Rapport

Zoidberg2.0

From X-ray images, an A.I that can detect pneumonia

Résumé

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas ac semper enim. Mauris at rhoncus ex. Vestibulum non ante sem. Suspendisse eget aliquam purus. Sed lorem massa, elementum at fringilla et, rutrum eu enim. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam rutrum hendrerit pharetra. Fusce nec gravida diam. Cras non nulla consectetur, dictum nisi a, viverra erat. Maecenas dignissim venenatis tellus et ullamcorper. Nullam tincidunt risus a risus consectetur, a convallis tortor commodo. Proin aliquet enim velit, sit amet pretium ex efficitur pulvinar.

Sections

A. FrereJacques, M. Dupont, X. Nouaille, G. Collin, P.A. Bolteau

1 Introduction

1.1 Contexte & Objectifs

1.2 Organisation du projet

2 Le Machine Learning en bref

Avant de rentrer dans les détails de ce projet, nous présentons une rapide introduction au machine learning. Le but est avant tout de fournir les définitions utiles et de donner une compréhension intuitive sur ce sujet. Ainsi de nombreux aspects ne seront pas abordés (notamment l'existence de solutions et leur convergence).

2.1 Machine Learning : Kezako ?

Dans plusieurs domaines dans lesquelles l'informatique est utilisé, il peut être utile que l'on ne décrive pas explicitement les règles d'un algorithme mais plutôt que l'ordinateur les apprend par lui-même. C'est exactement ce que l'on cherche à faire quand on parle de **Machine Learning** (ou apprentissage automatisé en français).
La définition de T. Mitchell fournie une explication plus pratique de ce domaine :

> *"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."*
>
> Tom Mitchell, 1997

Illustrons ça tout de suite avec un des exemples le plus populaire : *La classification des emails en spam ou non-spam.*
Pour déterminer si un email est un spam, on pourrait tenter de définir des règles explicites sur le titre du mail, une blacklist de mots, etc. Cependant, l'algorithme serait excessivement grand si l'on veut qu'il soit efficace, et sa maintenance serait très complexe au fur et à mesure que de nouveaux spams soient créés. C'est là que l'on se rend compte qu'il nous faut passer par le machine learning. En reprenant la définition de T. Mitchell, nous aurions :

- la tâche T : classifier un email en spam ou non-spam
- l'expérience E : examiner une banque de données d'un ensemble d'emails étiquetés comme spam ou non-spam
- la performance P : la fraction d'emails correctement classés comme spam par l'algorithme.

Il existe de nombreux modèles de machine learning que l'on peut classer en sous-domaines présentés dans la Figure 1.
Comme énoncé précédemment, dans ce projet, nous devons réaliser une classification en 3 classes. Nous allons donc nous restreindre à l'apprentissage supervisé, et en particulier à la classification [1].

2.2 De l'apprentissage à la prédiction

Essayons de formaliser la problématique de l'apprentissage supervisé :
A partir d'observations données $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y},\ 1 \leq i \leq m$ de *entées*[2]*/sorties* (appelées **données**

1. A noté une équivalence entre classification et régression. En effet, la plupart des modèles s'adaptent très bien aux deux problématiques (SVM, decision tree, etc.).

2. En générale, $x_i \in \mathbb{R}^n$ i.e. x_i est un vecteur de taille n où n correspond au nombre de *features* (variables caractéristiques de la donnée d'entrée). Dans notre projet, n correspond au nombre de pixels dans les images.

