Notes sur la décomposition QR et les moindres carrés

Maximilien

July 2020

Notions MIASHS fonctions, $\mathbb{R}^n$, calcul de dérivés, gradient

1 Introduction

En *machine learning*, le problème peut être formulé comme la recherche d'une fonction de $\mathcal{X}$ dans $\mathcal{Y}$. Sans rentrer dans les détails, une manière d'atteindre cet objectif consiste à définir une forme paramétrique de notre fonction et une notion d'erreur. Il s'agit de la vision *machine learning* qui fait finalement écho à la vision statistique. En effet, la notion d'erreur en *machine learning* fait souvent référence à l'inverse de la log-vraisemblance. Ainsi, l'objectif est de trouver la paramétrisation de notre modèle qui maximiserait la vraisemblance d'un jeu de données. Revenons en à une vision plus *machine learning*.

Soit $\mathcal{X} = \mathbb{R}^d$, et $\mathcal{Y} = \mathbb{R}$ et soit un jeu de données sur $\mathcal{X} \times \mathcal{Y}$, noté $\mathcal{D} = \{(x_i, y_i)\}_{i \leq N}$ (de taille N). Supposons que nous disposions d'une notion d'erreur quadratique et que nous restreignions notre modèle aux paramétrisations de la forme :

$$y = \langle \beta, x \rangle.$$

Cette vue se généralise aux modèles affines. En effet il suffit de considérer une dimension supplémentaire qui prendrait la valeur 1 pour chaque élément de notre jeu de données.

L'erreur quadratique se formule ainsi de la manière suivante :

$$Re(\beta) = \|X\beta - Y\|_2^2,$$

où $X \in \mathbb{R}^{N \times d}$ et $Y \in \mathbb{R}^N$ sont définis comme suit :

$$X = \begin{bmatrix} x_1^T \\ \dots \\ x_N^T \end{bmatrix} \text{ et } Y = \begin{bmatrix} y_1 \\ \dots \\ y_N \end{bmatrix}$$

Le point de vu statistiques nous dirait que la variable à expliquer y a une dépendence linéaire avec la variable explicative x à un bruit gaussien ϵ près.

Minimiser le risque quadratique revient ici à maximiser la log-vraisemblance de notre modèle "Gaussien".

L'objectif de *machine learning* associé devient donc le suivant :

$$\hat{\beta} = \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \, Re(\beta) = \underset{\beta \in \mathbb{R}^d}{\operatorname{argmin}} \, \|X\beta - Y\|_2^2 .$$

De manière un peu plus générale, soit un système d'équations linéaires $X\beta = Y$ qu'on cherche à résoudre en β. Selon la situation et en particulier s'il y a plus d'équations que d'inconnues, ce système n'admet pas forcément de solution et on cherchera $X\beta \approx Y$ où l'approximation est prise dans le sens où on minimisera l'écart quadratique.

2 Les équations normales

Les équations normales sont celles obtenues lorsqu'on cherche à annuler le gradient de notre erreur afin d'en trouver le minimum (notre problème étant strictement convexe et coercif si X est plein rang). On cherche en particulier à trouver le minimum de la fonction :

$$Re(\beta) = (X\beta - Y)^T(X\beta - Y) = \beta^T X^T X\beta + Y^T Y - 2\beta^T X^T Y.$$

Pour cela, on cherche β tel que le gradient est nul. On obtient ainsi :

$$X^T X\beta = X^T Y$$

Il s'agit des équations normales. $X^T X$ est la matrice de Gram. Toute solution $\hat{\beta}$ du problème d'optimisation doit les satisfaire. Le problème d'optimisation étant convexe, toute solution locale est une solution globale. Cependant, il existe certaines situations où une infinité de solutions sont possibles. Toutes ces solutions rendent la matrice $X^T X$ singulière et donc non inversible. Tout d'abord, $X \in \mathbb{R}^{N \times d}$. Si $N < d$ alors il y a plus d'inconnues que d'équation et $X^T X$ est singulière. Si la famille des vecteurs colonnes de X est liées, alors, à nouveau, $X^T X$ est singulière. Cependant, dans les autres cas de figure, $X^T X$ est inversible. Et on obtient :

$$\beta = (X^T X)^{-1} X^T Y$$

Où $X^\dagger = (X^T X)^{-1} X^T$ est ce qu'on peut appeler le pseudo-inverse.

