Theory of Statistical Learning
Part II

Damien Garreau

Université Côte d'Azur

2021

Outline

1. Linear predictors
   - Linear classification
   - Linear regression
   - Ridge regression
   - Polynomial regression
   - Logistic regression
2. Tree-based classifiers
   - Partition rules
   - Random forests
3. Boosting
   - Adaboost
   - XGBoost
4. Nearest neighbors

1. Linear predictors

1.1. Linear classification

Linear functions

- $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$
- thus $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d})^\top$
- we consider no bias term (otherwise *affine*):

$$\{h : x \mapsto w^\top x, w \in \mathbb{R}^d\}.$$

- **Reminder:** given two vectors $u, v \in \mathbb{R}^d$,

$$\langle u, v \rangle = u^\top v = \sum_{j=1}^{d} u_i v_i .$$

- binary classification: 0-1 loss, $\mathcal{Y} = \{-1, +1\}$
- **Important:** compose h with $\phi : \mathbb{R} \to \mathcal{Y}$ (typically the sign)

The sign function

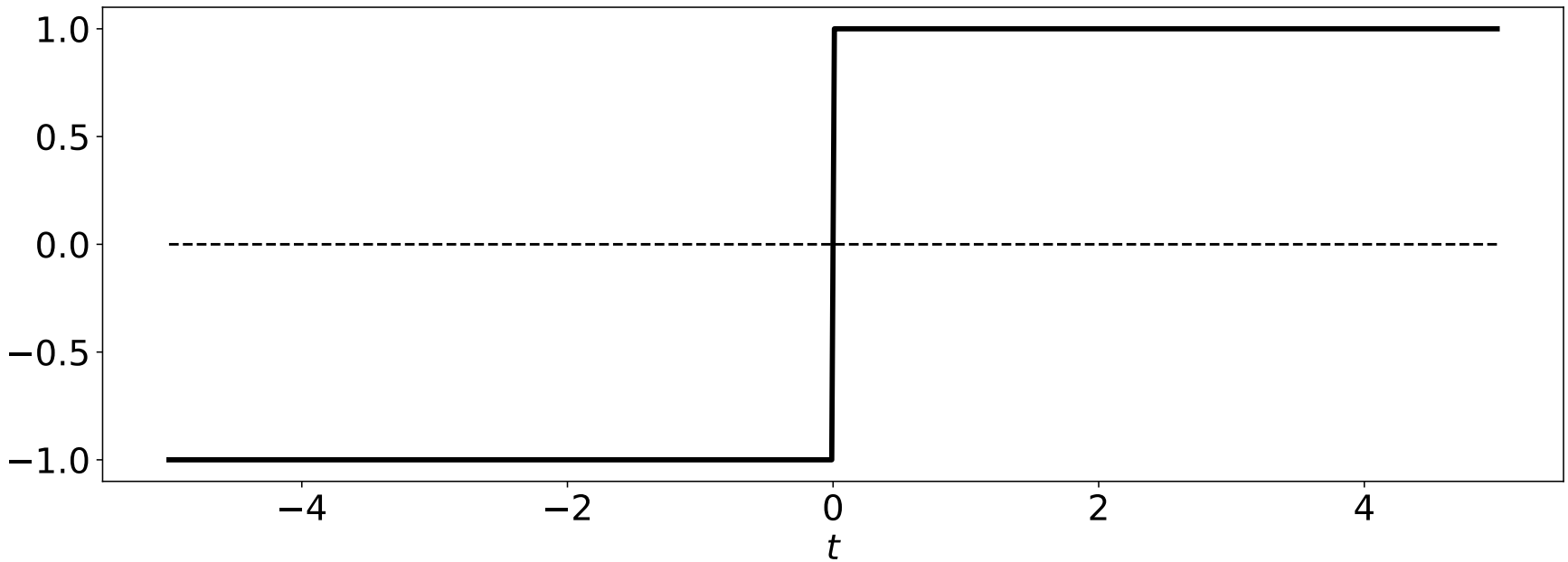


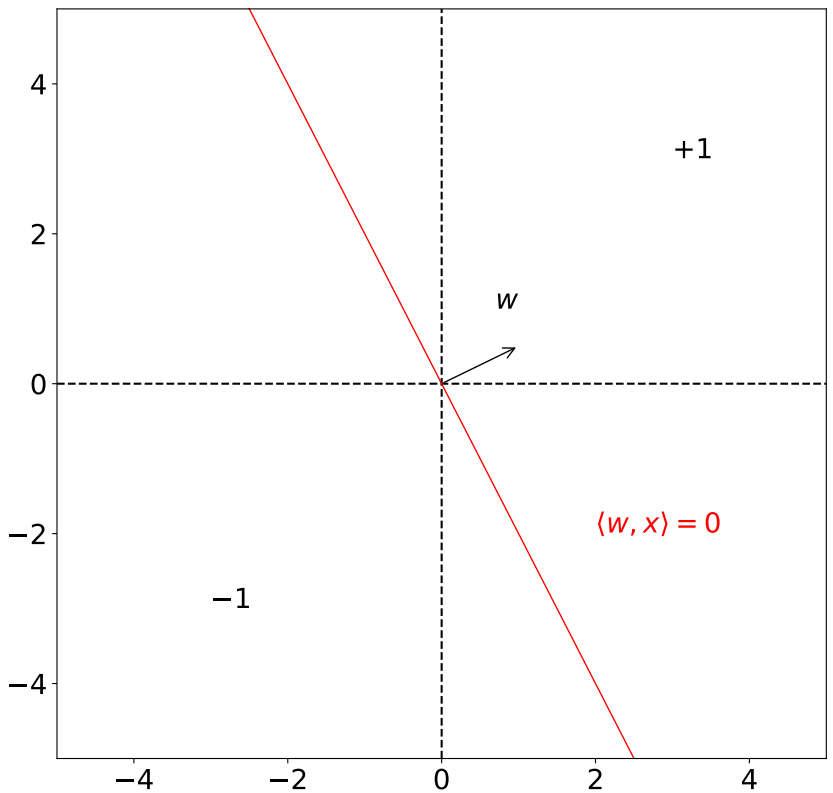
Figure: the sign function

Halfspaces

- thus our function class is

$$\mathcal{H} = \{x \mapsto \operatorname{sign}\left(w^\top x\right), w \in \mathbb{R}^d\}.$$

- gives label $+1$ to vector pointing in the same direction as w



VC dimension of halfspaces

Proposition: the VC dimension of halfspaces in dimension d is exactly $d + 1$.

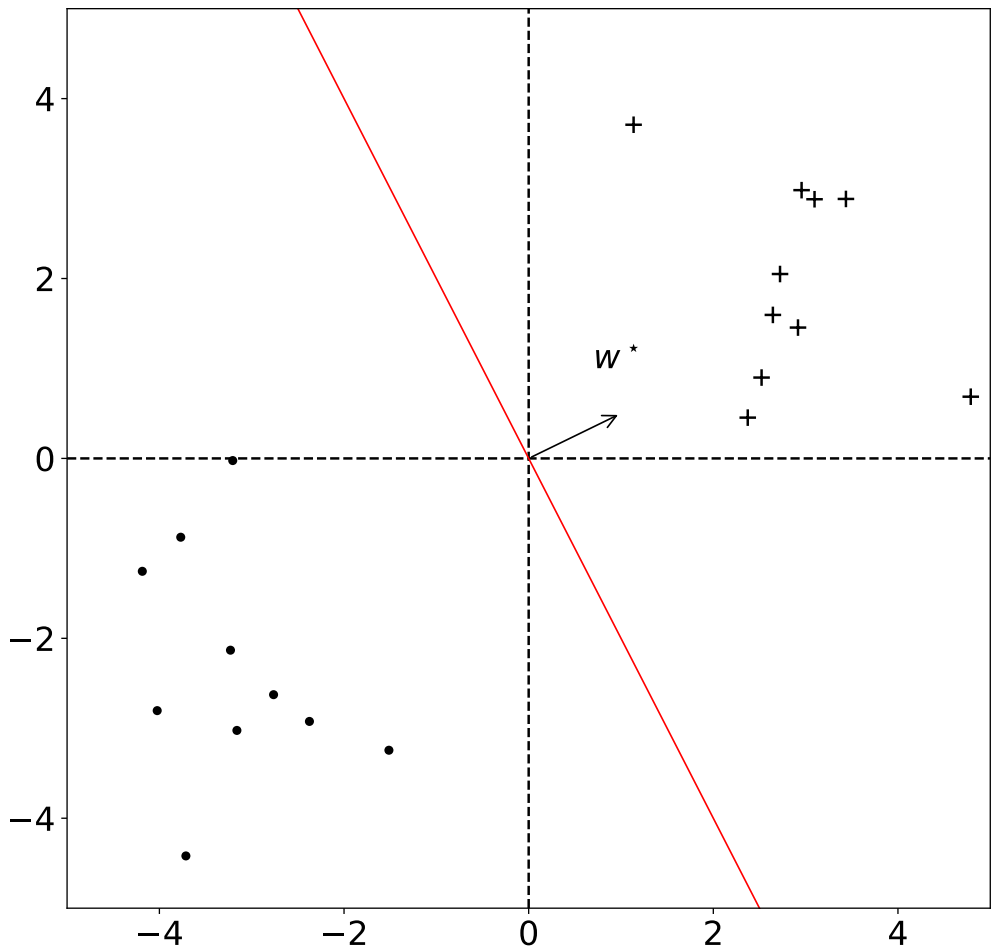
- **Consequence:** $\mathcal{H}$ is PAC learnable with sample complexity

$$\Omega\left(\frac{d + \log(1/\delta)}{\varepsilon}\right).$$

Linearly separable data

- **Important assumption:** data is linearly separable
- that is, there is a $w^\star \in \mathbb{R}^d$ such that

$$y_i = \operatorname{sign}\left(\langle w^\star, x_i \rangle\right) \quad \forall 1 \leq i \leq n\,.$$



Linear programming

- **Empirical risk minimization:** recall that we are looking for w such that

$$\hat{\mathcal{R}}_S(w) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{y_i \neq \mathrm{sign}(w^\top x_i)}$$

  is minimal
- **Question:** how to solve this?
- we want $y_i = \mathrm{sign}\left(w^\top x_i\right)$ for all $1 \leq i \leq n$
- equivalent formulation: $y_i \langle w, x_i \rangle > 0$
- we know that there is a vector that satisfies this condition $(w^\star)$
- let us set $\gamma = \min_i \{y_i \langle w^\star, x_i \rangle\}$ and $\overline{w} = w^\star / \gamma$
- we have shown that there is a vector such that $y_i \langle \overline{w}, x_i \rangle \geq 1$ for any $1 \leq i \leq n$ (and it is an ERM)

Linear programming, ctd.

