Modèle prédictif de classification d'image

TRABET Clément AKNIN Louis MARCHAND Josef

June 27, 2021

Contents

1 introduction au machine learning

Qu'est ce que le machine learning

Le machine learning consiste à programmer des ordinateur afin d'optimiser les prédictions des résultats d'un problèmes en se servant de données issue d'expériences dont on connait le résultat.

Pourquoi parle t-on d'apprentissage ?

On ne parle pas d'apprentissage quand on résous un calcul ou quand on tris une liste. On parle d'apprentissage quand : les humains ne sont pas capable d'expliquer à l'ordinateur ce qu'il faut faire (déplacement d'un robot sur une planète), quand ils ne sont pas capable d'expliquer comment ils analysent un signal (reconnaissance vocal), quand les solutions change au cours du temps ou encore quand on a besoin d'appliquer une même méthode à des cas particuliers.

Application divers du machine learning

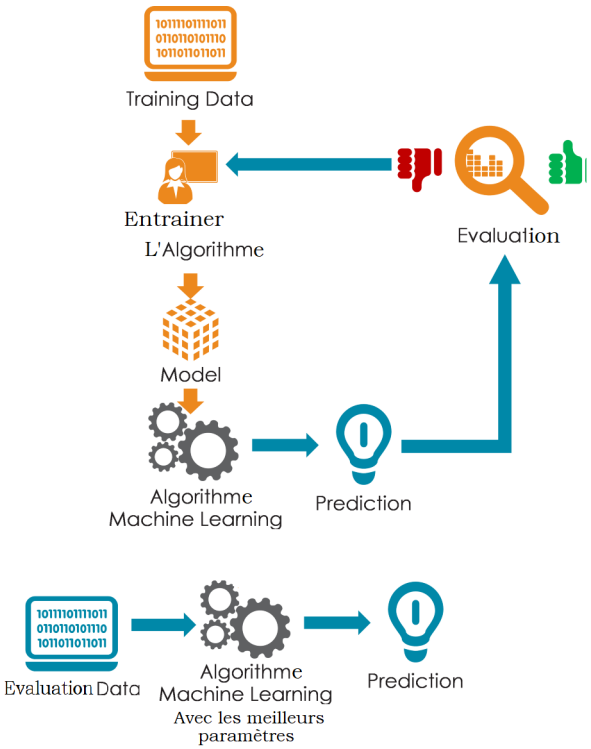
Le machine learning est de plus en plus présent dans notre vie. Il permet la reconnaissance faciale et vocale sur nos smartphone, la prédiction de la bourse, de trouver les résultats les plus cohérents lors d'une recherche sur internet, le diagnostique médical de cellules cancereuses, la détection de spam de malwares.

2 régression linéaire

2.1 qu'est-ce que la régression linéaire ?

La régression linéaire est la construction d'un modèle affine à partir d'exemples dont l'issue est connue dans le but de pouvoir par la suite donner une prédiction la plus proche de la réalité.

Il convient d'appliquer cette méthode uniquement à des variables dont le modèle nous semble déjà linéaire:



2.2 Dataset

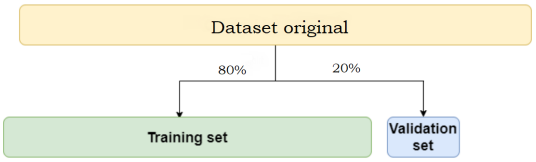
Tableau de données (X,Y) qui contient deux types de variables y appelés Target et les X appelés features.

$$X = \begin{pmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(m)} \end{pmatrix}$$

$$Y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

La première chose à faire est de découper ce dataset en deux parties le training-set et le validation-set. Le training-set est le jeu de données sûr lequel on va entrainer l'algorithme et une fois entrainé on va valider notre algorithme sur un jeu de données dit d'évaluation dont on ne se sera par servi lors de l'apprentissage de l'algorithme.

En notant m le nombre d'exemples que contient notre tableau (le nombre de ligne) et n le nombre de features (le nombre de colonnes de X)



Alors X est une matrice à m colonnes. $X \in \mathbb{R}^m$
Et y est un vecteur à m lignes. y dans $\mathbb{R}^m$ la feature **j** de l'exemple **i** est noté x_j^i

Figure 1: dataset exemple k

Grâce à notre dataset on peut ensuite construire un modèle

2.3 Modèle

Fonction mathématique qui associe X à y, telle que $\forall x in X \ f(x) = y$. On note θ le vecteur qui contient les paramètres de notre modèle. Pour une régression linéaire, la formulation matricielle de la régression linéaire est alors $F(X) = X^T \times \theta$ où $\theta = \begin{pmatrix} a \\ b \end{pmatrix}$ $X^T = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix}$

$$\begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{pmatrix}$$

$$\begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

$\alpha_k = \frac{\alpha}{k}$ $\alpha_k = \frac{\alpha}{(k)^2}$

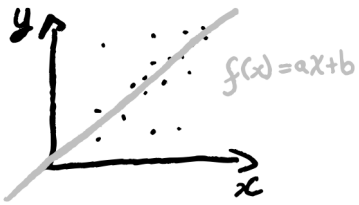


Figure 2: modèle linéaire

Par exemple en reprenant notre dataset précedent on peut construit notre modèle comme un fonction affine $y = ax + b$. Notre but est maintenant de trouver ces deux paramètres a et b.

2.4 Fonction coût

La fontcion Coût $J(\theta)$ mesure les erreurs entre le modèle et le Dataset. C'est en minimisant cette fonction que l'on arrive à approximer les paramètres. Plusieurs méthode peuvent être utiliser pour créer cette fonction coût. Nous utiliserons la méthode des moindre carré nommé *Mean Squared Error* . Cette méthode consiste à prendre en compte la norme euclidienne au carré c'est à dire la distance à notre droite comme erreur. L'erreur liée au kème exemple sera: $erreur^k = (f(x^k) - y^k)^2$

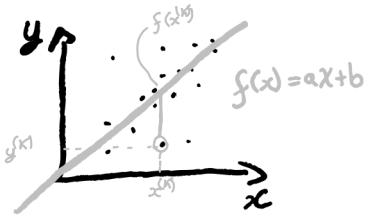


Figure 3: erreur euclidienne

Puis on somme les erreurs pour nos m exemples et on divise par m pour obtenir l'erreur moyenne. D'où la fonction coût qui s'écrit:

erreur total = $\mathbf{J(a,b)} = \sum_{k=1}^{m}(f(x^k) - y^k)^2$

Maintenant il nous faut minimiser cette fonction coût pour trouver les paramètres qui rendent notre modèle le plus proche de la réalité (avec le moins d'erreurs.

