Un aperçu sur quelques méthodes en apprentissage automatique supervisé

Aboubacar Konaté*

Cette note est une introduction aux algorithmes en apprentissage automatique supervisé. Les méthodes et leurs justifications mathématiques y sont traitées. Nous étudierons particulièrement les méthodes telles que l'algorithme Perceptron, l'algorithme SVM ou encore les réseaux artificiels. Des exemples réalistes y sont illustrés.

Table des matières

*Laboratoire Jacques-Louis Lions, Université Pierre et Marie Curie

1 Introduction

L'apprentissage automatique fait partie de ce qui est communément appelé intelligence artificielle. Il s'agit un ensemble de méthodes qui cherche à exécuter des tâches comme l'identification d'un numéro sur une photo, la détection de cancer sur une image ou comment trouver la meilleure stratégie dans une situation. Si ces tâches sont plus ou moins naturelles pour l'humain, d'un point de vue informatique il relève d'une véritable prouesse intellectuelle.

L'intelligence artificielle est aujourd'hui présente dans presque tous les domaines (médecine, finance, sciences sociales). Il existe plusieurs publications. Nous nous sommes proposé, dans cette note, de donner un aperçu sur quelques une des ces méthodes et de les illustrer avec des exemples. Nous nous limiterons à la théorie de l'apprentissage automatique dite supervisée. Il s'agit des méthodes qui cherchent à apprendre à partir d'exemples annotés (voir Figure 1).

2 Le problème de la classification

Nous allons commencer avec le problème de classification en particulier la classification binaire. Il s'agit de trouver une fonction qui sépare les éléments de deux groupes avec des labels ou sorties différents. Nous nous sommes inspirés des notes de cours de l'université MIT [Sin06].

Formellement, le problème de la classification binaire se formalise comme suit :
soit $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$ une base de données où $x_t \in \mathbb{R}^d$ est la $t^{\text{ième}}$ donnée d'entrée, $y_t \in \{-1, 1\}$ la $t^{\text{ième}}$ donnée de sortie et n la taille de l'échantillon. Nous cherchons à construire une fonction f dite classifieur (si cela est possible) telle que

$$\text{sign} f(x_t) = y_t, \qquad \forall t = 1, 2, \cdots, n. \tag{1}$$

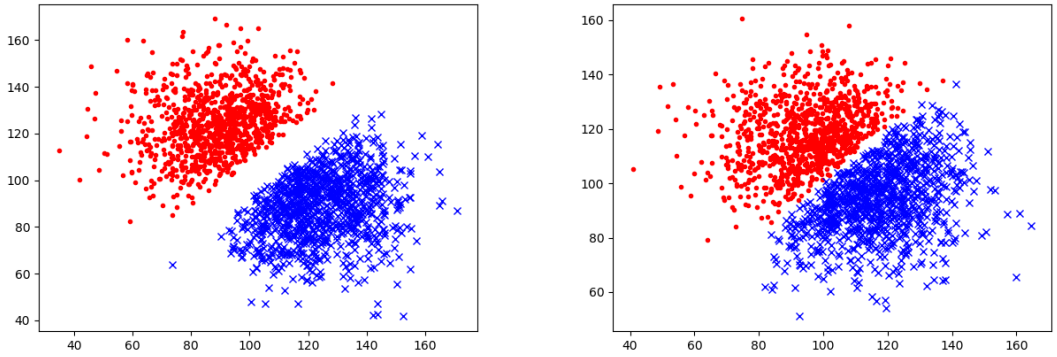
où la fonction sign est définie comme suit :

$$\text{sign}(z) = \begin{cases} +1 & \text{si} \quad z \geqslant 0 \\ -1 & \text{sinon.} \end{cases}$$

Bien sûr, il existe éventuellement une infinité de solutions pour construire le classifieur si aucune hypothèse supplémentaire est faite sur la nature de la fonction f. Nous allons considéré, dans un premier temps, uniquement des classifieurs linéaires (linéaires par rapport aux données de la base d'apprentissage). Formellement, nous cherchons des fonctions de la forme suivante :

$$f(x; \theta) = \theta^T x = \theta_1 x_1 + \cdots + \theta_d x_d \tag{2}$$

où $\theta = [\theta_1, \cdots, \theta_d]^T$ est une vecteur de paramètres réels.



(a) Données 1 (b) Données 2

FIGURE 1 – Exemples de base d'apprentissage. Les points en couleur rouge correspondent aux données d'entrée dont la sortie est 1 et ceux en couleur bleue aux données d'entrée dont la sortie est -1.

Dans la suite, on suppose qu'une telle fonction existe. En d'autres termes, il existe une fonction f telle que

$$\text{sign} f(x_t;\theta) = \text{sign}(\theta^T x_t) = y_t, \qquad \forall t = 1, ..., n. \tag{3}$$

2.1 La méthode Perceptron

Pour calculer les valeurs des paramètres θ, nous utiliserons la méthode dite Perceptron proposée en 1958 par Frank Rosenblatt dans [Ros58]. Elle consiste à initialiser la valeur des paramètres à 0 et ensuite boucler sur toutes les données d'entrée x_t et à ajuster les valeurs comme suit :

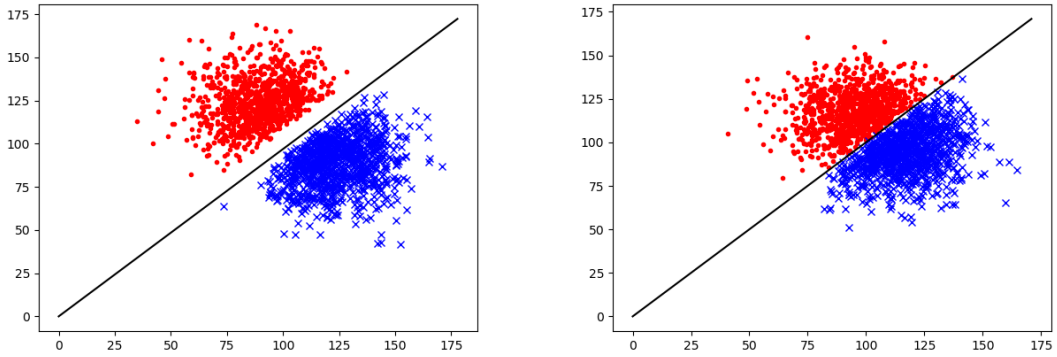
$$\theta^{'} \rightarrow \theta + y_t x_t \qquad \text{si} \quad y_t \neq f(x_t, \theta); \tag{4}$$

Par exemple, si l'élément x_t n'est pas correctement classifié alors le produit $y_t \theta^T x_t$ est négatif ; Ainsi, en ajustant la valeur des paramètres selon la méthode décrite ci-dessus, on a

$$y_t \theta^T x_t = y_t(\theta + y_t x_t)x_t = y_t \theta x_t + y_t^2 \|x_t\| = y_t \theta x_t + \|x_t\|. \tag{5}$$

En autres termes, on fait accroître la valeur du produit $y_t \theta^T x_t$. En répétant ainsi, l'élément x_t devient correctement classifié.

Nous avons appliqué la méthode Perceptron aux bases d'apprentissage présentées en Figure 1. Les résultats sont représentés en Figure 2. Nous avons obtenu 16 boucles sur toutes les données la première base et 154 dans la deuxième. Ce qui s'expliquent, comme nous allons le voir dans l'analyse de cette méthode, par le fait que la complexité est inversement proportionnelle à la marge (voir 3) c'est à dire plus la distance entre les deux groupe est importante plus la méthode est efficace. Nous avons implémenté cette méthode en Python (voir 6.1).



(a) Données 1 (b) Données 2

FIGURE 2 – Résultats obtenus avec la méthode Perceptron. Les points en couleur rouge correspondent aux données d'entrée dont la sortie est 1 et ceux en couleur bleue aux données d'entrée dont la sortir est -1. La droite noire est la fonction obtenue avec la méthode Perceptron.

2.2 L'analyse de l'algorithme Perceptron

Nous nous proposons, ici, de faire l'étude de convergence de la méthode Perceptron. Plus précisément, nous souhaitons montrer que les mises à jour des paramètres θ s'arrêtent au bout de quelques itérations.

