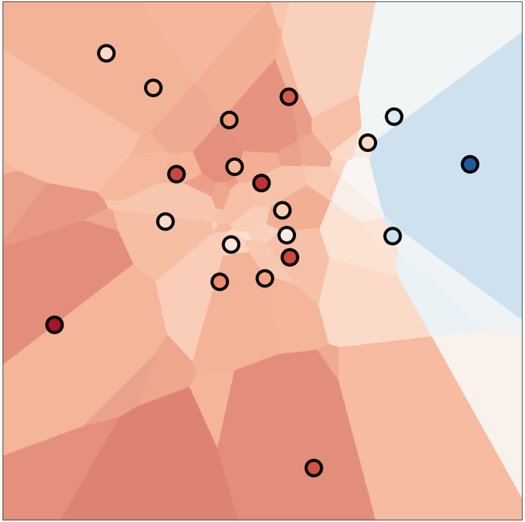
With **K = 3** neighbors,
the cells of the diagram
become smaller.



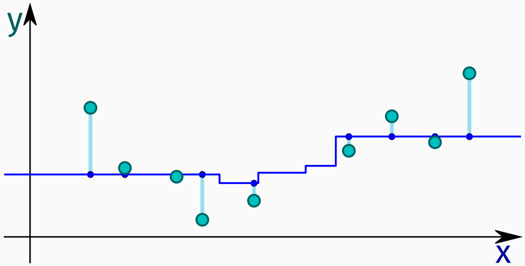
Average value among the K-Nearest Neighbors

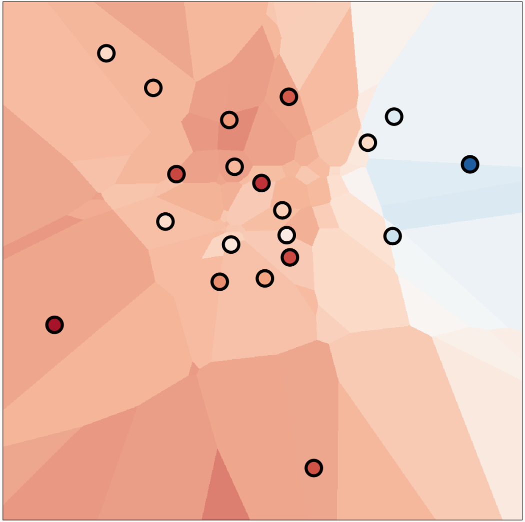


With **K = 4** neighbors,
the cells of the diagram
become smaller.



Average value among the K-Nearest Neighbors





With **K = 5** neighbors,
the model looks **smoother** and smoother
but is still **piecewise constant**.

K-Nearest Neighbors: the main selling points

K-NN models are:

- **Interpretable**.
- **Isotropic** – which may or may not be a good thing!
- **Easy** to deploy.
- **Fast**, parallel and GPU-friendly – see our MVA Lecture 7 on algorithms.
- **Well-packaged** and scalable: FAISS, KeOps, (big-)ann-benchmarks.com…

Major weakness: K-NNs require a good scaling of the input features

Unlike tree-based models, the Euclidean distance is sensitive
to the **precise values** of the features x.

Out-of-the-box, K-NNs are not even robust to the **choice of the units**
for the columns of our dataset!

We must **normalize** the input features using:

- A feature-wise rescaling using e.g. the **standard deviation**.
- A **multivariate** normalization using e.g. Principal Component Analysis.
  The Euclidean distance with a normalized PCA is known as the **Mahalanobis** metric.
- Alternatively, a robust **equalization** of the feature histograms.

Summary on trees and K-NNs

Tree-based and **K-NN** models are:

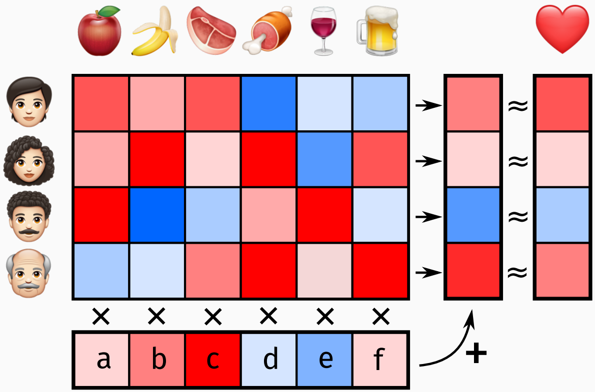
- **Interpretable** methods with **heterogeneous** / **homogeneous features**.
- Well-understood, well-packaged and easy to deploy.
- Excellent baselines for **interpolation**.

Unfortunately, both methods :

- Produce **non-smooth**, piecewise constant decision rules.
- Are **local** and do not estimate global trends.
  They are not a natural fit for **extrapolation**, **forecasting**.

Linear regression

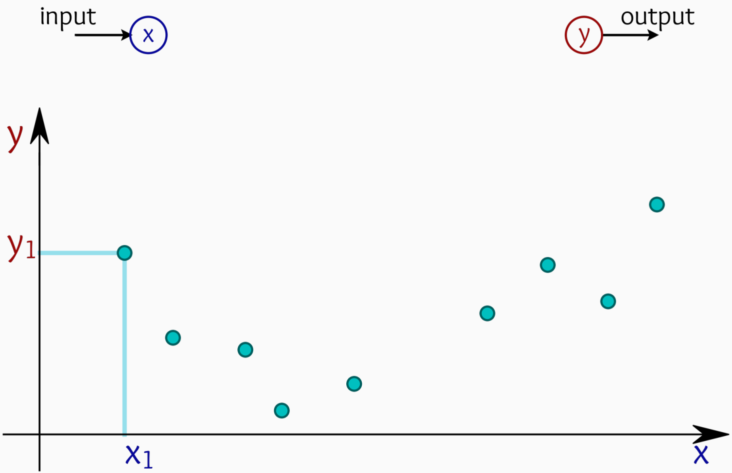
A simple model: linear regression



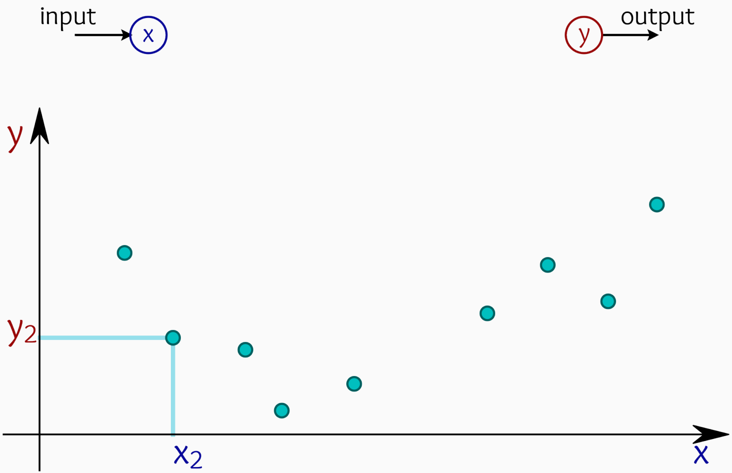
We choose the weights **a**, **b**, …, **f**

by minimizing a least squares error.

