Chapter 3 : Linear and Logistic Regressions

Contents :
1. Linear regression
2. Gradient descent
3. Classification with logistic regression
4. Newton's method

TP 25/02 : implementation of simple methods by hand.
TP 05/03 : Usage of scikit-learn.

Example : Houses

1. Linear regression

For regression problems : $\mathcal{Y} = \mathbb{R}$.

What hypothesis set for linear regressions?

If $\mathcal{X} = \mathbb{R}^p$,

$$\mathcal{H} = \left\{ h_\beta(x) = \boxed{\beta_0} + \beta_1 x_1 + \dots + \beta_p x_p \;\middle|\; \beta_0, \beta_1, \dots, \beta_p \in \mathbb{R} \right\}.$$

β_0 : called the bias or intercept term.

Technically, affine functions, not linear!

To make it easier to manipulate :

$$h_\beta(x) = \sum_{i=0}^{p} \beta_i x_i, \quad \boxed{\text{where } x_0 := 1.}$$

$$= x^T \beta, \text{ where } \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} \in \mathbb{R}^{p+1}, \; x = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix} \in \mathbb{R}^{p+1}.$$

$(x^{(i)}, y^{(i)})$: i^{th} training example $(1 \le i \le n)$.

Problem: Choose β such that $h_\beta(x^{(i)}) \approx y^{(i)}$ $\forall 1 \le i \le n$.

Loss function: $L(h_\beta(x), y) = (h_\beta(x) - y)^2$ (squared error).

So,

$$\underset{\beta}{\operatorname{argmin}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2}_{= J(\beta)}$$

$\rightsquigarrow$ MSE, mean squared error.

How to minimize $J(\beta)$?

---

2. Gradient descent

We want to minimize a function $J(\cdot)$.

- Start with some β (e.g., $\beta = \vec{0}$).
- Repeat until "convergence":
  for $j = 0, \dots, p$, simultaneously:

$$\beta_j \leftarrow \beta_j - \alpha \frac{\partial J(\beta)}{\partial \beta_j}$$

} One iteration of GD.

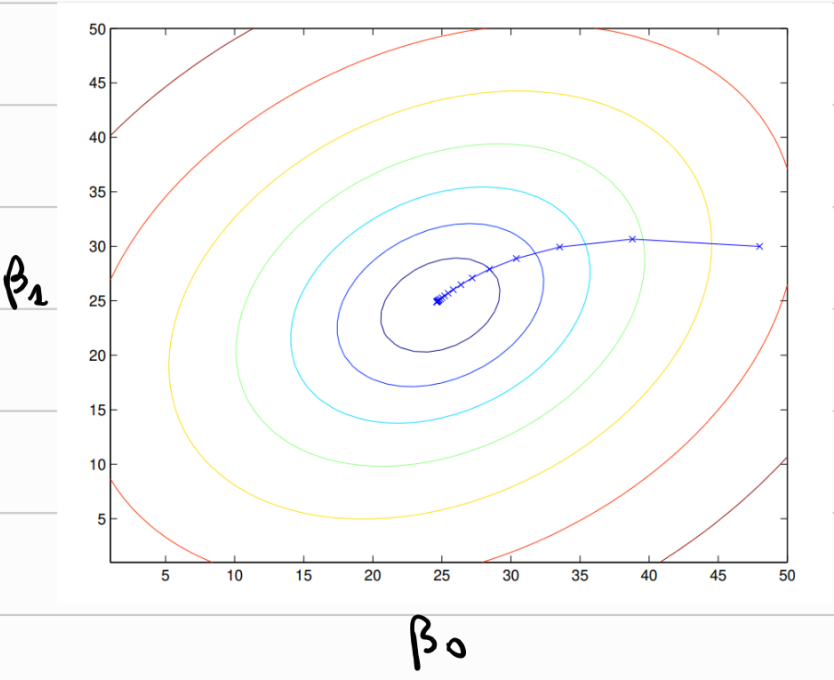
α → Learning rate (hyperparameter)