2.5 Descente de Gradient

L'algorithme que nous utilisons est l'algorithme de descente de gradient (*gradient descent*

l'algorithme se déroule en trois parties:

Calculer la pente de la fonction coût, la dérivée de $\frac{J(a,b)}{\partial a}$

Avancé d'un pas α appelé learning rate dans la direction de la pente la plus forte

Recommencer les étapes 1 et 2 jusqu'à atteindre le minimum avec un certain écart ϵ

Il faut tout de même faire attention à deux points dans cette algorithme : le pas α , si il est trop petit le temps de calcul sera très long car il faudra itérer de nombreuses fois et ϵ qui si trop petit peu ne jamais faire aboutir le calcul.

$$ak = a - \alpha\frac{\partial J(a,b)}{\partial a}$$
$$bk = b - \alpha\frac{\partial J(a,b)}{\partial b}$$

où les dérivés partielles sont

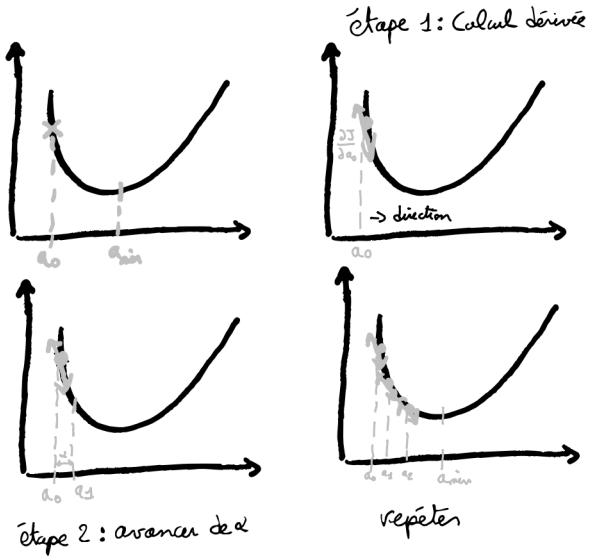


Figure 4: fonctionnement de l'algorithme de descente de gradient

$$\frac{\partial J(a,b)}{\partial a} = \frac{2}{m}\sum_{k=1}^{m}(ax^k + b - y^k)x^k$$
$$\frac{\partial J(a,b)}{\partial b} = \frac{2}{m}\sum_{k=1}^{m}(ax^k + b - y^k)$$

Pour simplifier les calculs on passe sous forme matricielle on a alors:
le modèle: $F(X) = X \times \theta$ où $\theta = \begin{pmatrix} a \\ b \end{pmatrix}$
La fonction coût :

$$J(\theta) = \frac{2}{m}\Sigma_{k=1}^{m}(F(X) - y)^2$$

Notre variant de boucle pour l'algorithme de gradiant:

$$\theta = \theta - \alpha \times \frac{\partial J(\theta)}{\partial \theta}$$
où
$$\frac{\partial J(\theta)}{\partial \theta} = \frac{2}{m}X^T \times (F(X) - Y)$$

2.6 méthode de Newton-Raphson

Soit f une fonction définie sur un intervalle $I =]a, b[$, qui vérifie les hypothèses suivantes :

- il existe deux réels c et d tels que $a < c < d < b$ et $f(c)f(d) < 0$;
- f est dérivable dans l'intervalle I ;

la dérivée de f ne prend que des valeurs positives stricte dans I - f' est continue dans I.

Alors l'équation $f(x) = 0$ admet une unique solution s sur l'intervalle $[c, d]$. Et on définit par récurrence une suite (Xn) à partir de tout point $X0$ de l'intervalle $]s, d]$ en posant $Xn+1 = Xn - \frac{f(xn)}{f'(xn)}$

Cette suite est strictement décroissante et converge vers s.

fonctionnement pratique:

La méthode consiste à introduire une suite (Xn) d'approximation successives de l'équation $f(x) = 0$.

On part d'un $X0$ quelconque.

À partir de $X0$, on calcule un nouveau terme $X1$ de la manière suivante : on trace la tangente à la courbe de f en $X0$. Cette tangente coupe l'axe des abscisses en $X1$

On réitère ce procédé en calculant $X2$ en remplaçant $X0$ par $X1$, puis $X3$ en remplaçant $X1$ par $X2$ et ainsi de suite

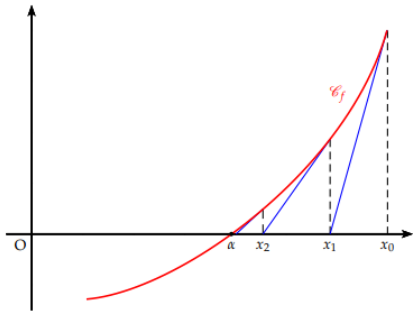
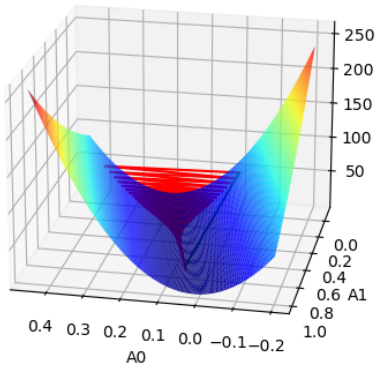


Figure 5: 2 itérations de la méthode de Newton

2.7 comparaison méthode de newton et descente de gradient

Après implémentation des deux méthodes sur notre datasets, la différence de convergence est énorme, 688 itérations pour la méthode de descente de gradient alors que nous avons seulement 3 itérations pour la méthode de newton pour des résultat quasi identique



theta final Newton => $a = 0.06388117 b = 0.75016254$
theta final Gradient => $a = 0.06473147 b = 0.74548894$

Alors que choisir ?

Dans le cas d'une régression linéaire ou polynomiale la méthode de Newton est beaucoup plus intéressante car la fonction du modèle vérifie toute les propriétés pour l'appliquer mais dans des cas plus complexes ce sera la descente de gradient la plus efficace car celle de Newton divergera et ne s'arrêtera jamais.

2.8 Implémentation de la régression linéaire

On importe des valeurs aléatoires pour former notre dataset

```
Entrée [3]: x,y = make_regression(n_samples=100, n_features=1, noise=10)
            plt.scatter(x,y)

Out[3]: <matplotlib.collections.PathCollection at 0x25eb3946670>
```

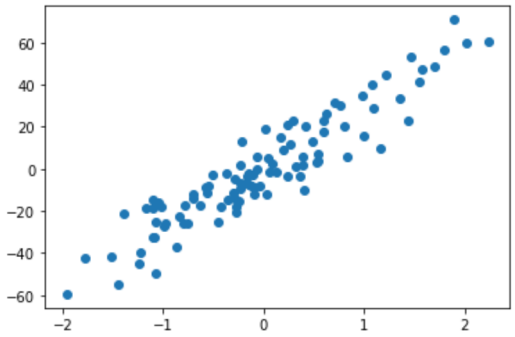


Figure 6: récupération d'un dataset aléatoire

Voici ce que notre modèle donne associée au dataset

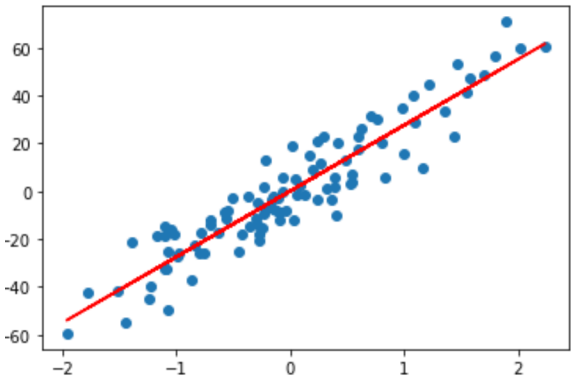


Figure 7: résultat

3 régression polynomial

Une fois que l'on est capable de mettre en place une régression linéaire il est facile de mettre en place une régression avec n'importe quelle modèle. Regardons les modèles polynomiale