Nous noterons par k le nombre de mise à jour et $\theta^{(k)}$ les paramètres correspondant. Initialement, $k = 0$ et $\theta^{(k)} = 0$. L'algorithme boucle sur les éléments de la base de donnée et corrige la valeur des paramètres θ uniquement lorsqu'un élément x_t est mal classifié. Précisément, nous avons

$$\theta^{(k+1)} = \theta^{(k)} + y_t x_t$$

si $y_t(\theta^{(k)})^T x_t < 0$.

2.2.1 Convergence

Nous supposons que les éléments de la base sont bornés au sens de la norme euclidienne c'est à dire que $\|x_t\| \leqslant R < \infty$ pour tout t et avec R une constante réelle. Ce qui est le cas en pratique. Nous supposons également qu'il existe au moins une solution au problème du classifieur notée $f(x; \theta^*)$. Plus précisément, nous supposons qu'il existe une constante réelle $\gamma > 0$ telle que $y_t(\theta^*)^T x_t \geqslant \gamma$ pour $t = 1, \cdots, n$. Le réel $\gamma > 0$ est utilisé pour s'assurer que les éléments de la base sont bien classifiés avec une distance (marge) finie. La preuve de la convergence est basée sur les deux résultats suivants :

1. le produit $(\theta^*)^T \theta^{(k)}$ croît au moins linéairement après chaque mise à jour
2. la norme $\|\theta^{(k)}\|^2$ croit au moins linéairement en fonction du nombre de mise à jour k. En combinant ces deux résultats, nous montrons que le cosine de l'angle entre $\theta^{(k)}$ et θ^* croît avec les itérations. Puisque le cosine est borné par 1, on en déduit que le nombre de mise à jour est fini.

Étape 1 : nous calculons le produit scalaire $(\theta^*)^T\theta^{(k)}$ avant et après chaque mise à jour. Après k mise à jour, par exemple lorsque l'élément x_t est mal classifié, nous obtenons

$$(\theta^*)^T\theta^{(k)} = (\theta^*)^T(\theta^{k-1} + y_t x_t) \geqslant (\theta^*)^T\theta^{(k-1)} + \gamma \tag{6}$$

où, par hypothèse, $y_t(\theta^*)^T x_t \geqslant \gamma$ pour tout t (θ^* est toujours bien classifié). Alors, nous en déduisons qu'après k mise à jour

$$(\theta^*)^T\theta^{(k)} \geqslant k\gamma. \tag{7}$$

Étape 2 : Pour la deuxième étape de la démonstration, nous avons :

$$\begin{aligned}
\|\theta^{(k)}\|^2 &= \|\theta^{(k-1)} + y_t x_t\|^2 \\
&= \|\theta^{(k-1)}\|^2 + 2y_t(\theta^{(k-1)})^T x_t + \|x_t\|^2 \\
&\leqslant \|\theta^{(k-1)}\|^2 + \|x_t\|^2 \\
&\leqslant \|\theta^{(k-1)}\|^2 + R^2.
\end{aligned}$$

puisque $y_t(\theta^{(k-1)})^T x_t < 0$ dès lors qu'une mise à jour est effectuée et que $\|x_t\| \leqslant R$. Nous en déduisons que

$$\|\theta^{(k)}\|^2 \leqslant kR^2.$$

En combinant **Étape 1)** et **Étape 2)**, nous obtenons :

$$cos(\theta^*, \theta^{(k)}) = \frac{(\theta^*)^T\theta^{(k)}}{\|\theta^*\|\|\theta^{(k)}\|} \geqslant \frac{k\gamma}{\|\theta^*\|\|\theta^{(k)}\|} \geqslant \frac{k\gamma}{\|\theta^*\|\sqrt{kR^2}}. \tag{8}$$

En utilisant le fait que le cosinus est borné par 1, nous obtenons :

$$1 \geqslant \frac{k\gamma}{\|\theta^*\|\sqrt{kR^2}} \qquad \text{autrement dit} \qquad k \leqslant \frac{\|\theta^*\|^2 R^2}{\gamma^2}. \tag{9}$$

Ce qui termine la démonstration.

2.2.2 Marge et géométrie

Nous avons vu que le terme $\dfrac{\|\theta^*\|}{\gamma}$ détermine la complexité de l'algorithme Perceptron. Nous allons montrer qu'il s'agit de l'inverse de la distance entre la base de donnée et le classifieur et quantifie le degré de séparation entre les classes de la base (voir Figure 3). Dans la suite, nous l'appeleront la marge géométrique et elle sera notée γ_{geom}.

Nous pouvons montrer facilement que la droite qui passe par un élément x_t de la base et perpendiculaire à la droite $(\theta^*)^T x = 0$ est caractérisée par

$$x(\xi) = x_t - \xi\frac{y_t\theta^*}{\|\theta^*\|}, \qquad \xi \in \mathbb{R}.$$

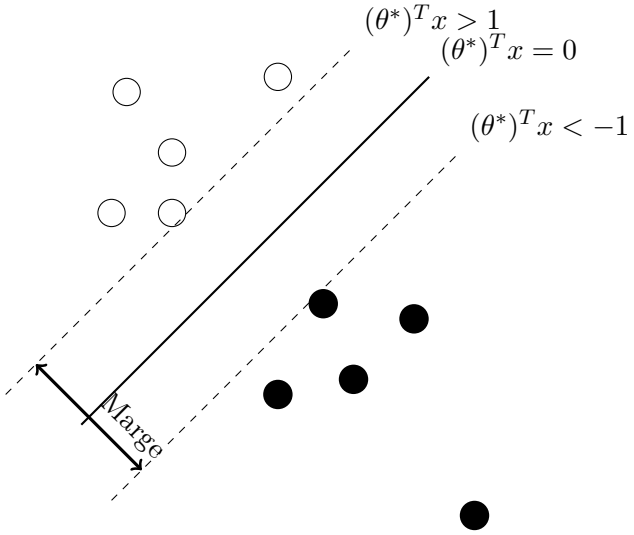


FIGURE 3 – Illustration géométrique de la méthode SVM.

Évaluer γ_{geom} revient à calculer la distance entre la droite $(\theta^*)^T x = 0$ et un élément x_t de la base pour lequel $y_t(\theta^*)^T x_t = \gamma$. Pour cela, il nous faut déterminer la valeur de ξ telle que $(\theta^*)^T x(\xi) = 0$, ou de manière équivalente, $y_t(\theta^*)^T x(\xi) = 0$. Ainsi, nous obtenons :

$$\begin{aligned} y_t(\theta^*)^T x(\xi) &= y_t(\theta^*)^T x(0) - y_t(\theta^*)^T \xi \frac{y_t \theta^*}{\|\theta^*\|} \\ &= y_t(\theta^*)^T x_t - y_t(\theta^*)^T \xi \frac{y_t \theta^*}{\|\theta^*\|} \\ &= y_t(\theta^*)^T x_t - \xi \frac{\|\theta^*\|^2}{\|\theta^*\|} \\ &= y_t(\theta^*)^T x_t - \xi \|\theta^*\| \\ &= \gamma - \xi \|\theta^*\| = 0 \end{aligned}$$

ce qui implique $\xi = \dfrac{\gamma}{\|\theta^*\|}$. Ce qui termine la démonstration.

3 The Support Vector Machine (ou en français Séparateurs à Vaste Marge)

Le Support Vector Machine ou simplement SVM est la méthode qui consiste à trouver le classifieur linéaire $f(x; \theta^*)$ (s'il existe) avec la marge maximale.