For the J above:

$$\frac{\partial J(\beta)}{\partial \beta_j} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial \beta_j} \left(h_\beta(x^{(i)}) - y^{(i)}\right)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} 2 \cdot \left(h_\beta(x^{(i)}) - y^{(i)}\right) \cdot \frac{\partial}{\partial \beta_j} \left(h_\beta(x^{(i)}) - y^{(i)}\right)$$

$$= \frac{2}{n} \sum_{i=1}^{n} \left(h_\beta(x^{(i)}) - y^{(i)}\right) \cdot x_j^{(i)}.$$



The $J(\cdot)$ for linear regression has a single (global) minimum.

← Application of GD on the contours of J, with $\rho = 1$.

How to choose α?

- If too large, you may "overshot".
  ↝ If the value of J increases, good sign that α is too big.
- If too small, lots of iterations.

→ **Try a few values!**

Gradient descent above : **Batch** gradient descent.

→ Uses <u>all the data</u> for each iteration.

<u>Problem</u>? If the dataset is large, every step is slow.

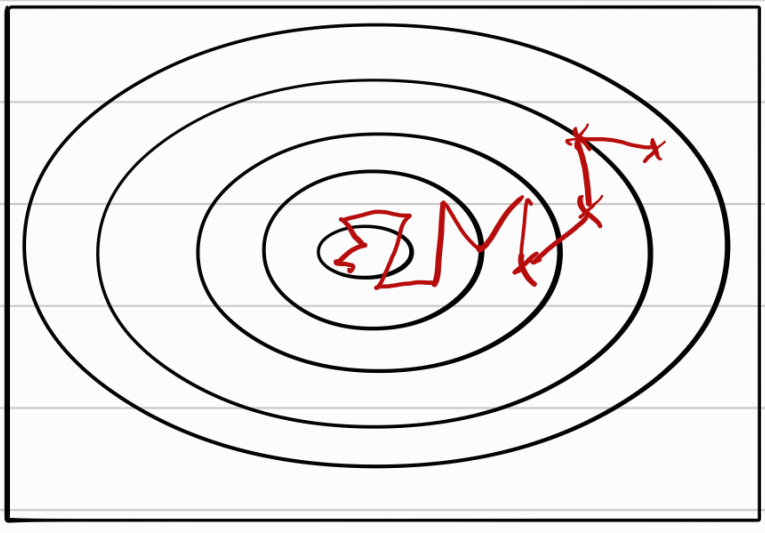
Alternative: **Stochastic** gradient descent.

Repeat:

  for $i = 1, \dots, n$:

    for $j = 0, \dots, p$, simultaneously:

$$\beta_j \leftarrow \beta_j - \alpha \frac{\partial}{\partial \beta_j} L\left(h_\beta(x^{(i)}), y^{(i)}\right)$$



Faster iterations, but slower "convergence".

Intermediate algorithm:

Minibatch gradient descent

↳ Use a small, fixed number of examples at a time.

Back to linear regression

A specificity of linear regression (as opposed to most other algorithms in this course) is that it has a closed-form solution ("solution de forme fermée").

Derivation using linear algebra:

$$\nabla_\beta J(\beta) = \begin{bmatrix} \frac{\partial}{\partial \beta_0} J(\beta) \\ \vdots \\ \frac{\partial}{\partial \beta_p} J(\beta) \end{bmatrix} \in \mathbb{R}^{p+1} \quad \text{(gradient of } J\text{)}.$$

If $X = \begin{bmatrix} — & (x^{(1)})^T & — \\ & \vdots & \\ — & (x^{(n)})^T & — \end{bmatrix} \in \mathbb{R}^{n \times (p+1)}$, $\beta = \begin{bmatrix} \beta_0 \\ | \\ \beta_p \end{bmatrix}$, $y = \begin{bmatrix} y^{(1)} \\ | \\ y^{(n)} \end{bmatrix}$