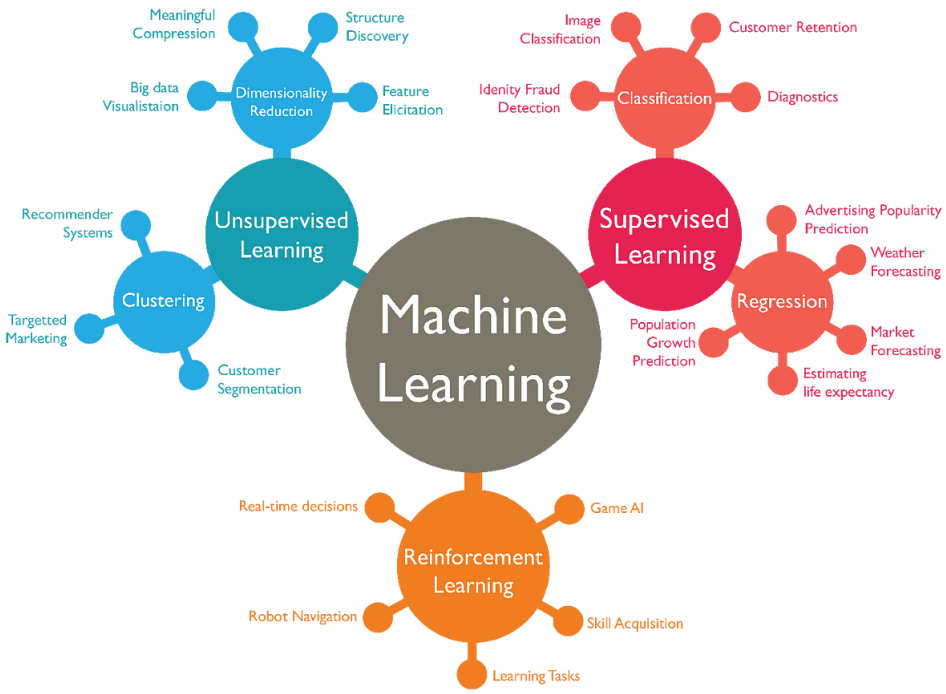


FIGURE 1. Cartographie non exhaustive des modèles de machine learning

d'entraînements), notre but est de prédire un nouveau label $y \in \mathcal{Y}$ à partir d'une nouvelle donnée $x \in \mathcal{X}$ (ces nouvelles données étant appelée **données de test**).
La prédiction n'est autre qu'une fonction dépendant de paramètres θ appelé **fonction d'hypothèse** :

$$h_\theta : \mathcal{X} \to \mathcal{Y}$$

Pour Zoidberd2.0, $\mathcal{X}$ est l'ensemble des images et $\mathcal{Y} = \{$ *normal*, *virus*, *bacteria* $\}$.
La forme de h_θ dépend du modèle choisie [3] : cela peut-être un simple polynôme dans le cas de la régression linéaire ou une fonction plus complexe comme ... un réseau de neurones (détaillé à la section 3). En revanche, θ est justement ce que l'on cherche à trouver via l'apprentissage.

En résumé, on cherche à trouver un algorithme $\mathcal{A}$ tel que $\mathcal{A}$ prend en entrée la distribution $\mathcal{D}(x_i)$ et nous renvoie θ tout en cherchant à ce que $h_\theta(x_i)$ soit le plus proche de y_i.

2.3 Risque & Fonction de perte

Le coeur de l'apprentissage supervisé réside dans le fait de mesurer à quel point la valeur prédite d'une donnée d'entraînement est éloignée de sa valeur réelle. Pour quantifier cet écart, on définit la **fonction de perte** (loss function en anglais) :

$$\begin{aligned} \mathcal{L} : \mathcal{Y} \times \mathcal{Y} &\to \mathbb{R}^+ \\ (\overline{y}\,,\, y) &\mapsto \mathcal{L}(\overline{y}, y) \end{aligned}$$

avec $\overline{y_i} = h_\theta(x_i)$. $\mathcal{L}(\overline{y}, y)$ tend vers zéro si la prédiction est proche de la valeur réelle et tend vers l'infini dans le cas contraire.

En calculant la moyenne de la fonction de perte sur l'ensemble des données d'entraînement, on obtient le risque du modèle :

$$\mathcal{R}_\mathcal{A}[\mathcal{D}(x_i)] = \mathbb{E}\Big[\mathcal{L}(h_\theta(x_i), y_i)\Big] = \frac{1}{m}\sum_i^m \mathcal{L}(h_\theta(x_i), y_i)$$

3. La fonction d'hypothèse ne peut pas prendre n'importe quelle forme : elle doit être un *concept que l'on peut apprendre* et mesurable par la VC dimension. [1] [2]

Ainsi, l'algorithme $\mathcal{A}$ cherche $\tilde{\theta}$ tel que :

$$\tilde{\theta} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{R}_{\mathcal{A}}[\mathcal{D}(x_i)]$$

La définition de ce risque [4] permet de transposer notre problématique d'apprentissage en un problème d'optimisation. En effet, tout en gardant une très grande capacité de généralisation, nous avons pu réduire notre problème d'apprentissage au simple fait de minimiser une fonction.