- define the matrix $A \in \mathbb{R}^{n \times d}$ such that

$$A_{i,j} = y_i x_{i,j} \, .$$

- **Intuition:** observations $\times$ labels
- remember that we have the ± 1 label convention
- define $v = (1, \ldots, 1)^\top \in \mathbb{R}^n$
- then we can rewrite the above problem as

$$\text{maximize} \quad \langle u, w \rangle \text{ subject to } Aw \leq v \, ,$$

  with $u = 0$ for instance
- we call this sort of problems **linear programs**[1]
- solvers readily available, e.g., `scipy.optimize.linprog` if you use Python

[1] Boyd, Vandenberghe, *Convex optimization*, Cambridge University Press, 2004

The perceptron

- another possibility: the *perceptron*[2]
- **Idea:** iterative algorithm that constructs $w^{(1)}, w^{(2)}, \ldots, w^{(T)}$
- update rule: at each step, find i that is misclassified and set

$$w^{(t+1)} = w^{(t)} + y_i x_i \,.$$

- **Question:** why does it work?
- pushes w in the right direction:

$$y_i \langle w^{(t+1)}, x_i \rangle = y_i \langle w^{(t)} + y_i x_i, x_i \rangle = y_i \langle w^{(t)}, x_i \rangle + \|x_i\|^2$$

- remember, we want $y_i \langle w, x_i \rangle > 0$ for all i

[2] Rosenblatt, *The perceptron, a perceiving and recognizing automaton*, tech report, 1957

1.2. Linear regression

Least squares

- regression $\Rightarrow$ squared-loss function

$$\ell(y, y') = (y - y')^2 \,.$$

- still looking at linear functions:

$$\mathcal{H} = \{h : x \mapsto \langle w, x \rangle \text{ s.t. } w \in \mathbb{R}^d\} \,.$$

- empirical risk in this context:

$$\hat{\mathcal{R}}_S(h) = \frac{1}{n} \sum_{i=1}^{n} (w^\top x_i - y_i)^2 = F(w) \,.$$

- also called **mean squared error**
- empirical risk minimization: we want to minimize $w \mapsto F(w)$ with respect to $w \in \mathbb{R}^d$
- F is a convex, smooth function

Least squares, ctd.

- let us compute the gradient of F:

$$\begin{aligned}
\frac{\partial F}{\partial w_j}(w) &= \frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial w_j}(w^\top x_i - y_i)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n} 2\cdot\frac{\partial}{\partial w_j}\left(w^\top x_i - y_i\right)\cdot\left(w^\top x_i - y_i\right) \\
&= \frac{1}{n}\sum_{i=1}^{n} 2\cdot\frac{\partial}{\partial w_j}\left(\cdots + w_j x_{i,j} + \cdots - y_i\right)\cdot\left(w^\top x_i - y_i\right) \\
\frac{\partial F}{\partial w_j}(w) &= \frac{2}{n}\sum_{i=1}^{n} x_{i,j}\cdot\left(w^\top x_i - y_i\right).
\end{aligned}$$

Least squares, ctd.

- it is more convenient to write $\nabla F(w) = 0$ in matrix notation
- define $X \in \mathbb{R}^{n \times d}$ the matrix such that line i of X is observation x_i
- one can check that, for any $1 \leq j, k \leq d$,

$$\left(X^\top X\right)_{j,k} = \sum_{i=1}^{n} x_{i,j} x_{i,k} \,.$$

- thus

$$\begin{aligned}
\left(X^\top X w\right)_j &= \sum_{k=1}^{d} \left(X^\top X\right)_{j,k} w_k \\
&= \sum_{k=1}^{d} \sum_{i=1}^{n} x_{i,j} x_{i,k} w_k \\
&= \sum_{i=1}^{n} x_{i,j} w^\top x_i \,.
\end{aligned}$$

Least squares, ctd.

- thus, if we define

$$A = X^\top X = \sum_{i=1}^{n} x_i x_i^\top \in \mathbb{R}^{d\times d} \text{ and } b = X^\top y = \sum_{i=1}^{n} y_i x_i \in \mathbb{R}\,,$$

  solving $\nabla F(w) = 0$ is equivalent to solving

$$Aw = b\,.$$

- if A is invertible, straightforward:

$$\boxed{\hat{w} = A^{-1} b}$$

- computational cost: $\mathcal{O}\left(d^3\right)$ (inversion is actually a bit less)
- what happens when A is not invertible?

Singular value decomposition

- since A is symmetric, it has an eigendecomposition

$$A = VDV^\top ,$$

  with $D \in \mathbb{R}^d$ diagonal and V orthonormal
- define D^+ such that

$$D^+_{i,i} = 0 \text{ if } D_{i,i} = 0 \text{ and } D^+_{i,i} = \frac{1}{D_{i,i}} \text{ otherwise.}$$

- define $A^+ = VD^+V^\top$
- then we set

$$\boxed{\hat{w} = A^+ b\,.}$$

Singular value decomposition, ctd.

- why did we do that?
- let v_i denote the ith column of V, then

$$\begin{aligned} A\hat{w} &= AA^{+}b && \text{(definition of } \hat{w}\text{)} \\ &= VDV^{\top}VD^{+}V^{\top}b && \text{(definition of } A^{+}\text{)} \\ &= VDD^{+}V^{\top}b && (V \text{ is orthonormal}) \end{aligned}$$

$$A\hat{w} = \sum_{i:D_{i,i}\neq 0} v_i v_i^{\top} b\,.$$

- in definitive, $A\hat{w}$ is the projection of b onto the span of v_i such that $D_{i,i} \neq 0$
- since the span of these v_i is the span of the x_i and b is in the linear span of the x_i, we have $A\hat{w} = b$
- cost of SVD: $\mathcal{O}\left(dn^2\right)$ if $d > n$ (SVD of X)

Exercise

Exercise: Of course, one does not have to use the squared loss. Instead, we may prefer to use

$$\ell(y, y') = |y - y'| \,.$$

1. show that, for any $c \in \mathbb{R}$,

$$|c| = \min_{a \geq 0} a \quad \text{subject to} \quad a \geq c \text{ and } a \geq -c \,.$$

2. use the previous question to show that ERM with the absolute value loss function is equivalent to minimizing the linear function $\sum_{i=1}^{n} s_i$, where the s_i satisfy linear constraints
3. write it in matrix form, that is, find $A \in \mathbb{R}^{2n \times (n+d)}$, $v \in \mathbb{R}^{d+n}$, and $b \in \mathbb{R}^{2n}$ such that the LP can be written

$$\text{minimize } c^\top v \quad \text{subject to} \quad Av \leq b \,.$$

Correction of the exercise

1. The absolute value is the smallest positive number larger than both c and $-c$ for any real number c.
2. In that case, the empirical risk can be written

$$\hat{\mathcal{R}}_S(w) = \frac{1}{n} \sum_{i=1}^{n} \left| y_i - w^\top x_i \right| .$$

We deduce the result from question 1.

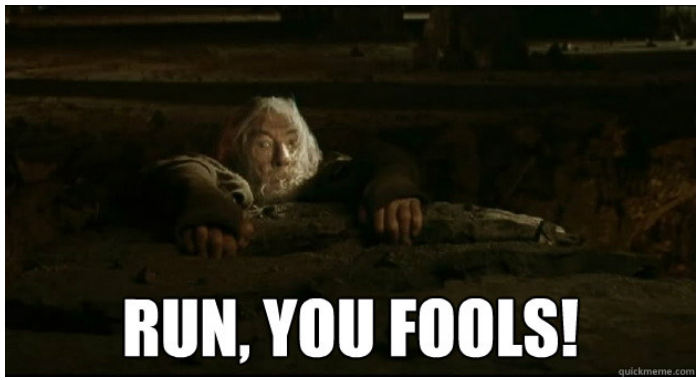
3. One possibility is to define $v = (w_1, \dots, w_d, s_1, \dots, s_n)^\top \in \mathbb{R}^{n+d}$, $c = (0, \dots, 0, 1, \dots, 1)^\top \in \mathbb{R}^{d+n}$, $b = (y_1, \dots, y_n, -y_1, \dots, -y_n)^\top \in \mathbb{R}^{2n}$, and

$$A = \begin{pmatrix} -X & -I_n \\ X & -I_n \end{pmatrix} \in \mathbb{R}^{2n \times (n+d)} ,$$

with $X \in \mathbb{R}^{n \times d}$ the matrix whose lines are the x_is and I_n the identity matrix.

Recap

- **What happens when we invoke** `sklearn.linear_model.LinearRegression` with default parameters?
- `fit_intercept` is True $\rightarrow$ assumes that the data is not centered (our maths are not totally accurate)
- `normalize` is False $\rightarrow$ we are responsible for the normalization of our data
- behind the scenes, calls `scipy.linalg.lstsq` when fitting, which itself calls LAPACK (Linear Algebra PACKage)[3]
- LAPACK is coded in Fortran90



[3] `http://www.netlib.org/lapack/`

1.3. Ridge regression

Ridge regression

- same hypothesis class: linear functions

$$\mathcal{H} = \{h : x \mapsto w^\top x, w \in \mathbb{R}^d\}$$

- squared loss:

$$\ell(y, y') = (y - y')^2 .$$

- **Idea:** regularization:

$$\text{minimize} \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - w^\top x_i)^2 + \lambda \|w\|^2 \right\},$$

with $\|u\|^2 = u_1^2 + \cdots + u_d^2$ and $\lambda > 0$ a *regularization parameter*

Exercise

Exercise: Let $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$ be n given training samples. For any $w \in \mathbb{R}^d$, set

$$F(w) = \frac{1}{n} \sum_{i=1}^{n} (y_i - w^\top x_i)^2 + \lambda \|w\|^2 \ .$$

Notice that F is a convex smooth function.

1. show that the minimizer $\hat{w}$ satisfies

$$\left(X^\top X + n\lambda I_d\right) w = X^\top y \, .$$

2. show that $X^\top X + n\lambda I_d$ is an invertible matrix

Correction of the exercise

1. Let $1 \leq j \leq d$ and let us compute $\partial_j F$:

$$\begin{aligned}\frac{\partial F}{\partial w_j}(w) &= \frac{\partial}{\partial w_j}\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - w^\top x_i)^2\right) + \frac{\partial}{\partial w_j}(\lambda(w_1^2 + \cdots w_d^2)) \\ &= \frac{2}{n}\sum_{i=1}^{n} x_{i,j} \cdot (w^\top x_i - y_i) + 2\lambda w_j \,,\end{aligned}$$

where we used the derivation for the least squares. We deduce the result by setting to zero and multiplying by n.

Correction of the exercise, ctd.