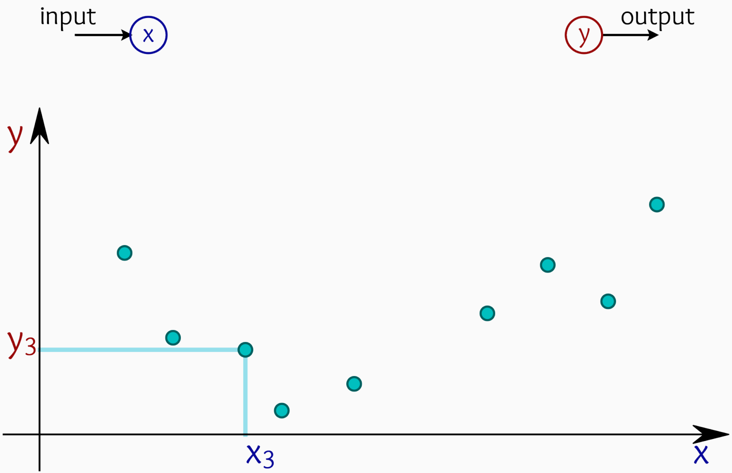
A toy dataset, in dimension 1



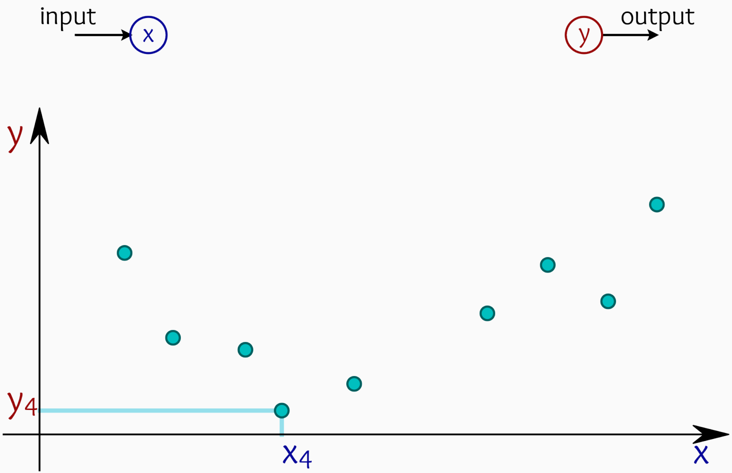
A toy dataset, in dimension 1



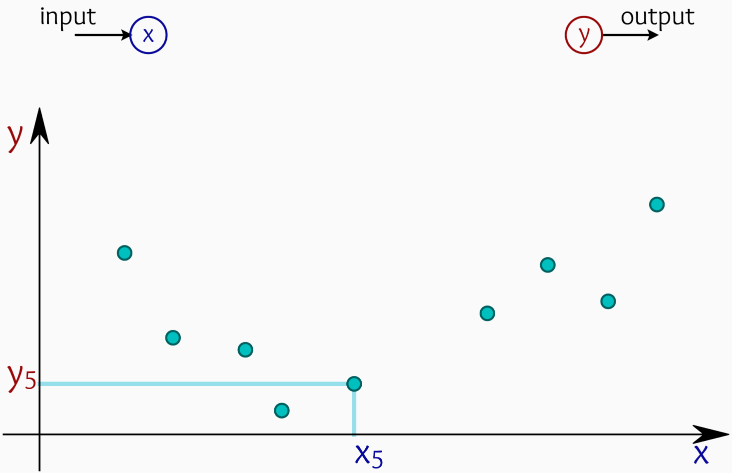
A toy dataset, in dimension 1



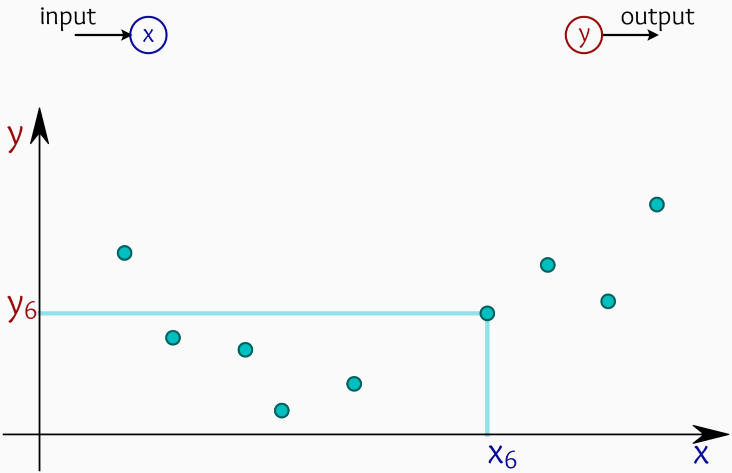
A toy dataset, in dimension 1



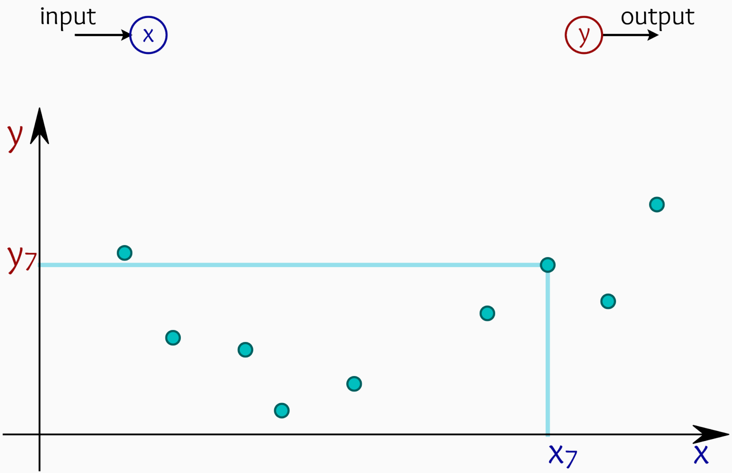
A toy dataset, in dimension 1



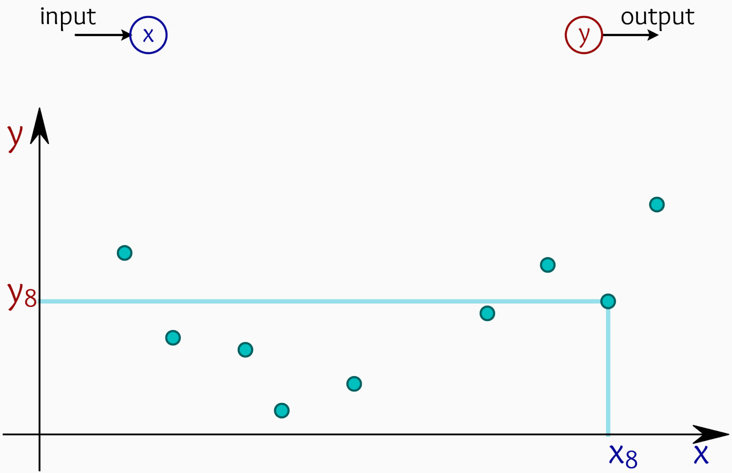
A toy dataset, in dimension 1



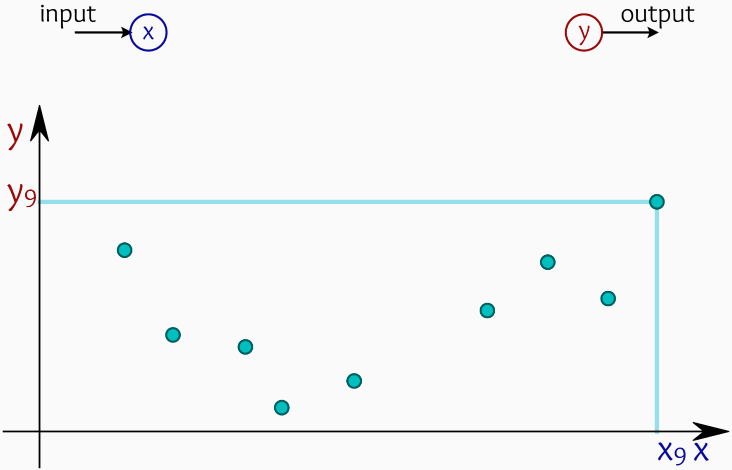
A toy dataset, in dimension 1



A toy dataset, in dimension 1

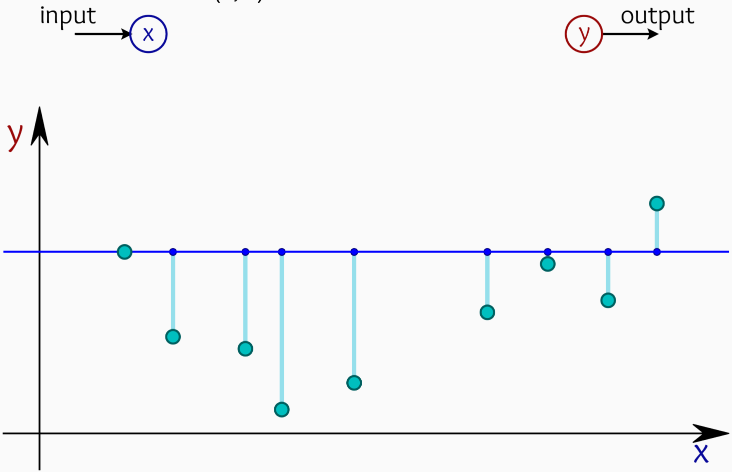


A toy dataset, in dimension 1



Iterative least squares fitting

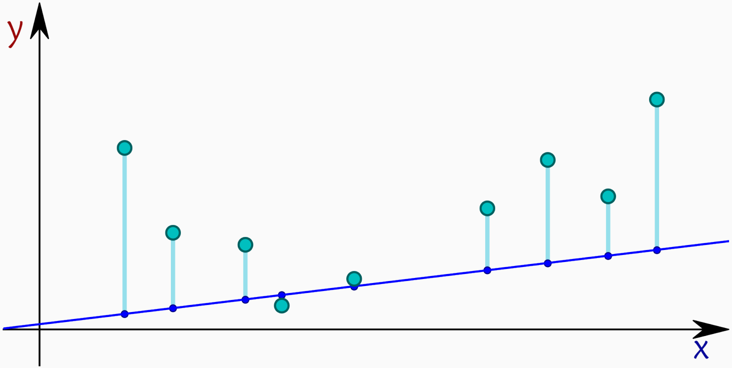
$$F(a, b; x) = a \cdot x + b$$
$$(a, b) = +0.00,\ +0.25$$



Iterative least squares fitting

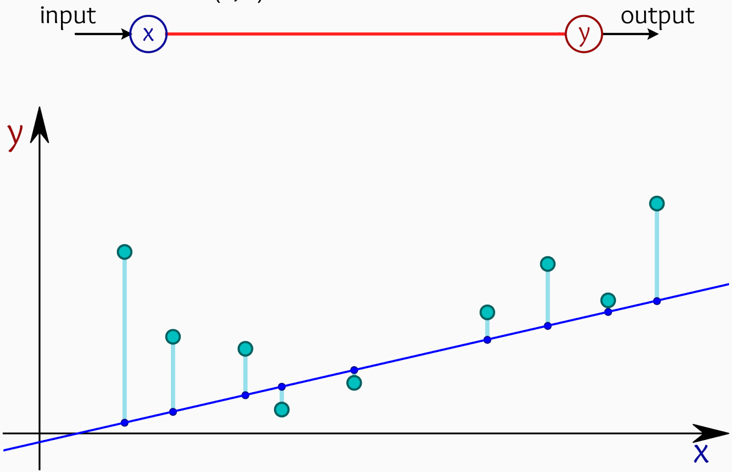
$$F(a, b; x) = a \cdot x + b$$

$$(a, b) = +0.12,\ +0.01$$



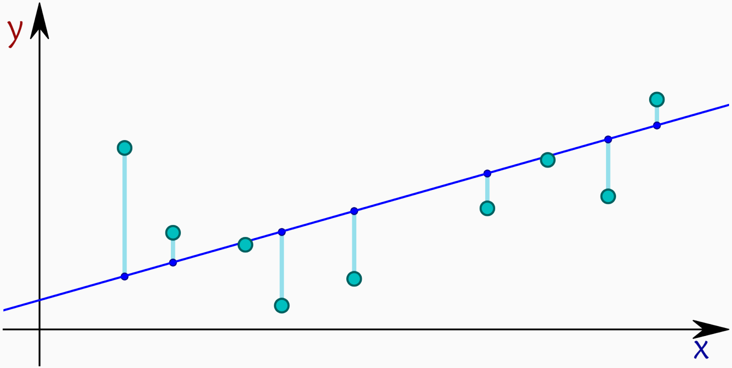
Iterative least squares fitting

$$F(a,b;x) = a \cdot x + b$$
$$(a,b) = +0.23, -0.01$$



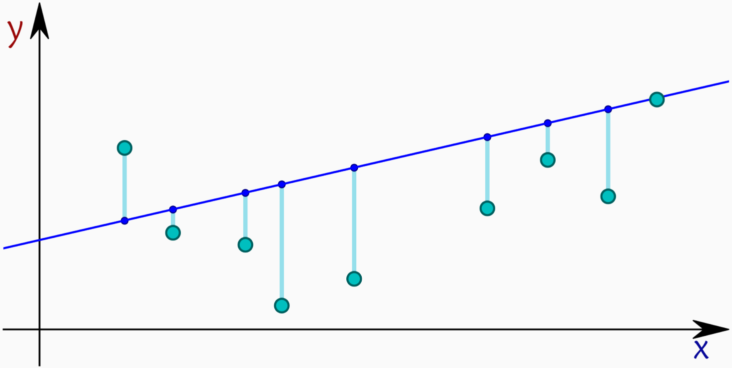
Iterative least squares fitting

$$F(a, b; x) = a \cdot x + b$$
$$(a, b) = +0.28,\ +0.04$$



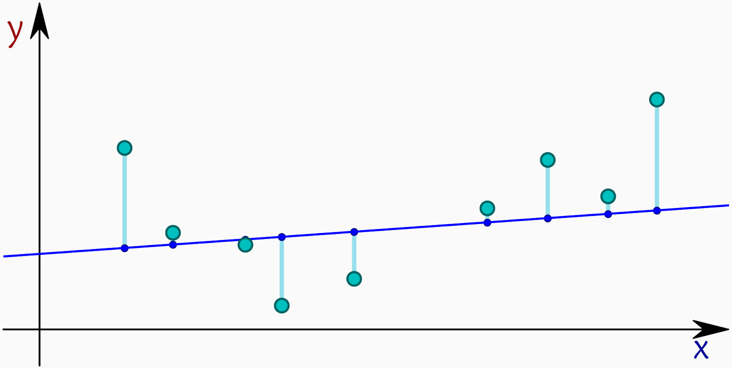
Iterative least squares fitting

$$F(a, b; x) = a \cdot x + b$$
$$(a, b) = +0.23, +0.12$$



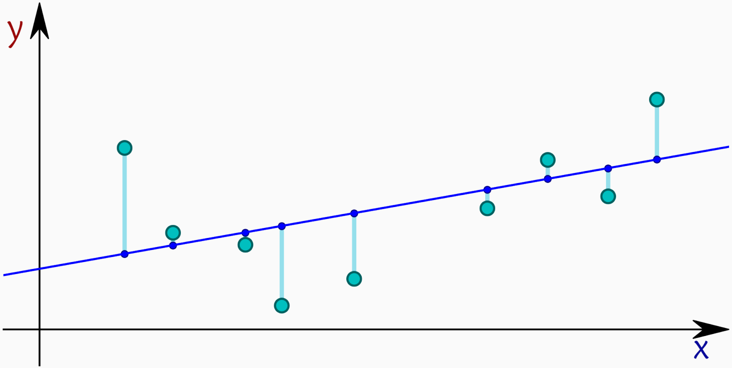
Iterative least squares fitting

$$F(a, b; x) = a \cdot x + b$$
$$(a, b) = +0.07,\ +0.10$$



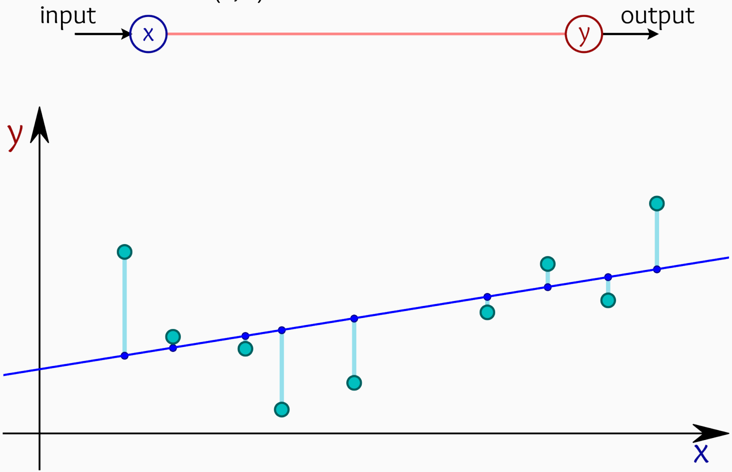
Iterative least squares fitting

$$F(a, b; x) = a \cdot x + b$$
$$(a, b) = +0.18, +0.08$$



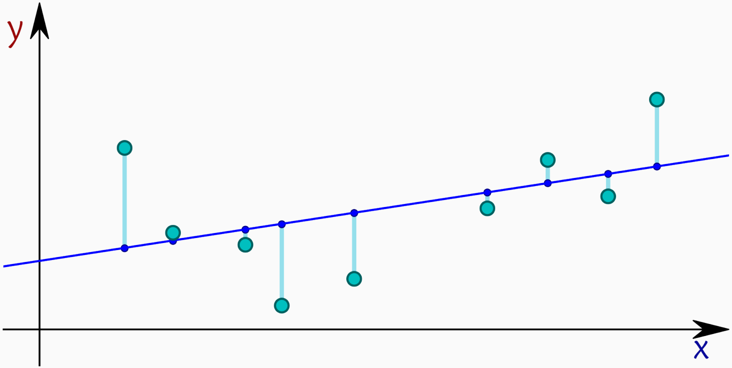
Iterative least squares fitting

$$F(a, b; x) = a \cdot x + b$$
$$(a, b) = +0.16,\ +0.09$$

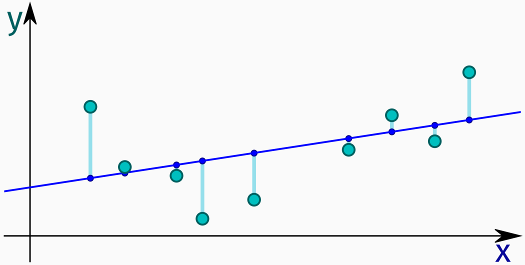


Iterative least squares fitting

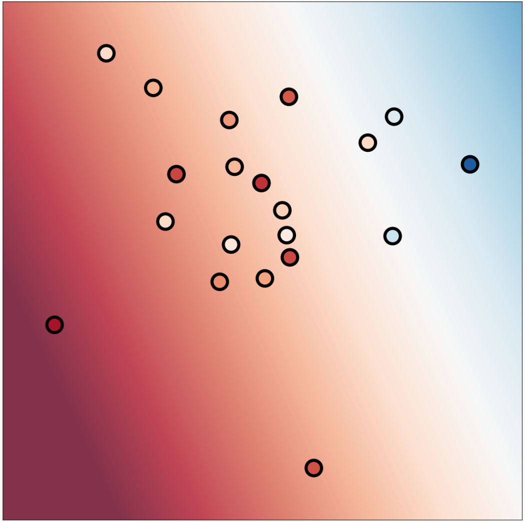
$$F(a, b; x) = a \cdot x + b$$
$$(a, b) = +0.15,\ +0.09$$



Linear regression



Linear regression models
a **monotonic trend**.
It cannot handle complex relationships
between the input x and the ouput y.

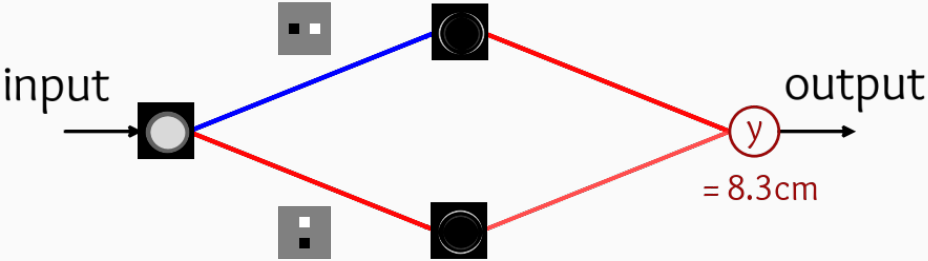


What should we do if the problem is complex?

