Rapport: TP optimisaton

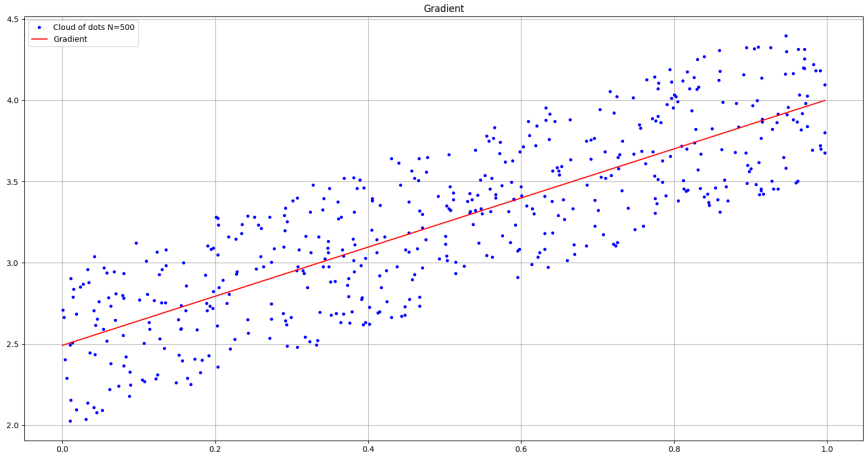
Maxence Montavon & Damian Boquete Costa

2021 - 2022

Introduction

Dans le cadre de ce travail nous avons crée un programme permettant d'effectuer une regression linéaire, à partir d'un ensemble de données, grâce à la méthode de descente de gradient. Celui ci est implémenté dans le language C et les resultats du programme sont visualisabes grace a la librairie matplotlib de python3. Nous commencerons par détailler l'aspect théorique du concept, suivi par l'aspect technique. Ensuite nous vous présenterons nos résultats en répondant aux questions qui nous ont été posées.

Petit aperçu de ce que produit notre programme:



Partie théorique

La regression linéraire est un concept mathématique permettant d'estimer un passage, dit optimal, d'une droite à travers un ensemble de données plus ou moins différentes les unes des autres. Dans le but de représenter au mieux, graphiquement, un résumé grossier de la tendance qu'ont ces données a prendre telle ou telle direction. Ceci permettant donc d'émettre des prédictions à partir d'un ensemble de données. Cette technique est notament très utilisée dans le domaine des statistiques et du machine learning.

Ceci étant, une question legitime se pose à partir de maintenant : Comment ça marche ? Premièrement, il faut savoir qu'il existe plusieures techniques pour arriver à nos fins. De notre côté nous avons utilisé la descente de gradient. La descente de gradient est un algorithme permettant de trouver le minimum ($pente = 0$) d'une fonction convexe (ou le maximum d'une fonction concave si on inverse le signe). Dans notre cas, nous cherchons à minimiser la fonction d'erreur quadratique suivante:

$$E(a,b) = \sum_{i=1}^{N} (x_i \cdot a + b - y_i)^2$$

Pour se faire, il faut commencer par dériver cette fonction par rapport à nos deux variables (a, b):

$$\frac{\partial E}{\partial a} = 2 \sum_{i=1}^{N} (a \cdot x_i + b - y_i) \cdot x_i$$

$$\frac{\partial E}{\partial b} = 2 \sum_{i=1}^{N} (a \cdot x_i + b - y_i)$$

Une fois les derivées faites, il est possible de remanier les formules afin de donner celles que nous utilisons dans notre programme:

$$a = a \cdot \sum_{i=1}^{N} x_i^2 + b \cdot \sum_{i=1}^{N} x_i - \sum_{i=1}^{N} x_i y_i$$

$$b = a \cdot \sum_{i=1}^{N} x_i + N \cdot b - \sum_{i=1}^{N} y_i$$

C'est ainsi que nous consituons notre gradient $\vec{\nabla} E\binom{a}{b}$. Celui-ci est utilisé pour chaque point constituant notre nuage de données. De ce fait, pour trouver la droite optimale passant par notre nuage de points, il est maintenant nécessaire d'effectuer la descente de gradient.