2. By contradiction, suppose that $X^\top X + n\lambda I_d$ is not invertible. Then

$$\det\left(X^\top X + n\lambda I_d\right) = 0\,.$$

In other words, $-n\lambda$ is an eigenvalue of $X^\top X$. Since $X^\top X$ is a symmetric matrix, its spectrum is $\subseteq \mathbb{R}$. Moreover, it is positive definite, thus all eigenvalues are non-negative. Since $\lambda > 0$, we deduce that $-n\lambda$ cannot be an eigenvalue of $X^\top X$ and we can conclude. $\square$

Recap

- **What happens when we invoke** `sklearn.linear_model.Ridge` with default settings?
- `alpha` $= 1 \rightarrow \lambda = 1/n$ with our notation, barely any regularization if n large
- `fit_intercept` is True $\rightarrow$ does not consider centered data (so our analysis is not entirely accurate)
- `normalize` is False $\rightarrow$ we decide whether we normalize our data
- `solver` is auto $\rightarrow$ `sklearn` will decide how to solve the minimization problem depending on the size of the data: the solution could be not exact!
- `tol` $= 0.001 \rightarrow$ tolerance threshold on the residuals

1.4. Polynomial regression

Polynomial regression

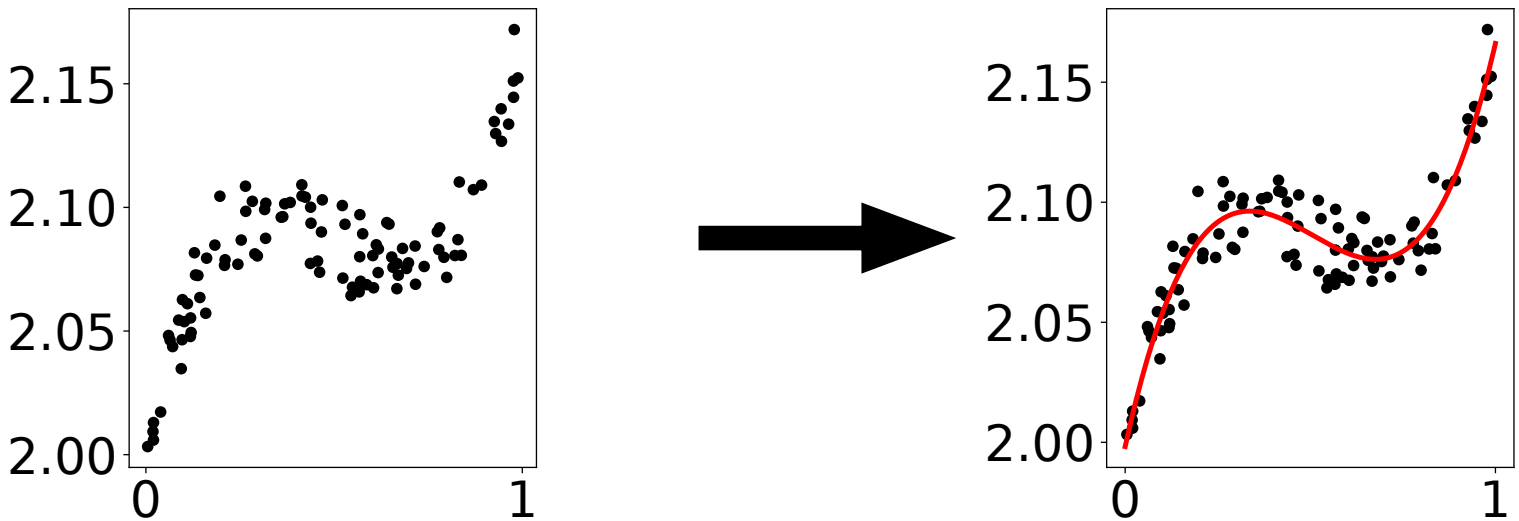
- linear regression is a powerful tool, especially because we can transform the inputs in a non-linear fashion
- **Example:** polynomial regression in $\mathbb{R}$
- inputs $x_1, \ldots, x_n \in \mathbb{R}$
- define the mapping $\phi(x) = (1, x, x^2, \ldots, x^p)^\top$
- then

$$\langle w, \phi(x) \rangle = w_0 + w_1 x + w_2 x^2 + \cdots + w_p x^p \,,$$

  and we can find the best coefficients by linear regression
- `numpy.polyfit` $\rightarrow$ very handy when we want to fit univariate data

Polynomial regression, ctd.

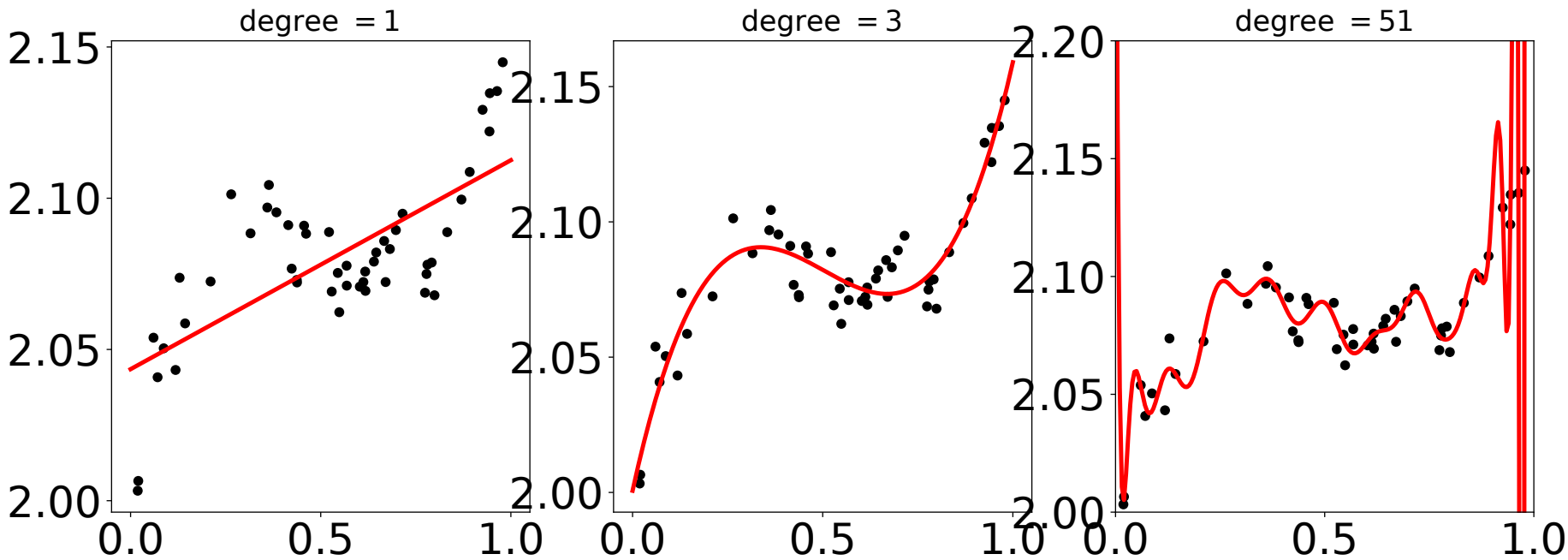
- **Example:** data = degree three polynomial + Gaussian noise with small variance
- fit a degree 3 polynomial:



- **Remark:** in practice, we do not know the degree of the polynomial!

Polynomial regression, ctd.

- typical case of under / overfitting:
  - when degree too low, poor fit
  - when degree too high, wiggly function ($n + 1 \Rightarrow$ interpolation)



1.5. Logistic regression

Logistic regression

- classification with $\mathcal{Y} = \{0, 1\}$
- however, we predict the probability of belonging to class 1
- hypothesis class:

$$\mathcal{H} = \{x \mapsto \phi(\langle w, x \rangle), w \in \mathbb{R}^d\},$$

with ϕ the *logistic function* (aka *sigmoid* function)

$$\phi(z) = \frac{1}{1 + \mathrm{e}^{-z}}.$$

- **Intuition:** squeeze the score between 0 and 1 to transform it into a probability
- $\mathbb{P}(y = 1 \,|\, x) = \phi(w^\top x)$ and $\mathbb{P}(y = 0 \,|\, x) = 1 - \phi(w^\top x)$

Logistic function

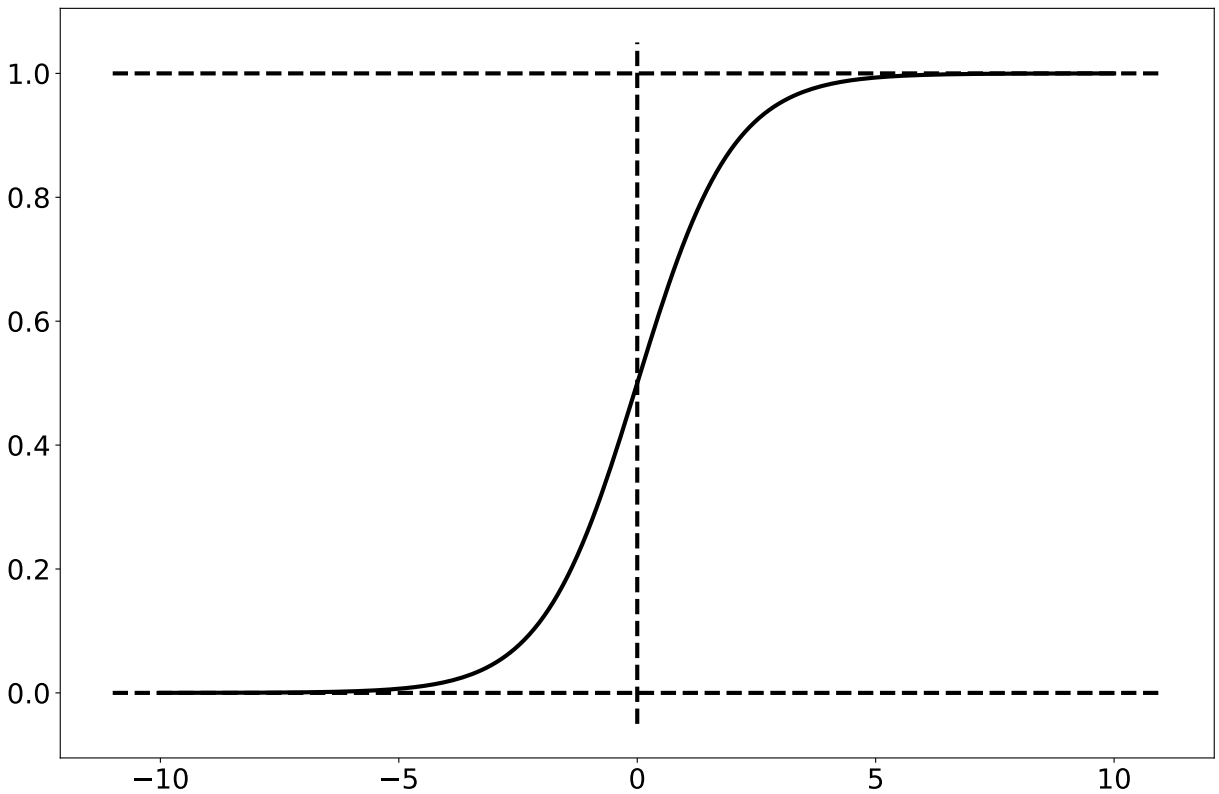
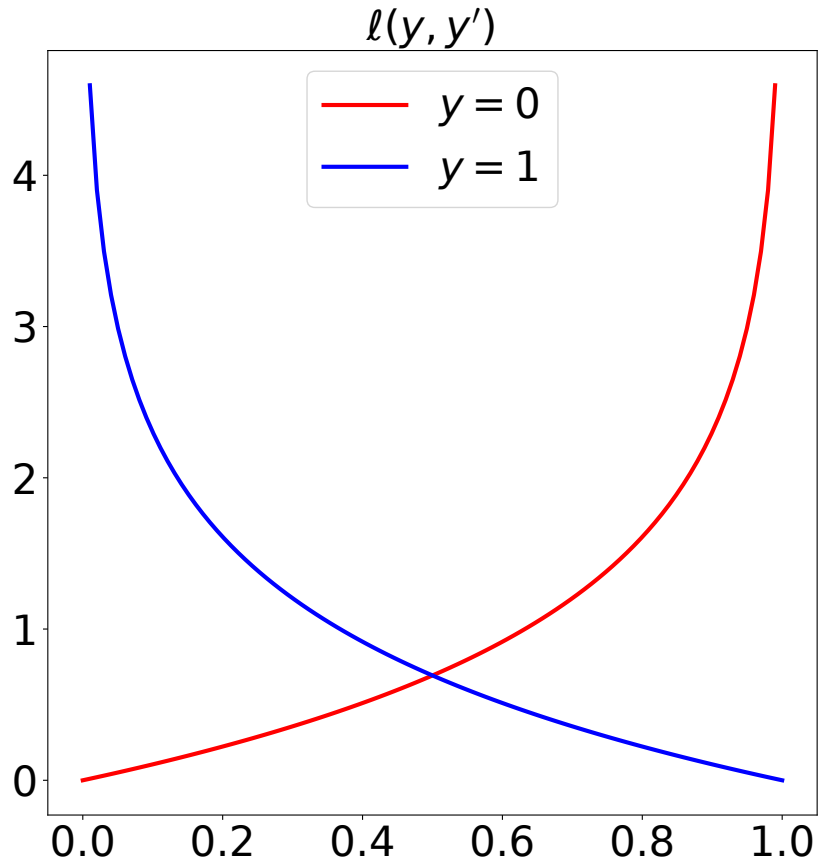


Figure: the logistic function $\phi : t \mapsto 1/(1 + \mathrm{e}^{-t})$.