Then $X\beta = \begin{bmatrix} h_\beta(x^{(1)}) \\ | \\ h_\beta(x^{(n)}) \end{bmatrix}$ and $J(\beta) = \frac{1}{n} \sum_{i=1}^{n} \left(h_\beta(x^{(i)}) - y^{(i)} \right)^2$

$$= \frac{1}{n} (X\beta - y)^T (X\beta - y).$$

$$\nabla_\beta J(\beta) = \nabla_\beta \frac{1}{n} (X\beta - y)^T (X\beta - y)$$

$$= \frac{1}{n} \nabla_\beta (\beta^T X^T - y^T)(X\beta - y)$$

$$= \frac{1}{n} \nabla_\beta (\beta^T X^T X\beta - \beta^T X^T y - y^T X\beta + y^T y)$$

$$= \frac{1}{n} (2X^T X\beta - 2X^T y).$$

So $\nabla_\beta J(\beta) = \vec{0} \iff$ $X^T X\beta = X^T y$.

("normal equations for linear least squares")

If $X^T X$ is invertible, $\boxed{\beta = (X^T X)^{-1} X^T y}$.

↳ requires $n \geq p+1$.

$X^T X$ is invertible iff all $p+1$ predictors are linearly independent.

Probabilistic interpretation.

Why the least squares for the loss function?

If we assume

$$y^{(i)} = \beta^T x^{(i)} + \varepsilon^{(i)}$$

↳ error term, "random noise"

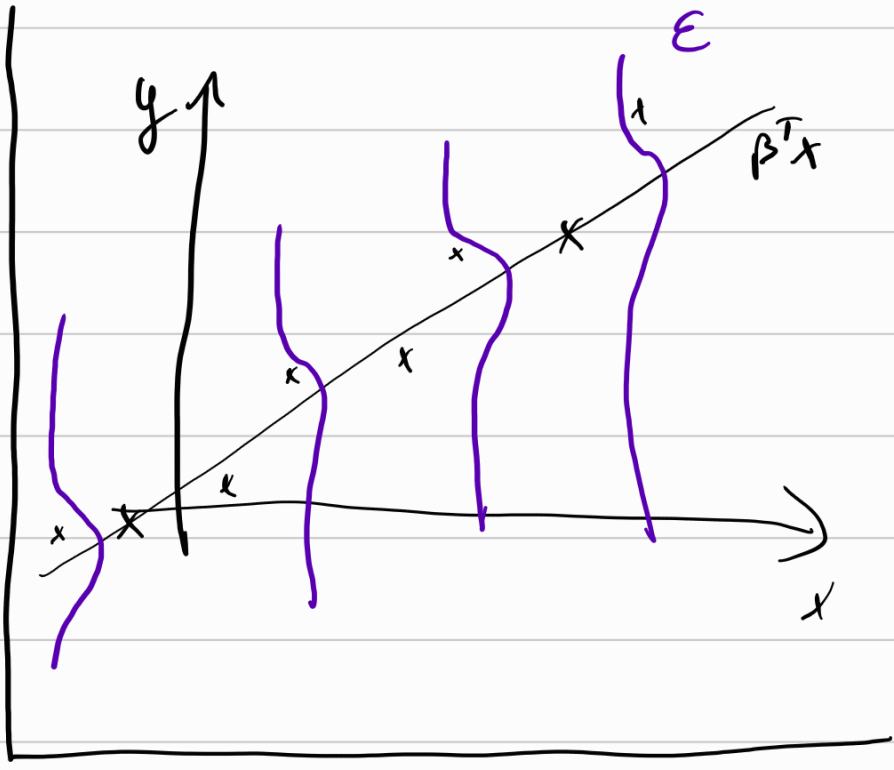
with $\varepsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$, then

$$f(y^{(i)} \mid x^{(i)}; \beta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \beta^T x^{(i)})^2}{2\sigma^2}\right).$$

Finding β that maximizes the likelihood

$$\mathcal{L}(\beta) = \prod_{i=1}^{n} f(y^{(i)} | x^{(i)}; \beta)$$

is equivalent to minimizing the least squares (see Lab 2, Ex. 7).