Pour ce faire, commençons par choisir une valeur lambda λ (la taille de nos "pas"), une valeur epsilon ϵ (une valeur d'arrêt) et un point de départ:

$$\lambda \in]0,1] \quad p.e\ \lambda = 0.001$$
$$\epsilon \in]0,1] \quad p.e\ \epsilon = 0.001$$
$$\vec{depart} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

La descente de gradient consiste simplement à soustraire le vecteur courant par le gradient, lui-même multiplié par la valeur de λ:

$$\vec{x}_{i+1} = \vec{x}_i - \lambda \cdot \vec{\nabla} E \begin{pmatrix} a \\ b \end{pmatrix}$$

On s'assure à la fin de chacunes des itérations que la différence entre notre nouveau vecteur et le vecteur de l'itération précédente ne soit pas plus petite que la valeur de ϵ fixée au préalable:

$$\|\vec{x}_{i+1} - \vec{x}_i\| > \epsilon$$

Quand la valeur est plus petite que notre ϵ, alors on s'arrête et on garde la dernière valeur trouvée. On se retrouve donc avec la pente et l'ordonnée à l'origine de notre droite "optimale". De ce fait, il nous est donc possible de dessiner cette droite et constater visuellement le resultat de nos efforts.

Dans le but de valider la solution dite "numérique" de notre programme, il est possible d'effectuer une solution "analytique" puis de les comparer afin de savoir si ce que nous faisons est cohérent.

Pour ce faire, il faut partir des équations déjà définies précédemment et créer un système d'équations dans lequel il faudra isoler a et b.

$$1: \quad a = a \cdot \sum_{i=1}^{N} x_i^2 + b \cdot \sum_{i=1}^{N} x_i - \sum_{i=1}^{N} x_i y_i$$

$$2: \quad b = a \cdot \sum_{i=1}^{N} x_i + N \cdot b - \sum_{i=1}^{N} y_i$$

A des fins de lisibilité, nous allons commencer par remplacer toutes les sommes par des lettres majuscules:

$$A = \sum_{i=1}^{N} x_i \qquad B = \sum_{i=1}^{N} y_i \qquad C = \sum_{i=1}^{N} x_i y_i \qquad D = \sum_{i=1}^{N} x_i^2$$

Nous allons maintenant isoler a en foncion de b dans la seconde équation du système d'équation ci-dessus:

$$aA + Nb - B = 0 \quad \Leftrightarrow \quad aA = B - Nb \quad \Leftrightarrow \quad a = \frac{B - Nb}{A}$$

Par la suite nous devons remplacer l'équation trouvée dans la première équation afin d'obtenir une équation où b est indépendant de a:

$$\frac{B - Nb}{A} \cdot D + bA - C = 0 \quad \Leftrightarrow \quad \frac{BD - NbD}{A} + bA = C$$
$$\Leftrightarrow \quad \frac{BD - NbD + bA^2}{A} = C \quad \Leftrightarrow \quad BD - NbD + bA^2 = AC$$
$$\Leftrightarrow \quad -NbD + bA^2 = AC - BD \quad \Leftrightarrow \quad bA^2 - NbD = AC - BD$$
$$\Leftrightarrow \quad b(A^2 - ND) = AC - BD \quad \Leftrightarrow \quad \frac{b(A^2 - ND)}{A^2 - ND} = \frac{AC - BD}{A^2 - ND}$$
$$\Leftrightarrow \quad b = \frac{AC - BD}{A^2 - ND}$$

Pour terminer de résoudre notre système d'équation, il nous faut à présent, trouver une équation où a est indépendant de b. Il nous suffit de remplacer b dans l'équation précédemment trouvée pour a :

$$a = \frac{B - Nb}{A} \quad \Leftrightarrow \quad a = \frac{B - N \cdot (\frac{AC - BD}{A^2 - ND})}{A}$$