Logistic loss

- **Next:** we need to define a loss function
- for any y, y', we define the *logistic loss*:

$$\ell(y, y') = -(1 - y) \log(1 - y') - y \log y' .$$



Logistic regression

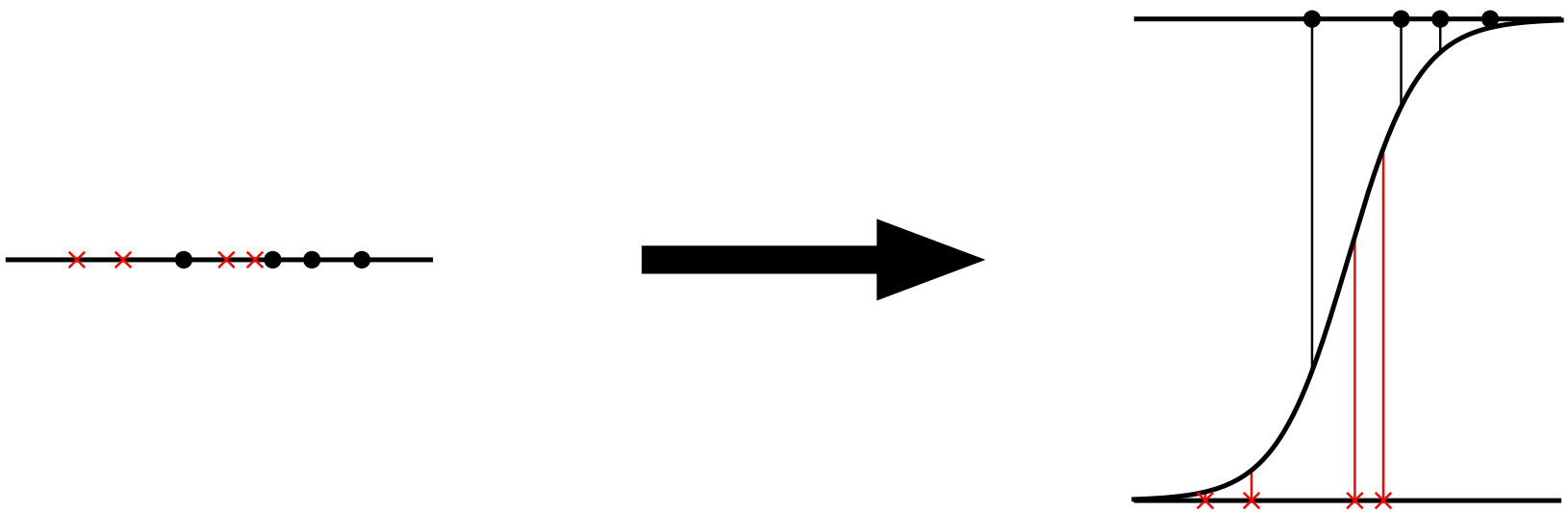
- finally, logistic regression = empirical risk minimization with the logistic loss
- that is, minimize for $w \in \mathbb{R}^d$

$$\hat{\mathcal{R}}_S(w) = \sum_{i=1}^{n} \left\{ -(1-y_i) \log(1-\phi(w^\top x_i)) - y_i \log \phi(w^\top x_i) \right\} .$$

- **Remark (i):** we can show that this is equivalent to maximum likelihood for a certain prior distribution
- **Remark (ii):** complicated to optimize (see exercise)

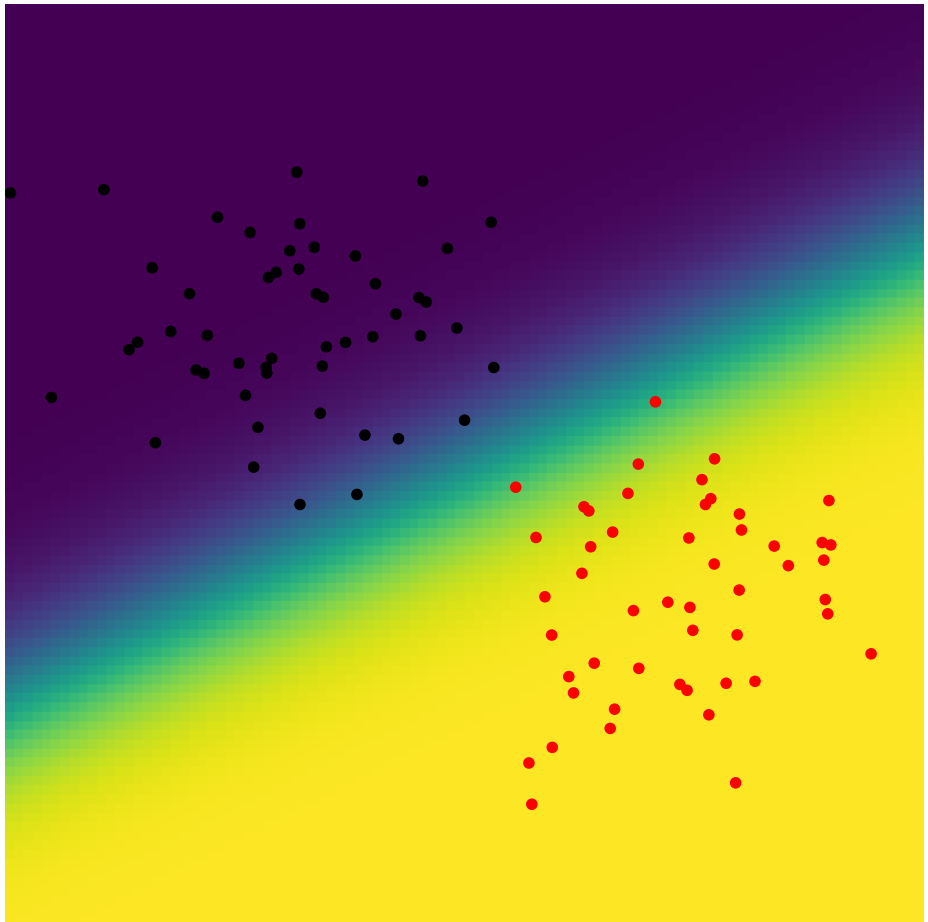
Logistic regression in dimension 1

- **Example:** in dimension one:



Logistic regression in dimension 2

- **Example:** in dimension two:



Exercise

Exercise: Recall that we defined the logistic loss by

$$\ell(y, y') = -(1-y)\log(1-y') - y\log y' .$$

1. Show that ERM with the logistic loss is equivalent to minimizing

$$F(w) = \sum_{i=1}^{n} \log\left(1 + \exp\left(-\tilde{y}_i \langle w, x_i \rangle\right)\right),$$

   where $\tilde{y}_i = \operatorname{sign}(y_i - 0.5)$. Deduce that $\hat{\mathcal{R}}$ is a convex function of w.
2. Compute the gradient of $\hat{\mathcal{R}}$ with respect to w. *Hint:* show that $\phi'(z) = \phi(z)(1-\phi(z))$.
3. Can you solve $\nabla\hat{\mathcal{R}}(w) = 0$? If not, propose a strategy for finding a good w.

Correction of the exercise

1. Let us set $1 \leq i \leq n$. We write

$$\begin{aligned}
\ell(y_i, \phi(w^\top x_i)) &= -(1-y_i)\log(1-\phi(w^\top x_i)) - y_i \log \phi(w^\top x_i) \\
&= -(1-y_i)\log \frac{\mathrm{e}^{-w^\top x_i}}{1+\mathrm{e}^{-w^\top x_i}} - y_i \log \frac{1}{1+\mathrm{e}^{-w^\top x_i}} \\
&= -(1-y_i)\log \mathrm{e}^{-w^\top x_i} + \log(1+\mathrm{e}^{-w^\top x_i})\,.
\end{aligned}$$

If $y_i = 0$, the last display equals

$$\log(1+\exp\left(w^\top x_i\right))\,,$$

if $y_i = 1$, it is

$$\log(1+\exp\left(-w^\top x_i\right))\,.$$

One can check directly that $x \mapsto \log(1+\mathrm{e}^{-x})$ is convex. By composition, F is a sum of convex functions, thus convex.

Correction of the exercise, ctd.

2. Let $1 \leq j \leq d$. We write

$$\begin{aligned}
\frac{\partial \hat{\mathcal{R}}(w)}{\partial w_j} &= -\sum_{i=1}^{n} \frac{\partial}{\partial w_j} \left\{ (1-y_i) \log(1-\phi(w^\top x_i)) + y_i \log \phi(w^\top x_i) \right\} \\
&= -\sum_{i=1}^{n} \left\{ \frac{-(1-y_i)}{1-\phi(w^\top x_i)} + \frac{y_i}{\phi(w^\top x_i)} \right\} \frac{\partial}{\partial w_j} \phi(w^\top x_i) \\
&= -\sum_{i=1}^{n} \left\{ \frac{-(1-y_i)}{1-\phi(w^\top x_i)} + \frac{y_i}{\phi(w^\top x_i)} \right\} \phi(w^\top x_i)(1-\phi(w^\top x_i)) x_{i,j} \\
\frac{\partial \hat{\mathcal{R}}(w)}{\partial w_j} &= -\sum_{i=1}^{n} \left(y_i - \phi(w^\top x_i) \right) x_{i,j} \, .
\end{aligned}$$

3. It does not seem possible to solve $\nabla F(w) = 0$ in closed-form, one has to use gradient descent. □

Recap

- **What happens when we call** `sklearn.linear_model.LogisticRegression`?
- `penalty` is $\ell_2 \rightarrow$ there is regularization by default! (not much though, $C = 1$)
- `fit_intercept` is True $\rightarrow$ again, our maths are not entirely accurate
- `solver` is liblinear $\rightarrow$ since there is no closed-form, a solver will be used
- liblinear uses coordinate descent
- will default soon to lbfgs (limited memory Broyden-Fletcher--Goldfarb-Shanno)
- do not worry too much about the solvers, just change if you see that it is not converging

2. Tree-based classifiers

2.1. Partition rules

Introduction

- let $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$
- in this section, we consider partition-based classifiers:

$$\mathcal{H} = \left\{ h : x \mapsto \sum_{j=1}^{p} h_j \mathbb{1}_{x \in A_j} \right\},$$

  where $a_j \in \mathbb{R}$ and $A_1, \ldots, A_p$ form a *partition* of the space
- that is,

$$A_1 \cup \cdots \cup A_p = \mathcal{X} \quad \text{and} \quad A_i \cap A_j = \emptyset \forall i \neq j \,.$$