3.1 La méthode

Cette méthode est formulée sous forme de problème d'optimisation. Elle est basée sous l'hypothèse que tous les éléments de la base de donnée sont bien classifiés, ie $y_t \theta^T x_t \geqslant \gamma > 0$ pour

tout $t = 1, \cdots, n$. Tenant compte de cette dernière l'hypothèse, nous voudrions maximiser $\frac{\gamma}{\|\theta|}$. Alternativement, on peut minimiser l'inverse $\frac{\|\theta\|}{\gamma}$ ou bien alors minimiser l'inverse de la norme au carré $\frac{1}{2}\left(\frac{\|\theta\|}{\gamma}\right)^2$. On aboutit au problème d'optimisation suivant pour trouver θ^* :

$$\begin{aligned} \min_{\theta} \quad & \frac{1}{2}\left(\frac{\|\theta\|}{\gamma}\right)^2 \\ & y_t(\theta)^T x_t \geqslant \gamma \quad \text{pour tout} \quad t = 1, \cdots, n \end{aligned}$$

Ce dernier problème peut être réécrit comme suit :

$$\begin{aligned} \min_{\theta} \quad & \frac{1}{2}\|\theta/\gamma\|^2 \\ & y_t \left(\frac{\theta}{\gamma}\right)^T x_t \geqslant 1 \quad \text{pour tout} \quad t = 1, \cdots, n \end{aligned}$$

Nous pouvons voir que le problème ne dépend que du rapport $\frac{\theta}{\gamma}$. Puisque γ est juste un scalaire, nous décidons de l'éliminer en lui assignant la valeur. Nous obtenons enfin :

$$\begin{aligned} \min_{\theta} \quad & \frac{1}{2}\|\theta\|^2 \\ & y_t(\theta^T x_t) \geqslant 1 \quad \text{pour tout} \quad t = 1, \cdots, n \end{aligned}$$

Il s'agit d'un problème d'optimisation convexe quadratique. Ce type de problème a été abondamment etudié dans la littérature. Pour plus de détailes sur les aspects théoriques et algorithmiques, nous pourrons consulter le livre [BV04]. Nous avons implémenté la méthode SVM en python en utilisant la librairie cvxopt (voir 6.2).

3.2 Formulation générale

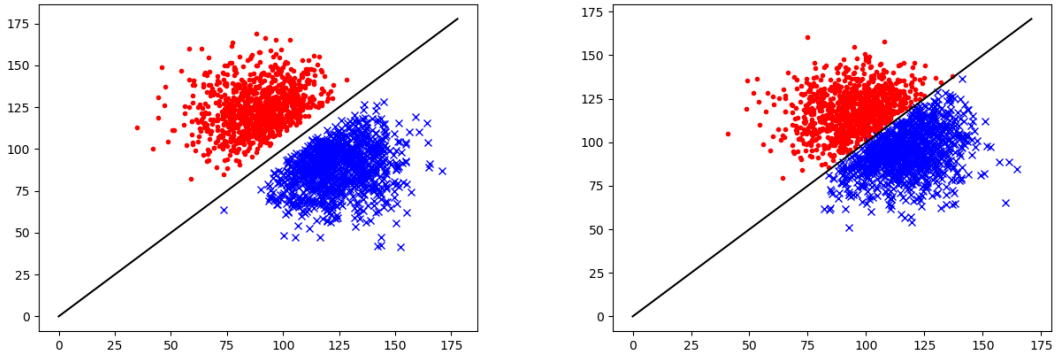
Nous allons, dans cette section, donner la formulation générale du problème de classification. Jusqu'à là, nous avons considéré que des droite passant à l'origine. Or dans la pratique, toutes les droites ne passent pas nécessairement à l'origine. La formulation générale du problème du classifieur linéaire est comme suit :

$$f(x; \theta, \theta_0) = \theta^T x + \theta_0 \tag{10}$$

avec θ les paramètres et le paramètre à l'origine θ_0.

De manière analogue, nous pouvons déduire la formulation générale de la méthode SVM :

$$\begin{aligned} \min_{\theta} \quad & \frac{1}{2}\|\theta\|^2 \\ & y_t(\theta^T x_t + \theta_0) \geqslant 1 \quad \text{pour tout} \quad t = 1, ..., n. \end{aligned}$$



(a) Données 1 (b) Données 2

FIGURE 4 – Résultats obtenus avec la méthode SVM. Les points en couleur rouge correspondent aux données d'entrée dont le label est 1 et ceux en couleur bleue aux données d'entrée dont le label est -1. La droite noire est la fonction obtenue avec la méthode SVM.

3.3 Exemple 1 : Détection de la présence d'un piéton sur une photo

Nous allons appliquer la méthode SVM sur un exemple de base beaucoup plus réaliste introduit dans [Bey17]. L'objectif consiste à détecter la présence d'un piéton sur une photo. L'idée consiste à former une base d'images telle que

1. les images comportant une personne sont labellisées par 1
2. et cells ne comportant pas une personne seront, elles, labellisées par -1.

La base d'images labellisées par 1 (http://cbcl.mit.edu/software-datasets/PedestrianData.html) est composée de 399 images et chacune de taille 64×128. Quelques une sont représentées en Figure 5.

La principale difficulté avec cet exemple c'est le choix de la base de données avec les éléments labialisés avec -1. Nous avons considéré la base introduite dans (http://cvcl.mit.edu/database.htm). Elle est composée de 250 images. Quelques une sont représentées en Figure 6. Puisque les images sont de taille grande que 64×128, pour chacune d'elles nous allons en considéré une région de taille 64×128.

Pour appliquer la méthode SVM, un traitement préalable est nécessaire. En effet, chaque image sera représentée par un grand vecteur de taille $128 \times 64 \times 3$ où les composantes sont la valeur RBG de chaque pixel.

Pour tester la méthode sur une image x_{test} de taille supposée plus grande que la taille des images de la base de donnée 64×128, nous allons balayer toutes les régions de x_{test} de taille 64×128 pour détecter la présence de personnne. Nous avons implémenté la méthode avec la librairie scikit-learn [PVG+11] et avons testé la méthode. Quelques résultats sont représentés en Figure 7.

FIGURE 5 – Exemple d'images labélisées par 1

FIGURE 6 – Exemple d'images labellisées par -1

(a) test 1 (b) test 2 (c) test 2

FIGURE 7 – Quelques résultats

Dans cet exemple, nous avons pu détecter des piétons sur toutes les images. La méthode est efficace mais peut être améliorée. Il existe dans la littérature des extensions amélioriées de la méthode SVM étudiée ici (pour plus de détailes sur ces méthodes, voir [CL11]). Pour beaucoup plus d'exemples, nous pourrons consulter le livre d'Aurélien Geron [Gér17].

4 Approche probabiliste

Dans la pratique, les données les plus réalistes comportent des incertitudes. Elles sont souvent modélisées par des lois probabilistes.

Ici, nous supposons que les éléments d'entrée $x_1, \cdots, x_n$ et les éléments de sortie $y_1, \cdots, y_n$ sont liées comme suit :

$$y_t = \theta^T x_t + \theta_0 + \epsilon, \qquad t = 1, \cdots, n \tag{11}$$

où $e_t \sim N(0, \sigma^2)$ (c'est à dire e_t est distribuée suivant la loi gaussienne de moyenne zero et de variance σ^2) et e_i est indépendant de e_j pour tout $i \neq j$. Nous pouvons remarquer que :

$$\epsilon \sim N(y_t - \theta^T x_t - \theta_0, \sigma^2). \tag{12}$$

Il est alors très naturel que nous cherchons une fonction d'apprentissage de la forme :

$$f(x; \theta) = \hat{\theta}^T x + \hat{\theta}_0 + \epsilon. \tag{13}$$

Nous allons, à présent, voir comment estimer les valeurs de paramètres $\hat{\theta}$, $\hat{\theta}_0$ et $\hat{\sigma}$.

4.1 Maximum de vraisemblance

Pour estimer les valeurs des paramètres $\hat{\theta}$, $\hat{\theta}_0$ et $\hat{\sigma}$, il est commode d'utiliser la méthode du maximum de vraisemblance (pour plus de details sur cette méthode, voir [RVH12]).