Si $N \geq d$ mais que la famille des vecteurs colonnes est liées, il suffit de perturber certains éléments de X de manière infinitésimale pour qu'elle devienne à inversible. Cependant, l'opération $X^T X$ va rendre ces petite perturbation encore plus petite, et un ordinateur risquerait d'arrondir et de rendre à nouveau $X^T X$ singulière. Ce problème peut se produire dans la réalités lorsque certaines variables apportent "peu" d'information par rapport à des variables existantes. Autant, si la famille de vecteurs est liée, il suffit d'en extraire une famille libre,

autant, s'il s'agit d'un problème numérique qui apparaît au moment de la résolution, cela devient plus embêtant. Au-delà de la non inversibilité, cela crée des problèmes de stabilité numérique (arrondis avec effets significatifs).

La méthode via une décomposition QR permet d'obtenir une optimisation stable en évitant d'inverser la matrice $X^T X$.

3 Décomposition QR

Afin de faciliter la lecture, travaillons avec une matrice appelée A (et non X) dans $\mathbb{R}^{m \times n}$. La décomposition QR de la matrice A est :

$$A = QR,$$

où Q est une matrice orthogonale (par colonne si rectangulaire) et R une matrice diagonale supérieure. On retrouve d'ailleurs parfois l'appellation "décomposition QU" où U signifie *Upper triangular*. Une matrice orthogonale par colonne implique $Q^T Q = I$ et donc $m \geq n$. En effet, si la famille de vecteur est plus grande que la dimension de l'espace, alors elle est forcément liée.

3.1 Procédé de Gram-Schmidt

Il existe plusieurs stratégies permettant de réaliser cette décomposition et nous utiliserons celle basée sur le procédé (ou algorithme) de Gram-Schmidt. Soit $F = \{u_1, ..., u_n\}$ une famille de vecteurs libres. Le procédé de Gram-Schmidt a pour objectif de construire une base orthonormale $B = \{e_1, ..., e_n\}$ à partir de F.

L'opération de base du procédé de Gram-Schmidt est l'opérateur de projection :

$$\text{proj}_u(v) = \Pi_u(v) = \frac{\langle v, u \rangle}{\langle u, u \rangle} u,$$

où le vecteur v est projeté orthogonalement sur u.

Le procédé itère sur l'ensemble des vecteurs de la famille F de la manière suivante.

1. $v_1 = u_1$ et $e_1 = v_1 / \|v_1\|_2$,
2. $v_2 = u_2 - \Pi_{e_1}(u_2)$ et $e_2 = v_2 / \|v_2\|_2$,
3. ...
4. $v_n = u_n - \sum_{i=1}^{n-1} \Pi_{e_i}(u_n)$ et $e_n = v_n / \|v_n\|_2$.

La famille $\{e_1, ..., e_n\}$ ainsi construite est une base orthonormée.

3.2 Décomposition QR

Soit $A \in \mathbb{R}^{m \times n}$ où les vecteurs colonnes sont libres. Notons $\{a_1, ..., a_n\}$ l'ensemble des vecteurs colonnes. Soit $\{e_1, ..., e_n\}$ une base orthonormale résultant du procédé de Gram-Schmidt appliqué aux vecteurs colonnes de A.

De manière assez directe, on observe que $a_1 = \langle a_1, e_1 \rangle e_1$. Dit autrement, a_1 est un vecteur co-linéaire à e_1 dont la norme est $\langle e_1, a_1 \rangle$ (sachant que e_1 est unitaire). Le vecteur a_2 est un peu plus complexe à reconstruire : $a_2 = \langle a_2, e_2 \rangle e_2 + \langle a_2, e_1 \rangle e_1$. Autrement dit, a_2 est une combinaison linéaire de e_1 et e_2, ce qui est logique puisque e_2 est construit en retirant la composante non orthogonale à e_1 de a_2. On réitère l'opération jusqu'à $a_n = \sum_i \langle e_i, a_n \rangle e_i$.

En notant :

$$Q = [e_1, ..., e_n] \text{ et } R = \begin{bmatrix} \langle e_1, a_1 \rangle & \langle e_1, a_2 \rangle & \langle e_1, a_3 \rangle & \dots & \langle e_1, a_n \rangle \\ 0 & \langle e_2, a_2 \rangle & \langle e_2, a_3 \rangle & \dots & \langle e_2, a_n \rangle \\ 0 & 0 & \langle e_3, a_3 \rangle & \dots & \langle e_3, a_n \rangle \\ 0 & 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & 0 \dots & \langle e_n, a_n \rangle \end{bmatrix}$$

on retrouve bien $A = QR$.