- the A_js are often called *cells*
- generally, for practical reasons the A_js are rectangles

ERM for partition rules

- assume that the partition is fixed and set $A(x) =$ cell containing x
- **Regression:** with squared loss, ERM rule gives

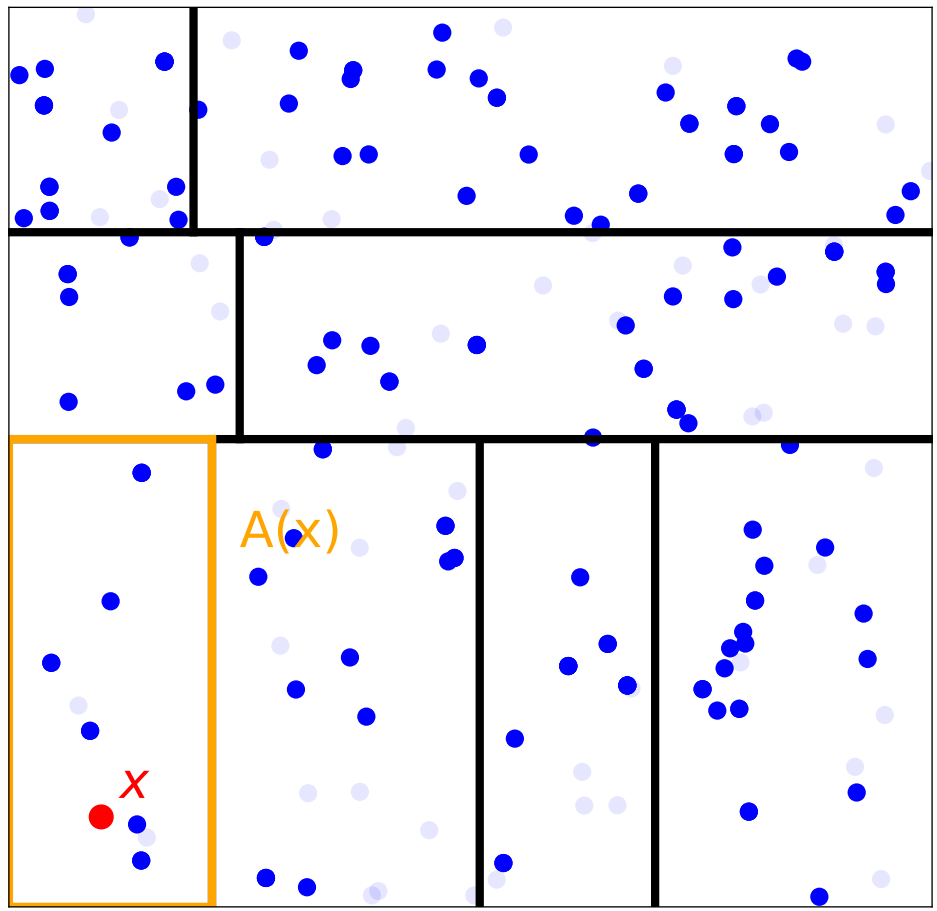
$$f(x) = \frac{1}{|\{j \text{ s.t. } x_j \in A(x)\}|} \sum_{i=1}^{n} x_i \mathbb{1}_{y_i \in A(x)} \,,$$

  that is the average of the observations on each cell
- **Classification:** majority vote:

$$f(x) = \begin{cases} 1 & \text{if } |\{i \text{ s.t. } x_i \in A(x) \text{ and } y_i = 1\}| \geq \\ & \qquad |\{i \text{ s.t. } x_i \in A(x) \text{ and } y_i = 0\}| \\ 0 & \text{otherwise.} \end{cases}$$

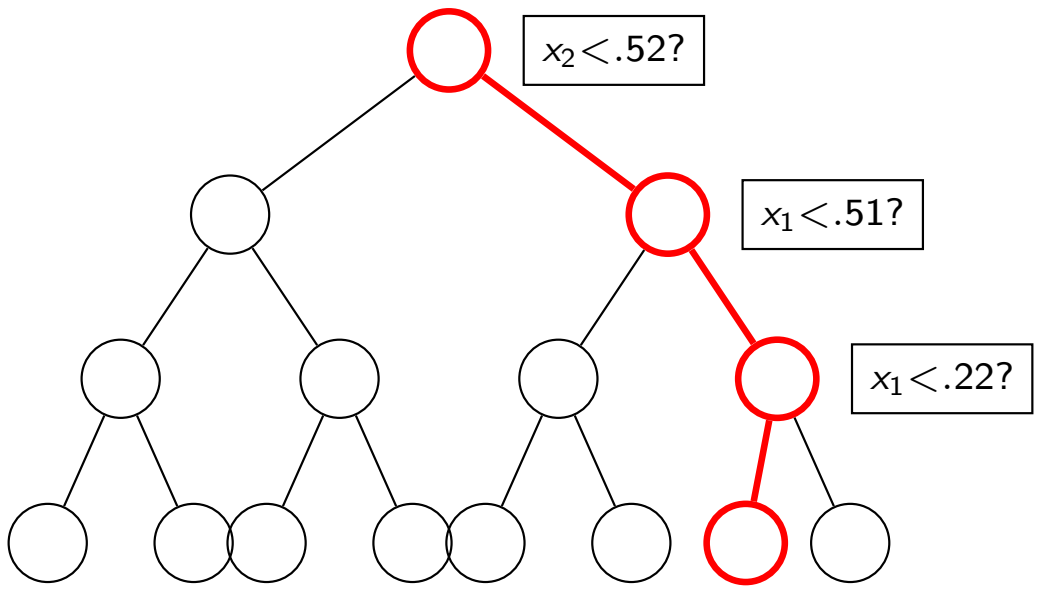
- thus ERM $\Leftrightarrow$ finding the best partition (for a fixed p)
- **Problem:** this is computationally very hard! p^n possibilities to compare
- even if we restrict ourselves to rectangles, intractable

Example of a partition-based predictor

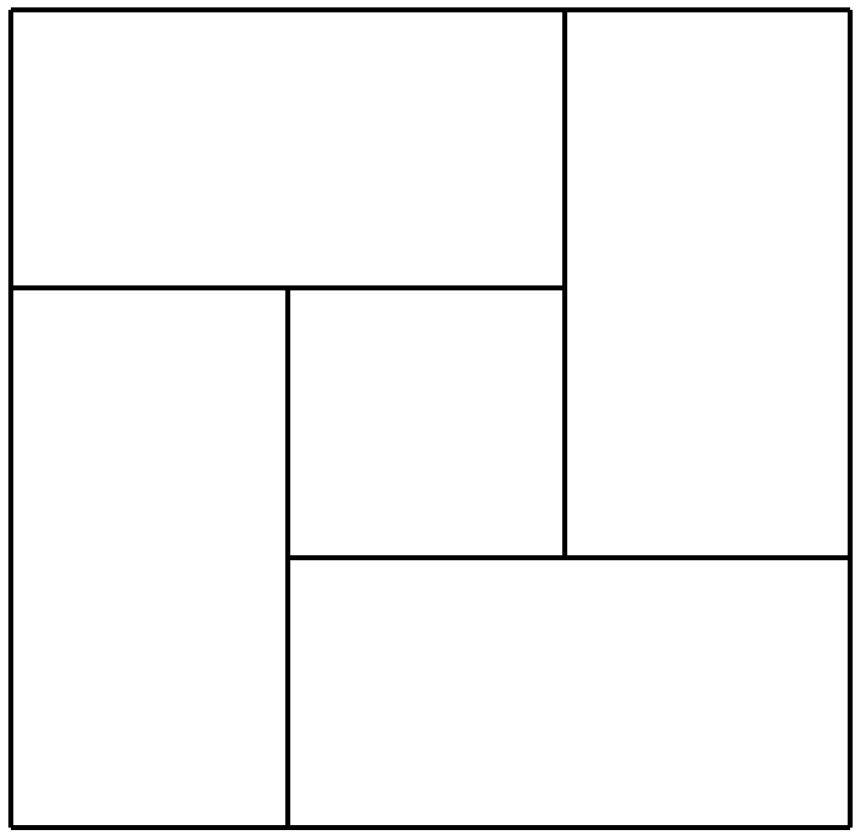


Tree structures

- one possible solution: start from $\mathcal{X}$ and *split* the cells iteratively
- we obtain a tree-like structure
- **Remark:** not necessary to split in two, but very common
- another advantage in doing so: root the new data efficiently



Not a tree



- **Figure:** this partition of $[0,1]^2$ can not be obtained by recursive binary splitting

Growing trees

- **Question:** how do we make the splits?
- **general answer:** take an heuristic that makes sense
- each heuristic yields a different algorithm, completed with stopping criterion (do a split only if gain greater than γ)
- complete reference on such procedures: the *yellow book*[4]
- good splitting rules:
  - create many cells (enough to capture the local variations of the distribution);
  - create cells that are large enough (we want sufficiently training data in the cells to compute a relevant average)
- **Notation:** I current node, I_L (resp. I_R) left (resp. right) node after the split

[4] Devroye, Györfi, Lugosi, *A probabilistic theory of pattern recognition*, 1996

ID3[6] and C4.5

Definition: Let S be a finite set of points. Then we define the *entropy* of S by

$$H(S) = \sum_{y \in \mathcal{Y}} -p(y) \log_2 p(y)\,,$$

where $p(y)$ is the proportion of elements of S classified as y.

- easy to see that $H(S) = 0$ means that the node is *pure* = only one label ($0 \log 0 = 0$)
- C4.5 criterion:[5] find direction and split that maximizes

$$H(I) - H(I_L) - H(I_R)\,.$$

[5] Quinlan, *C4.5: Programs for Machine Learning*, 1993
[6] Quinlan, *Induction of decision trees*, Machine Learning, 1986

CART trees, classification

- later supplanted by CART trees[7]