Dataset polynomial:

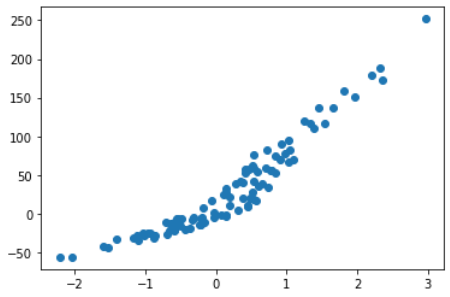


Figure 8: résultat

Toutes les fonctions restent inchangées seul la taille de notre matrice de départ X et la taille de la matrice théta sont à changer.

```
Entrée [10]: #matrice X
X=np.hstack((x, (np.ones(x.shape))) )#colle deux vectteur numpy l'un à côté de l'autre
X=np.hstack((x**2, X))
print(X.shape)
print(X[:10])

(100, 3)
[[ 0.36990509  0.60819824  1.        ]
 [ 0.1417135   0.37644853  1.        ]
 [ 1.10994355  1.05353858  1.        ]
 [ 1.0528476   1.02608362  1.        ]
 [ 0.13310148 -0.36483076  1.        ]
 [ 0.76011786 -0.87184738  1.        ]
 [ 2.76360523  1.66240946  1.        ]
 [ 0.43082988 -0.65637632  1.        ]
 [ 0.51214389 -0.71564229  1.        ]
 [ 0.05235546 -0.22881315  1.        ]]

Entrée [11]: theta=np.random.randn(3,1)
theta

Out[11]: array([[ 0.86641891],
                [-0.72287965],
                [-0.02895819]])
```

Figure 9: résultat

Et on obtient facilement le résultat suivant

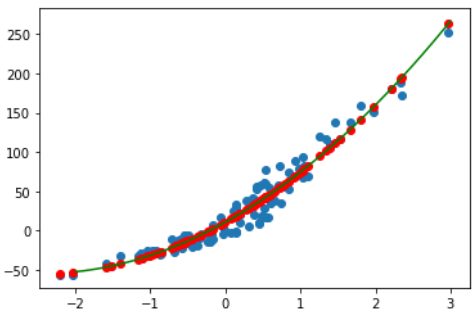


Figure 10: résultat

3.1 Exemple d'application

On étudie le nombre de cas de Covid-19 en France du 25 février au 15/03 mars 2020. Soit les 19 premiers jours où la france a commencé à tester le covid.

On récupère notre dataset selon les informations données sur le site du ministère de la santé (https://dashboard.covid19.data.gouv.fr/vue-d-ensemble?location=FRA)

Jours	2	3	4	5	6	7	8	9	10
Cas de Covid	191	212	285	423	613	949	1126	1412	1784
Jours	11	12	13	14	15	16	17	18	19
Cas de Covid	2281	2876	3661	4500	5423	6633	7730	9134	10995

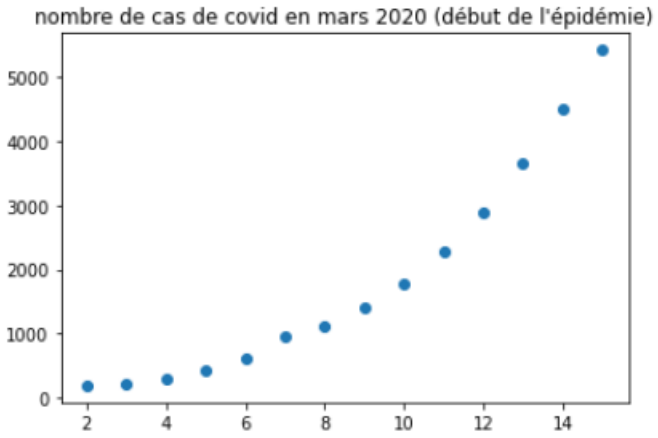


Figure 11: résultat

On utilise seulement les données du 2 au 15 comme training set et on conserve le reste pour un evaluation set. Puis on met en place une régression polynomial d'ordre 4 (la valeur de l'ordre est arbitraire). On obtient alors le résultat suivant.

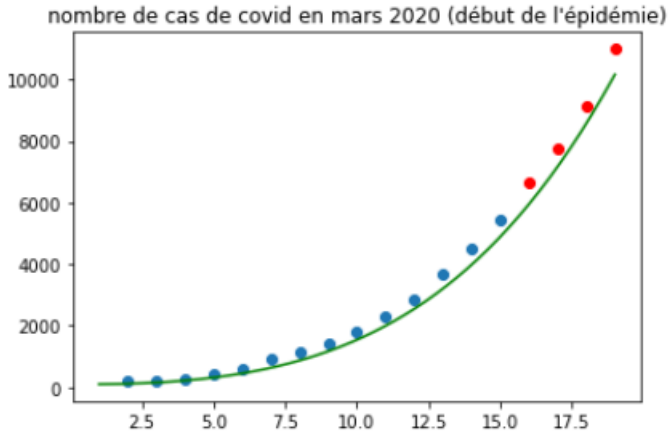


Figure 12: résultat

Les données en rouge sont les données des jours suivants ceux avec lesquels on a pas entraîner l'algorithme. On se rend compte que les résultat son assez proche Mais à plus long terme notre modèle sera totalement erroné.

4 regression multiple

Une fois que l'on est capable de mettre en place une régression linéaire il est facile de mettre en place une régression sur un grand nombre de dimensions.

Prenons un exemple pour comprendre ce que sont les dimensions:

Voici un dataset a deux dimensions:

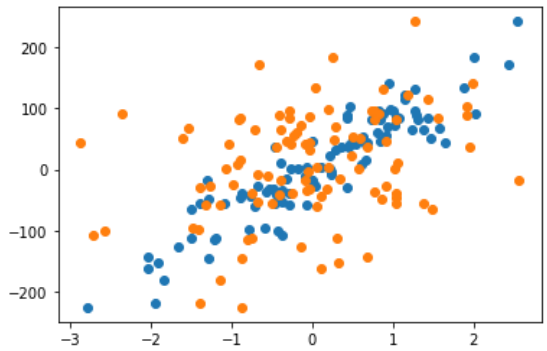


Figure 13: résultat

On voit bien que le nuage de point en bleu est bien linéaire le jaune l'est tout autant seulement pas sur sur le même dimensions. En réutilisant le même méthode que la régression polynomial au lieu d'avoir des colonnes avec X, X^2, .. on aura $X_{dimension1}$,$X_{dimension2}$, ... On change le theta en une matrice de taille $n \times 1$ et on calcul comme dans la regression linéaire et on obtient les courbes suivantes.