Il existe de plusieurs techniques d'optimisation (comme l'équation normale) mais la plus utilisée est le **Gradient Descent** et ses dérivés [5] :

L'algorithme de Gradient Descent [a]

1. On initialise θ aléatoirement.
2. On met à jour θ pour réduire $J(\theta)$ jusqu'à ce qu'on atteigne un minimum :

```
repeat until convergence {
```

$$\theta_j := \theta_j - \eta \frac{\partial}{\partial \theta_j} J(\theta) \quad (\text{pour } j = 0, \dots, n)$$

```
}
```

où $J(\theta) = \mathcal{R}_{\mathcal{A}}[\mathcal{D}(x_i)]$ et η est le taux d'apprentissage (learning rate en anglais) qui contrôle la taille des pas que l'algorithme prend à chaque étape.

a. Le Gradient Descent est développé en annexe A.1.

Cette partie étant très théorique, on propose un exemple de classification avec la régression logistique en annexe A.2.

2.4 Performances

Comme la tâche principale du machine learning consiste à sélectionner un modèle et à l'entraîner sur des données, les deux choses qui peuvent mal tourner sont soit un "mauvais modèle", soit de "mauvaises données".

2.4.1 Validation

Avant de regarder les problèmes énoncés, voyons comment peut-on mesurer si un modèle performe bien ou non.
La première des choses est de séparer notre base de donnée en deux (usuellement en 80%/20%) :

— données d'entraînement : données sur lesquelles le modèle apprend
— données de test : données sur lesquel le modèle final est testé. Ces données ne devront être **utilisées qu'à la toute fin** pour voir si le modèle performera bien une fois déployé.

Lorsque l'on entraîne un modèle, on cherche également à trouver ces meilleurs *hyper-paramètres* [6]. Pour ce faire, nous utilisons la validation :

— ensemble de validation : on redivise notre ensemble d'entraînement en 80%/20% et on test ces hyper-paramètres sur ces 20%
— cross-validation : on répète plusieurs fois le découpage entraînement/validation de manière aléatoire (souvent préféré pour maximiser les données d'entraînement et réduire la variabilité de la procédure de validation).

4. En réalité, le risque $\mathcal{R}_{\mathcal{A}}$ tend vers une valeur optimale et ce que l'on cherche à minimiser est $\mathcal{R}_{\mathcal{A}} - \mathcal{R}^*$ où $\mathcal{R}^*$ est le risque de Bayes. Pour plus de détails, on se référa à l'excellent cours de Francis Bach [3].

5. Il existe de nombreuses variantes du Gradient Descent que nous n'aborderons pas ici comme le Gradient Descent Stochastique ou Adam.

6. Les hyper-paramètres du modèle sont des paramètres qui ne sont pas appris par le modèle lui-même, mais qui doivent être choisis avant l'entraînement pour déterminer la structure et le comportement du modèle. Exemples : le learning rate dans le Gradient Descent, le nombre de neurones dans un réseau, etc.

2.4.2 Gérer les données

On peut maintenant regarder les problématiques face aux mauvaises données :

1. *Manque de données* : C'est probablement le problème le plus courant. La plupart des modèles nécessitent des milliers de données (voire des millions pour les réseaux de neurones).
2. *Données non représentative* : Les données d'entraînement doivent être les plus représentatives des cas que nous souhaitons généraliser.
3. *Mauvaise qualité des données* : Trop d'erreurs, de bruits ou d'outliers dans nos données créeront inexorablement un biais dans notre modèle.
4. *Features non pertinentes* : Le système ne peut apprendre que si les features fournies permettent de caractériser l'output attendu. Pour Zoidberg2.0, la date de création des images ne nous permettrait en rien de classifier nos images.

2.4.3 Underfitting & Overfitting

Côté modèle, les erreurs viennent du dilemme *biais / variance*.