Nous définissons la fonction de vraisemblance (qui n'est rien d'autres que le produit des fonctions densités) comme suit :

$$L(\theta, \theta_0, \sigma^2) = \prod_{t=1}^{n} \frac{1}{\sqrt{2\,\pi\sigma^2}} \exp\left(-\frac{(y_t - \theta^T x_t - \theta_0)^2}{2\sigma^2}\right) \tag{14}$$

Pour faciliter le calcul, il est souvent commode de travailler avec le logarithme de vraisemblance :

$$\begin{aligned} l(\theta, \theta_0, \sigma^2) &= \sum_{i=1}^{n} \log\left[\frac{1}{\sqrt{2\,\pi\sigma^2}} \exp\left(-\frac{(y_t - \theta^T x_t - \theta_0)^2}{2\sigma^2}\right)\right] \\ &= \sum_{i=1}^{n} \left[-\frac{\log(2\pi)}{2} - \frac{\log(\sigma^2)}{2} - \frac{\left(y_t - \theta^T x_t - \theta_0\right)^2}{2\sigma^2}\right] \\ &= -\frac{n\,\log(2\pi)}{2} - \frac{n\,\log(\sigma^2)}{2} - \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(y_t - \theta^T x_t - \theta_0\right)^2 \end{aligned}$$

La valeur des paramètres $\hat{\theta}$ et $\hat{\theta}_0$ est calculée en minimisant simplement l'erreur suivante :

$$\sum_{i=1}^{n} \left(y_t - \theta^T x_t - \theta_0\right)^2$$

Ce dernier problème de minimisation peut se réécrire sous forme matricielle. Pour cela, nous définissons X comme la matrice ayant sur chaque ligne t le vecteur $[x_t^T, 1]$. Le problème devient alors :

$$\begin{aligned}
\sum_{i=1}^{n} \left(y_t - \begin{bmatrix} \theta \\ \theta_0 \end{bmatrix}^T \begin{bmatrix} x_t \\ 1 \end{bmatrix}\right)^2 &= \sum_{i=1}^{n} \left(y_t - \begin{bmatrix} x_t^T & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \theta_0 \end{bmatrix}\right)^2 \\
&= \left\| \begin{bmatrix} y_1 \\ \cdots \\ y_n \end{bmatrix} - \begin{bmatrix} x_1^T & 1 \\ \cdots & \cdots \\ x_n^T & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \theta_0 \end{bmatrix} \right\|^2 \\
&= \left\| \mathbf{y} - \mathbf{X} \begin{bmatrix} \theta \\ \theta_0 \end{bmatrix} \right\|^2 \\
&= \mathbf{y}^T\mathbf{y} - 2 \begin{bmatrix} \theta \\ \theta_0 \end{bmatrix} \mathbf{X}^T y + \begin{bmatrix} \theta \\ \theta_0 \end{bmatrix}^T \mathbf{X}^T\mathbf{X} \begin{bmatrix} \theta \\ \theta_0 \end{bmatrix}
\end{aligned}$$

où $y = [y_1, \cdots, y_n]^T$. Nous pouvons montrer facilement :

$$\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{15}$$

Nous en déduisons la valeur optimale $\hat{\sigma}$ comme suit :

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} \left(y_t - \hat{\theta}^T x_t - \hat{\theta}_0\right)^2. \tag{16}$$

4.2 Biais et variance des paramètres estimés

Nous nous sommes proposé, ici, d'évaluer l'efficacité de la valeur des paramètres estimés en (15) et en (16).

Nous définissons θ^*, θ_0^*, et ${\sigma^*}^2$ comme les vraies valeurs des paramètres c'est à dire que :

$$y_t = {\theta^*}^T x_t + \theta_0^* + \epsilon_t, \qquad t = 1, \cdots, n. \tag{17}$$

En forme matricielle :

$$\mathbf{y} = \mathbf{X} \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} + \mathbf{e} \tag{18}$$

où $\mathbf{e} = [\epsilon_1, \cdots, \epsilon_n]^T$, $\mathbb{E}\{e\} = 0$ et $\{E\}\{ee^T\} = {\sigma^*}^2\mathbf{I}$.

En remplaçant la valeur de $\mathbf{y}$ dans (15), nous obtenons

$$\begin{aligned}\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\left(\mathbf{X}\begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} + \mathbf{e}\right) \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{e} \\ &= \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{e}.\end{aligned}$$

En prenant la moyenne, nous obtenons

$$\mathbb{E}\left\{\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix}|\mathbf{X}\right\} = \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} + (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}\left\{\mathbf{e}|\mathbf{X}\right\} = \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix}$$

c'est à dire l'estimation est correcte en moyenne et nous disons qu'elle est non biaisée.

En utilisant les inégalités établies précédemment, nous pouvons calculer la matrice de covariance des paramètres comme suit :

$$\begin{aligned}Cov\left\{\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix}|\mathbf{X}\right\} &= Cov\left\{\left(\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} - \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix}\right)\left(\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} - \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix}\right)^T|\mathbf{X}\right\} \\ &= Cov\left\{\left((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{e}\right)\left((\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{e}\right)^T|\mathbf{X}\right\} \\ &= Cov\left\{(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{e}\mathbf{e}^T\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}|\mathbf{X}\right\} \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbb{E}(\mathbf{e}\mathbf{e}^T|\mathbf{X})\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} \\ &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\sigma^{*2}\mathbf{I}\mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1} \\ &= \sigma^{*2}(\mathbf{X}^T\mathbf{X})^{-1}.\end{aligned}$$

Ce qui signifie que la covariance ne dépend que de la matrice $\mathbf{X}$.

Nous allons à présent évaluer la moyenne de l'erreur de l'estimation au carrée. Pour cela, nous utiliserons le résultat suivant :

$$\begin{aligned}\mathbb{E}\left\{\|\mathbf{z}-\mathbf{z}^*\|^2\right\} &= \mathbb{E}\left\{\|\mathbf{z}-\mathbb{E}(\mathbf{z})+\mathbb{E}(\mathbf{z})-\mathbf{z}^*\|^2\right\} \\ &= \mathbb{E}\left\{\|\mathbf{z}-\mathbb{E}(\mathbf{z})\|^2 + 2(\mathbf{z}-\mathbb{E}(\mathbf{z}))^T(\mathbb{E}(\mathbf{z})-\mathbf{z}^*) + \|\mathbb{E}(\mathbf{z})-\mathbf{z}^*\|^2\right\} \\ &= \mathbb{E}\left\{\|\mathbf{z}-\mathbb{E}(\mathbf{z})\|^2\right\} + 2\mathbb{E}\left\{(\mathbf{z}-\mathbb{E}(\mathbf{z}))^T\right\}(\mathbb{E}(\mathbf{z})-\mathbf{z}^*) + \|\mathbb{E}(\mathbf{z})-\mathbf{z}^*\|^2 \\ &= \underline{\mathbb{E}\left\{\|\mathbf{z}-\mathbb{E}(\mathbf{z})\|^2\right\}}_{variance} + \underline{\|\mathbb{E}(\mathbf{z})-\mathbf{z}^*\|^2}_{biais}\end{aligned}$$

où $\mathbf{z}^*$ est une constante.

Par ailleurs, nous avons :

$$\begin{aligned}
\mathbb{E}\left\{\|\mathbf{z}-\mathbb{E}(\mathbf{z})\|^2\right\} &= \mathbb{E}\left\{(\mathbf{z}-\mathbb{E}(\mathbf{z}))^T(\mathbf{z}-\mathbb{E}(\mathbf{z}))\right\} \\
&= \mathbb{E}\left\{Tr\left[(\mathbf{z}-\mathbb{E}(\mathbf{z}))^T(\mathbf{z}-\mathbb{E}(\mathbf{z}))\right]\right\} \\
&= \mathbb{E}\left\{Tr\left[(\mathbf{z}-\mathbb{E}(\mathbf{z}))(\mathbf{z}-\mathbb{E}(\mathbf{z}))^T\right]\right\} \\
&= Tr\left[\mathbb{E}\left\{(\mathbf{z}-\mathbb{E}(\mathbf{z}))(\mathbf{z}-\mathbb{E}(\mathbf{z}))^T\right\}\right] \\
&= Tr\left[Cov(\mathbf{Z})\right]
\end{aligned}$$

où $Tr[\cdot]$ est la fonction trace. Nous avons utilisé le fait que $Tr[a^T b] = Tr[ab^T]$ pour tous vecteurs a et b.

A présent, en adaptant ce résultat dans notre contexte, nous obtenons

$$\begin{aligned}
Cov\left\{\left\|\begin{bmatrix}\hat{\theta}\\ \hat{\theta}_0\end{bmatrix}-\begin{bmatrix}\theta^*\\ \theta_0^*\end{bmatrix}\right\|^2|\mathbf{X}\right\} &= \underbrace{Tr\left[Cov\left\{\begin{bmatrix}\hat{\theta}\\ \hat{\theta}_0\end{bmatrix}|\mathbf{X}\right\}\right]}_{variance} + \underbrace{\left\|\mathbb{E}\left\{\begin{bmatrix}\hat{\theta}\\ \hat{\theta}_0\end{bmatrix}|\mathbf{X}\right\}-\begin{bmatrix}\theta^*\\ \theta_0^*\end{bmatrix}\right\|^2}_{bias} \\
&= \sigma^{*2}Tr\left[(\mathbf{X}^T\mathbf{X})^{-1}\right].
\end{aligned}$$

Ce qui signifie que pour avoir une meilleure fonction de prédiction, il faut diminuer ce dernier terme.