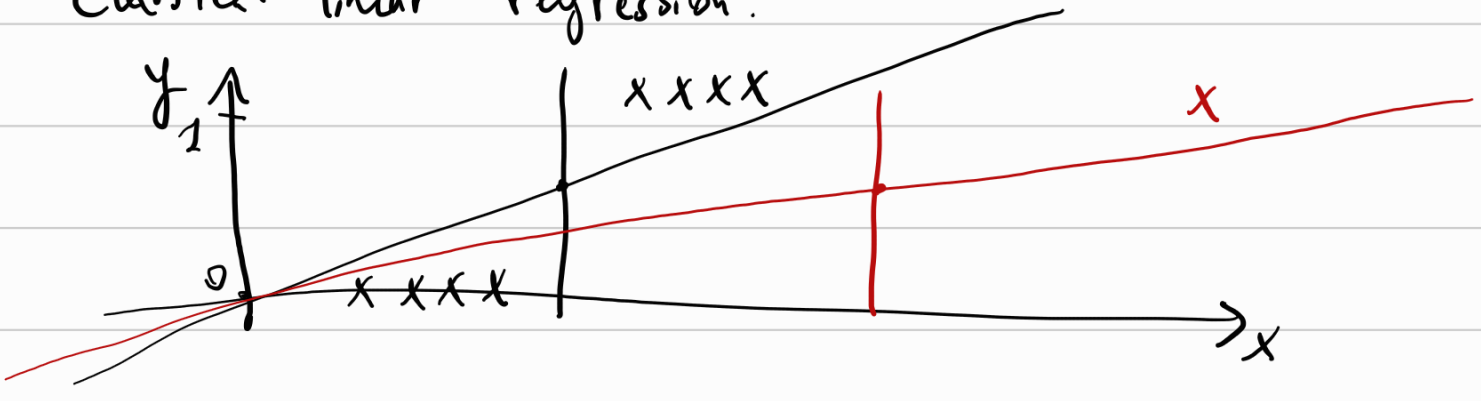


---

3. Classification with logistic regression

$y = \{0, 1\}$ (binary classification).

Classical linear regression?



- High dependence on "outliers".
- Outputs > 1 or < 0 : how to interpret them ???

→ Ordinary linear regression is not suited.

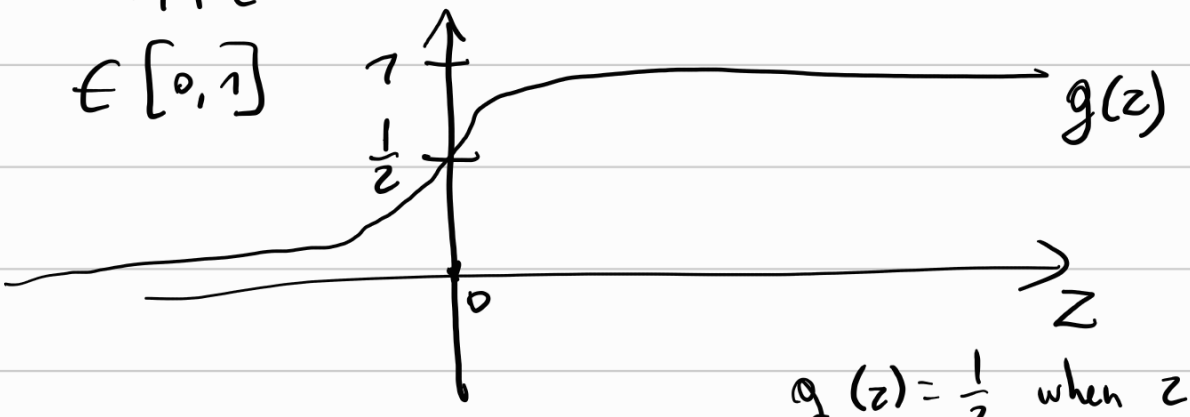
Logistic regression

We want $h_\beta(x) \in [0,1]$.

