Data Science

Marc Tommasi

September 26, 2023

Outline

Outline

Main ideas

- Used to explain as well as to predict. Example: explain fuel consumption as a function of car weight.
- Take the class of straight lines $y = ax + b$. Find a and b given weights (x) and observed consumption (y).
- We also call b *intercept* (ordonnée à l'origine) and a is a coefficient (which defines the slope of the line).
- If we know that the data x are centered at 0, then the intercept is zero.
- Generalization to the case where data are represented in a space of dimension d. Example: predict fuel consumption as a function of car weight and engine power, weather, speed, etc.
- We take a class $\mathcal{H}$ of linear functions: we look for w_i such that $y = w_0 + w_1x_1 + \cdots + w_dx_d$.
- Note: we can pose $x_0 = 1$ and compute

$$y = w_0x_0 + w_1x_1 + \cdots + w_dx_d = \boldsymbol{w}^\top \boldsymbol{x} = \langle \boldsymbol{w}, \boldsymbol{x} \rangle$$

.

Formal setting

- $\mathcal{X} \subseteq \mathbb{R}^d$, $\mathcal{Y} \subseteq \mathbb{R}$
- $\mathcal{H} = \{\boldsymbol{x} \rightarrow \langle \boldsymbol{w}, \boldsymbol{x} \rangle \mid \boldsymbol{w} \in \mathbb{R}^d\}$ is the class of hypothesis.
- There are m examples of the form $((x_1, x_2, \dots, x_d), y) = (\boldsymbol{x}, y)$:

$$S = \{(\boldsymbol{x}^{(1)}, y_1), (\boldsymbol{x}^{(2)}, y_2), \dots, (\boldsymbol{x}^{(m)}, y_m)\}$$

- S can be represented by a matrix X where each row is an instance of S
- We use the quadratic loss (squared error) to measure the empirical error:

$$\ell(h, (\boldsymbol{x}, y)) = (h(\boldsymbol{x}) - y)^2$$

ERM and the objective function

- The aim is to minimize the squared error for each example, and the loss function, the objective function, is the Mean squared error (MSE)
$$\frac{1}{m}\sum_i(\langle \boldsymbol{w}, \boldsymbol{x}^{(i)}\rangle - y_i)^2 = \frac{1}{m}\|X\boldsymbol{w} - \boldsymbol{y}\|^2$$
- The ERM principle is to find the parameters $\boldsymbol{w}$ that minimize the objective function on S:
$$\operatorname*{argmin}_w \frac{1}{m}\|X\boldsymbol{w} - \boldsymbol{y}\|^2$$
- The objective is convex: there is a global minimum and it can be calculated analytically.

Derivatives and gradient

- The derivative f' of a function $f : \mathbb{R} \to \mathbb{R}$ evaluated at x is defined by
$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x + \epsilon) - f(x)}{\epsilon}$$

.

- The derivative cancels for an x that minimizes f.
- For $f : \mathbb{R}^d \to \mathbb{R}$, we can consider the derivative with respect to each component of $\boldsymbol{x} = (x_1, \ldots, x_d)$. This is denoted $\frac{\partial f}{\partial x_i}$.
- The d -dimensional vector of all these derivative functions is the gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d})$.
- Remember: the derivative of $f(x) = x^n$ is $f'(x) = nx^{n-1}$ and the derivative of a constant is zero;

Conclusion

- We can find an analytical solution which will be $(X^\top X)^{-1}X^\top \boldsymbol{y}$.
- Informally, we're looking for $X\boldsymbol{w} = \boldsymbol{y}$, which we can rewrite as $X^\top X\boldsymbol{w} = X^\top \boldsymbol{y}$, then $\boldsymbol{w} = (X^\top X)^{-1}X^\top \boldsymbol{y}$.
- Note
  - The matrix $(X^\top X)$ may not be invertible, so we can instead calculate $X^\dagger \boldsymbol{y}$ where $X^\dagger$ is the Moore-Penrose pseudo-inverse, which is calculated by an SVD (If $X = U\Sigma V^\top$ then $X^\dagger = V\Sigma^\dagger U^\top$ and $\Sigma^\dagger$ is calculated by taking the inverse of the values except for the 0s, which remain 0).

Complexity

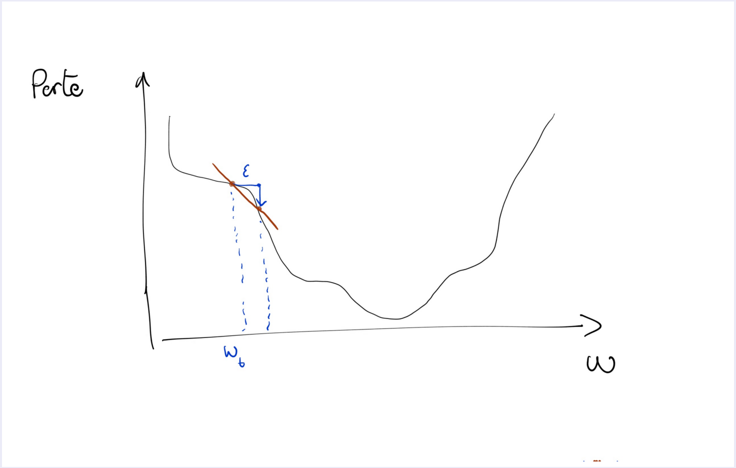
- The matrix inversion is costly for very large datasets
- Matrix multiplication is $O(n^3)$ for simple algorithms ($O(n^{2.4})$ for the best ones).
- The gradient descent approach yields a quadratic algorithm that works quite well in practice.
- (and can be accelerated with GPUs...)