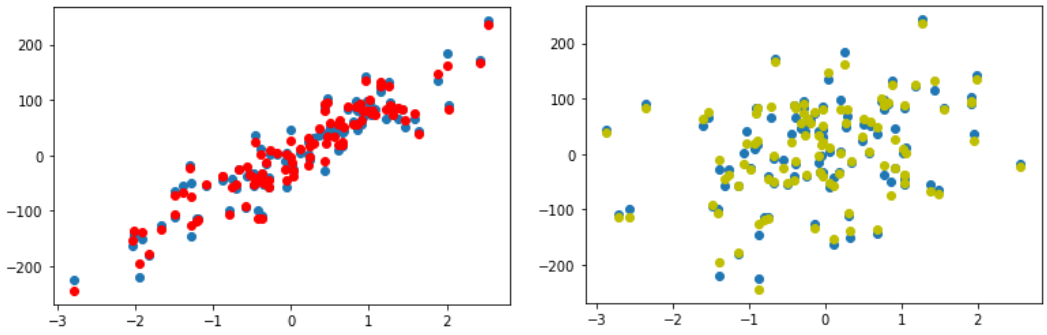


Figure 14: solution dimension 1 à gauche et 2 à droite

Pour ce rendre mieux compte de la performance de notre modèle on peut calculer le célèbre coéfficient R^2. On constate qu'il est de 0.986 ce qui est très élevé car comme vu précédemment si il est plus proche de un on risque de tomber dans l'overfitting et avoir un modèle totalement erroné.

5 Overfitting et Underfitting

Le plus gros problème du machine learning est l'overfitting et l'under fitting. En effet nous recherchons le modèle qui est le plus perfomant pour prédire une valeur mais le bruit (des données écarté de la variance) est inévitable ainsi notre modèle ne sera jamais capable de prédire à 100% la valeur rechercher.

Si on a une prédiction proche de 100% sur notre dataset nous somme dans le cas de l'overfitting (troisième image du graphique) notre modèle détecte le bruit. Il est trop complexe. Et donc il ne pourra pas se généraliser à des valeurs autres. Par exemple dans l'industrie si vous voulez faire un détecteur de pomme et que vous prenez une image de bonne qualité d'une pomme de couleur rouge

et que votre système de régression est en over-fitting alors il sera incapable de détecter des pommes un peu abimés ou dont l'image est flou.

C'est l'exemple de l'interpolation de lagrange en mathématique efficace sur les points recherché mais pas du tout autres part.

Le deuxième problème antagoniste à l'overfitting est l'underfitting dans ce cas la le modèle est trop simple par exemple si on modélise par un droite un évènement non linéaire.

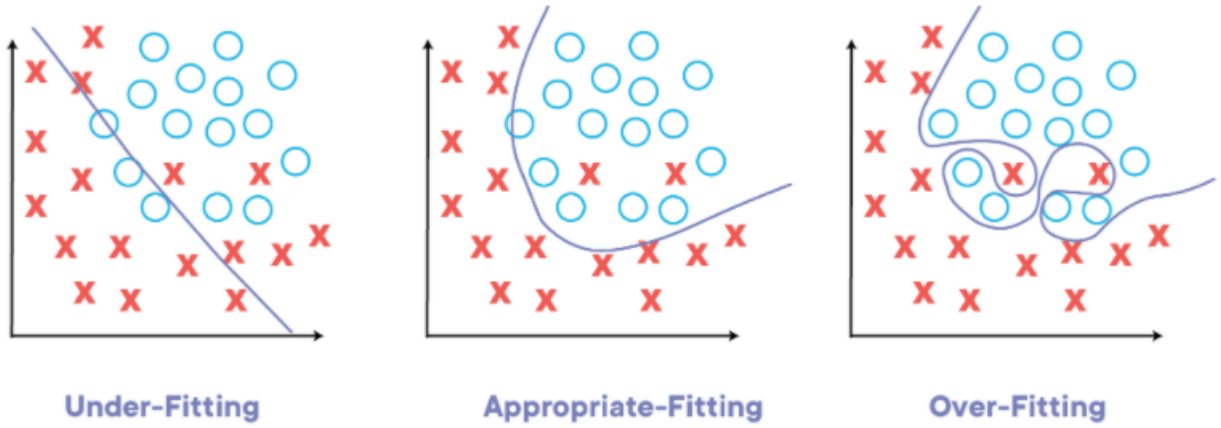


Figure 15: overfitting et underfitting

6 application de la régression

6.1 Exemple avancé: Détection d'image

Situation: imaginons qu'une ambulance veuille rejoindre un hopitâl le plus rapidement possible. Il serais préférable pour elle d'éviter les bouchons. Pour cela elle à besoin de savoir en temps réel si il y a des véhicules sur les routes.
Matériel à disposition : une caméra sur en hauteur permettant d'avoir une image net de la route.

Objectif : être capable de compter le nombre de véhicule et différencier les voitures et les motos

Figure 16: détection de voiture

7 Algorithme général

7.1 Principe de fonctionnement

La méthode que l'on utilise est dites de Viola & Jones (du nom des inventeurs). Cette méthode ce base sur la forme de l'objet récupéré avec des calcul de caractéristiques facile a calculer mais en grand nombre.

Elle consiste à balayer une image à l'aide d'une fenêtre de détection de taille initiale $24px$ par $24px$ (dans l'algorithme original). Et de déterminer la présence de l'objet sur la fenêtre. On balaye l'image en décalant successivement la fenêtre pixel par pixel (on peut augmenter le décalage pour diminuer les calcul). Lorsque l'image a été parcourue entièrement, la taille de la fenêtre est augmentée (facteur 1.25) et le balayage recommence, on fait cela jusqu'à ce que la fenêtre fasse la taille de l'image. Afin de mieux comprendre le fonctionnement de l'algorithme je vous d'aller voir la vidéo qui montre comment fonctionne l'algorithme (**??** disponible ici https://youtu.be/hPCTwxF0qf4). Cette vidéo est ralentie pour montrer le fonctionnement, la vitesse réel est quasi instantanée.