Take a break :-)

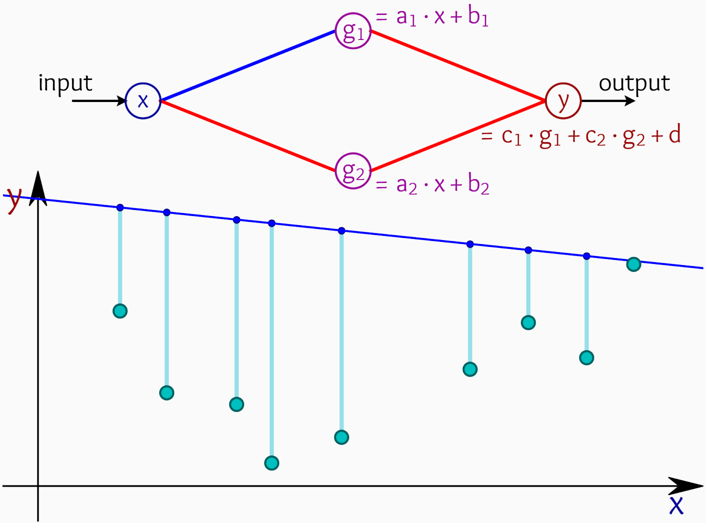
Neural networks

Maybe, we could introduce some intermediate variables in our model?

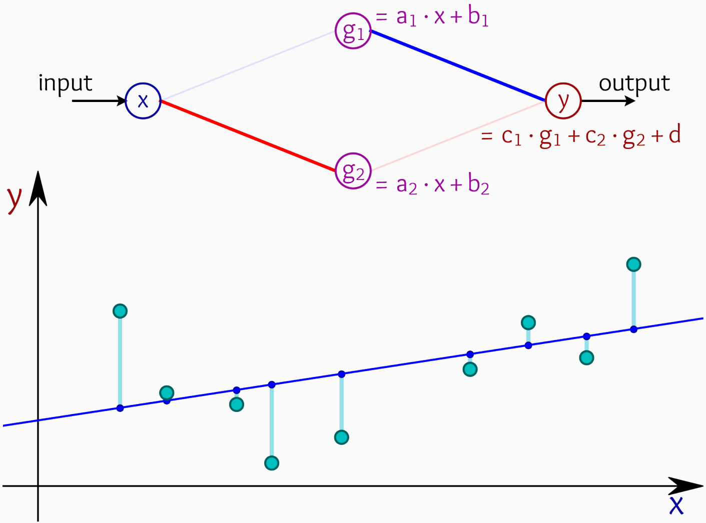


Domain experts may have suggested a **step-by-step** process
to compute the quantity of interest – say, the perimeter of an organ.

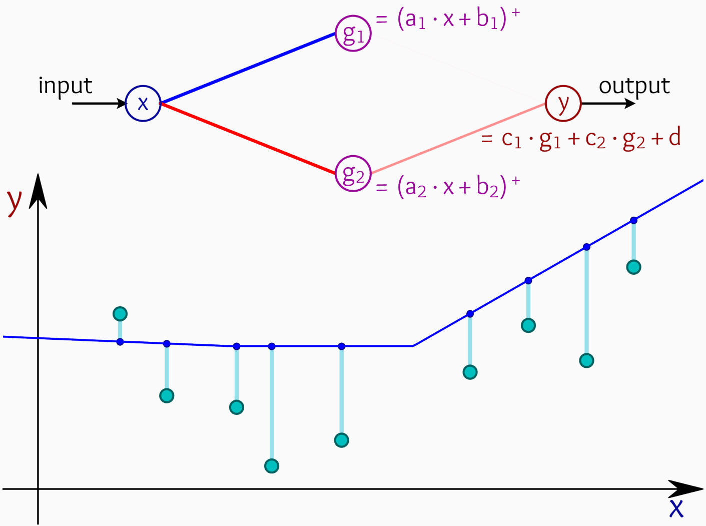
Let's complexify our model with intermediate variables...



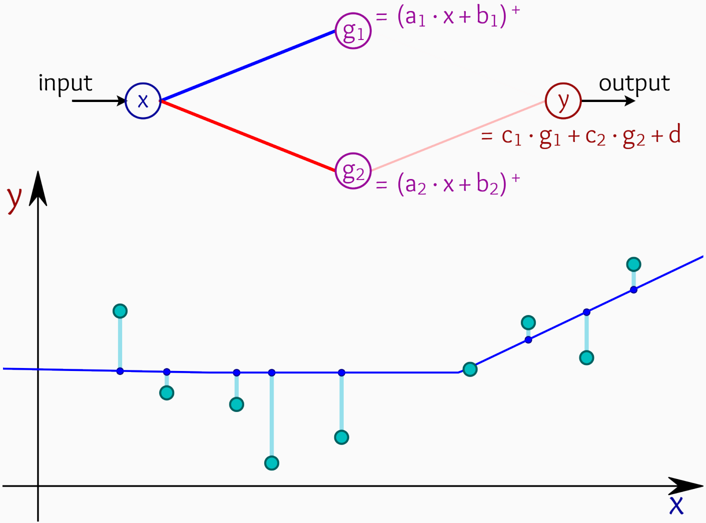
Let's complexify our model with intermediate variables...



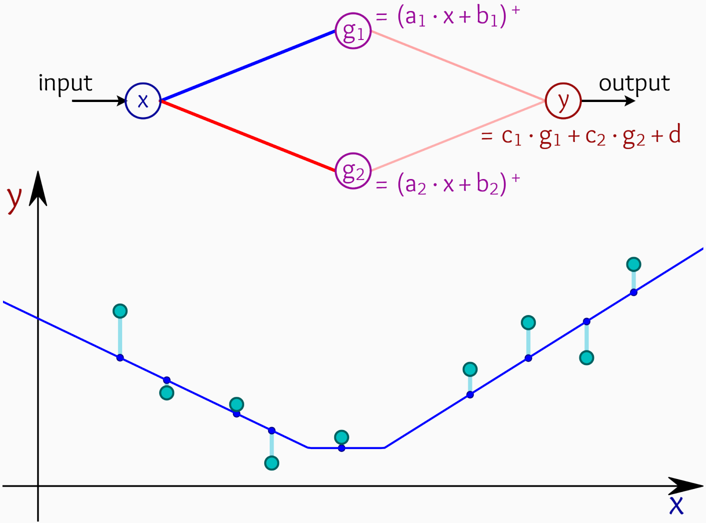
Let's complexify our model with intermediate variables... and non-linearities!



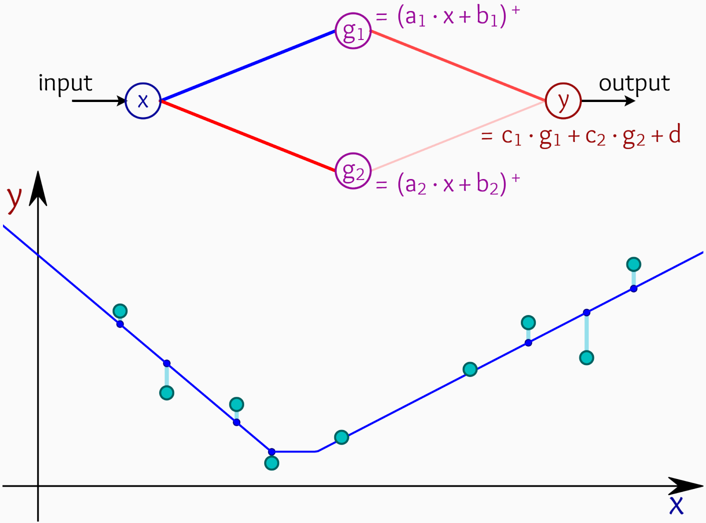
Let’s complexify our model with intermediate variables... and non-linearities!



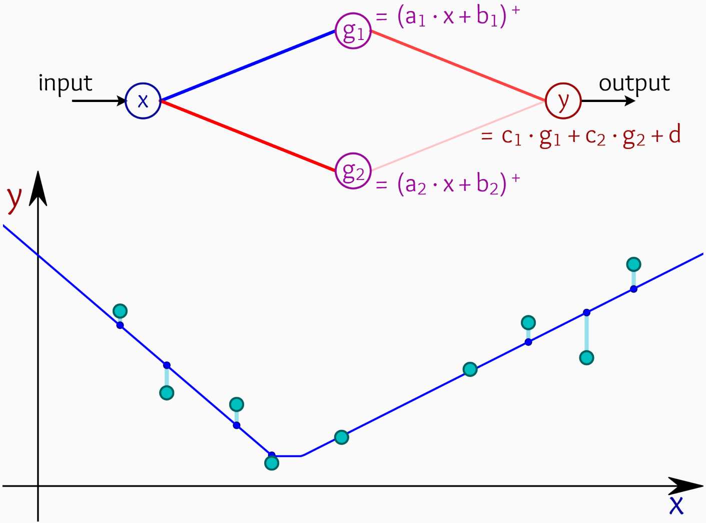
Let’s complexify our model with intermediate variables... and non-linearities!



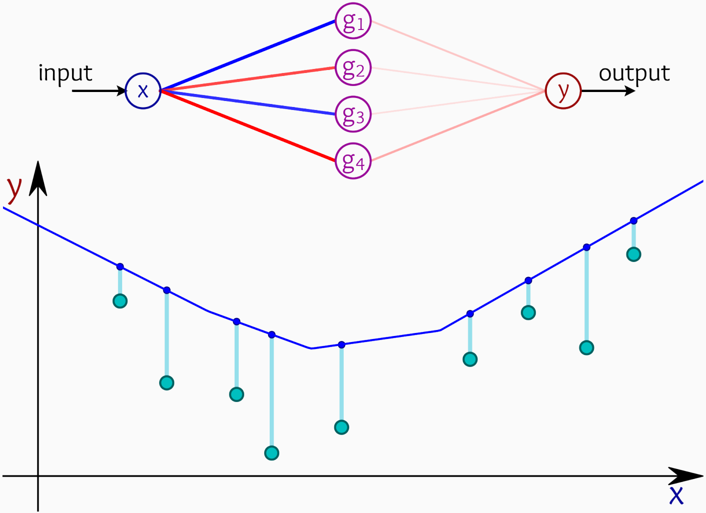
Let's complexify our model with intermediate variables... and non-linearities!



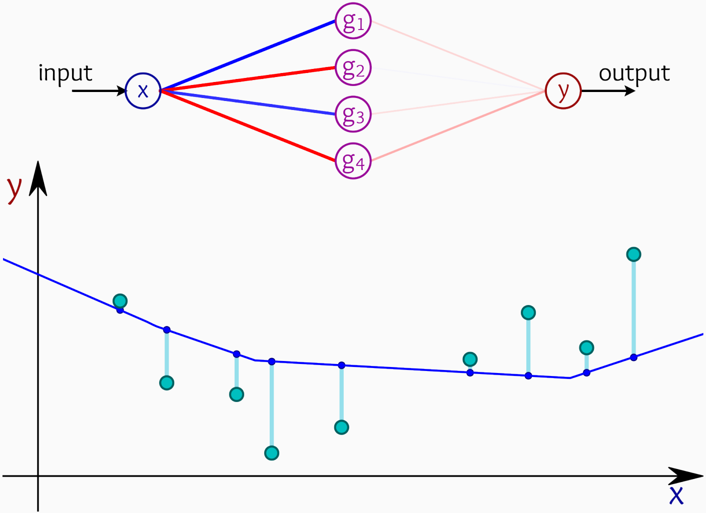
Let’s complexify our model with intermediate variables… and non-linearities!



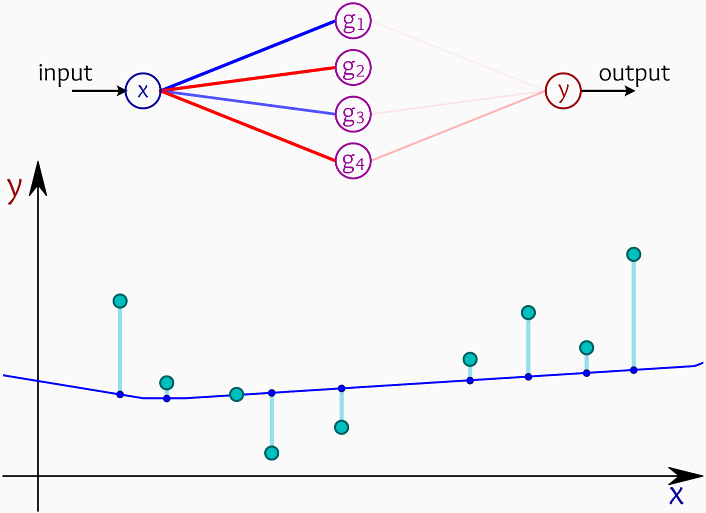
We can then increase the number of “neurons”...



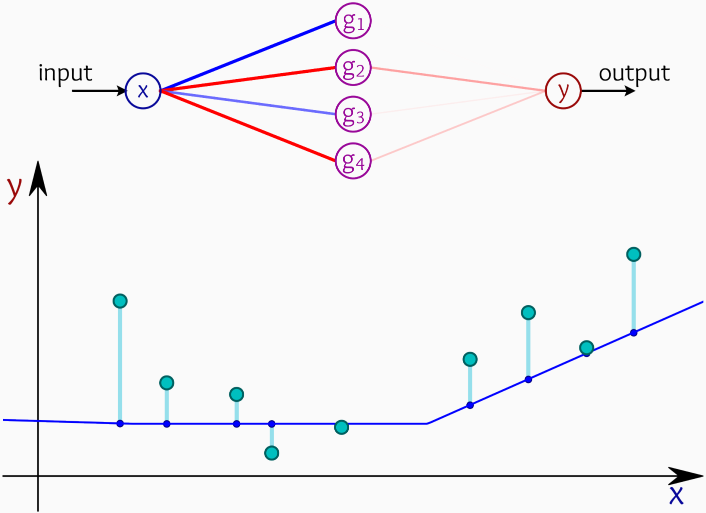
We can then increase the number of “neurons”...