3.3 Exemple

Soit la matrice suivante :

$$A = \begin{bmatrix} 1 & -1 \\ 2 & 0 \\ 2 & 2 \end{bmatrix} = [a_1, a_2]$$

Commençons la procédure de Gram-Schmidt.On note :

$$v_1 = a_1 \text{ et } e_1 = v_1 / \|v_1\|_2 = \left[\frac{1}{3}, \frac{2}{3}, \frac{2}{3}\right]^T$$

Et :

$$v_2 = a_2 - \Pi_{e_1}(a_2) \text{ et } e_2 = v_2 / \|v_2\|_2 = \left[-\frac{2}{3}, -\frac{1}{3}, \frac{2}{3}\right]^T$$

On vérifie assez bien que e_1 et e_2 sont orthogonaux et unitaires. Calculons maintenant les produits scalaires :

$$\langle e_1, a_1 \rangle = 3, \ \langle e_1, a_2 \rangle = 1 \text{ et } \langle e_2, a_2 \rangle = 2$$

Nous avons donc

$$Q = \begin{bmatrix} 1/3 & -2/3 \\ 2/3 & -1/3 \\ 2/3 & 2/3 \end{bmatrix} \text{ et } R = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$$

On vérifie facilement qu'on a bien l'égalité $A = QR$.

4 Application aux moindres carrés

L'objectif initial est de trouver une solution β au système d'équations linéaires $X\beta = Y$ (formulation régression linéaire) ou $Ax = b$ (formulation qu'on

peut retrouver ailleurs). En pratique, le système d'équations linéaires est souvent sur-déterminés et il n'existe pas de solution satisfaisant l'égalité. On cherchera alors $\hat{\beta}$ tel que $\|X\beta - Y\|_2$ est minimisé. Nous avons vu que la solution était comme suit :

$$\begin{aligned}\hat{\beta} &= (X^T X)^{-1} X^T Y \\ &= ((QR)^T QR)^{-1} (QR)^T Y \\ &= ((R^T Q^T QR)^{-1} R^T Q^T Y \\ &= (R^T R)^{-1} R^T Q^T Y \\ &= R^{-1} (R^T)^{-1} R^T Q^T Y \\ &= R^{-1} Q^T Y.\end{aligned}$$

où $X = QR$. Cette stratégie, comme nous allons le voir, est beaucoup plus stable.

5 Exemple comparatif

Soit la matrice suivante :

$$X = \begin{bmatrix} 1 & -1 \\ 0 & 10^{-5} \\ 0 & 0 \end{bmatrix} \text{ et } Y = \begin{bmatrix} 0 \\ 10^{-5} \\ 0 \end{bmatrix}$$

Supposons que l'algorithme tourne sur une machine où les nombres sont arrondis après 8 décimales (i.e. si $|x| < 10^{-8}$ alors $x := 0$).

La première étape consiste à calculer le produit :

$$X^T X = \begin{bmatrix} 1 & -1 \\ -1 & 1 + 10^{-10} \end{bmatrix} \approx \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

Or

$$\left| \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \right| = 0$$

et n'est donc pas inversible. On ne peut ainsi pas déduire de notre algorithme la solution.

La stratégie QR nous donne :

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \text{ et } R = \begin{bmatrix} 1 & -1 \\ 0 & 10^{-5} \end{bmatrix}$$

On observe que pour l'instant, il n'y a pas d'arrondis. Calculons les différents éléments de notre solution :

$$R^{-1} = \begin{bmatrix} 1 & 10^5 \\ 0 & 10^{i5} \end{bmatrix} \text{ et } Q^T \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\hat{\beta} = R^{-1}Q^TY = \begin{bmatrix}1\\1\end{bmatrix}$$

Sans avoir eu besoin d'approximation. Notons que `numpy` n'arrondit pas après 8 décimales, mais l'instabilité numérique joue. Nous avons ainsi :

$$\hat{\beta}_{\texttt{numpy}} = \begin{bmatrix}\texttt{0.99999992}\\\texttt{0.99999992}\end{bmatrix} \text{ et } \hat{\beta}_{\texttt{numpy,QR}} = \begin{bmatrix}\texttt{1.}\\\texttt{1.}\end{bmatrix}$$

Cette exemple simple montre déjà les avantages de la décomposition QR dans le cadre des moindres carrés. On image sans mal son intérêt dans des exemples beaucoup plus compliqués où les dépendances linéaires sont peut-être plus difficiles à discerner au milieu des perturbations et du bruit.