We define $h_\beta(x) = g(\beta^T x) = \frac{1}{1+e^{-\beta^T x}}$.

$g(z) = \frac{1}{1+e^{-z}}$ "logistic" or "sigmoid" function

$\in [0,1]$



$g(z) = \frac{1}{2}$ when $z = \beta^T x = 0$.

Why this function spacifically?

Part of "Generalized Linear Models".
(not covered here)

Probabistic interpretation

We assume that $\begin{cases} p(y=1 \mid x;\beta) = h_\beta(x), \\ p(y=0 \mid x;\beta) = 1 - h_\beta(x). \end{cases}$

$\rightarrow p(y \mid x;\beta) = h_\beta(x)^y \cdot (1-h_\beta(x))^{1-y}$.

MLE:

$$L(\beta) = \prod_{i=1}^{n} p(y^{(i)} \mid x^{(i)};\beta)$$

$$= \prod_{i=1}^{n} h_\beta(x^{(i)})^{y^{(i)}} \cdot (1-h_\beta(x^{(i)}))^{1-y^{(i)}}$$

Log-likelihood:

$$\ell(\beta) = \log L(\beta)$$
$$= \sum_{i=1}^{n} y^{(i)} \log\left(h_\beta(x^{(i)})\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\beta(x^{(i)})\right).$$

Choose β to **maximize** $\ell(\beta)$. It is concave (single global maximum).

No closed form solution ☹.

Batch gradient **ascent**:

$$\beta_j \leftarrow \beta_j \boxed{+} \alpha \frac{\partial}{\partial \beta_j} \ell(\beta).$$

Exercise: compute the partial derivatives of $\ell(\beta)$.

You should find $$\boxed{\frac{\partial}{\partial \beta_j} \ell(\beta) = \sum_{i=1}^{n} \left(y^{(i)} - h_\beta(x^{(i)})\right) x_j^{(i)}}.$$

Exactly the same as linear regression (with a different hypothesis function h_β)!