Definition: Let S be a finite set of points. We define the *Gini impurity* by

$$G(S) = \sum_{y \in \mathcal{Y}} p(y)(1 - p(y)) \,.$$

- $G(S) = 0$ if the leaf is pure
- CART trees: find direction and split that maximizes

$$G(I) - G(I_L) - G(I_R) \,.$$

[7] Breiman et al., *Classification and Regression Trees*, 1984

CART trees for regression

- slightly different in the regression setting: look at the *variance*

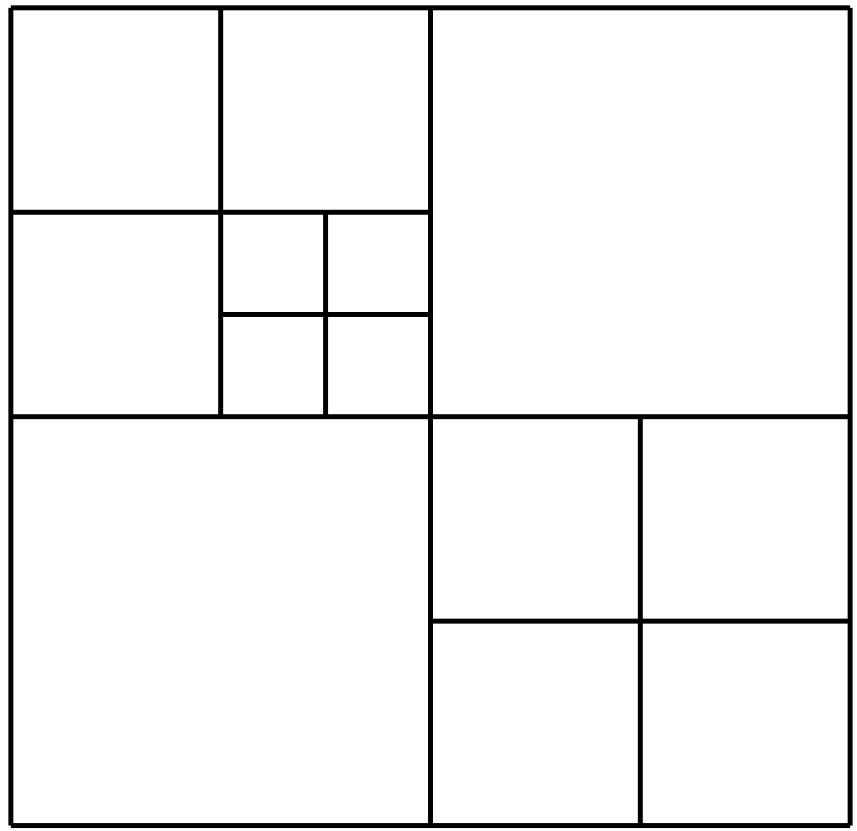
$$V(S) = \frac{1}{|S|} \sum_{i \in S} (y_i - \bar{y}_S)^2 = \frac{1}{|S|^2} \sum_{i,j \in S} \frac{1}{2} (y_i - y_j)^2 \,.$$

- the criterion is the variance reduction due to the split:

$$V(I) - \frac{|I_L|^2}{|I|^2} V(I_L) - \frac{|I_R|^2}{|I|^2} V(I_R) \,.$$

- **Intuition:** maximal if data is homogeneous left and right of the split (then $V(I_L) = V(I_R) = 0$)

Other examples



- **Figure:** quad trees[8]

[8] Finkel, Bentley, *Quad trees: a data structure for retrieval on composite keys*, Acta Informatica, 1974

Other examples, ctd.

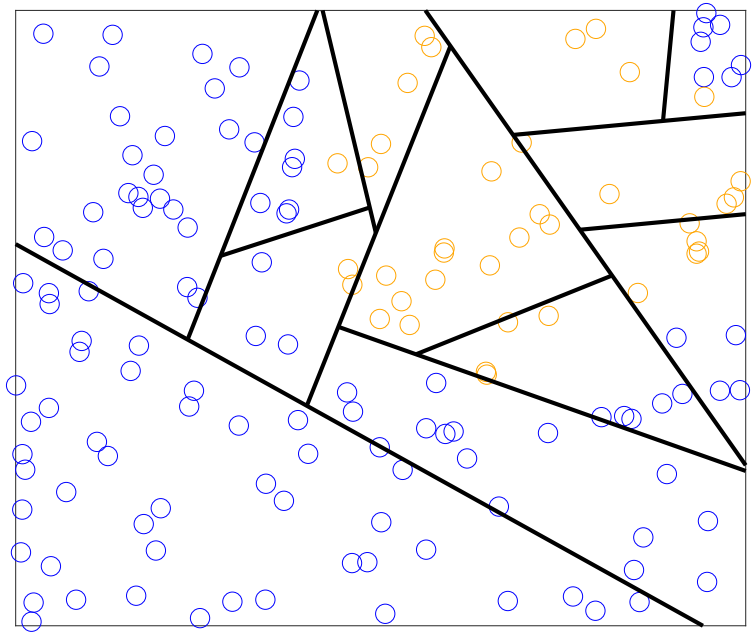


Figure: comparison-based splits[9]

[9] Haghiri et al., *Comparison-based random forests*, ICML, 2018

When to stop?

- usually, many direction to try: CART reduces to a random subset of directions
- also possible to specify T a max height for the tree
- other strategy: grow the trees to the full extent, and then **pruning**
- one possibility = reduced error pruning
- starting at the leaves, each node replaced by its most common class
- if prediction accuracy does not change, ditch the node
- **Remark:** error computations on a *validation set*

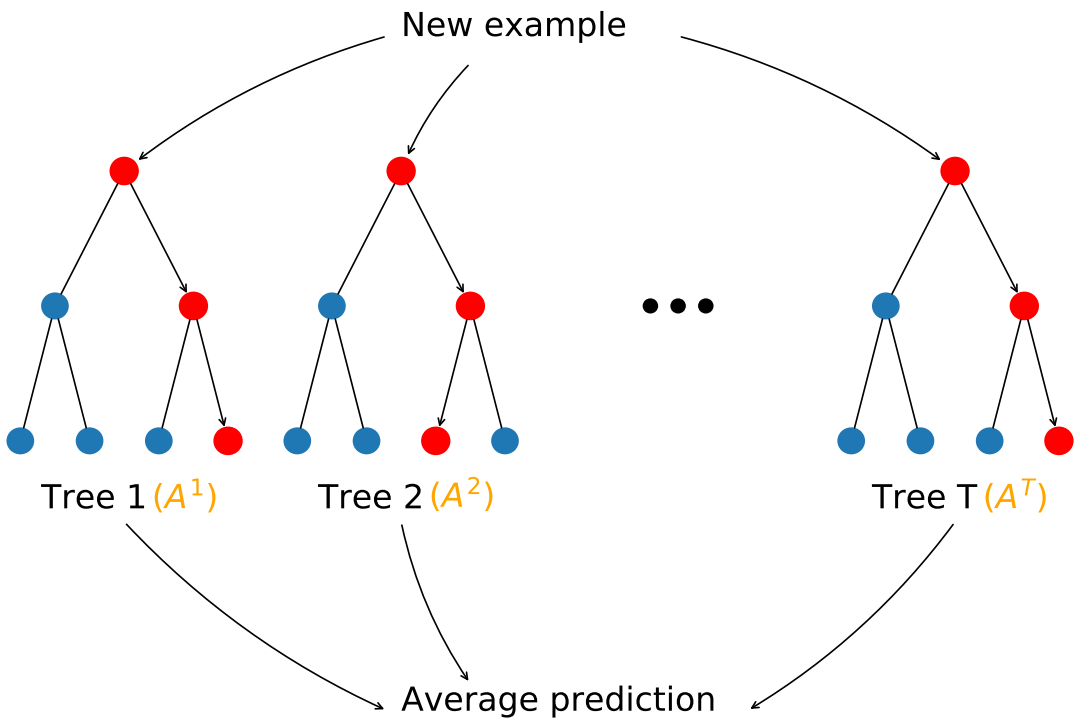
Recap

- **What happens by default when we invoke the function** `sklearn.tree.DecisionTreeClassifier`? let us look at least at the main options
- `criterion` is set to Gini $\rightarrow$ we are using CART trees
- `splitter` is set to best $\rightarrow$ looking at the best split at each step
- `max_depth` is None $\rightarrow$ splitting until leaves are pure or contain less than `min_samples_split`
- `min_samples_split` $= 2$
- `max_features` is None $\rightarrow$ no max number of features, log could be a reasonable choice if we have many features
- `max_leaf_nodes`: None $\rightarrow$ many leaves, we could also restrict this
- `min_impurity_decrease` $= 0 \rightarrow$ continues to split even if very small gain

2.2. Random forests

Random forests

- one possible problem with tree classifiers: overfitting
- **Solution:** train many trees and aggregate the prediction
- **Classification:** majority vote
- **Regression:** return the mean



Bagging

- **Additional idea:**[10] train each tree on a random subsample of the data
- usual strategy = bagging
- bagging means bootstrap aggregation
- sample *with replacement* a proportion αn of the training data
- train the tree classifier on this subset
- resample for each tree

[10] Breiman, *Random forests*, Machine Learning, 2001

Recap

What happens by default when we invoke the function `sklearn.ensemble.RandomForestClassifier`?

- `n_estimators` $= 100$ (T in our notation)
- `criterion` $=$ 'Gini' $\rightarrow$ we are using CART trees
- `max_depth` $=$ None $\rightarrow$ trees are grown until leaves are pure
- `max_features` $=$ auto $\rightarrow \sqrt{d}$ features considered
- `bootstrap` $=$ True $\rightarrow$ taking subsamples of the data, but since `max_samples` is set to None actually sampling the whole data

3. Boosting

Introduction

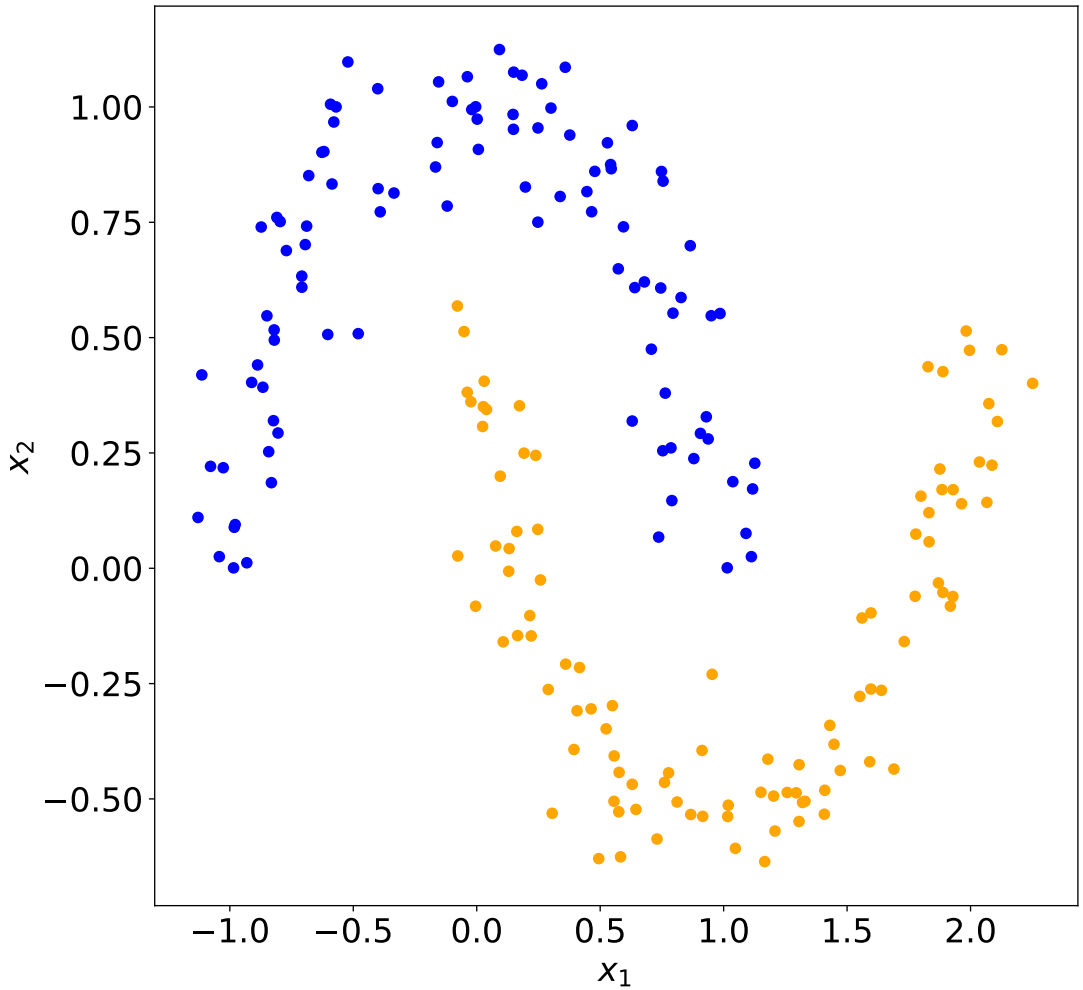
- **Important:** classification setting, $\mathcal{Y} = \{-1, +1\}$
- **Idea:** aggregate many classifiers together, then majority voting:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) ,$$

  where h_t are classifiers and α_t weights
- weak classifier = barely better than random guessing
- **Examples:** linear classifier, small trees,...
- **Question:** how do we decide which weights to put?
- different strategies, different algorithms[11]