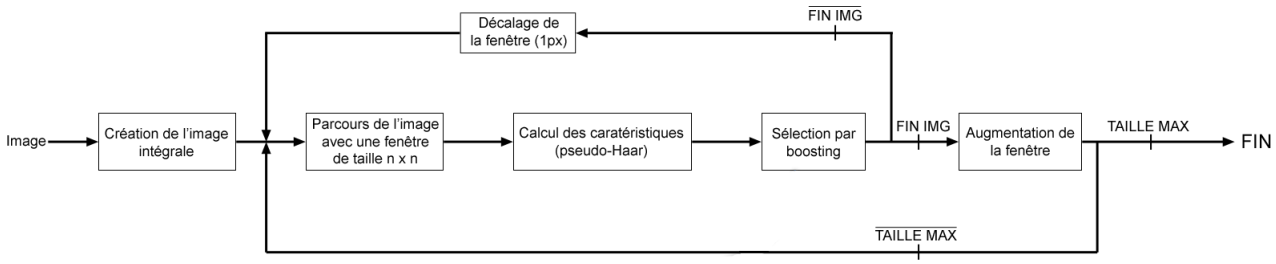


Figure 17: schéma de fonctionnement de l'algorithme

Comme on peut le voir sur la *Figure 1* l'algorithme se décompose en plusieurs étapes:

1. la création d'une image intégral qui permet de simplifier les calculs des caractéristiques
2. le parcours de l'image par des caractéristiques (pseudo-haar)
3. le test de présence proprement dit via la selection par boosting
4. l'augmentation de la taille de la fenêtre de détection

7.2 Image intégral

définition: représentation d'une image sous la forme d'un tableau de valeur, permettant de calculer rapidement des sommes de valeurs dans des zones rectangulaires. Cette représentation est de même taille que l'image initiale et chacun de ses point est la somme des pixels situé au situés au-dessus et à gauche de ce point.

Plus formellement en notant $M_{n,p}$ la matrice contenant en chaque point l'intensité du pixel correspondent sur l'image d'origine, $I_{n,p}$ la matrice que l'on construit.

Pour $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant p$
Le point $I_{i,j} = M_{i,j} + I_{i,j-1} + I_{i-1,j} - I_{i-1,j-1}$

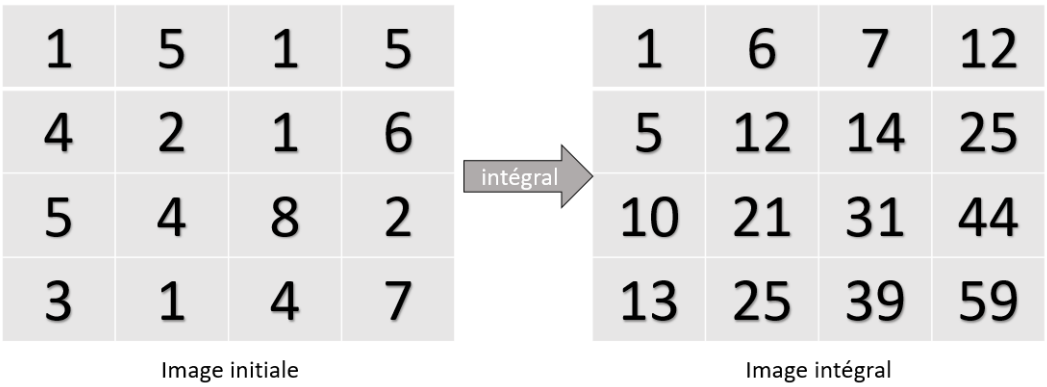
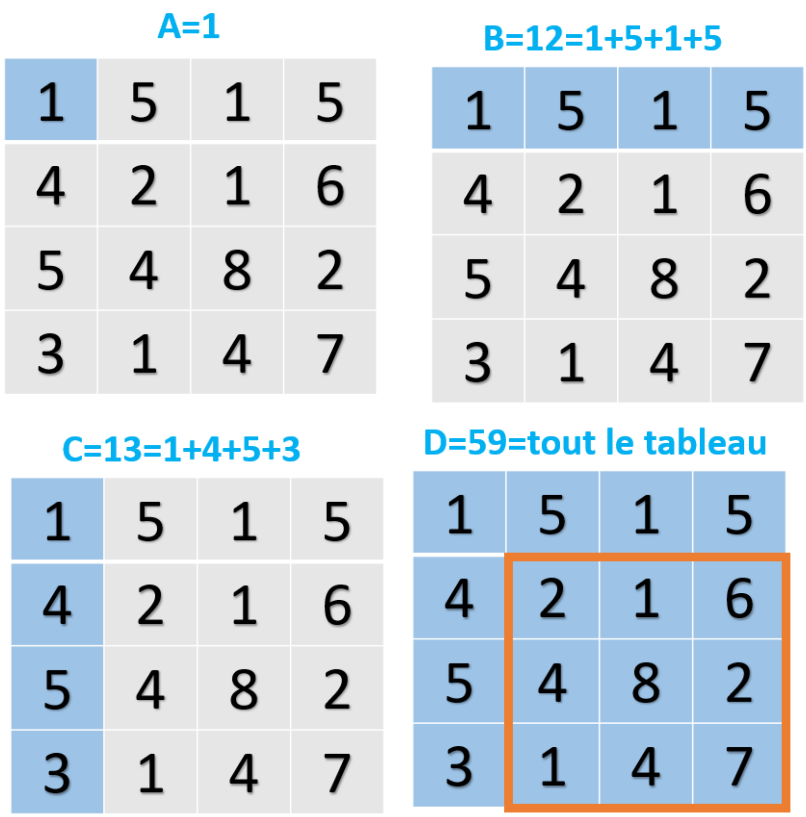


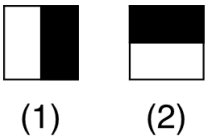
Figure 18: transformation de l'image initial en image intégral



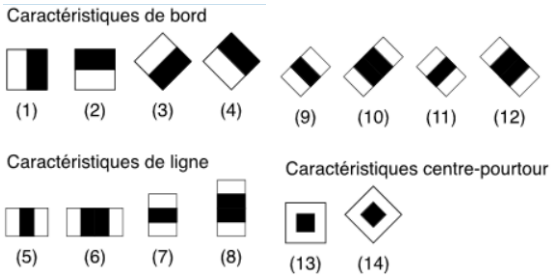
7.3 Caractéristique pseudo-Haar

Définition: Une caractéristique est une représentation synthétique et informative, calculée à partir des valeurs des pixels. Les caractéristiques utilisées dans cette algorithme sont dîtes pseudo-haar. Elle sont calculées par différence des sommes de pixels de deux ou plusieurs rectangles adjacents.

Prenons l'exemple suivant:



Pour calculer cette caractéristique on soustrait la somme des intensités des pixels dans le rectangle noir à celle dans le rectangle blanc. En utilisant l'image intégrale le calcul est très simple On fait 6 accès au tableau pour les sommets des rectangles puis quatre calculs. C'est encore un calcul en temps constant $O(1)$.
Ces deux caractéristiques sont des caractéristiques de bord car si la valeur de la caractéristique est proche de zéro cela signifie que la différence d'intensité est faible donc il est très probable que les deux rectangles soit sur le même objet. Au contraire si la valeur de la caractéristique est élevé alors alors on se trouve probablement sur un bord entre deux objets d'où son nom de caractéristique de bord.
Il existe différents types de caractéristiques pseudo-haar en voici quelques une.