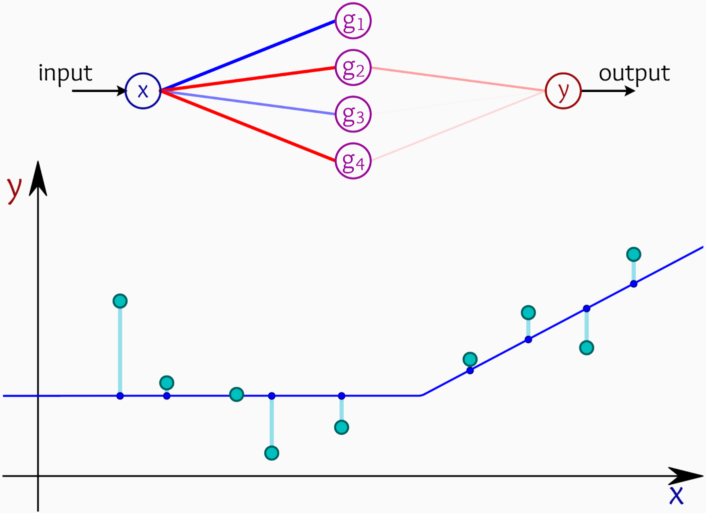
We can then increase the number of “neurons”...



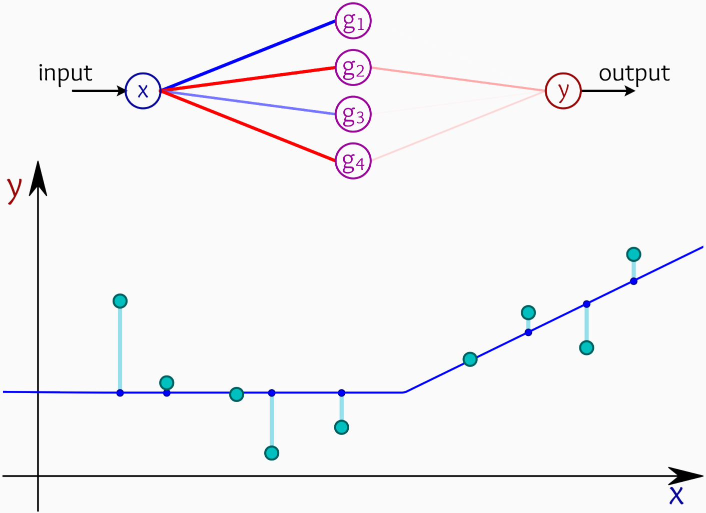
We can then increase the number of “neurons”...



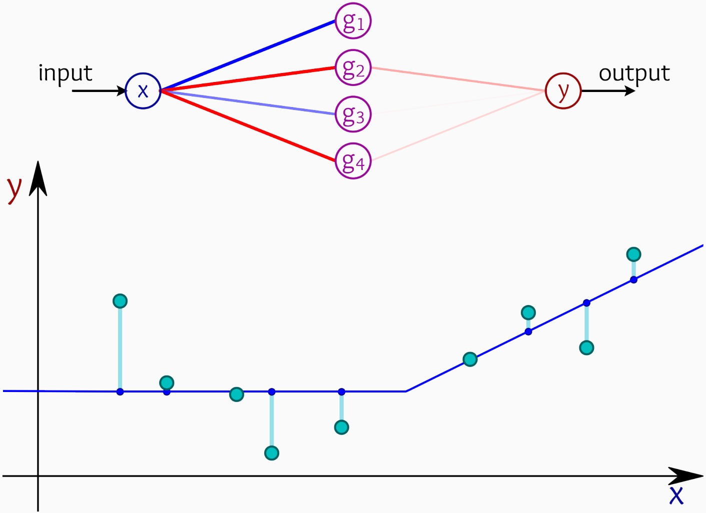
We can then increase the number of “neurons”...



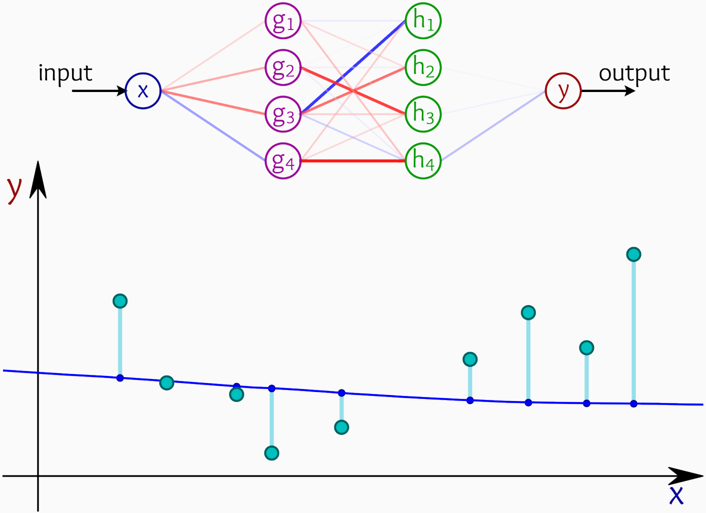
We can then increase the number of “neurons”...



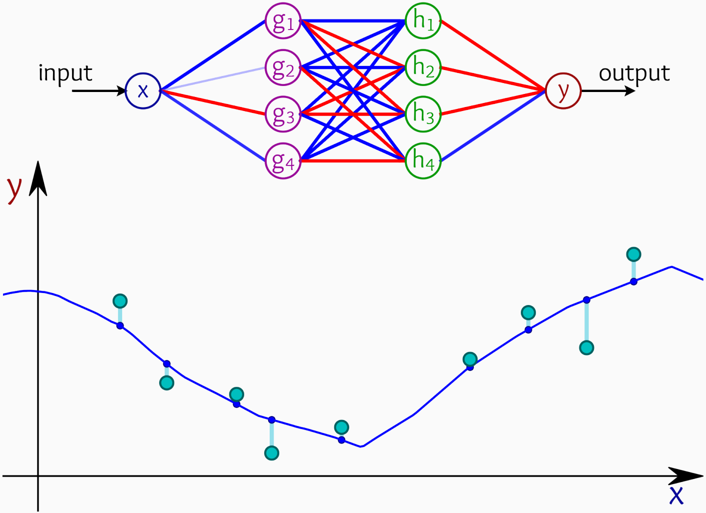
We can then increase the number of “neurons”...



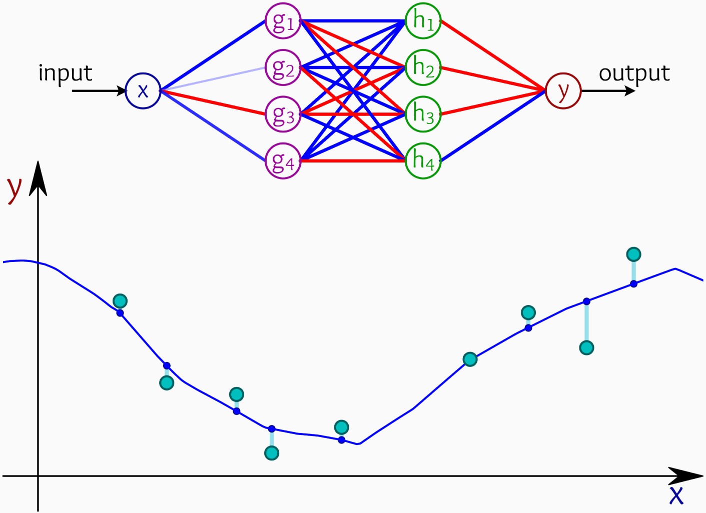
We can then increase the number of “neurons”... and layers!



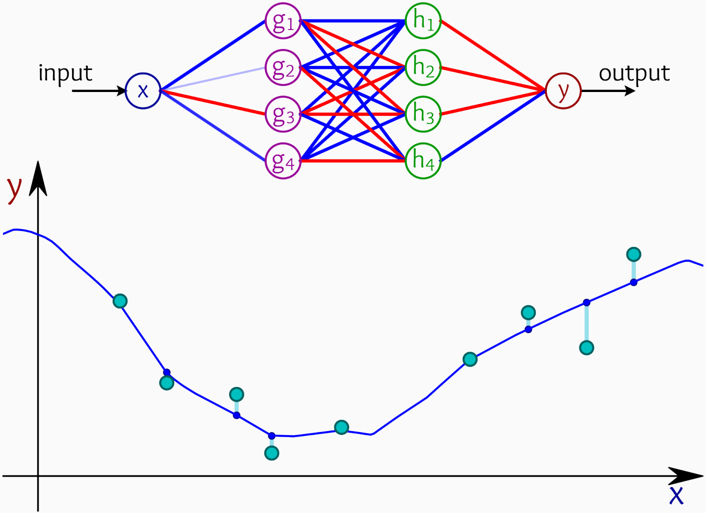
We can then increase the number of “neurons”... and layers!



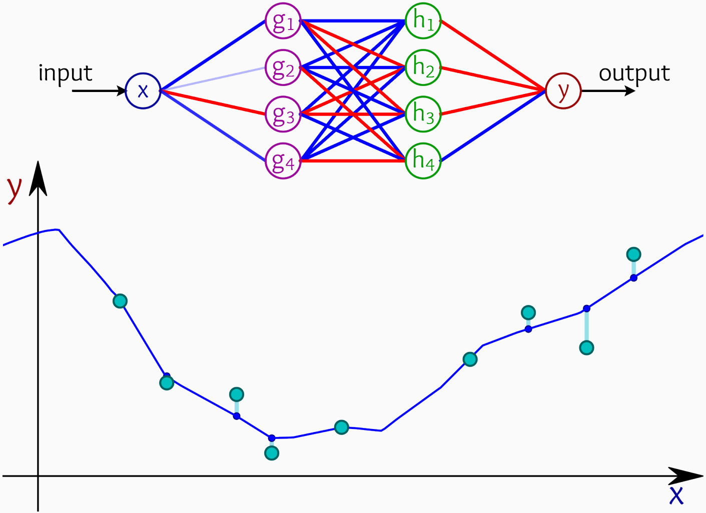
We can then increase the number of “neurons”... and layers!



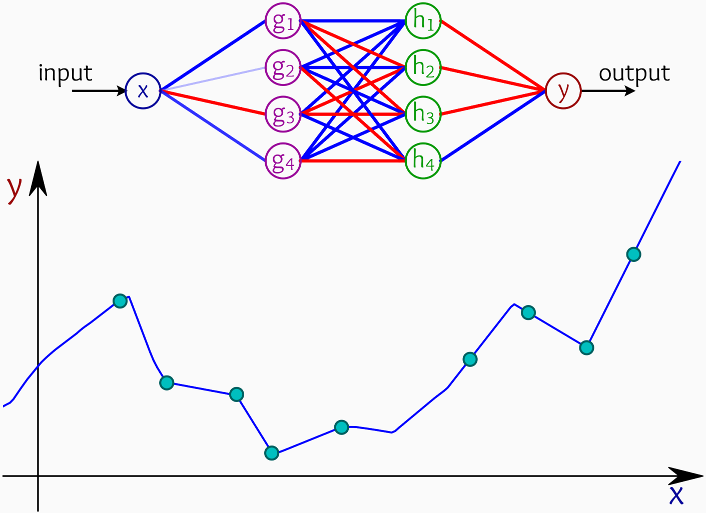
We can then increase the number of “neurons”... and layers!



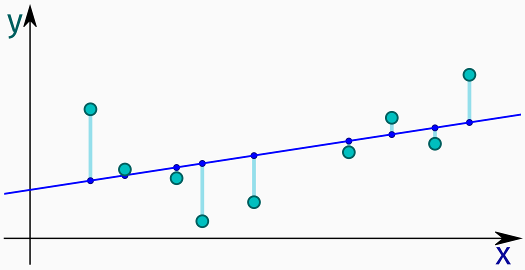
We can then increase the number of “neurons”... and layers!

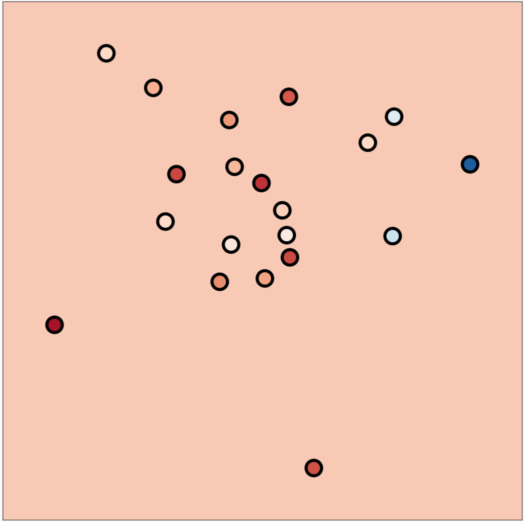


We can then increase the number of “neurons”... and layers!



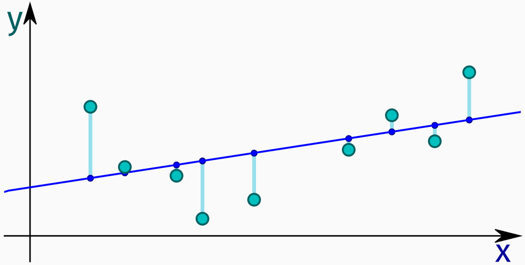
Fully connected neural networks, aka. multi-layer perceptrons





MLP with **1 hidden neuron**.
This is a piecewise linear model
with at most 1 hinge.
The optimizer doesn't use them all.