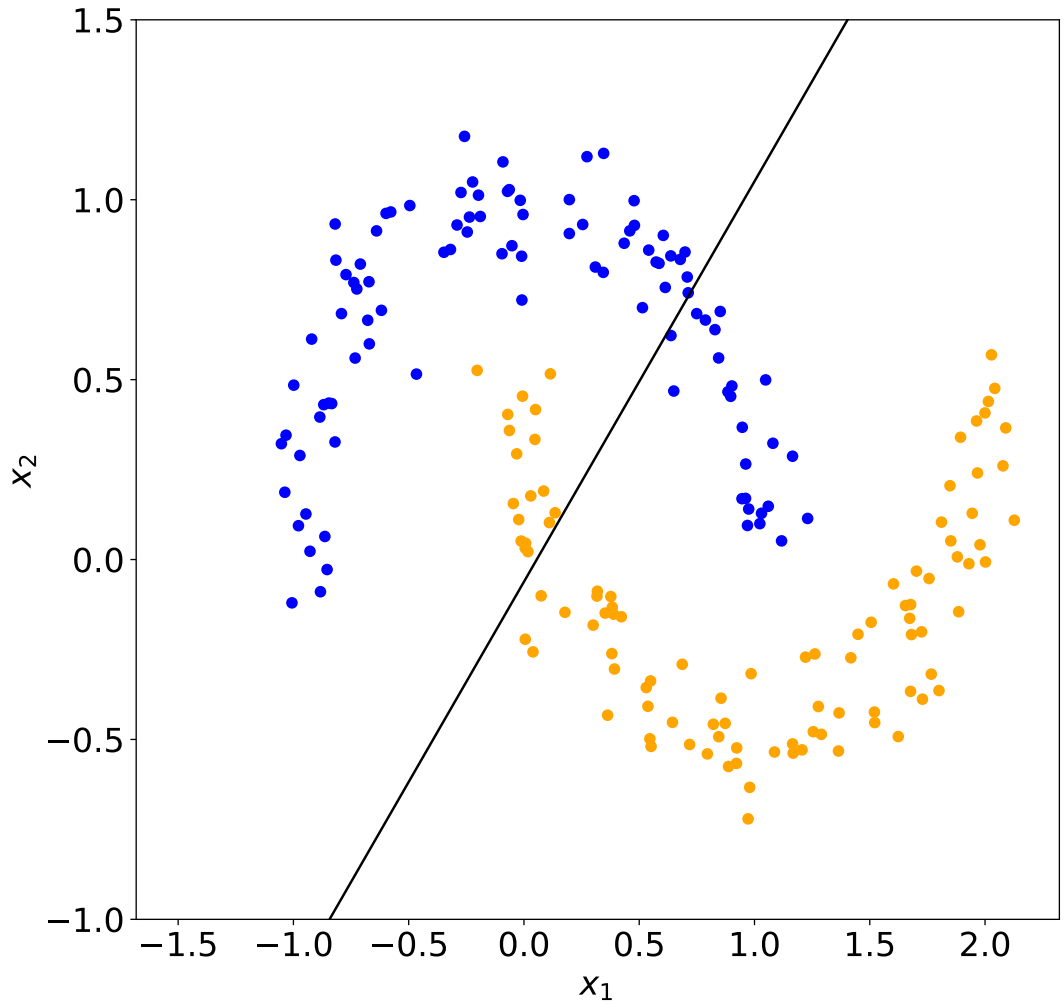
[11] Schapire, Freund, *Boosting: foundations and algorithms,* MIT Press, 2012

Non-linearly separable datasets



- **Figure:** moons datasets from `sklearn`

Weak classifier



- **Figure:** moons datasets from `sklearn`

3.1. Adaboost

Introduction

- we first look at AdaBoost[12] *(short for adaptative boosting)*
- AdaBoost maintains a *distribution* D_t over time
- start with uniform distribution, then increase the weights of misclassified examples
- at each step, we pick a classifier that minimizes the *weighted error*

$$\begin{aligned}\varepsilon_t &:= \mathbb{P}_{i\sim D_t}\left(h_t(x_i) \neq y_i\right)\\ &= \sum_{i=1}^{n} D_t(i) \cdot \mathbb{1}_{h_t(x_i)\neq y_i}\,.\end{aligned}$$

- adjust the weights by multiplying by a quantity depending on ε_t, larger than one if misclassified, smaller if correctly classified

[12] Freund and Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, Journal of computer and system science, 1997

AdaBoost

Algorithm 1: AdaBoost algorithm

Input: n training examples $(x_1, y_1), \ldots, (x_n, y_n)$ where $x_i \in \mathcal{X}$ and $y_i \in \{-1, 1\}$

Initialize the distribution to $D_1(i) = \frac{1}{n}$

for $t = 1$ *to* T **do**

Train weak learner using distribution D_t

Get weak hypothesis $h_t : \mathcal{X} \to \{-1, 1\}$

h_t minimizes the weighed error $\varepsilon_t := \mathbb{P}_{i \sim D_t} \left(h_t(x_i) \neq y_i \right)$. Set $\alpha_t := \frac{1}{2} \log \left(\frac{1-\varepsilon_t}{\varepsilon_t} \right)$

Update, for $i = 1 \ldots n$,

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \mathrm{e}^{-\alpha_t} & \text{if } h_t(x_i) = y_i \,, \\ \mathrm{e}^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \,. \end{cases}$$

end

Result: final classifier $H(x) := \operatorname{sign} \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$.

Exercise

Exercise: With the notation of the previous slide,

1. show that
$$\mathbb{P}_{i\sim D_{t+1}}\left(h_t(x_i)\neq y_i\right)=\frac{\sqrt{\varepsilon_t(1-\varepsilon_t)}}{Z_t}\,.$$
2. show that
$$Z_t=2\sqrt{\varepsilon_t(1-\varepsilon_t)}\,.$$
Deduce that
$$\mathbb{P}_{i\sim D_{t+1}}\left(h_t(x_i)\neq y_i\right)=\frac{1}{2}\,.$$

In other terms, with the weights chosen by AdaBoost, the weak classifier obtained at step t does not better than random guessing at step $t+1$.

Correction of the exercise

1. We write

$$\begin{aligned}
\mathbb{P}_{i\sim D_{t+1}}\left(h_t(x_i)\neq y_i\right) &= \sum_{i=1}^{n} \mathbb{1}_{h_t(x_i)\neq y_i}\cdot D_{t+1}(i) && \text{(total expectation)}\\
&= \sum_{i=1}^{n} \mathbb{1}_{h_t(x_i)\neq y_i}\cdot \frac{D_t(i)}{Z_t}\cdot \mathrm{e}^{\alpha_t} && \text{(definition of } D_{t+1})\\
&= \frac{1}{Z_t}\sum_{i=1}^{n} \mathbb{1}_{h_t(x_i)\neq y_i}\cdot D_t(i)\cdot \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}} \\
& && \text{(definition of } \alpha_t)\\
&= \frac{1}{Z_t}\cdot\sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}\cdot\varepsilon_t = \frac{\sqrt{\varepsilon_t(1-\varepsilon_t)}}{Z_t}\,.
\end{aligned}$$

Correction of the exercise, ctd.

2.

$$
\begin{aligned}
Z_t &= \sum_{\substack{i=1 \\ h_t(x_i)\neq y_i}}^{n} D_t(i)\cdot \mathrm{e}^{\alpha_t} + \sum_{\substack{i=1 \\ h_t(x_i)= y_i}}^{n} D_t(i)\cdot \mathrm{e}^{-\alpha_t} \\
&= \mathrm{e}^{\alpha_t}\sum_{i=1}^{n} \mathbb{1}_{h_t(x_i)\neq y_i}\cdot D_t(i) + \mathrm{e}^{-\alpha_t}\left(1-\sum_{i=1}^{n}\mathbb{1}_{h_t(x_i)\neq y_i}\cdot D_t(i)\right) \\
&= \sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}\cdot\varepsilon_t + \sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}}\cdot(1-\varepsilon_t) \\
Z_t &= 2\sqrt{\varepsilon_t(1-\varepsilon_t)}\,.
\end{aligned}
$$

□

Training error of AdaBoost

- the choice of α_t and the update of the weights is mysterious
- actually, are we even sure that the train error goes to zero?

Proposition: with the above notation, let $\gamma_t := \frac{1}{2} - \varepsilon_t$. Then the weighted training error of the combined classifier H with respect to D_1 satisfies

$$\mathbb{P}_{i\sim D_t}\left(H(x_i) \neq y_i\right) \leq \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right).$$

- **Consequence:** suppose that our weak classifiers are always doing a bit better than random, *i.e.*, $\gamma_t \geq \gamma > 0$
- then $\mathbb{P}_{i\sim D_1}\left(H(x_i) \neq y_i\right) \leq (1-4\gamma)^{T/2} \to 0$ when $T \to +\infty$

Training error of AdaBoost, ctd.

- **Proof:** we let $F(x) := \sum_{t=1}^{T} \alpha_t h_t(x)$, thus $H(x) = \operatorname{sign}(F(x))$
- by definition of D_t, for any $i \in \{1, \ldots, n\}$

$$
\begin{aligned}
D_{T+1}(i) &= D_1(i) \times \frac{\mathrm{e}^{-y_i \alpha_1 h_1(x_i)}}{Z_1} \times \cdots \times \frac{\mathrm{e}^{-y_i \alpha_T h_T(x_i)}}{Z_T} \\
&= \frac{D_1(i) \cdot \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)\right)}{\prod_{t=1}^{T} Z_t} \\
&= \frac{D_1(i) \cdot \exp\left(-y_i F(x_i)\right)}{\prod_{t=1}^{T} Z_t} \, .
\end{aligned}
$$

Training error of AdaBoost, ctd.

- since $H(x) = \operatorname{sign}(F(x))$, if $H(x) \neq y$, then $yF(x) \leq 0$
- in that case, $\mathrm{e}^{-yF(x)} \geq 1$
- therefore $\mathbb{1}_{H(x)\neq y} \leq \mathrm{e}^{-yF(x)}$

$$\begin{aligned}
\mathbb{P}_{i\sim D_1}(H(x_i) \neq y_i) &= \sum_{i=1}^{n} D_1(i) \cdot \mathbb{1}_{H(x_i)\neq y_i} \\
&\leq \sum_{i=1}^{n} D_1(i) \cdot \exp(-y_i F(x_i)) \\
&= \sum_{i=1}^{n} D_{T+1}(i) \cdot \prod_{t=1}^{T} Z_t \\
&= \prod_{t=1}^{T} Z_t \,.
\end{aligned}$$

Training error of AdaBoost, ctd.