On peut ainsi mieux comprendre comment est ce que l'ordinateur voit les formes. Prenons l'exemple de la voiture à gauche on peut grossièrement trouver les contours en utilisant des caractéristiques (image de droite).

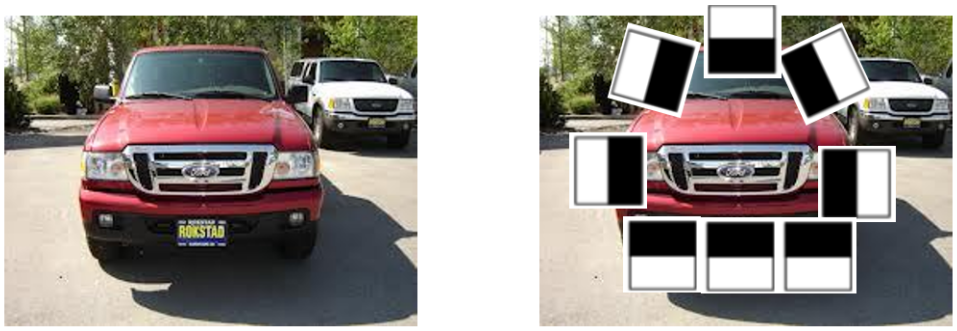


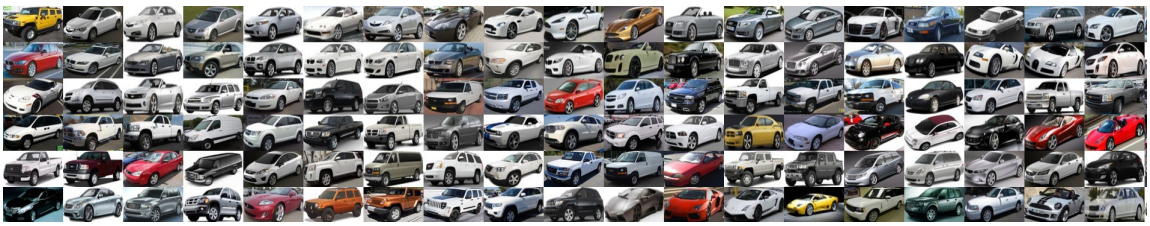
Figure 19: définition des contour de la voiture grâce aux caractéristiques

Je vous conseil de regarder la même vidéo que précédemment citée si-dessus pour bien voir le balayage de caractéristiques.(https://youtu.be/hPCTwxF0qf4)

7.4 Construction d'un classifieur

Définition: un classifieur est un dossier, synthèse issue de la comparaison de plusieurs milliers d'image par machine learning.

Pour réaliser un classifieur il faut avoir un **très** grand nombre d'image. Heureusement l'université de Stanford met en libre accès des base des données conséquentes où l'on peut trouver des dizaines de miliers d'images de voitures. (Dans notre projet le classifeur se base sur près de 15 000 images).



Il faut avoir deux types d'image les positives qui contiennent un voiture et les négatives qui ne contiennent pas de voiture ni aucune partie de voiture.

7.5 Sélection par boosting

La méthode de sélection par boosting

est une méthode qui divise de classfieur en plusieurs petits classifieurs pour gagner en temps de calcul. En effet, si on n'utilisait qu'un très gros classifeur dit "fort" il faudrait attendre que le programme est analyser l'entièreté de la fenêtre de détection pour savoir si il y a une voiture. Ce qui donnerait :

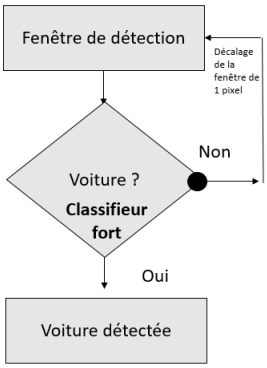
Afin d'éviter cela nous allons mettre en série de classifieurs plus petit en série appelé "haarcascade" afin de diviser la tâche en plusieurs étage chaque étage pouvant soit détecter qu'il y a une voiture et dans ce cas transmettre l'image à son voisin soit dire il n'y a pas d'image et dans ce cas rejeter l'image puis décaler la fenêtre de détection de 1 pixel. Ce qui donne alors :

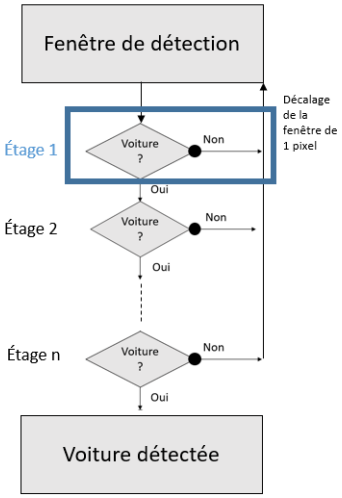
Il y a donc trois sorties possible pour un étage:

1. VRAI POSITIF Il dit qu'il y a une voiture sur l'image et c'est vrai. Ce cas est idéal, ça fonctionne bien.

2. FAUX POSITIF Il dit qu'il y a une voiture sur l'image et c'est faux. Ce cas n'est pas très grave car il transmet l'image à l'étage suivant qui sera peut-être capable de ne pas faire l'erreur.

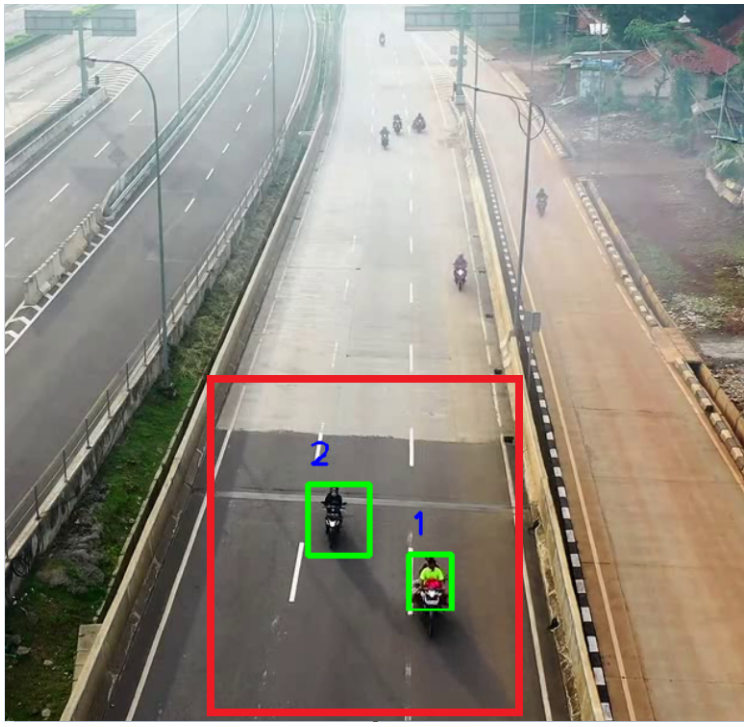
3. FAUX NEGATIF Il dit qu'il n'y a pas de voiture sur l'image et il y a une voiture. C'est le pire des cas, il faut à tout prix l'éviter.