Fully connected neural networks, aka. multi-layer perceptrons

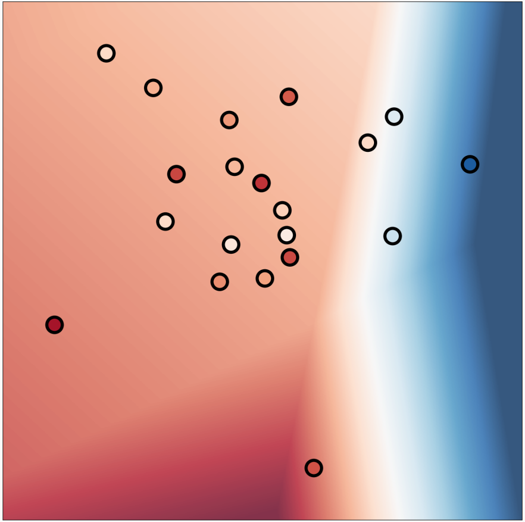


MLP with **10 hidden neurons**.

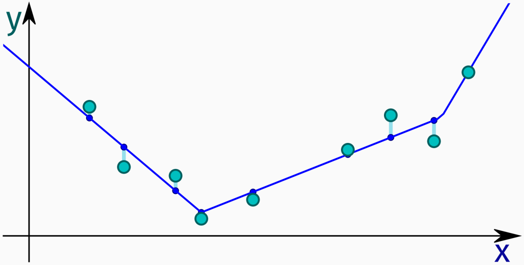
This is a piecewise linear model

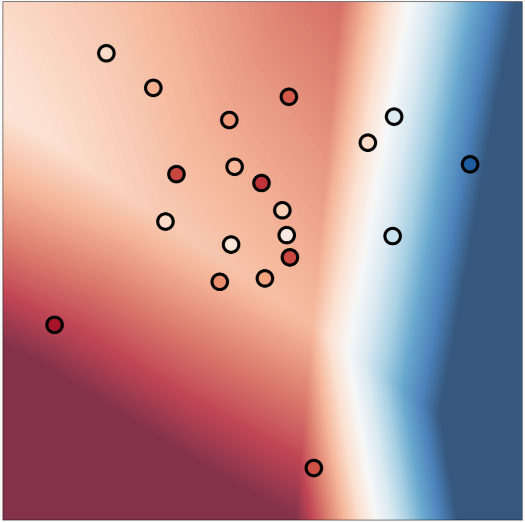
with at most 10 hinges.

The optimizer doesn't use them all.



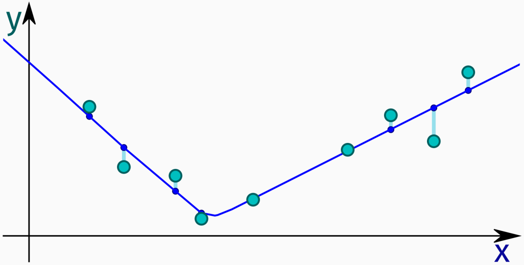
Fully connected neural networks, aka. multi-layer perceptrons





MLP with **20 hidden neurons**.
This is a piecewise linear model
with at most 20 hinges.
The optimizer doesn’t use them all.

Fully connected neural networks, aka. multi-layer perceptrons

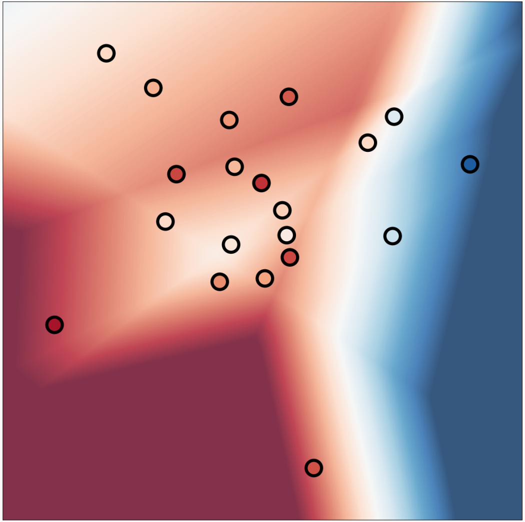


MLP with **50 hidden neurons**.

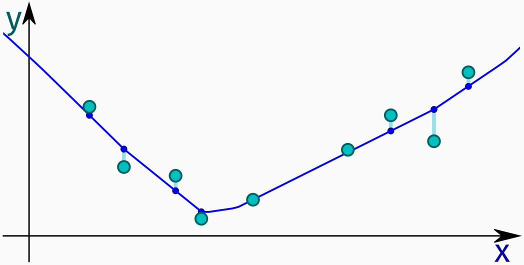
This is a piecewise linear model

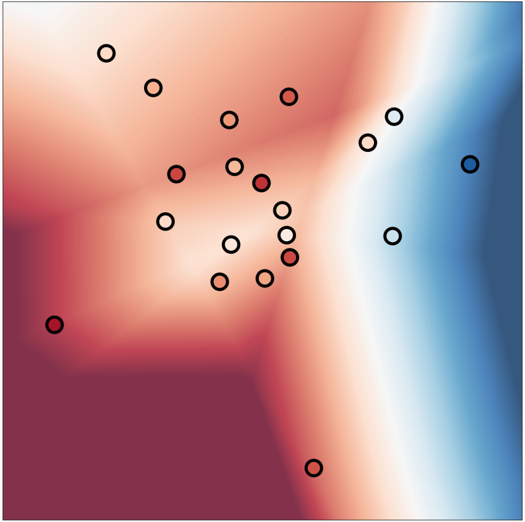
with at most 50 hinges.

The optimizer doesn't use them all.



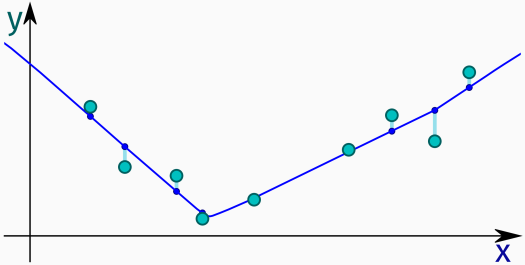
Fully connected neural networks, aka. multi-layer perceptrons

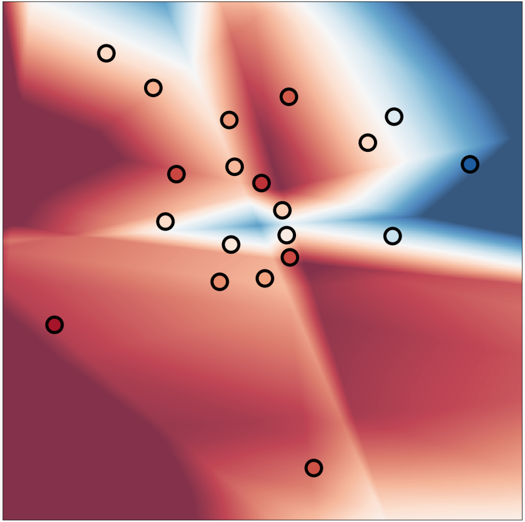




MLP with **100 hidden neurons**.
This is a piecewise linear model
with at most 100 hinges.
The optimizer doesn't use them all.

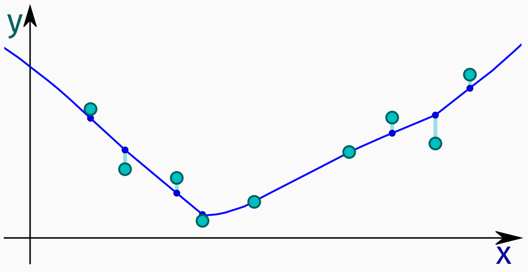
Fully connected neural networks, aka. multi-layer perceptrons

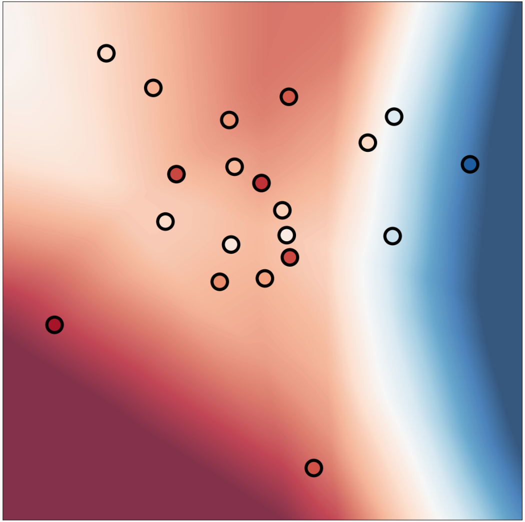




Deeper MLP with 2 hidden layers and **100 + 100 hidden neurons**, i.e. at most 100 x 100 hinges. The **non-convex, stochastic** optimization is **unreliable** and not reproducible.

Fully connected neural networks, aka. multi-layer perceptrons





Deeper MLP with 3 hidden layers and

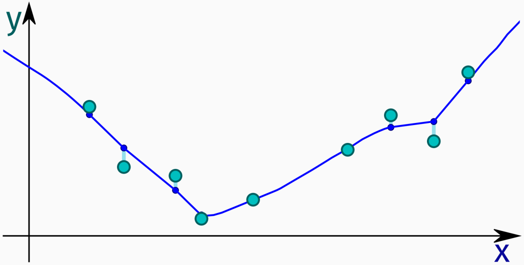
100 + 100 + 100 hidden neurons,

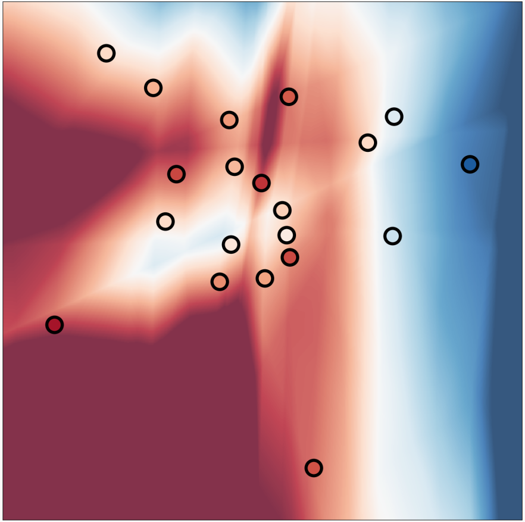
i.e. at most 100 x 100 x 100 hinges.

The **non-convex, stochastic** optimization is

unreliable and not reproducible.

Fully connected neural networks, aka. multi-layer perceptrons





Deeper MLP with 4 hidden layers and
100 + 100 + 100 + 100 hidden neurons,
i.e. at most 100 x 100 x 100 x 100 hinges.

Starting to look like a smooth origami ;-)

(Vanilla, fully connected) neural networks – strengths and weaknesses

- **Modular** and easy to extend.
- Simplest way of implementing **high-dimensional piecewise linear** models.
- Extremely **well-supported** on CPU and GPU: PyTorch, TensorFlow…

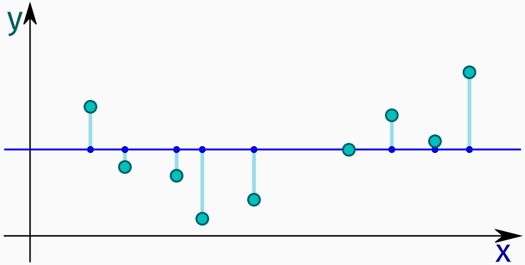
Unfortunately, the optimization of the "neural" weights corresponds to a **non-convex** optimization problem.

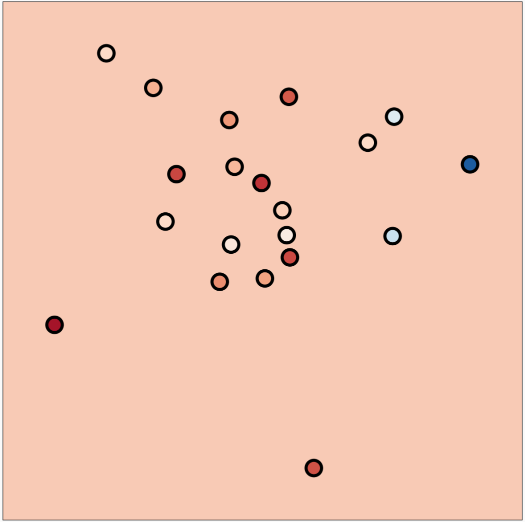
We must rely on **non-deterministic**, stochastic solvers. Performance and smoothness are **not** simply correlated to the number of neurons and layers.

In most applications, this lack of reproducibility and interpretability is a **deal-breaker**.

Polynomial interpolation

Polynomial interpolation



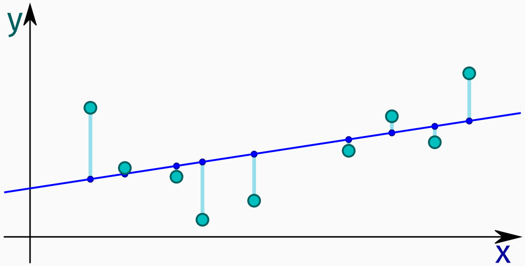


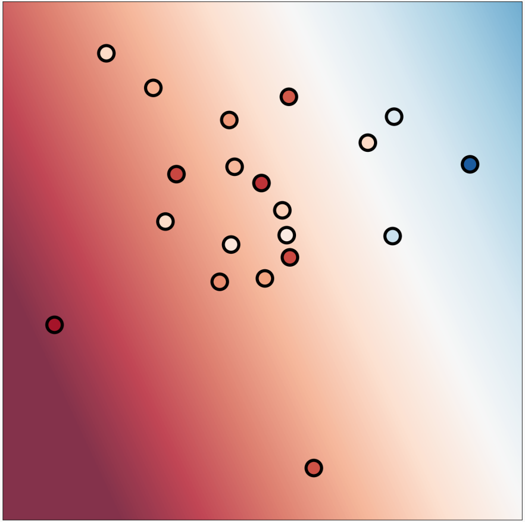
Constant polynomials of **degree 0**:

D=1 – 1 constant.