4.3 Régularisation

Quand le nombre d'éléments de la base de données est insuffisant c'est à dire pas assez fourni pour bien estimer la valeur des paramètres, il est alors bénéfique de régulariser la fonction d'apprentissage. Nous considérons ici une forme de régularisation en assignant une fonction de distribution sur les paramètres $P(\theta, \theta_0, \sigma'^2)$. L'objectif de cette distribution de probabilité est de privilégier les petites valeurs. Particulièrement, nous choisirons une distribution gaussienne de moyenne nulle

$$P(\theta,\theta_0,\sigma'^2) = N\left(\begin{bmatrix}\theta\\ \theta_0\end{bmatrix};\begin{bmatrix}0\\ 0\end{bmatrix},\sigma'^2\mathbf{I}\right) = N(\theta;0,\sigma'^2)\prod_{i=1}^{n}N(\theta_j;0,\sigma'^2)$$

où la variance σ'^2 de la distribution mesure combien nous voudrions que ces paramètres soient petits.

Le logarithme de vraisemblance devient :

$$\begin{aligned}
l'(\theta,\theta_0,\sigma^2) &= \sum_{t=1}^{n}\log\left[\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_t-\theta^T x-\theta_0)^2}{2\sigma^2}\right)\right] + \log P(\theta,\theta_0,\sigma'^2) \\
&= \text{const.} - \frac{n}{2}\log\sigma^2 - \sum_{t=1}^{n}\left(\frac{(y_t-\theta^T x-\theta_0)^2}{2\sigma^2}\right) - \frac{1}{2\sigma'^2}\left(\theta_0^2+\sum_{j=1}^{d}\theta_j\right) - \frac{d+1}{2}\log\sigma'^2
\end{aligned}$$

Nous avons choisi $\sigma'^2 = \sigma^2/\lambda$. Ce qui a pour but de pénaliser la valeur de σ lorsque celle-ci est très large. Ce qui donne :

$$l'(\theta, \theta_0, \sigma^2) = \text{const.} - \frac{n}{2}\log\sigma^2 - \sum_{t=1}^{n}\left(\frac{(y_t - \theta^T x - \theta_0)^2}{2\sigma^2}\right) - \frac{\lambda}{2\sigma^2}\left(\theta_0^2 + \sum_{j=1}^{d}\theta_j\right) - \frac{d+1}{2}\log\frac{\sigma^2}{\lambda}$$

$$= \text{const.} - \frac{n+d+1}{2}\log\sigma^2 + \frac{d+1}{2}\log\lambda - \left(\sum_{t=1}^{n}\frac{(y_t - \theta^T x - \theta_0)^2}{2\sigma^2} - \frac{\lambda}{2\sigma^2}\left(\theta_0^2 + \sum_{j=1}^{d}\theta_j\right)\right)$$

Comme précédemment, nous obtenons :

$$\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} = (\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{19}$$

Ici, l'estimation des paramètres θ et θ_0 est biaisée. En effet, nous avons :

$$\begin{aligned}
\mathbb{E}\left\{\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} |\mathbf{X}\right\} &= (\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}\begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} \\
&= (\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}(\mathbf{X}^T\mathbf{X} - \lambda\mathbf{I} + \lambda\mathbf{I})\begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} \\
&= \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} - \lambda(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix} \\
&= \left(\mathbf{I} - \lambda(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\right)\begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix}
\end{aligned}$$

avec $\left(\mathbf{I} - \lambda(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\right)$ une matrice définie positive avec toutes les valeurs propres toutes positives et plus petites que 1 et qui tendent vers 1 lorsque λ tend vers 0.
Évaluons à présent la matrice de covariance :

$$\begin{aligned}
Cov\left\{\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} |\mathbf{X}\right\} &= \sigma^{*2}(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{X}(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1} \\
&= \sigma^{*2}(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X} - \lambda\mathbf{I})(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1} \\
&= \sigma^{*2}(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1} - \lambda\sigma^{*2}(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-2}
\end{aligned}$$

Le carré de la moyenne est par :

$$\begin{aligned}
\mathbb{E}\left\{\left\|\begin{bmatrix} \hat{\theta} \\ \hat{\theta}_0 \end{bmatrix} - \begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix}\right\|^2 |\mathbf{X}\right\} &= \sigma^{*2}Tr[(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1} - \lambda(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-2}] \\
&\quad + \lambda^2\begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix}(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X})^{-2}\begin{bmatrix} \theta^* \\ \theta_0^* \end{bmatrix}
\end{aligned}$$

Nous pouvons diminuer cette quantité en faisant varier la valeur de λ pour ainsi obtenir de meilleures estimations des paramètres.

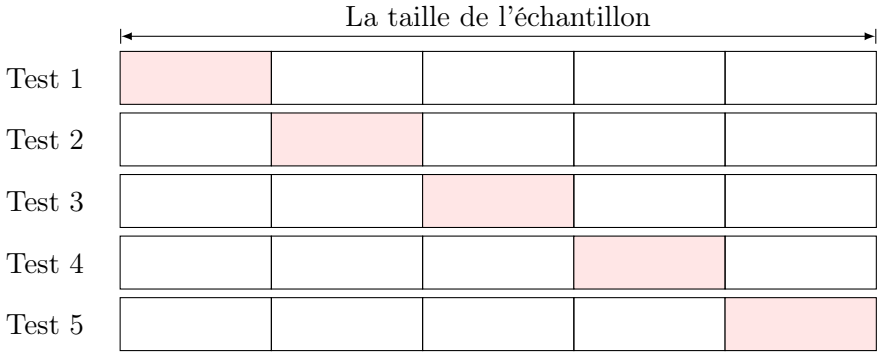


FIGURE 8 – Illustration de la méthode de validation croisée

4.4 Généralisation

Les différentes méthodes d'apprentissage que nous avons vues jusque là peuvent être étendues à des méthodes beaucoup plus complexes. Par exemple, nous pouvons chercher des fonctions de la forme suivante :

$$f(x;\theta) = \sum_{t=1} \phi(x_t) w_t + w_0 \tag{20}$$

où la fonction ϕ peut être choisie comme étant polynomiale, gaussienne etc.. Pour plus de détails sur ces méthodes, nous pouvons consulter [Anz12, FHT01].

4.4.1 Surestimation

Lorsque l'ordre utilisé est de plus en plus important, la fonction d'apprentissage devient plus efficace. Mais malheureusement, arriver à un certain point, la qualité des prédictions se détériore considérablement. Déterminer l'ordre optimal est un vrai défi en machine Learning. Pour pallier à cette difficulté, nous rappelons ci-après quelques méthodes couramment utilisées en pratique.

4.4.2 Données de validation

Une méthode couramment utilisée pour pallier à la difficulté évoquée précedemment consiste à utiliser une deuxième base de données dite base de test pour évaluer l'efficacité de la fonction de prédiction. Pour plus de détails sur cette méthode, nous pourrons consulter [RG16][p.57].

4.4.3 Validation croisée

Lorsque les bases d'apprentissage n'est pas assez fournie, alors il n'est pas efficace de la diviser en base d'apprentissage et base de test. Mais à la place, on utilise une méthode dite validation croisée. Elle consiste à diviser la base en K sous ensemble et ensuite, tour à tour, nous apprenons sur une partie et nous testons sur les autres (voir l'illustration graphique en Figure 8). L'efficacité sera mesuré en calculant sa moyenne. Pour plus de détails, nous pourrons consulter [RG16][p.57].

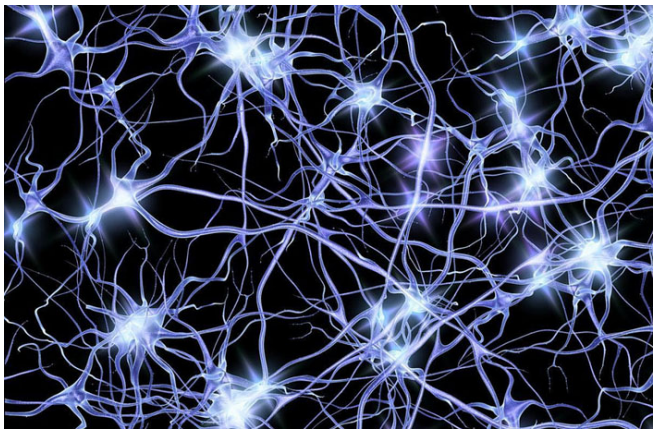


FIGURE 9 – Réseaux de neurones du cerveau humain. Source :https://www.psypost.org/2017/07/antisocial-personality-disorder-linked-impairments-white-matter-microstructure-49349

5 Les réseaux de neurones artificiels

Le cerveau humain est composé par des milliers de neurones connectés entre eux par des synapses (voir Figure 9). A chaque fois que l'homme interréagit avec son environnement via ses sens (la vue, l'ouie, etc), certains de ses neurones sont sollicités. Ce qui lui permet, par exemple, de distinguer une voiture, un cheval, un avion, bus etc...