- finally, recall that $\varepsilon_t = \frac{1}{2} - \gamma_t$ and $Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$ according to the previous exercise
- we deduce that $Z_t = \sqrt{1-4\gamma_t^2}$
- the exponential bound is obtained using the classical inequality $\mathrm{e}^x \geq 1 + x$ with $x = -4\gamma_t^2$ and taking the logarithm $\square$

Links with optimization

- there is a deeper reason why the training error of AdaBoost is decreasing
- more precisely, let us look at the training error with *exponential loss*:

$$\hat{\mathcal{R}}_S(F) = \frac{1}{n}\sum_{i=1}^{n} \exp\left(-y_i F(x_i)\right) .$$

- optimizing directly on F can be complicated
- greedy[13] procedure: assume that F_t is already known and pick $\alpha > 0$ and $h \in \mathcal{H}$ that minimize

$$\sum_{i=1}^{n} \exp\left(-y_i(F_t(x_i) + \alpha h(x_i))\right) \propto \sum_{i=1}^{n} D_t(i) \cdot \exp\left(-y_i \alpha h(x_i)\right) .$$

[13] = make the optimal choice at each step

Links with optimization, ctd.

- let us rewrite this objective:

$$\sum_{i=1}^{n} D_t(i) \cdot \exp\left(-y_i \alpha h(x_i)\right) = \sum_{y_i = h(x_i)} D_t(i) \cdot \mathrm{e}^{-y_i \alpha h(x_i)} + \sum_{y_i \neq h(x_i)} D_t(i) \cdot \mathrm{e}^{-y_i \alpha h(x_i)}$$
$$= \mathrm{e}^{-\alpha} \cdot \sum_{y_i = h(x_i)} D_t(i) + \mathrm{e}^{\alpha} \cdot \sum_{y_i \neq h(x_i)} D_t(i)$$
$$= (\mathrm{e}^{\alpha} - \mathrm{e}^{-\alpha}) \sum_{y_i \neq h(x_i)} D_t(i) + \mathrm{e}^{-\alpha}\,.$$

- for fixed α, since $\mathrm{e}^{\alpha} - \mathrm{e}^{-\alpha} > 0$, this is equivalent to minimizing in h

$$\sum_{y_i \neq h(x_i)} D_t(i) = \sum_{i=1}^{n} D_t(i) \cdot \mathbb{1}_{h(x_i) \neq y_i} = \varepsilon_t \text{!!!}$$

Links with optimization, ctd.

- let us go back one step: we want to minimize

$$(\mathrm{e}^{\alpha} - \mathrm{e}^{-\alpha}) \cdot \varepsilon_t + \mathrm{e}^{-\alpha} = \varepsilon_t \mathrm{e}^{\alpha} + (1 - \varepsilon_t)\mathrm{e}^{-\alpha}$$

  with respect to α

- this is a smooth, convex function of α, standard procedure:

$$\frac{\mathrm{d}}{\mathrm{d}\alpha}\left(\varepsilon_t \mathrm{e}^{\alpha} + (1 - \varepsilon_t)\mathrm{e}^{-\alpha}\right) = \varepsilon_t \mathrm{e}^{\alpha} - (1 - \varepsilon_t)\mathrm{e}^{-\alpha} = 0\,,$$

  which yields

$$\mathrm{e}^{2\alpha} = \frac{1 - \varepsilon_t}{\varepsilon_t}\,.$$

- finally,

$$\alpha = \frac{1}{2}\log\frac{1 - \varepsilon_t}{\varepsilon_t} = \alpha_t !!!$$

Conclusion

- `sklearn.ensemble.AdaBoostClassifier` in Python
- `base_estimator` defaults to None, which calls `DecisionTreeClassifier` $\rightarrow$ CART tree
- `n_estimator` is 50 by default (you can increase this value)
- learning rate fixed to one

3.2. XGBoost

Introduction

- boosting is a very powerful framework
- AdaBoost is no longer state-of-the-art
- XGBoost[14] is the latest *Wunderkind*
- remarkable empirical results, winning a lot of Kaggle compatitions
- does not require as much tuning as deep neural nets
- not very well understood
- **Important:** we present XGBoost in the *regression* setting

[14] Chen and Guestrin, *XGBoost: a scalable tree boosting system*, SIGKDD, 2016

Tree boosting

- **First important idea:** as we have seen, at each step t, boosting optimizes

$$\sum_{i=1}^{n} \ell(y_i, F_t(x_i) + f(x_i))$$

for $f \in \mathcal{H}$ (the class of models that we are considering)

- loss function can be complicated: just take a Taylor approximation (at order 2) of ℓ!
- formally, replace $\ell(y_i, F_t(x_i) + f(x_i))$ by

$$\ell(y_i, F_t(x_i)) + g_i f(x_i) + \frac{1}{2} h_i f(x_i)^2 ,$$

where

$$\begin{cases} g_i &= \frac{\partial}{\partial y'} \ell(y, y') \Big|_{y=y_i, y'=F_t(x_i)} \\ h_i &= \frac{\partial^2}{\partial y'^2} \ell(y, y') \Big|_{y=y_i, y'=F_t(x_i)} \end{cases}$$

Quadratic loss

- **Example:** let us consider the quadratic loss

$$\ell(y, y') := (y - y')^2 \,.$$

- then we can compute

$$\frac{\partial}{\partial y'} \ell(y, y') = -2(y - y') \,,$$

and

$$\frac{\partial^2}{\partial y'^2} \ell(y, y') = 2 \,.$$

- in that particular case, the approximation is exact

Regularization

- after simplifications, the objective becomes:

$$\sum_{i=1}^{n} \left[g_i f(x_i) + \frac{1}{2} h_i f(x_i)^2 \right] .$$

- **Second important idea:** regularization
- very typical to work with trees, we do not want them to grow to deep
- add $\Omega(f)$, a term that *penalizes* such situations:

$$\mathrm{obj}(f) := \sum_{i=1}^{n} \left[g_i f(x_i) + \frac{1}{2} h_i f(x_i)^2 \right] + \Omega(f) .$$

- in XGBoost:

$$\boxed{\Omega(f) = \gamma L_f + \frac{1}{2} \lambda \sum_{j=1}^{L_f} w_j^2 ,}$$

where L_f is the number of leaves of f, $\lambda > 0$ is a constant, and w_j is the value of f on leaf j

Structure score

Exercise:

1. assume that the structure of the tree f is known. Show that the objective function can be written

$$\sum_{j=1}^{L_f} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma L_f \,,$$

   where I_j is the set of points falling inside leaf j.

2. set $G_j := \sum_{i \in I_j} g_i$ and $H_j := \sum_{i \in I_j} h_i$. Show that, for a given tree structure, the best w_js and the best objective function value are given by

$$w_j^\star := \frac{-G_j}{H_j + \lambda}, \qquad \text{and} \qquad \text{obj}^\star := \frac{-1}{2} \sum_{j=1}^{L_f} \frac{G_j^2}{H_j + \lambda} + \gamma L_f \,.$$

Learning the tree structure

- essentially, the exercise tells us that we just need to optimize over all possible tree structures
- unfortunately, as for CART, this is not realistic
- instead, we use a heuristic to choose where to split a node:

$$\text{gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right]$$

- **Intuition:** score on the new left + right leaves − score on the node
- then iterative splitting and stopping criterion

Conclusion

- Python implementation: `xgboost` package
- `XGBRegressor` object
- `n_estimator` $= 100$ by default
- $\gamma = 0$ by default (!)
- `max_depth` $= 6$
- $\lambda = 1$ by default

4. Nearest neighbors

Introduction

- $\mathcal{X}$ some metric space
- there is a *distance function* $\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$
- properties:
  - $\delta(x, y) = 0$ if, and only if, $x = y$;
  - $\delta(x, y) = \delta(y, x)$;
  - $\delta(x, z) \leq \delta(x, y) + \delta(y, z)$.
- **Example (i):** Euclidean distance:

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

- **Example (ii):** Mahalanobis distance:

$$\delta(x, y) = \sqrt{(x - y)^\top S^{-1}(x - y)} \, .$$

- **Example (iii):** Hamming distance: number of substitutions between two strings

Nearest neighbors rule

- training examples $x_1, \ldots, x_n \in \mathcal{X}$
- **Idea:** memorize the training set and predict the label of a new instance according to the labels of its k nearest neighbors[15]
- for any $x \in \mathcal{X}$, define $\pi_\cdot(x)$ an ordering according to $\delta(x, x_i)$
- namely:

$$\delta(x, x_{\pi_i(x)}) \leq \delta(x, x_{\pi_{i+1}(x)}) \,.$$

- **Regression:** empirical average:

$$h(x) = \frac{1}{k} \sum_{i=1}^{k} y_{\pi_i(x)} \,.$$

- **Classification:** majority vote
- when $k = 1$, $1 - NN$ rule: $h(x) = y_{\pi_1(x)}$

[15] Fix, Hodges, *Discriminatory analysis. Nonparametric discrimination: consistency properties*, tech report, 1951

Weighted nearest neighbors

- possible to take into account the distances
- for instance:

$$h(x) = \sum_{i=1}^{k} \frac{\delta(x, x_{\pi_i(x)})}{\sum_{j=1}^{k} \delta(x, x_{\pi_j(x)})} y_{\pi_i(x)} \, .$$

- possible to extend to arbitrary weights

Generalization error of $1 - NN$

- focus on $k = 1$, possible to generalize[16]
- binary classification ($\mathcal{Y} = \{0, 1\}$) on $[0, 1]^d$
- $0 - 1$ loss: $\ell(y, y') = \mathbb{1}_{y \neq y'}$
- recall

$$\eta(x) = \mathbb{P}(y = 1 \,|\, x) \ ,$$

and

$$h^\star(x) = \mathbb{1}_{\eta(x) > 1/2} \, .$$

- **Assumption:** the conditional probability of the labels is c-Lipschitz:

$$\forall x, x' \in \mathcal{X}, \quad |\eta(x) - \eta(x')| \leq c \, \|x - x'\| \ .$$

[16] Cover, Hart, *Nearest neighbor pattern classification*, IEEE Transactions on Information Theory, 1967

Generalization error, ctd.

Theorem: let $\mathcal{X} = [0,1]^d$ and $\mathcal{Y} = \{0,1\}$. Let $\mathcal{D}$ be a distribution on $\mathcal{X} \times \mathcal{Y}$ such that the conditional distribution η is c-Lipschitz. Let $S = (x_1, y_1), \ldots, (x_n, y_n)$ be an i.i.d. sample from $\mathcal{D}$ and let h_S be its $1-NN$ associated hypothesis. Let $h^\star$ be the Bayes optimal rule for η. Then

$$\mathbb{E}_S[\mathcal{R}(h_S)] \leq 2\mathcal{R}(h^\star) + 4c\sqrt{d}\, n^{\frac{-1}{n+1}} .$$

- **Intuition:** risk of $1-NN$ is bounded by twice the Bayes error + some term depending on the regularity of η
- if we want consistency, we actually have to take $k \to +\infty$ with $k/n \to 0$ (Stone's theorem[17])

[17] Stone, *Consistent nonparametric regression*, The Annals of Statistics, 1977

Conclusion

- `sklearn.neighbors.KNeighborsClassifier`
- `n_neighbors` $= k = 5$ by default
- `weights` defaults to uniform
- default distance is Euclidean