D=2 – 1 constant.

Polynomial interpolation



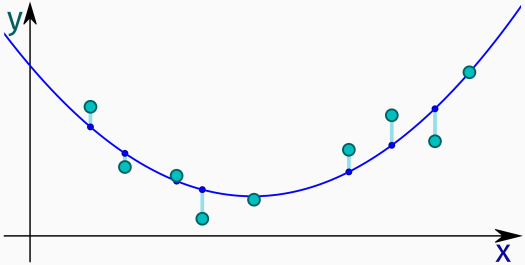


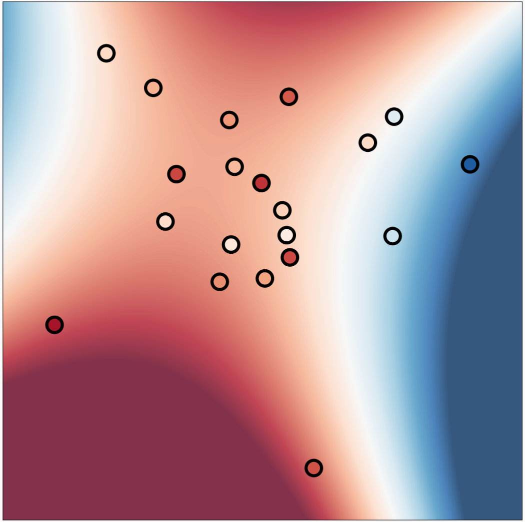
Linear polynomials of **degree 1**:

D=1 – 1, x.

D=2 – 1, x, y.

Polynomial interpolation



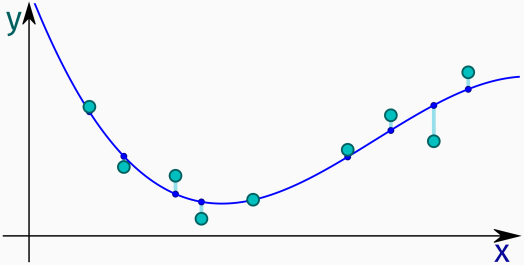


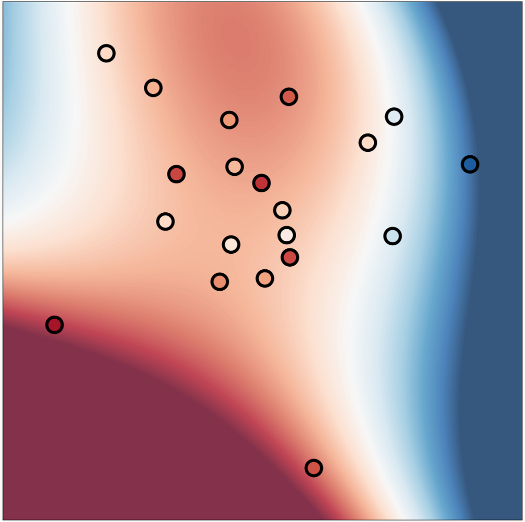
Quadratic polynomials of **degree 2**:

D=1 – 1, x, x^2.

D=2 – 1, x, y, x^2, xy, y^2.

Polynomial interpolation



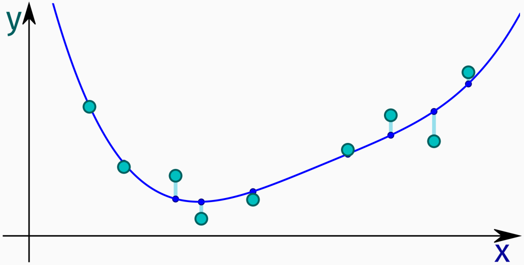


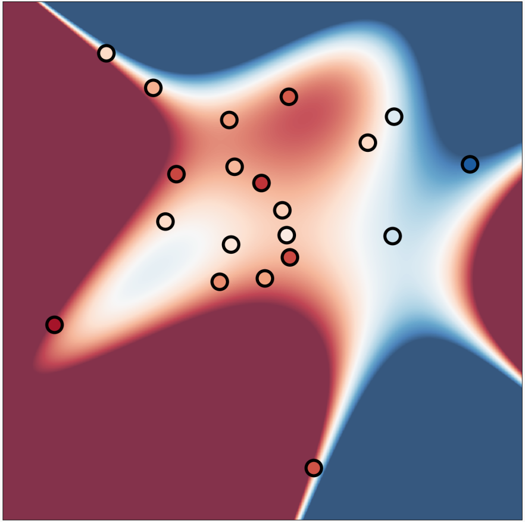
Cubic polynomials of **degree 3**:

D=1 – 1, x, x^2, x^3.

D=2 – 1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3.

Polynomial interpolation





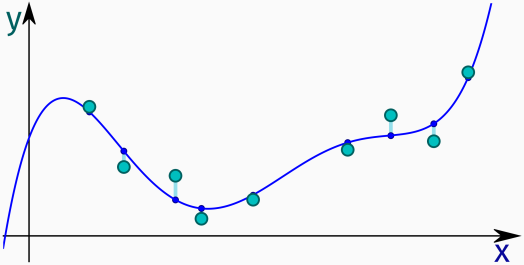
Quartic polynomials of **degree 4**:

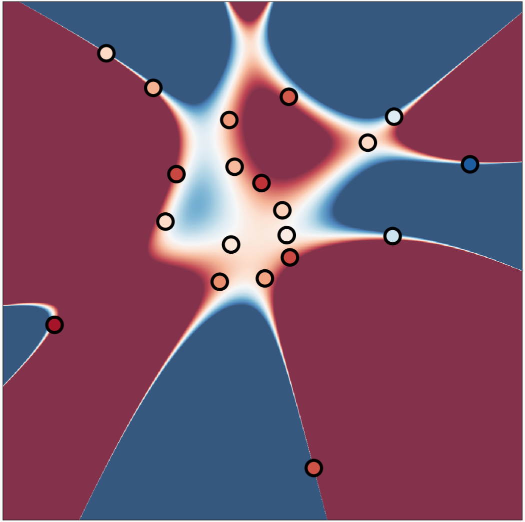
D=1 – 1, x, x^2, x^3, x^4.

D=2 – 1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^4, x^3y, x^2y^2, xy^3, y^4.

Starting to **overfit** in dimension D=2.

Polynomial interpolation





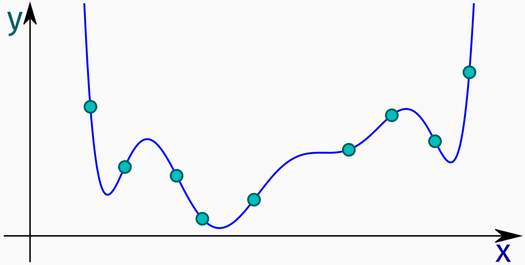
Polynomials of **degree 5**:

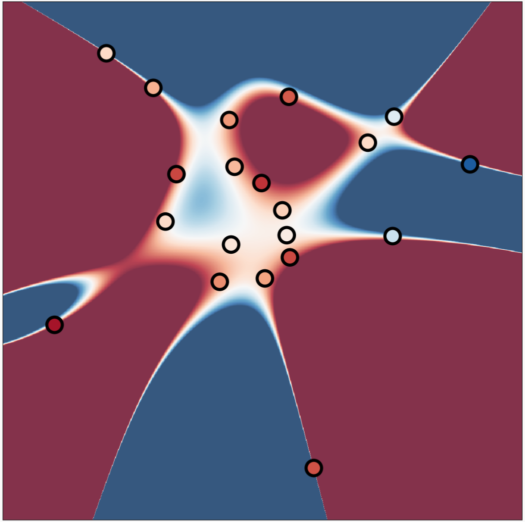
D=1 – $1, x, x^2, x^3, x^4, x^5$.

D=2 – $1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3,$
$x^4, x^3y, \ldots, y^4, x^5, x^4y, \ldots, y^5$.

Full **overfit** in dimension D=2.

Polynomial interpolation





Polynomials of **degree 10**:

D=1 – $1, x, x^2, x^3, x^4, x^5, \ldots, x^{10}$.

D=2 – $1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3,$
$x^4, x^3y, \ldots, y^4, x^5, x^4y, \ldots, y^5, \ldots, y^{10}$.

Full **overfit** in both examples.

Summary of the models that we have seen so far

“Non-parametric” methods:

- **Tree-based** models – robust, but with a bias along the axes.
- **K-Nearest Neighbors** models – isotropic, but requires a good scaling.

“Parametric” methods:

- **Linear** regression – useful, but often too simplistic.
- **Neural networks** – expressive, but unreliable.

Polynomial regression:

- Linear regression with polynomial features.
- Quadratic regression is fine – but we badly **overfit** beyond degree 4-5.

Kernel interpolation

User-specified smoothness

Let's specify directly a linear parametric form for the model:

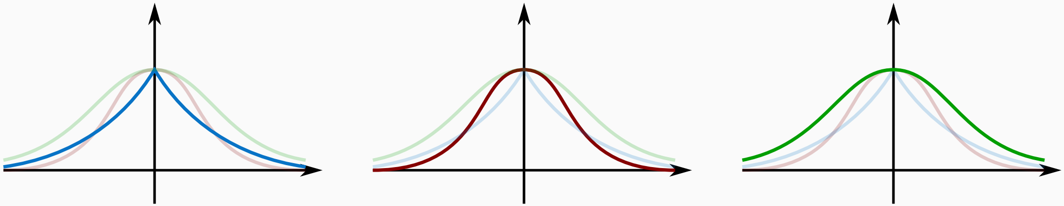
$$F(a_1, \dots, a_J; x) = a_1 F_1(x) + \cdots + a_J F_J(x).$$

In practice, we often use:

$$F(a_j; x) = \sum_j a_j \, k(x - x_j)$$

and say that $k(x - y)$ is the **kernel** of our method.

Some common kernels



Exponential

Gaussian

Cauchy

Two main criteria: is the kernel **smooth** or **peaky**?

Does the kernel have **compact support** or a **heavy tail**?

How do we choose the weights?

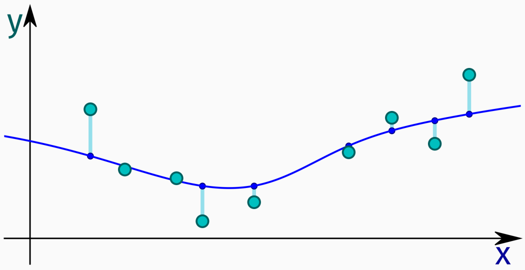
First method – just use a fraction instead of a linear combination:

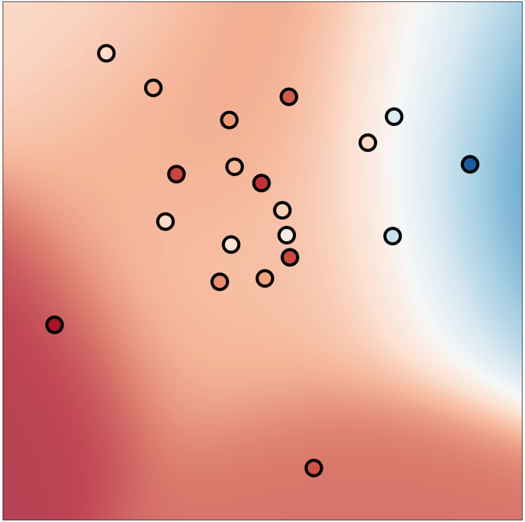
$$F(x) \;=\; \frac{\sum_j k(x - x_j)\, y_j}{\sum_j k(x - x_j)}$$

The Nadaraya–Watson method assumes that $k(x - y)$ takes positive values.

It corresponds to a **barycentric interpolation** between the values y_j,
with weights that are proportional to $k(x - x_j)$.

Nadaraya–Watson interpolation



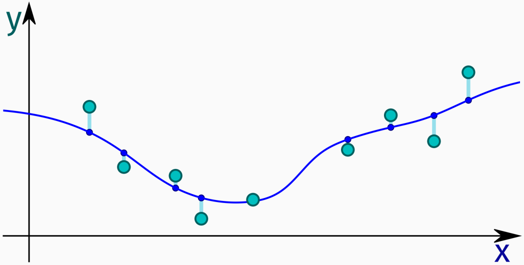


Smooth, local Gaussian kernel with **σ = 0.2**

$$k(x, y) = \exp(-\|x - y\|^2 / 2\sigma^2).$$

Smooth local averaging
on the unit interval and square.

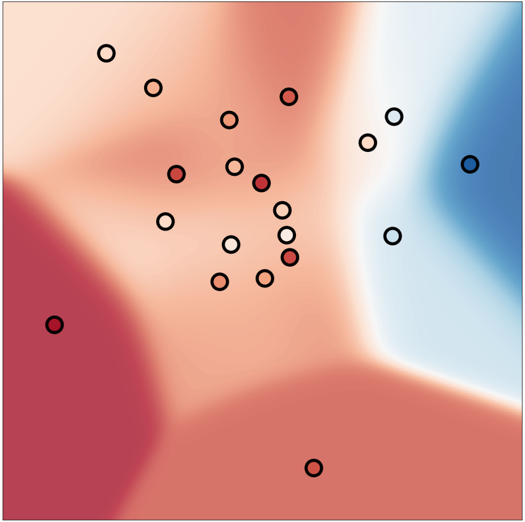
Nadaraya–Watson interpolation



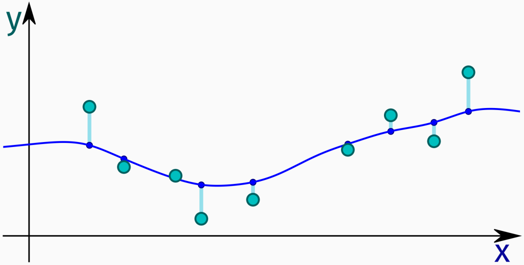
Smooth, **local** Gaussian kernel with **σ = 0.1**

$$k(x, y) = \exp(-\|x - y\|^2 / 2\sigma^2).$$

Sharper, K-NN-like decision boundaries.