8 résultat de notre algorithme

Notre algorithme répond tout à fait à nos objectifs énoncés précédemment. Il es parfaitement capable de différencier une voiture et une moto et est capable de compter le nombre de voiture avec moins de 2% d'erreur. Et de différencier voiture et moto avec 10% d'erreur



sur cette image on peut voir la portion de route sur laquelle fonctionne notre algorithme en rouge.

9 biliographie

livres et publications:

Apprendre le machine learning en une semaine
autheur: guillaume Saint-cirgue
utilité: découverte des algorithme de régression

https://www.merl.com/publications/docs/TR2004-043.pdf
méthode de viola et jones
utilité: méthode de construction de dataset pour machine learning à partir d'une image

https://transp-or.epfl.ch/slides/Optimisation/newton.pdf
autheur: Michel Bierlair professeur à l'Ecole Polytechnique Féderal de Lausanne utilité:

site:

https://machinelearnia.com/
utilité: site web expliquent le fonctionement des différents algorithme de machine learning

https://dashboard.covid19.data.gouv.fr/vue-d-ensemble?location=FRA utilité: Site du gouvernement français contenant les données liée à la Covid-19

https://data.cdc.gov/ utilité: Site du gouvernement américain contenant les données liée à la Covid-19

Cours:

cours de monsieur Réda DEHAK
professeur de machine learning à l'EPITA (école d'ingénieur)
utilité: étude théorique des algorithmes

https://openclassrooms.com/fr/courses/initiez-vous-au-machine-learning
cours dispenser par Yannis Chaouche ingénieur Machine learning et Chloé-Agathe Azencott enseignante à CentralSupélec.
utilité: étude théorique des algorithmes

10 annexe

10.1 démonstration du passage sous forme matricielle

$$(X^T X\theta - XY)$$

$$= \begin{pmatrix} x^{(1)} & \cdots & x^{(m)} \\ 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} x^{(1)} & 1 \\ | & | \\ x^{(m)} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} - \begin{pmatrix} x^{(1)} & \cdots & x^{(m)} \\ 1 & \cdots & 1 \end{pmatrix} \begin{pmatrix} y^{(1)} \\ y^{(m)} \end{pmatrix}$$

$$= \begin{pmatrix} x^{(1)2} + \cdots + x^{(m)2} & x^{(1)} + \cdots + x^{(m)} \\ x^{(1)} + \cdots + x^{(m)} & m \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} - \begin{pmatrix} x^{(1)}y^{(1)} + \cdots + x^{(m)}y^{(m)} \\ y^{(1)} + \cdots + y^{(m)} \end{pmatrix}$$

$$= \begin{pmatrix} ax^{(1)2} + \cdots + ax^{(m)2} + bx^{(1)} + \cdots + bx^{(m)} - x^{(1)}y^{(1)} - \cdots - x^{(m)}y^{(m)} \\ ax^{(1)} + \cdots + ax^{(m)} + bm - y^{(1)} - \cdots - y^{(m)} \end{pmatrix}$$

$$= \begin{pmatrix} \sum_{k=1}^{m} (ax^{(k)} + b - y^{(k)}) x^{(k)} \\ \sum_{k=1}^{m} ax^{(k)} + b - y^{(k)} \end{pmatrix}$$

d'où :

$$\frac{2}{m} (X^T X\theta - XY)$$

$$= \begin{pmatrix} \frac{2}{m} \sum_{k=1}^{m} (ax^{(k)} + b - y^{(k)}) x^{(k)} \\ \frac{2}{m} \sum_{k=1}^{m} ax^{(k)} + b - y^{(k)} \end{pmatrix} = \begin{pmatrix} \frac{\partial J(a,b)}{\partial a} \\ \frac{\partial J(a,b)}{\partial b} \end{pmatrix}$$

10.2 Image intégral

définition: représentation d'une image sous la forme d'un tableau de valeur, permettant de calculer rapidement des sommes de valeurs dans des zones rectangulaires. Cette représentation est de même taille que l'image initiale et chacun de ses point est la somme des pixels situé au situés au-dessus et à gauche de ce point.

Plus formellement en notant $M_{n,p}$ la matrice contenant en chaque point l'intensité du pixel correspondent sur l'image d'origine, $I_{n,p}$ la matrice que l'on construit.

Pour $1 \leqslant i \leqslant n$, $1 \leqslant j \leqslant p$
Le point $I_{i,j} = M_{i,j} + I_{i,j-1} + I_{i-1,j} - I_{i-1,j-1}$

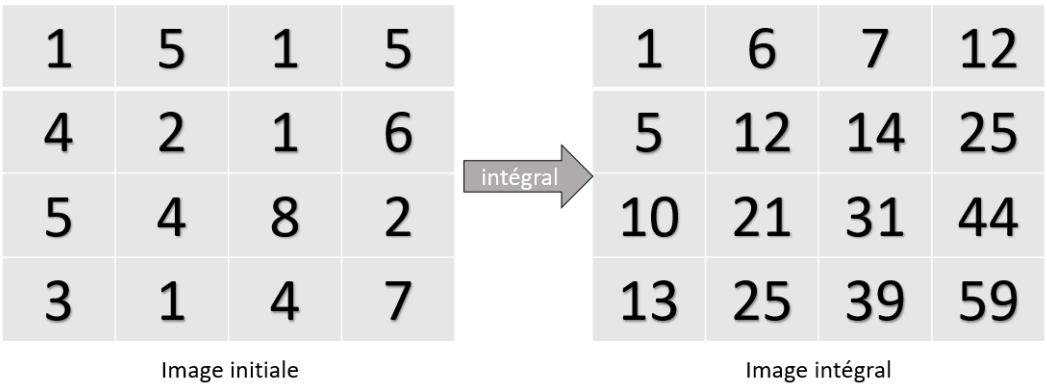


Figure 20: transformation de l'image initial en image intégral

Intéressons nous au calcul d'un point de l'image intégral

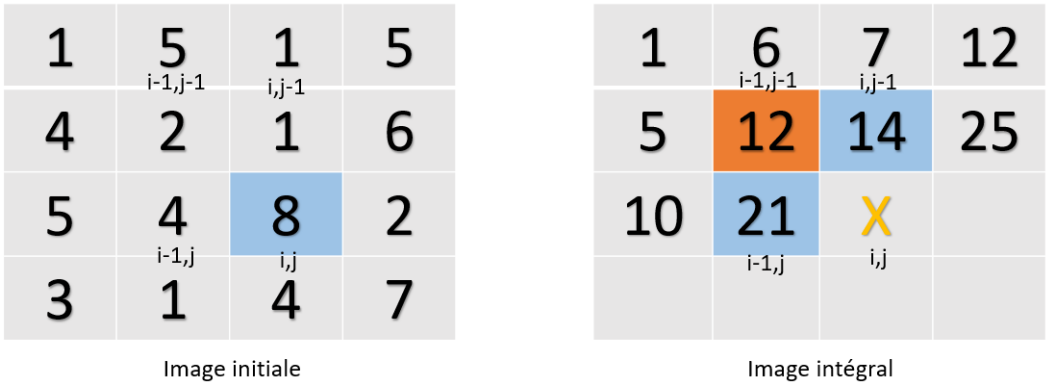


Figure 21: transformation de l'image initial en image intégral

Pour calculer la valeur de l'image intégral au point de cordonnées (i,j) on somme les cases bleu et on soustrait celle en orange ce qui est exactement la formule:

$$I_{i,j} = M_{i,j} + I_{i,j-1} + I_{i-1,j} - I_{i-1,j-1}$$
$$I_{i,j} = 8 + 14 + 21 - 12 = 31$$

On retrouve bien la valeur de la case (i,j) de l'image intégral de la *figure 2*

Intéressons nous maintenant à la récupération de la somme d'un rectangle de l'image original grâce à l'image intégral. Pour cela commençons par calculer la somme des valeurs dans le rectangle bleu (ci-dessous).

1 A	5	1	5 B
4	2	1	6
5	4	8	2
3 C	1	4	7 D

Image initiale

En faisant le calcul naïvement on doit : récupérer les valeurs des neufs cases puis les sommer. Donc $\sum = 2 + 1 + 6 + 4 + 8 + 2 + 1 + 4 + 7 = 35$
Soit 9 accès à l'image et 8 calcul de somme soit 17 opérations

Supposons que nous voulons calculer la somme d'un rectangle de taille L+l.
Il faut récupérer les $L \times l$ valeurs du tableau puis faire $L \times l - 1$ additions soit un calcul en $O(n^2)$

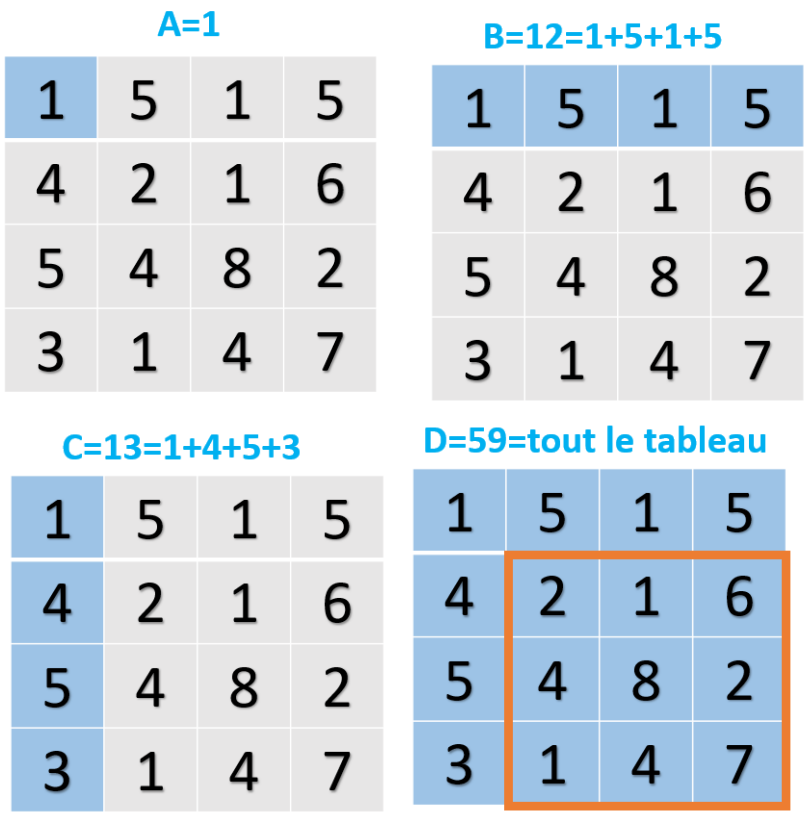
Refaisons le même calcul mais avec l'image intégral

1 A	6	7	B12
5	12	14	25
10	21	31	44
13C	25	39	D59

Image intégral

On a alors la formule de calcul $\sum = A + D - B - C = 59 - 13 - 12 + 1 = 35$
Ainsi on a récupérer la somme du rectangle en seulement quatre accès à l'image et trois calculs. Le calcul est donc en temps constant $O(1)$

Démontrons géométriquement cette formule. Pour cela il suffit de voir ce que représente chaque case.



Comme nous voulons récupérer l'aire dans le rectangle marron, il suffit de faire l'aire total (D) on lui enlève (B) et (C) et on lui rajoute la zone (A) qui est soustraite deux fois.
Ce qui nous donne : $\sum = A + D - B - C = 59 - 13 - 12 + 1 = 35$

Dans notre algorithme nous utilisons l'image intégral pour récupérer rapidement la somme des intensités des pixels d'une région rectangulaire dans le but de calculer en temps constant le multiple caractéristiques. Mais qu'est-ce qu'une caractéristique ?

10.3 implémentation de la régression linéaire

10.4 implémentation de R^2

$X^T\theta = Y$

$XX^T\theta = XY$

$\theta = (XX^T)^{-1}XY$

```
Entrée [9]: #Fonction cout

Entrée [10]: def cost_function(X,y, theta):
                 m=len(y)
                 return 1/(2*m)* np.sum((model(X,theta)-y)**2)

Entrée [11]: cost_function(X, y, theta)

Out[11]: 309.6215639729725

Entrée [12]: def grad(X,y, theta):
                 m=len(y)
                 return 1/m*X.T.dot(model(X,theta)-y)

Entrée [13]: def gradient_descent(X,y, theta, learning_rate, n_iterations):
                 cost_history=np.zeros(n_iterations)
                 for i in range (0, n_iterations):
                     theta=theta-learning_rate*grad(X,y,theta)
                     cost_history[i]=cost_function(X,y, theta)
                 return theta, cost_history

Entrée [14]: #machine learning

Entrée [15]: theta_final, cost_history= gradient_descent(X,y,theta, learning_rate=0.01, n_iterations=1000)

Entrée [16]: theta_final

Out[16]: array([[27.56492762],
                [ 0.057585  ]])

Entrée [17]: prediction=model(X, theta_final)
             plt.scatter(x, y)
             plt.plot(x, prediction, c='r')
```

Figure 22: implémentation des fonction coût, gradient, prédiction

```
#coefficient de determination

def coef_determination(y, pred):
    u=((y-pred)**2).sum()
    v=((y-y.mean())**2).sum()
    return 1-u/v

coef_determination(y, prediction)
```

```
0.9857239439730436
```

Figure 23: R^2