(4) Newton's method

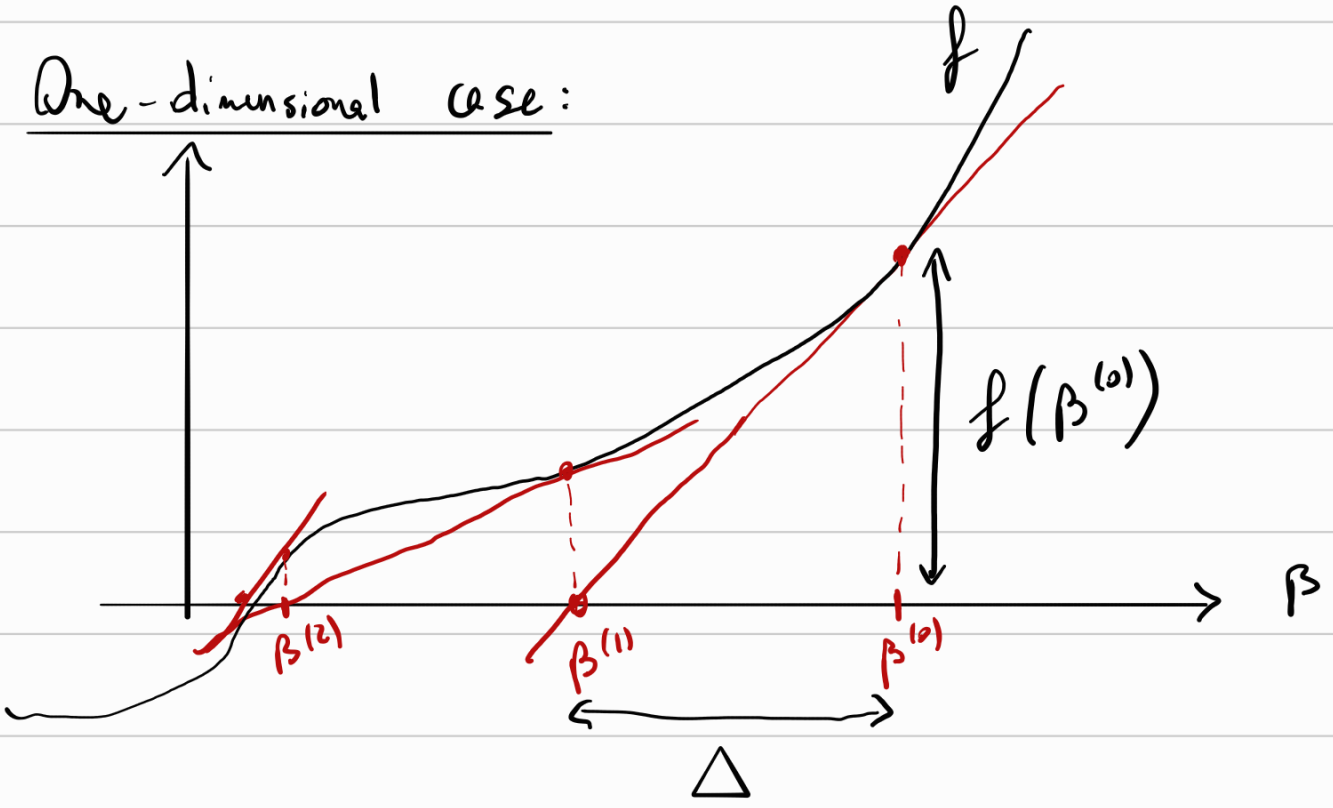
Other algorithm to find local extrema.

Used to find <u>roots</u> of a function f (β s.t. $f(\beta)=0$).

To maximize $\ell(\beta)$, you look for β such that

$$\boxed{\ell'(\beta)=0} \quad \rightsquigarrow f=\ell'.$$

<u>One-dimensional case:</u>



- $\beta^{(1)} \leftarrow \beta^{(0)} - \Delta$
- $f'(\beta^{(0)}) = \dfrac{f(\beta^{(0)})}{\Delta} \rightsquigarrow \Delta = \dfrac{f(\beta^{(0)})}{f'(\beta^{(0)})}$

In general:

$$\beta^{(t+1)} \leftarrow \beta^{(t)} - \frac{f(\beta^{(t)})}{f'(\beta^{(t)})} \left(= \beta^{(t)} - \frac{\ell'(\beta^{(t)})}{\ell''(\beta^{(t)})} \right).$$

Newton's method has <u>quadratic convergence</u> (the number of correct digits <u>doubles</u> at each iteration) under assumptions on f.

↳ $\cdot f' \neq 0$
$\cdot f''$ is continuous
(see Numerical Analysis course)
$\cdot \dots$

Faster convergence than gradient descent under stronger conditions; requires to know the second derivative. But each iteration is more expensive.

- <u>Multi-dimensional case</u> ($\beta \in \mathbb{R}^{p+1}$) to find an extremum of ℓ.

$$\beta^{(t+1)} \leftarrow \beta^{(t)} + \underbrace{H^{-1}}_{\mathbb{R}^{(p+1)\times(p+1)}} \underbrace{\nabla_\beta \ell}_{\in \mathbb{R}^{p+1}}$$

where H is the Hessian matrix:

$$H_{ij} = \frac{\partial^2 \ell}{\partial \beta_i \partial \beta_j}.$$

Fewer iterations than gradient descent, but requires to invert a $(p+1) \times (p+1)$ matrix at every step: prohibitive if p is big.