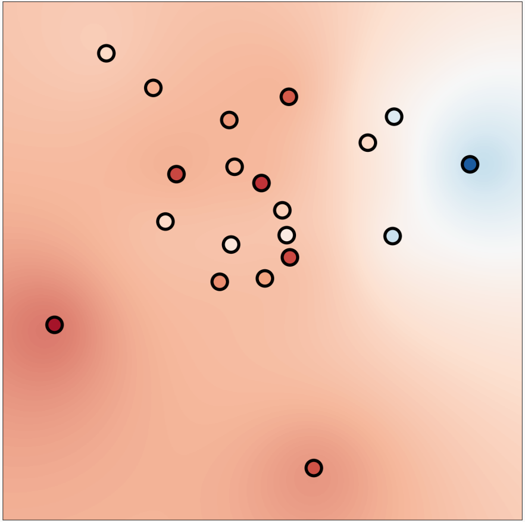
Nadaraya–Watson interpolation



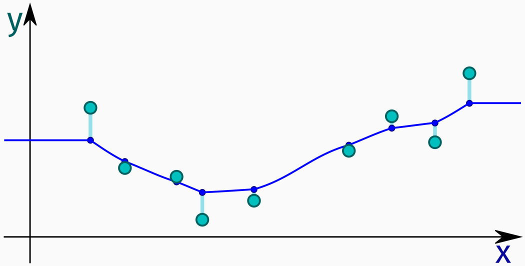
Heavy-tail Cauchy kernel with **σ = 0.1**

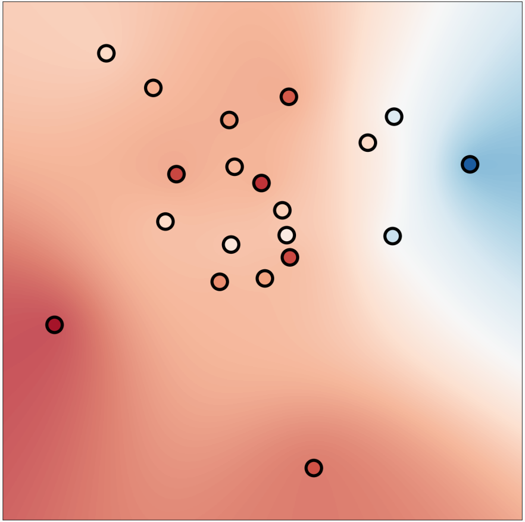
$$k(x, y) = 1 \,/\, (1 + \|x - y\|^2 / \sigma^2).$$

Dampened towards the global average.



Nadaraya–Watson interpolation



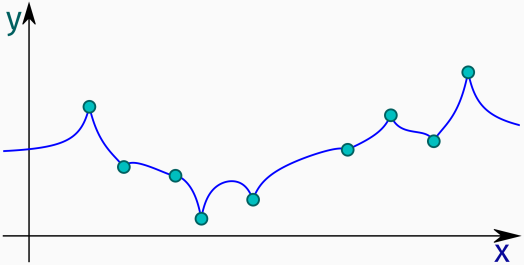


Pointy exponential kernel with **σ = 0.1**

$$k(x, y) = \exp(-\|x - y\|/\sigma).$$

Closer fit to the training data.

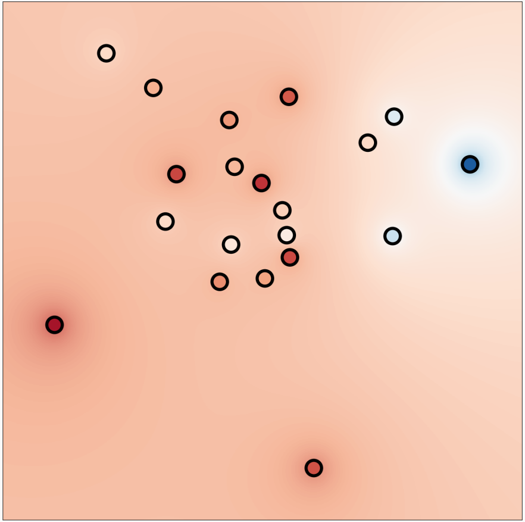
Nadaraya–Watson–Shepard interpolation – Inverse Distance Weighting



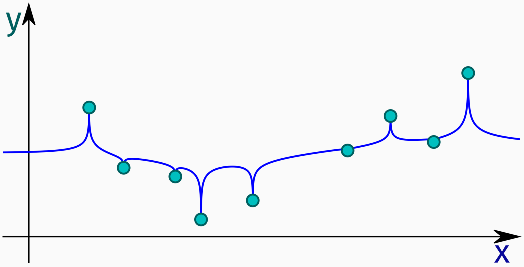
Singular Shepard kernel

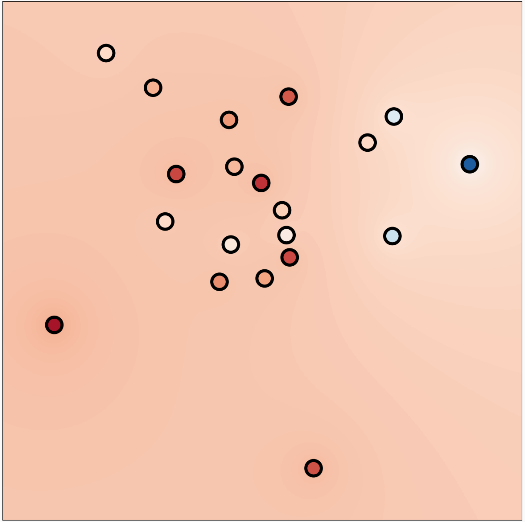
$$k(x, y) = 1 \,/\, \|x - y\|.$$

Perfect fit to the training data.



Nadaraya–Watson–Shepard interpolation – Inverse Distance Weighting



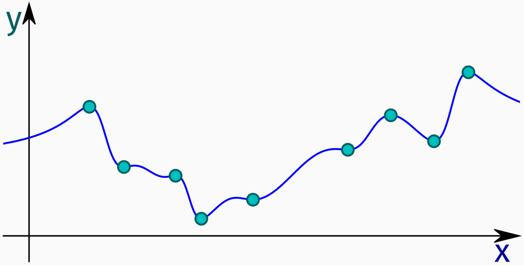


Singular and **heavy-tail** Shepard kernel

$$k(x, y) = 1 \,/\, \sqrt{\|x - y\|}.$$

Perfect fit to the training data,
dampening to the average value elsewhere.

Nadaraya–Watson–Shepard interpolation – Inverse Distance Weighting

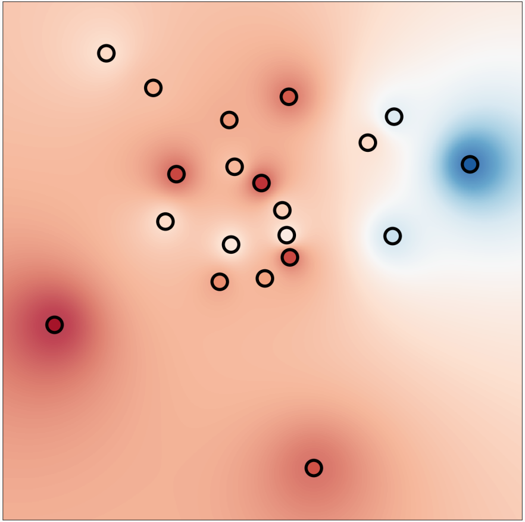


Highly singular Shepard kernel

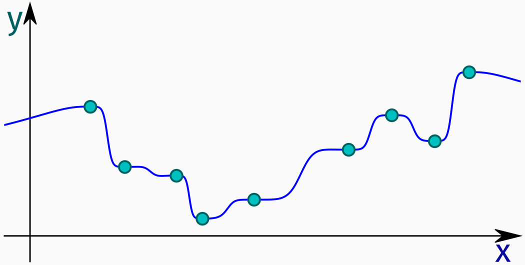
$k(x, y) = 1 \, / \, \|x - y\|^2.$

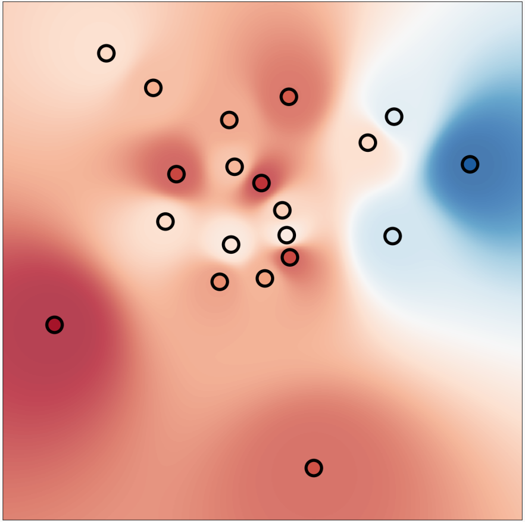
Perfect fit to the training data,

close to a **linear** interpolation.



Nadaraya–Watson–Shepard interpolation – Inverse Distance Weighting





Very highly singular Shepard kernel

$$k(x, y) = 1 \,/\, \|x - y\|^4.$$

Perfect fit to the training data,
close to a **nearest neighbor** interpolation.

Second method: solve a linear system

$$
\begin{aligned}
F(x_1) &= \mathbf{a_1}\,\varphi(x_1 - x_1) + \cdots + \mathbf{a_N}\,\varphi(x_1 - x_N) \simeq y_1 \\
F(x_2) &= \mathbf{a_1}\,\varphi(x_2 - x_1) + \cdots + \mathbf{a_N}\,\varphi(x_2 - x_N) \simeq y_2 \\
\vdots &= \qquad \vdots \qquad\quad + \ddots + \qquad \vdots \qquad\quad \simeq \vdots \\
F(x_N) &= \mathbf{a_1}\,\varphi(x_N - x_1) + \cdots + \mathbf{a_N}\,\varphi(x_N - x_N) \simeq y_N
\end{aligned}
$$

Linear system $\Phi\,\mathbf{a} = y$

Approximate kernel regression

Enforcing a **perfect fit** to the data may not be reasonable.

Instead, we target a trade-off between accuracy and smoothness:

$$\min_{a} \; \|\Phi a - y\|^2 + \mathrm{Reg}(a).$$

Popular regularization terms are **convex**:

- **Ridge**: $\alpha\|a\|^2 = \alpha(a_1^2 + \cdots + a_N^2)$.
- **Lasso**: $\lambda\|a\|_1 = \lambda(|a_1| + \cdots + |a_N|)$.
- **Elastic Net**: $\lambda\|a\|_1 + \alpha\|a\|^2$.

Kernel ridge regression

$$\begin{aligned}
\min_{\mathsf{a}} \ \|\Phi\,\mathsf{a}-\mathsf{y}\|^2 \ + \ \alpha\,\|\mathsf{a}\|^2 \ &= \ (\mathsf{a}^\top\Phi^\top-\mathsf{y}^\top)(\Phi\mathsf{a}-\mathsf{y})+\alpha\,\mathsf{a}^\top\mathsf{a} \\
&= \ \mathsf{a}^\top(\Phi^\top\Phi+\alpha\,\mathrm{Id}_N)\,\mathsf{a}-2\,\mathsf{y}^\top\Phi\mathsf{a}+\mathsf{y}^\top\mathsf{y}
\end{aligned}$$

$$\begin{aligned}
\Longrightarrow \ \mathsf{a} \ &= \ (\Phi^\top\Phi+\alpha\,\mathrm{Id}_N)^{-1}\,\Phi^\top\mathsf{y} \\
&= \ \Phi^\top(\Phi\Phi^\top+\alpha\,\mathrm{Id}_N)^{-1}\mathsf{y}
\end{aligned}$$

$$\Longrightarrow \ F(x) \ = \ \Phi\,\mathsf{a} \ = \ \Phi\Phi^\top(\Phi\Phi^\top+\alpha\,\mathrm{Id}_N)^{-1}\,\mathsf{y}\,.$$

A fundamental object appears:

the symmetric, positive, semidefinite **kernel matrix** $K=\Phi\Phi^\top$.

The kernel trick

$$K = \Phi\Phi^\top \quad \text{i.e.} \quad K(x_i, x_j) \;=\; \langle\, \Phi(x_i), \Phi(x_j) \,\rangle \;=\; \sum_{s=1}^{N} \varphi(x_i - x_s)\, \varphi(x_j - x_s)$$

This may be **expensive**: N terms for every coefficient of K.

Fortunately, we may use the continuous limit instead:

$$k(x_i, x_j) \;=\; \int_X \varphi(x_i - x)\, \varphi(x_j - x)\, dx$$

This dot product between **two translated copies** of φ is often known in **closed form**.

Computational perspective

We consider functions $k(x_i - x_j)$ that can be written as the previous integral for a suitable function φ.

Criterion: if the **Fourier** transform $\widehat{k}(\omega)$ is real-valued and positive, then $\widehat{\varphi}(\omega) = \sqrt{\widehat{k}(\omega)}$ works.

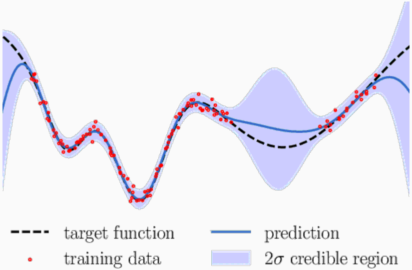
Then, **kernel ridge regression** simply relies on the model:

$$F(x) \;=\; K\,(K + \alpha\,\mathrm{Id}_N)^{-1}\,y.$$

On GPUs, we may solve this linear system efficiently using:

- **KeOps** – bruteforce methods scale to $N = 1{,}000{,}000$ in seconds.
- **FalkonML** – approximate methods scale to $N = 1{,}000{,}000{,}000$ in hours.