Imitant le même fonctionnement, les réseaux de neurones artificiels sont introduits. Nous nous somme proposés, dans cette section, d'en donner une description mathématique. Nous nous inspirerons des notes de cours [Nie17].

5.1 La formulation mathématique des réseaux de neurones

Les réseaux de neurones artificiels sont constitués par trois types de couches : la première couche dite couche d'entrée, la dernière couche dite couche de sortie et éventuellement d'autres couches dites des couches cachées (voir Figure 10). Pour faire la description, nous considérons un réseau de neurones avec $L+1$ couches. Nous commençons par définir les quantités suivantes :

$$z_j^l = \sum_k w_{jk} a_k^{l-1} + b_j^l, \qquad \forall l = 1, \cdots, L-1, L+1 \tag{21}$$

$$a_j^l = \sigma\left(\sum_k w_{jk} a_k^{l-1} + b_j^l\right) = \sigma(z_j^l), \qquad \forall l = 1, \cdots, L-1, L+1 \tag{22}$$

où $a_j^0 = x_j$ (x_j est la $j^{\text{ième}}$ donnée d'entrée), w_{jk}^l la valeur du poids du $k^{\text{ième}}$ neurone dans la $(l-1)^{\text{ième}}$ couche au $j^{\text{ième}}$ neurone dans la $l^{\text{ième}}$ couche. De manière similaire, nous définissons les biais b_j^l. La fonction d'activation σ peut être choisie parmi celles-ci :

— Logistique : $\sigma(z) = \dfrac{1}{1+\exp^z}$

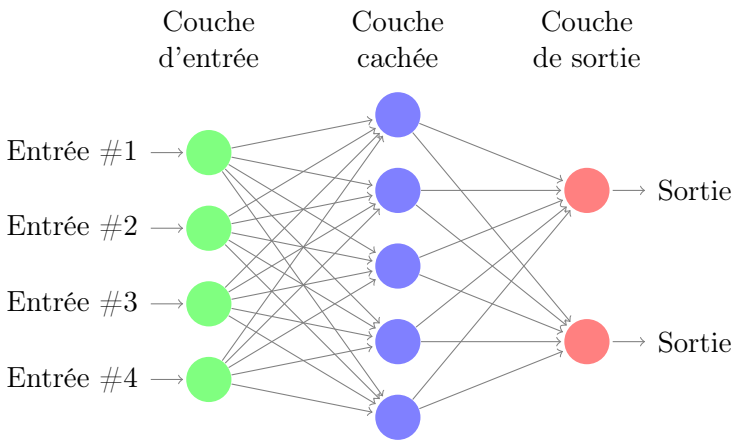


FIGURE 10 – Exemple de réseau de neurones avec 3 couches.

— Hyperbolic tangent function : $\tanh(z) = \dfrac{\exp^z - \exp^{-z}}{\exp^z + \exp^{-z}}$

— Rectified Linear neuron or rectified linear unit : $\mathrm{ReLU}(z) = \max(0, z)$

— Softmax : $\mathrm{softmax}(z_1, z_2, \cdots, z_n) = \left(\dfrac{\exp^{z_1}}{\sum_{i=1}^n \exp^{z_i}}, \cdots, \dfrac{\exp^{z_n}}{\sum_{i=1}^n \exp^{z_i}} \right)$.

En forme matricielle, nous obtenons :

$$z^l = w^l a^{l-1} + b^l, \qquad \forall l = 1, \cdots, L-1, L+1 \tag{23}$$

$$a^l = \sigma(z^l), \qquad \forall l = 1, \cdots, L-1, L+1. \tag{24}$$

Pour déterminer les poids et les biais, nous minimisons l'une des fonctions suivantes :

— Norme L^2 : $C = \dfrac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$

— Entropie : $C = -\dfrac{1}{n} \sum_x \left[y(x) \log(a^L(x)) + (1 - y(x)) \log(1 - a^L(x)) \right]$.

5.2 L'algorithme Backpropogation

Pour résoudre ce dernier problème de minimisation, nous considérons une méthode de type descente de gradient (voir [JAS18]). Ce qui nécessite le calcul du gradient. Nous présentons la méthode Backpropogation introduite par [HM94] pour un calcul rapide du gradient. Pour cela, nous définissons les éléments suivants :

— Une équation pour évaluer l'erreur dans la couche de sortie δ^L

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L). \tag{25}$$

— Une équation pour évaluer l'erreur dans la couche l

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}, \qquad \forall l = 0, \cdots, L-1. \tag{26}$$

— Une équation pour l'erreur δ^l en fonction δ^{l+1}

$$\delta^l = \left(\left(w^{l+1} \right)^T \delta^{l+1} \right) \odot \sigma^{'}(z^l). \tag{27}$$

où $\odot$ est le produit élément par élément.

— Une équation pour calculer la dérivée de la fonction à minimiser par rapport aux biais :

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{z_j^l}{\partial b_j^l} = \delta_j^l. \tag{28}$$

— Une équation pour calculer la dérivée de la fonction à minimiser par rapport aux poids :

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l. \tag{29}$$

L'algorithme qui permet de calculer le gradient est le suivant :
Data: Considérer une base d'apprentissage
Result: Calcul du gradient de la fonction d'apprentissage
Pour élément x de la base d'apprentissage : initialiser $a^{x,1}$ et réaliser les étapes suivantes :;
for $l = 1, 2, 3, \cdots, L$ **do**

$$z^{l,x} = w^l a^{x,l-1} + b^l, \qquad \text{et} \qquad a^{x,l} = \sigma(z^{x,l}) \tag{30}$$

end
Calculer le vecteur $\delta^{x,L} = \nabla_a C \odot \sigma^{'}(z^{x,L})$;
for $l = L-1, L-2, \cdots, 1$ **do**

$$\delta^{x,l} \left(\left(w^{x,l+1} \right)^T \delta^{x,l+1} \right) \odot \sigma^{'}(z^{x,l}). \tag{31}$$

end
for $l = L, \cdots, 1$ **do**
mettre à jour les poids selon la règle suivante :

$$w^l \leftarrow w^l - \frac{\eta}{n} \sum_x \delta^{x,l} \left(a^{x,l-1} \right)^T \tag{32}$$

et les biais comme suit :

$$b^l \leftarrow b^l - \frac{\eta}{n} \sum_x \delta^{x,l}. \tag{33}$$

end

Algorithm 1: Calcul du gradient

FIGURE 11 – Exemple d'images d'articles vestimentaires

5.3 Exemple 2 : classification d'articles vestimentaires

Nous considérons, ici, un exemple de problème en machine learning introduit dans https://www.tensorflow.org/tutorials/keras/basic_classification. Il consiste à classifier des articles vestimentaires selon leurs catégories. Nous allons utiliser les réseaux de neurones artificiels. La base (https://github.com/zalandoresearch/fashion-mnist) est composée de 70.000 images(voir Figure 11), de taille 28×28 pixels chacune, classées en 10 catégories [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] avec

- 0 : T-shirt/top
- 1 : Trouser
- 2 : Pullover
- 3 : Dress
- 4 : Coat
- 5 : Sandal
- 6 : Shirt
- 7 : Sneaker
- 8 : Bag
- 9 : Ankle boot

Comme mentionné plûtot, pour mieux évaluer l'efficacité de notre fonction d'apprentissage, il est commode de diviser la base d'images en deux sous-ensemble (base d'apprentissage et base de test). Ici, nous avons considéré les 60.000 premières images comme la base d'apprentissage et le reste comme base de test (10. 000 images).

