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 classifiers
   - Partition rules
3. Boosting

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)

Halfspaces

- thus our function class is

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

- it is possible to show that $VC(\mathcal{H}) = d + 1$
- **Consequence:** $\mathcal{H}$ is PAC learnable with sample complexity

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

- **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 $w^{(1)}, w^{(2)}, \dots, w^{(T)}$
- 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

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 $a \in \mathbb{R}$,

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

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

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

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:

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

Least squares, ctd.

- we can rewrite it, define

$$A = \sum_{i=1}^{n} x_i x_i^\top \text{ and } b = \sum_{i=1}^{n} y_i x_i \,,$$

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

$$Aw = b\,.$$

- if A is invertible, straightforward:

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

- 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.

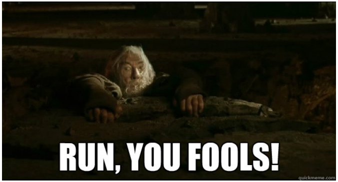
- why did we do that?
- let v_i denote the ith column of V, then

$$
\begin{aligned}
A\hat{w} &= AA^{+}b && \text{(definition of } \hat{w}\text{)} \\
&= VDV^{\top}VD^{+}V^{\top}b && \text{(definition of } A^{+}\text{)} \\
&= VDD^{+}V^{\top}b && (V \text{ is orthonormal}) \\
A\hat{w} &= \sum_{i:D_{i,i}\neq 0} v_i v_i^{\top} b\,.
\end{aligned}
$$

- 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 $A\hat{w} = b$

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:

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

with $\left\| u \right\|^2 = u_1^2 + \cdots + u_d^2$ 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

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

Notice that F is a convex smooth function and find its minimizer $\hat{w}$ in closed-form. Recall that we defined

$$A = \sum_{i=1}^{n} x_i x_i^\top \quad \text{and} \quad b = \sum_{i=1}^{n} y_i x_i .$$

Recap

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

1.4. Polynomial regression

Polynomial regression

- linear regression is a powerful tool, especially because we can transform the inputs
- **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` $\to$ very handy when we want to fit univariate data

1.5. Logistic regression

Logistic regression

- regression task, but the output is $\mathcal{Y} = [0, 1]$: 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)

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

- logistic loss:

$$\ell(y, y') = \frac{-1}{2}(1 + y) \log y' + \frac{1}{2}(1 - y) \log(1 - y') \, .$$

Logistic function

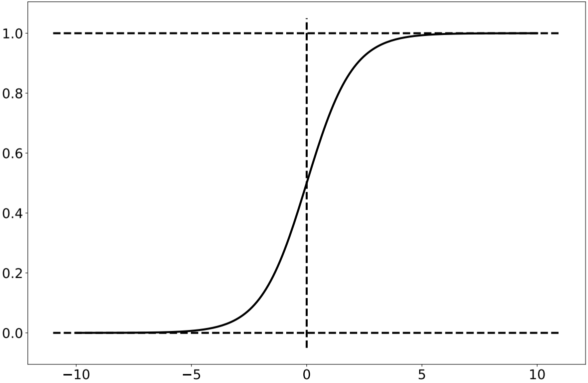


Figure: the logistic function $\sigma : t \mapsto 1/(1 + \mathrm{e}^{-t})$.

Exercise

Exercise: show that empirical risk minimization with the logistic loss is equivalent to minimizing

$$F(w) = \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle w, x_i \rangle)).$$

Is F a convex function of w? Compute the gradient of F with respect to w. Can you solve $\nabla F(w) = 0$?

Recap

- **What happens when we call** `sklearn.linear_model.LogisticRegression`?
- `penalty` is ℓ_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 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:

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

  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
- regression with squared loss, then the ERM rule gives

$$h_j = \frac{1}{|i \text{ s.t. } i \in A_j|} \sum_{i \in A_j} x_i \,,$$

  that is the average of the observations on each cell
- classification $\Rightarrow$ majority vote
- 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

Trees

- one possible solution: start from $\mathcal{X}$ and *split* iteratively
- we obtain a tree-like structure
- another advantage in doing so: root the new data efficiently
- **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)
- **Notation:** I current node, I_L (resp. I_R) left (resp. right) node after the split
- **Note:** we focus on classification ($\mathcal{Y} = \{0, 1\}$)

ID3[5] 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 S is perfectly classified ($0 \log 0 = 0$)
- C4.5 criterion:[4] find direction and split that maximizes

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

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

CART

- later supplanted by CART trees[6]

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

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

- CART trees: find direction and split that minimizes

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

- for regression, variance reduction criterion

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

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**

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

3. Boosting

Introduction

- **Idea:** aggregate many weak classifiers together, then majority voting
- **Examples:** linear classifier, trees,...