Pour terminer nous pouvons retirer les lettres majuscules et nous obtenons nos équations:

$$a = \frac{\sum_{i=1}^{N} y_i - N \cdot (\frac{\sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} y_i \cdot \sum_{i=1}^{N} x_i^2}{(\sum_{i=1}^{N} x_i)^2 - N \cdot \sum_{i=1}^{N} x_i^2})}{\sum_{i=1}^{N} x_i}$$
$$b = \frac{\sum_{i=1}^{N} x_i \cdot \sum_{i=1}^{N} x_i y_i - \sum_{i=1}^{N} y_i \cdot \sum_{i=1}^{N} x_i^2}{(\sum_{i=1}^{N} x_i)^2 - N \cdot \sum_{i=1}^{N} x_i^2}$$

Partie technique

Maintenant que la partie théorique est éclaircie, il est temps de se pencher sur la partie technique. Pour commencer, nous avons implémenté un système de génération de "nuage de points" (ensemble de N points éparpillés aléatoirement).

Pour ce faire, chaque point est généré en respectant les règles suivantes:

- Une pente et une ordonnée à l'origine doivent être choisi pour les appliquer sur l'enssemble des points à générer:

$$p = 2 \cdot g, g \in R, g \in [0, 1] \quad (pente)$$

$$o = 2 \cdot h, h \in R, h \in [0, 1] \quad (ordonnée\ à\ l'origine)$$

- Suite à cela, nous allons calculer les valeurs suivantes durant N itérations:

$$x_i \in R, x_i \in [0, 1] \quad (valeur\ en\ x\ aléatoire)$$

$$r_i \in R, r_i \in [0, 1] \quad (nombre\ aléatoire)$$

$$y_i = x_i \cdot p + o + r$$

Avec la génération de nuage de N points opérationnelle, il nous a été nécessaire d'implémenter deux fonctions s'occupant de calculer la dérivée en a et en b (explicité dans la partie théorique).

Avec ces prérequis, il est maintenant possible de s'attaquer à l'implémentation techique de la descente de gradient. Celui-ci nécessite donc un nuage de points de taille N, un point de départ arbitraire (généralement égal à $\vec{x} = \binom{0}{0}$), une valeur λ et une valeur ϵ. Le calcul est simplement le même que celui représenté dans la partie théorique.

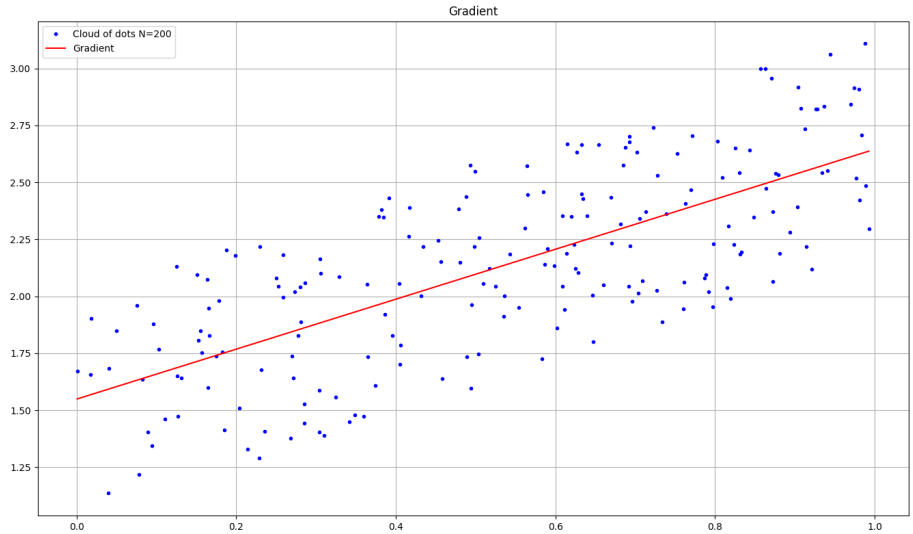
Le système d'affichage des résultats étant "fait maison", il nous semble nécessaire de brievement expliciter certains points le concernant. Celui-ci est constitué d'une fonction qui exporte la liste de points ainsi que le gradient obtenu (placé à la fin du fichier). Ce fichier est ensuite passé en argument d'un autre script (avec 3 autres fichiers concernant la validation croisée) dans le but de les afficher.