Si le nombre de neurones dans la couche d'entrée et celui de la couche de sortie sont imposés par la nature de l'application (Ici, on a un réseau de neurones avec une couche d'entrée avec 28×28 neurones et une couche de sortie avec 10 neurones), le choix du nombre de couches cachées (par conséquent le nombre de neurones de chacune d'elles) constitue un vrai défi en machine learning. Ceci n'est en aucun cas une surprise. En effet, nous avons vu précédemment qu'il n'existe pas une méthode pour déterminer de manière automatique l'ordre optimal (dans

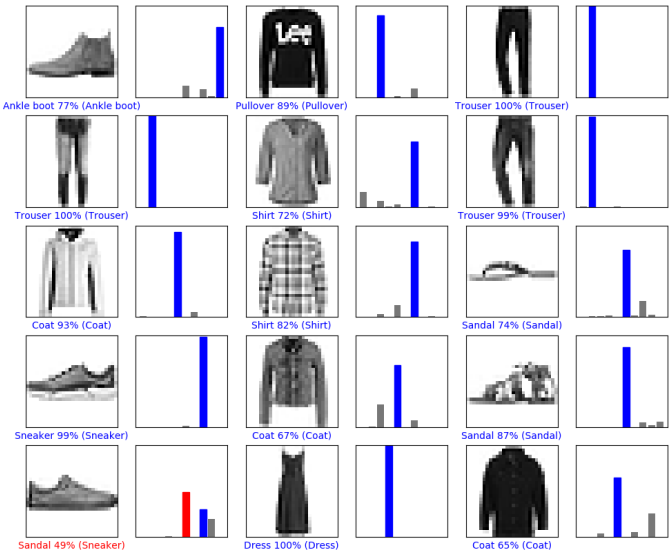


FIGURE 12 – Quelques prédiction avec un réseau de neurones avec uniquement une couche d'entrée et une couche de sortie. L'efficacité est de 0.8395.

la méthode SVM) et qu'un ordre supérieur n'est pas nécessairement un gage d'efficacité.

Nous allons considérer plusieurs configurations de réseaux de neurones artificiels. La méthode a été implémentée avec la librairie Tensorflow [AAB+15].

Quelques résultats sont représentés en Figure 12 et Figure 13. Le réseau constitué uniquement de couche d'entrée et de couche de sortie donne une efficacité de 0.8395 tandis que celui avec une couche cachée de 128 et de 300 neurones a, respectivement, une efficacité de 0.8733 et de 0.8739. Un réseau avec 2 couches cachées composées chacune de 128 neurones donne une efficacité de 0.877. Un réseau avec 3 couches cachées composées chacune de 128 neurones donne une efficacité de 0.8723.

Au vue de ces résultats, nous pouvons conclure qu'un nombre important de couches n'est pas nécessairement gage d'efficacité. Ce qui n'est pas une surprise. le choix de la configuration géométrique du réseau constitue un vrai problème en machine learning.

6 Implémentation

6.1 Implémention de la méthode Perceptron

```
import numpy as np
import matplotlib.pyplot as plt

def perceptron_train(X, y, theta0, k):
    num_correct = 0
    Shape_X = X.size
    n = 2000
```

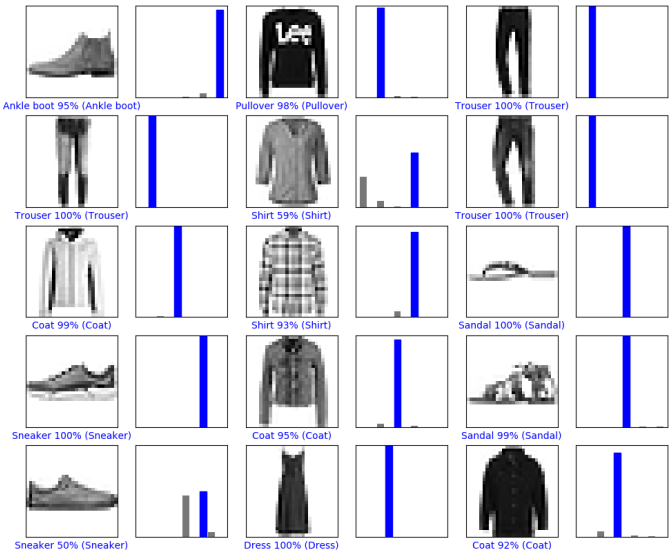


FIGURE 13 – Quelques prédiction avec un réseau de neurones avec une couche cachée avec 128 neurones. L'efficacité est de 0.8733.

```
    d = 2
    curr_index = 1
    theta = np.zeros(d)
    k = 0
    is_first_iter = 1
    while (num_correct < n):
        xt = X[curr_index,:]
        yt = y[curr_index]
        a = np.sign(yt*(np.dot(theta,xt)))
        if (is_first_iter == 1 or a < 0):
            num_correct = 0
            theta = theta + (yt*xt)
            k = k+1
            is_first_iter = 0;
        else:
            num_correct = num_correct + 1

        curr_index = 1 + curr_index

        if (curr_index == n):
            curr_index = 0

    print(k)
    theta0[:] = theta
    return
```

Listing 1 – Algorithme de perceptron

6.2 Implémentation de la méthode SVM

```
from cvxopt import matrix
```

```
from cvxopt.blas import dot
from cvxopt.solvers import qp

import matplotlib.pyplot as plt

def svm_train(X, y):
    X_cvxop  = matrix(X)

    y_cvxop = matrix(y)

    S = matrix(1.0, (d,d))

    G = matrix(0.0, (n,d))
    h = matrix(-1.0, (n,1))
    A = matrix(0.0, (1,d))
    b = matrix(0.0)
    q = matrix(0.0, (d,1))


    for i in range (n):
      G[i,:] = -y_cvxop[i]*X[i,:]


    for i in range (d):
      S[i,i] = 1.0


    theta_op = qp(S, q, G, h) ['x']

   print(theta_op)
```

6.3 Implémentation de l'approche probabiliste

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import inv

# Generate a synthetic dataset from a quadratic function
N = 200  # %Number of data points
#Generate random x values between -5 and 5
x = np.sort(np.random.uniform(-5,5,N))
#% $t = w_0 + w_1x + w_2x^2$
w_0 = 1
w_1 = -2
w_2 = 0.5
#%% Define t
t = w_0 + w_1*x + w_2*(x**2)
#%% Add some noise
t = t + 0.5*np.random.normal(0,1,N)
X = np.ones(N)
for k in range(5):
    if( k == 1):
        X=np.c_[X,x**k]
        linear_w = (inv(X.transpose().dot(X))).dot(X.transpose().dot(t))
```

```
    elif (k==2):
        X=np.c_[X,x**k]
        quad_w = (inv(X.transpose().dot(X))).dot(X.transpose().dot(t))
    elif (k==3):
        X=np.c_[X,x**k]
        third_order_w = (inv(X.transpose().dot(X))).dot(X.transpose().dot(t))
    elif (k==4):
        X=np.c_[X,x**k]
        forth_order_w = (inv(X.transpose().dot(X))).dot(X.transpose().dot(t))

print(linear_w)
print(quad_w)
print(third_order_w)
print(forth_order_w)
x_mesh = np.linspace(-5.0,5.0,N)
plt.plot(x,t, '*')
X_plot = np.ones(N)
X_plot = np.c_[X_plot,x_mesh**1, x_mesh**2,x_mesh**3, x_mesh**4]
plt.plot(x_mesh, X_plot[:,0:2].dot(linear_w), 'g', label='linear')
plt.plot(x_mesh, X_plot[:,0:3].dot(quad_w), 'r', label='quad')
plt.plot(x_mesh, X_plot[:,0:4].dot(third_order_w), '+', label='cubic')
plt.plot(x_mesh, X_plot[:,0:5].dot(forth_order_w), 'b', label='forth order')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Example of parameters estimation")
plt.legend()
plt.show()
```

6.4 Implémentation de l'exemple 1

```
datadir = "chapter6" % repertoire des images
dataset = "pedestrians128x64"
datafile = "%s/%s.tar.gz" % (datadir, dataset)

extractdir = "%s/%s" % (datadir, dataset)

def extract_tar(datafile, extractdir):
    try:
        import tarfile
    except ImportError:
        raise ImportError("You do not have tarfile installed. "
                          "Try unzipping the file outside of Python.")