Gradient descent I

- The aim is to reduce the empirical error step by step: find the parameter modification that will make the loss function take on a smaller value.
- Idea of descending through the steepest slope on the curve of loss as a function of parameters $\boldsymbol{w}$

Gradient descent II

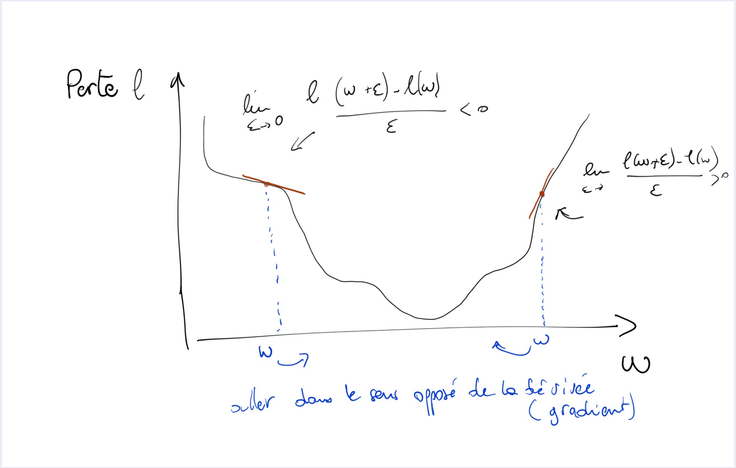
Derivative direction



Quick reminder about derivatives

Gradient descent III

Direction

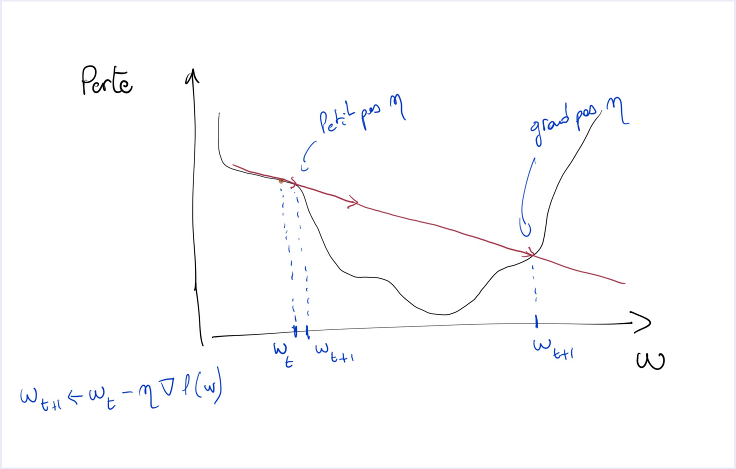


Perte ℓ
$\lim_{\varepsilon \to 0} \frac{\ell(w+\varepsilon) - \ell(w)}{\varepsilon} < 0$
$\lim_{\varepsilon \to} \frac{\ell(w+\varepsilon) - \ell(w)}{\varepsilon} > 0$
w
w
w
aller dans le sens opposé de la dérivée (gradient)

The gradient gives the direction: opposite to the gradient

Gradient descent IV

Step by step



We're taking a step in this direction, by a factor yet to be specified: the learning rate.

Algorithm

- Start with random weights, which define a point on the curve of the objective function.
- Calculate the derivative (gradient) of the objective function at this point
- One step is taken in the opposite direction to this gradient. The length of this step is determined by the learning rate.
- A termination check is performed on the norm of the gradient (tolerance) or on the error.
- Notes
  - In the convex case (as with MSE), a global minimum is guaranteed.
  - It is preferable to have normalized dimensions (use `StandardScaler` for example).
  - Dynamic gradient step can be used: the learning rate value decreases slowly (on the order of $1/t$).

More formally

- Input: Sample $S = \{(\boldsymbol{x}^{(1)}, y_1), (\boldsymbol{x}^{(2)}, y_2), \ldots, (\boldsymbol{x}^{(m)}, y_m)\}$, a loss function f, a learning rate η
- Init: $\boldsymbol{w}^{(0)}$ recieves random values
- Repeat until convergence
  - $t = t + 1$
  - $\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \nabla_{\boldsymbol{w}^{(t)}} f(\boldsymbol{w}, S)$
- Output : $\boldsymbol{w}^{(t)}$

Batch and stochastic

Gradient descent (Batch Gradient Descent)

- The algorithm requires each iteration to pass over the entire data set (reminder: the loss function is a sum over all examples).
- The cost of this calculation can still be prohibitive
- Sometimes difficult to escape from local minima

Stochastic Gradient Descent (SGD)

- A single example is selected at each round
- Advantages
  - fast
  - if the loss function is very unstable, SGD can get out of local minima

Mini Batch Gradient Descent

- a subset of the examples is selected
- suitable for GPUs, as we can now perform small matrix operations on a batch of data

Polynomial regression

- Linear models have been constructed (straight lines, hyperplanes, etc.).
- This class of hypotheses may be ill-suited to non-linear data.
- We can get around this problem by adding new attributes to the data: square, cube, pairwise products of attributes, etc.
- In the 1-dimension case, let's solve

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_k x^k$$

.

- The data are transformed, but the same linear regression algorithm is applied!
- The same applies if x is a vector in $\mathbb{R}^d$ and products can be made between different attributes ($x_i x_j$, $x_i x_j x_k \ldots$).
- NB: we can avoid materializing the construction of all these new attributes using kernel techniques (used in SVMs, ...).

Technically

- with `PolynomialFeatures`.
- we can use `Pipeline` class objects to compose different stages
  - feature transformation

    ```
    estimators = [('reduce_dim', PCA()), ('clf', SVC())]
    pipe = Pipeline(estimators)
    # two _ to access params
    pipe.set_params(clf__C=10)
    ```
  - application of a model with `pipe.fit`, (`score`, `predict`,...)