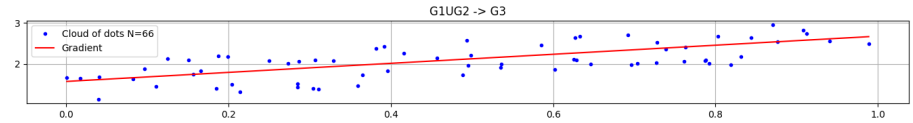
Au sujet de la validation croisée, nous avons segmenté notre nuage de points en 3 sous groupes. Dans lesquels les points sont répartis de façon aléatoire, dans le but de permettre plusieurs tests de validations croisées en une execution de programme si désiré.

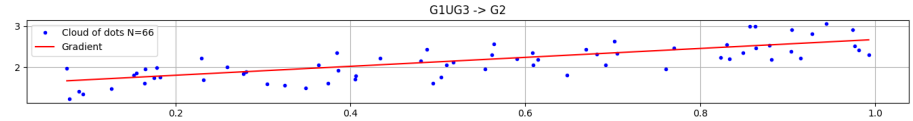
Ces groupes sont ensuite utilisés lors de ladite validation croisée et affichés par le même système brièvement décrit précédement. Les résultats concernant l'erreur quadratique sont visibles sur la console.

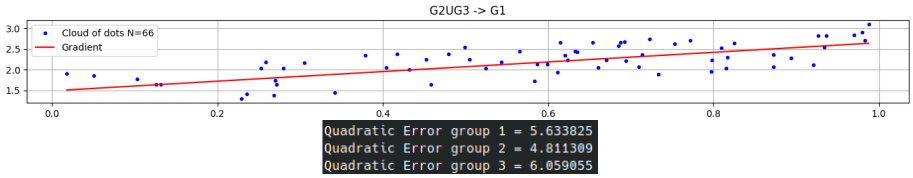
Quelques résultats

Voici un exemple de la sortie de notre programme avec la génération de 200 points aléatoires:





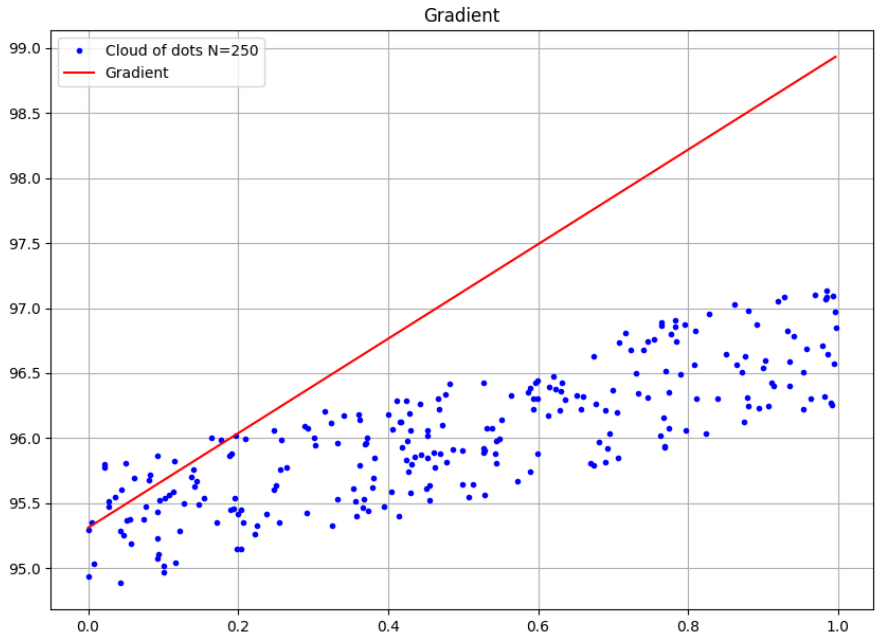




Nous allons maintenant répondre aux différentes questions qui nous ont été posées:

En augmentant c (la pente) et d (l'ordonnée à l'origine) les résultats de notre solution numérique sont toujours cohérent. Plus nous augmentons c, plus l'intervalle de l'axe y pour nos points sera grand. Plus notre d est grand, plus l'ordonnée à l'origine de notre gradient sera grand. Mais notre gradient ainsi que son erreur quadratique moyenne restent cohérents. **Par contre** si nous augmentons d et testons notre solution analytique, implémentée en C (programme test), la valeur pente est totalement faussée. Nous en avons déduis donc que

notre solution analytique possède des erreurs de calculs mathématique. Gradient calculé avec la solution analytique (d entre 0 et 50) :



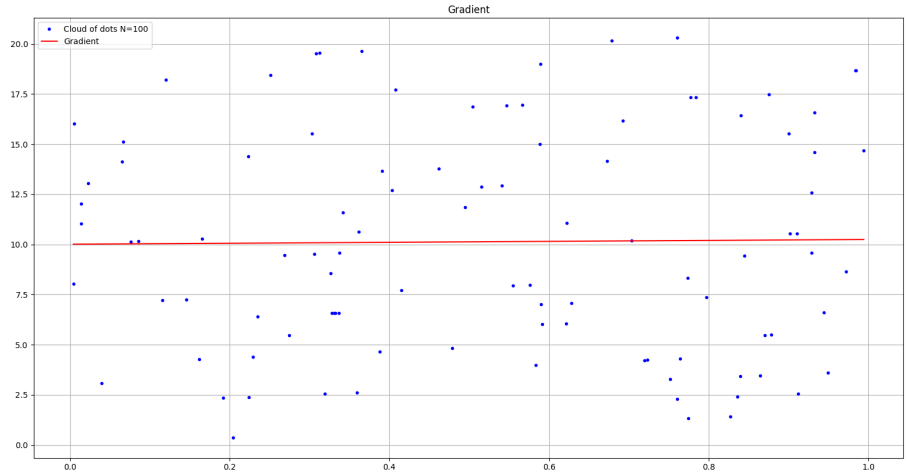

```
Analytic and numeric solutions comparison :
-------------------------------------------
Analytique solution slope : 3.635847
Numeric solution slope : 1.430261
Difference between absolute values of slopes = 2.205587

Analytique solution intercept : 95.310499
Numeric solution intercept : 95.309982
Difference between absolute values of intercept = 0.000518
```

L'erreur quadratique est une valeur numérique qui représent à quel point le modèle que nous avons calculé est correcte vis-à-vis du nuage de points actuel. Donc à quel point notre gradient est, en moyenne, proche de chacuns des points.

L'augmentation de r_i dans la génération des points agrandis "l'épaisseur" du nuage de points sur laquelle les points sont générés, et notre gradient en est grandement impacté ! Plus r_i augmente, plus la fonction d'erreur est mauvaise, car la droite est de plus en plus loin des points sur l'axe des ordonnées.

Exemple avec r_i aléatoire entre 0 et 20:



Conclusion

Pour terminer, notre programme effectue correctement une régression linéaire grâce à la méthode de descente de gradient et nous affiche en console, la valeur d'erreur de ce dernier, pour un nuage de points aléatoire donné. A l'inverse, notre solution analytique n'est quant à elle pas totalement correcte et pourrait être améliorée. L'affichage est simple et nous permet d'exemplifier nos résultats mais il pourrait, lui aussi, être retravaillé avec par exemple l'affichage d'un tableau contenant les points générés.