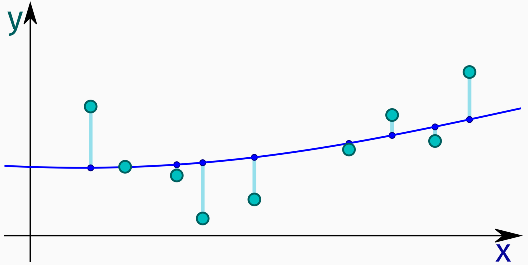
Kriging, Gaussian process regression [Lec18]



Kernel ridge regression has a **rich history** in applied mathematics.
It is especially popular in geostatistics to estimate smooth terrain models:
the approximation parameter α controls the **nugget** effect.

This theory is also behind **Sobolev** norms and **Gaussian** processes…
More about this in the MVA Lecture 6 on probability distributions!

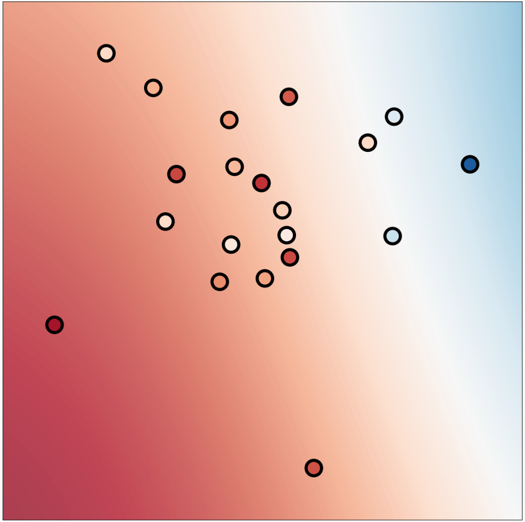
Kernel ridge regression – with $\alpha = 0.1$



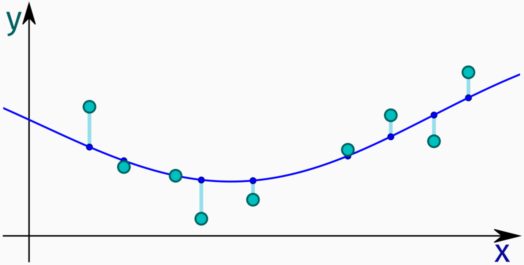
Smooth, **global** Gaussian kernel with $\boldsymbol{\sigma = 1.0}$

$$k(x, y) = \exp(-\|x - y\|^2 / 2\sigma^2).$$

Only models a **global** linear trend.



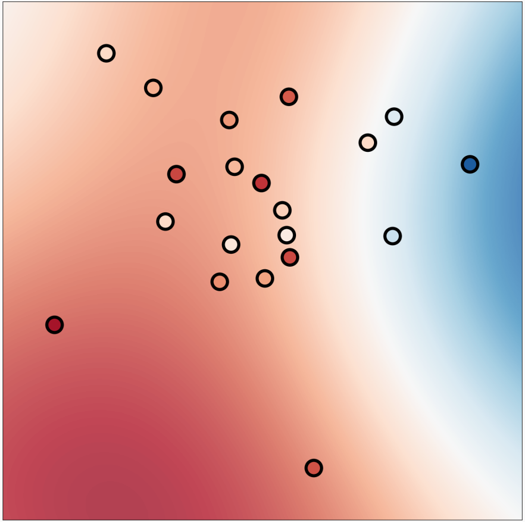
Kernel ridge regression – with α = 0.1



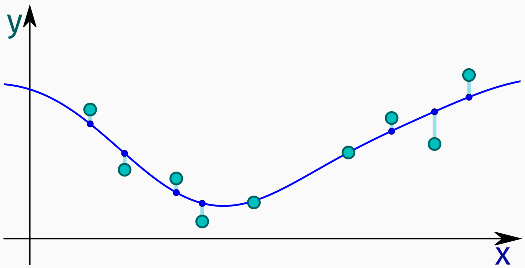
Smooth Gaussian kernel with **σ = 0.5**

$$k(x, y) = \exp(-\|x - y\|^2 / 2\sigma^2).$$

Starts to discern **different regions**.



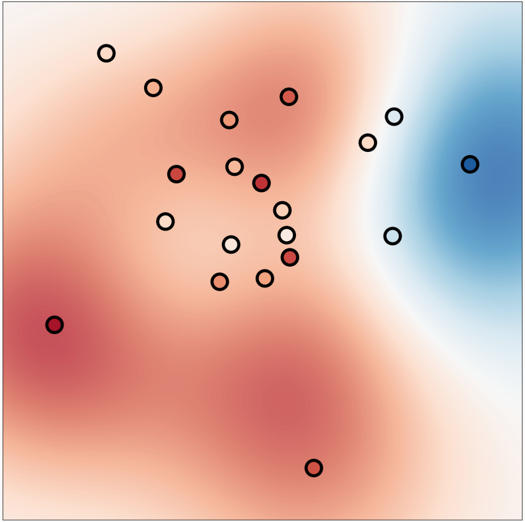
Kernel ridge regression – with α = 0.1



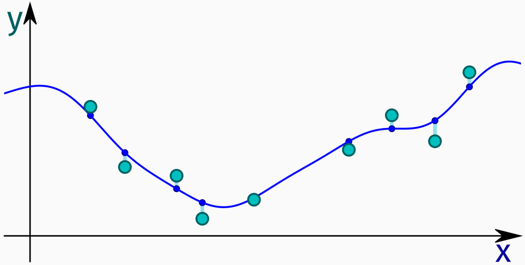
Smooth Gaussian kernel with **σ = 0.2**

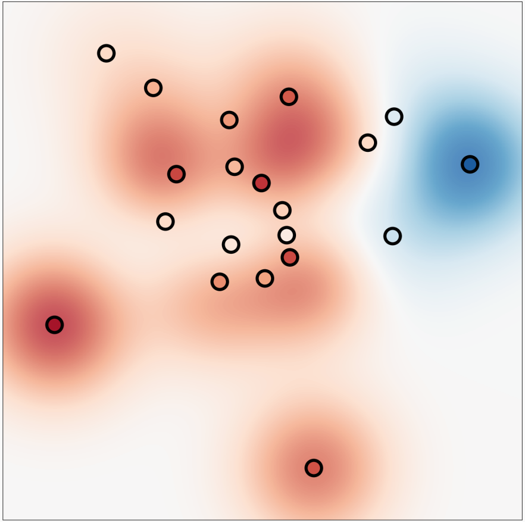
$$k(x, y) = \exp(-\|x - y\|^2 / 2\sigma^2).$$

Well-suited to the **sampling density**.



Kernel ridge regression – with α = 0.1



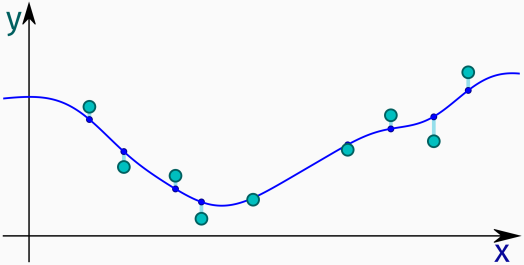


Smooth, **local** Gaussian kernel with **σ = 0.1**

$$k(x, y) = \exp(-\|x - y\|^2 / 2\sigma^2).$$

Overfits on individual sample values.

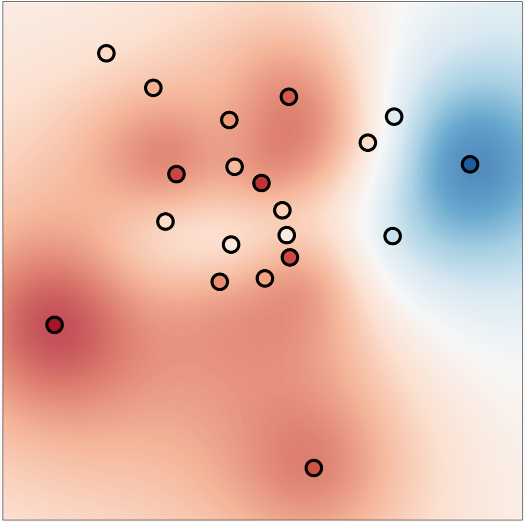
Kernel ridge regression – with $\alpha = 0.1$



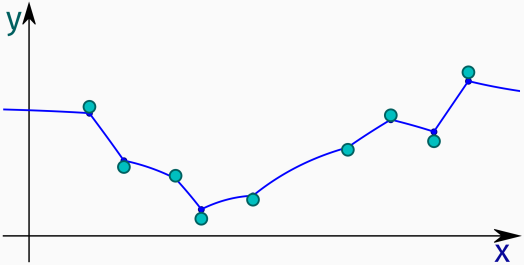
Heavy-tail Cauchy kernel with **σ = 0.2**

$$k(x, y) = 1 \,/\, (1 + \|x - y\|^2 / \sigma^2).$$

Extrapolates with more confidence.



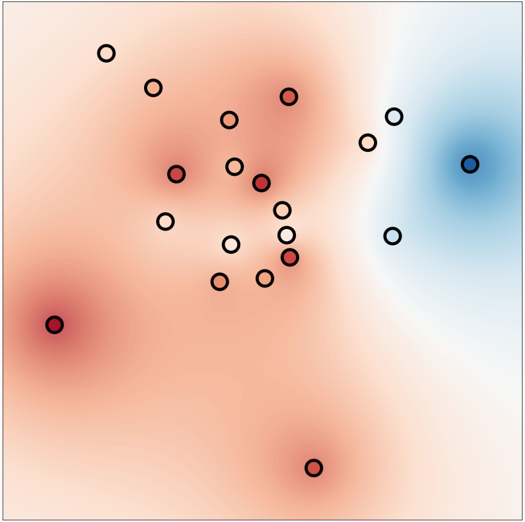
Kernel ridge regression – with α = 0.1



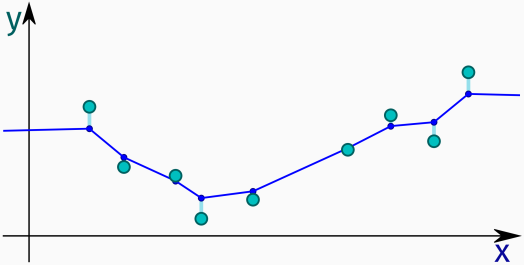
Pointy exponential kernel with **σ = 0.2**

$$k(x, y) = \exp(-\|x - y\|/\sigma).$$

Closer fit to the training data.



Kernel ridge regression – with α = 0.1

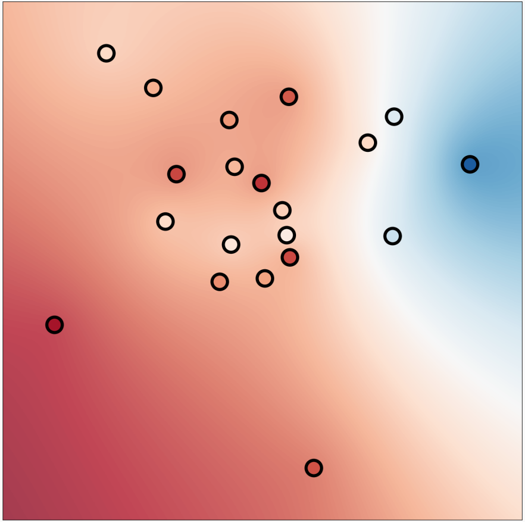


Pointy, **global** distance kernel

$$k(x, y) = -\|x - y\|.$$

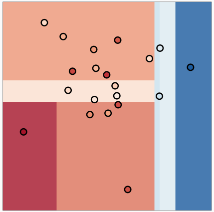
Models both **local** and **global** trends.

Excellent parameter-free baseline.

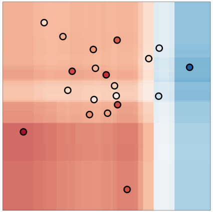


Conclusion

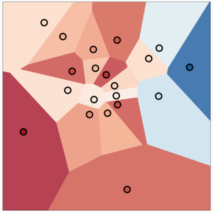
Numerous regression models... But what about the curse of dimensionality?



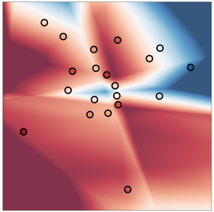
Decision tree.



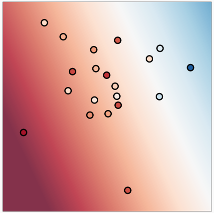
Random forest.



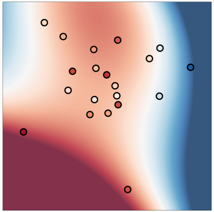
Nearest neighbors.



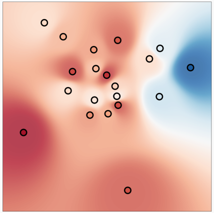
Neural network.



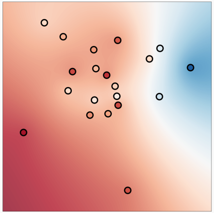
Linear.



Cubic.



Shepard.



Kernel.

References

References i

Olivier Ecabert, Jochen Peters, and Matthew Walker.

Segmentation of the heart and great vessels in ct images using a model-based adaptation framework.

Medical Image Analysis, (15):863–876, 2011.

Florent Leclercq.

Bayesian optimization for likelihood-free cosmological inference.

Physical Review D, 98(6):063511, 2018.

Withings.

About body mass index.

http://en.wikipedia.org/wiki/JPEG_2000www.withings.com.