    tar = tarfile.open(datafile)
    tar.extractall(path=extractdir)
    tar.close()
    print("%s successfully extracted to %s" % (datafile, extractdir))

extract_tar(datafile, datadir)

import cv2
import matplotlib.pyplot as plt
#%matplotlib inline

```

```
plt.figure(figsize=(10, 6))
for i in range(5):
    filename = "%s/per0010%d.ppm" % (extractdir, i+5)
    img = cv2.imread(filename)

    plt.subplot(1, 5, i + 1)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.axis('off')



win_size = (48, 96)
block_size = (16, 16)
block_stride = (8, 8)
cell_size = (8, 8)
num_bins = 9
hog = cv2.HOGDescriptor(win_size, block_size, block_stride, cell_size, num_bins)

import numpy as np
import random
random.seed(42)
X_pos = []
for i in random.sample(range(900), 400):
    filename = "%s/per%05d.ppm" % (extractdir, i)
    img = cv2.imread(filename)
    if img is None:
        print('Could not find image %s' % filename)
        continue
    X_pos.append(hog.compute(img, (64, 64)))

X_pos = np.array(X_pos, dtype=np.float32)
y_pos = np.ones(X_pos.shape[0], dtype=np.int32)
X_pos.shape, y_pos.shape


negset = "pedestrians_neg"
negfile = "%s/%s.tar.gz" % (datadir, negset)
negdir = "%s/%s" % (datadir, negset)
extract_tar(negfile, datadir)

import os
hroi = 128
wroi = 64
X_neg = []
plt.figure(figsize=(10, 6))
compteur = 0
for negfile in os.listdir(negdir):
    filename = '%s/%s' % (negdir, negfile)
    img = cv2.imread(filename)
    img = cv2.resize(img, (512, 512))
    if compteur < 5:
        plt.subplot(1, 5, compteur + 1)
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.axis('off')
```

```
        compteur += 1
    for j in range(5):
        rand_y = random.randint(0, img.shape[0] - hroi)
        rand_x = random.randint(0, img.shape[1] - wroi)
        roi = img[rand_y:rand_y + hroi, rand_x:rand_x + wroi, :]
        X_neg.append(hog.compute(roi, (64, 64)))



X_neg = np.array(X_neg, dtype=np.float32)
y_neg = -np.ones(X_neg.shape[0], dtype=np.int32)
X_neg.shape, y_neg.shape

X = np.concatenate((X_pos, X_neg))
y = np.concatenate((y_pos, y_neg))

from sklearn import model_selection as ms
X_train, X_test, y_train, y_test = ms.train_test_split(
    X, y, test_size=0.2, random_state=42
)

def train_svm(X_train, y_train):
    svm = cv2.ml.SVM_create()
    svm.train(X_train, cv2.ml.ROW_SAMPLE, y_train)
    return svm

def score_svm(svm, X, y):
    from sklearn import metrics
    _, y_pred = svm.predict(X)
    return metrics.accuracy_score(y, y_pred)

svm = train_svm(X_train, y_train)

score_svm(svm, X_train, y_train)

score_svm(svm, X_test, y_test)

score_train = []
score_test = []
for j in range(3):
    svm = train_svm(X_train, y_train)
    score_train.append(score_svm(svm, X_train, y_train))
    score_test.append(score_svm(svm, X_test, y_test))

    _, y_pred = svm.predict(X_test)
    false_pos = np.logical_and((y_test.ravel() == -1), (y_pred.ravel() == 1))
    if not np.any(false_pos):
        print('done')
        break
    X_train = np.concatenate((X_train, X_test[false_pos, :]), axis=0)
    y_train = np.concatenate((y_train, y_test[false_pos]), axis=0)

img_test = cv2.imread('chapter6/pedestrian_test_5.jpg')

```

```
stride = 16
found = []
for ystart in np.arange(0, img_test.shape[0], stride):
    for xstart in np.arange(0, img_test.shape[1], stride):
        if ystart + hroi > img_test.shape[0]:
            continue
        if xstart + wroi > img_test.shape[1]:
            continue
        roi = img_test[ystart:ystart + hroi, xstart:xstart + wroi, :]
        feat = np.array([hog.compute(roi, (64, 64))])
        _, ypred = svm.predict(feat)
        if np.allclose(ypred, 1):
            found.append((ystart, xstart, hroi, wroi))

hog = cv2.HOGDescriptor(win_size, block_size, block_stride, cell_size, num_bins)

rho, _, _ = svm.getDecisionFunction(0)
sv = svm.getSupportVectors()
hog.setSVMDetector(np.append(sv[0, :].ravel(), rho))

hogdef = cv2.HOGDescriptor()

hogdef.setSVMDetector(cv2.HOGDescriptor_getDefaultPeopleDetector())

found, _ = hogdef.detectMultiScale(img_test)

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111)
ax.imshow(cv2.cvtColor(img_test, cv2.COLOR_BGR2RGB))
from matplotlib import patches
for f in found:
    ax.add_patch(patches.Rectangle((f[0], f[1]), f[2], f[3], color='y', linewidth
    =3, fill=False))
plt.savefig('detected.png')

plt.show()
```

6.5 Implémentation de l'exemple 2

```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)


fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data
    ()
```

```
15
16 class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
17                'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
18
19 train_images = train_images / 255.0
20
21 test_images = test_images / 255.0
22
23
24 plt.figure(figsize=(10,10))
25 for i in range(25):
26     plt.subplot(5,5,i+1)
27     plt.xticks([])
28     plt.yticks([])
29     plt.grid(False)
30     plt.imshow(train_images[i], cmap=plt.cm.binary)
31     plt.xlabel(class_names[train_labels[i]])
32
33
34
35 #plt.savefig("AmazonBasis.png")
36 #plt.show()
37
38 model = keras.Sequential([
39     keras.layers.Flatten(input_shape=(28, 28)),
40     keras.layers.Dense(128, activation=tf.nn.relu),
41     keras.layers.Dense(128, activation=tf.nn.relu),
42     keras.layers.Dense(128, activation=tf.nn.relu),
43     keras.layers.Dense(10, activation=tf.nn.softmax)
44 ])
45
46
47 model.compile(optimizer=tf.train.AdamOptimizer(),
48               loss='sparse_categorical_crossentropy',
49               metrics=['accuracy'])
50
51
52 model.fit(train_images, train_labels, epochs=5)
53
54 test_loss, test_acc = model.evaluate(test_images, test_labels)
55
56 print('Test accuracy:', test_acc)
57
58 predictions = model.predict(test_images)
59
60 def plot_image(i, predictions_array, true_label, img):
61   predictions_array, true_label, img = predictions_array[i], true_label[i], img[i]
62   plt.grid(False)
63   plt.xticks([])
64   plt.yticks([])
65
66   plt.imshow(img, cmap=plt.cm.binary)
67
68   predicted_label = np.argmax(predictions_array)
```

```
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array[i], true_label[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')


# Plot the first X test images, their predicted label, and the true label
# Color correct predictions in blue, incorrect predictions in red
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions, test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions, test_labels)


plt.savefig("AmazonBasis_result.png")
plt.show()
```

Références

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow : Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[Anz12] Yuichiro Anzai. *Pattern recognition and machine learning.* Elsevier, 2012.

[Bey17] Michael Beyeler. *Machine Learning for OpenCV : Intelligent image processing with Python*. Packt Publishing, 2017.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[CL11] Chih-Chung Chang and Chih-Jen Lin. Libsvm : a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3) :27, 2011.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA :, 2001.

[Gér17] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. " O'Reilly Media, Inc.", 2017.

[HM94] Martin T Hagan and Mohammad B Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE transactions on Neural Networks*, 5(6) :989–993, 1994.

[JAS18] Daniel N Wilke Jan A Snyman. *Practical Mathematical Optimization : Basic Optimization Theory and Gradient-Based Algorithms*. Springer Optimization and Its Applications. Springer, 2nd ed. 2018 edition, 2018.

[Nie17] Michael Nielsen. *Neural Networks and Deep Learning*. 2017.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830, 2011.

[RG16] Simon Rogers and Mark Girolami. *A first course in machine learning*. CRC Press, 2016.

[Ros58] Frank Rosenblatt. The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386, 1958.

[RVH12] Allen T Craig Robert V. Hogg, Joeseph McKean. *Introduction to Mathematical Statistics*. 7th Edition. Pearson, 7 edition, 2012.

[Sin06] Rohit Singh. *Machine Learning*. MIT, 2006.