1. Biais : Il y a un biais lorsque les hypothèses de départ sont erronées. Cela se produit lorsque le modèle est trop simple par rapport aux données fournies. Le modèle aura une grande erreur sur les données d'entraînement. On parle alors de sous-apprentissage (*underfitting* en anglais).
2. Variance : Une grande variance du modèle correspond à sa grande sensibilité aux légères variations des données d'entraînement. Le modèle va très bien apprendre mais il aura des erreurs importantes sur de nouvelles données. On parle alors de sur-apprentissage (*overfitting* en anglais).

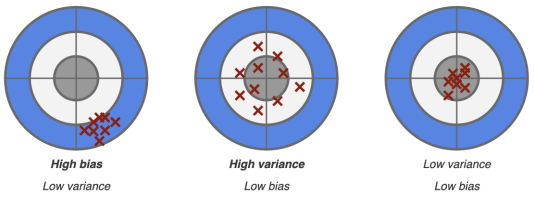


FIGURE 2. Exemple du dilemme Biais / Variance sur une cible

Dans les deux cas, il existe plusieurs solutions possibles en fonction du modèle utilisé. Voici quelques-unes des solutions possibles :

— Underfitting :
  1. choisir un modèle plus complexe avec plus de paramètres
  2. utiliser des caractéristiques (features) plus adaptées pour le modèle
  3. réduire les contraintes imposées au modèle.

— Overfitting :
  1. choisir un modèle plus simple avec moins de paramètres
  2. utiliser la validation croisée pour mieux évaluer la performance du modèle
  3. ajouter plus de données au modèle pour augmenter la variété des exemples
  4. arrêter l'apprentissage du modèle dès qu'il ne progresse plus pour éviter d'apprendre trop spécifiquement les exemples d'entraînement.

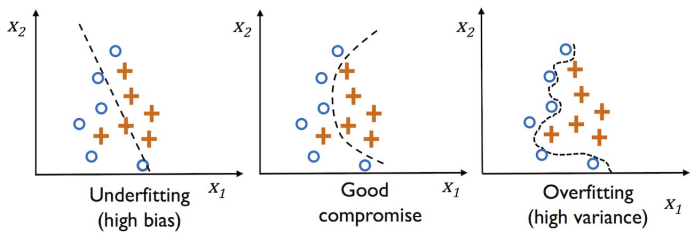


FIGURE 3. Underfitting et Overfitting dans le cas d'une classification

3 Plongée dans le Deep Learning

Deep Learning vs ML

3.1 Un neurone

linearly separable data
linear transformation : SVD
non-linear transformation : activation
Perceptron

3.2 Des neurones

réseau ANN
définition utiles
Backpropagation, GD
fully connected

3.3 Les Réseaux de Convolution

la convolution
relu
le pooling
CNN

4 Préparation des données

4.1 Préparation des données

4.2 Augmentation de données

5 Modèles utilisés

5.1 Modèle 1

5.2 Modèle 2

6 Résultats obtenus

6.1 Évaluation de la performance

6.2 Comparaison des modèles

Références

[1] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the vapnik-chervonenkis dimension," *J. ACM*, vol. 36, no. 4, p. 929–965, oct 1989. [Online]. Available : https://doi.org/10.1145/76359.76371

[2] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*, 2nd ed. Cambridge, MA : MIT Press, 2018.

[3] F. Bach, "Introduction to machine learning," Cours magistral, 2021, École Normale Supérieur, M2 ICFP. [Online]. Available : https://www.di.ens.fr/~fbach/ML_physique_2021.html

[4] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow : concepts, tools, and techniques to build intelligent systems*, 2nd ed. Sebastopol, CA : O'Reilly Media, 2019.

[5] Wikipedia, "Hyperplane separation theorem," 2006, accessed version 23/03/2023. [Online]. Available : https://en.wikipedia.org/wiki/Hyperplane_separation_theorem

A Annexes

A.1 Gradient Descent en détails

On note $J(\theta)$ le risque et θ est vecteur de dimension n. Comme on l'a expliqué dans notre discussion dans la section 2.3, nous souhaitons minimiser le risque au regard des paramètres θ.
Pour ce faire, nous allons revoir ensemble les principes de la dérivation. On commence avec un seul paramètre i.e. $\theta \in \mathbb{R}$

A.1.1 Dimension $n = 1$

Supposons que $J(\theta)$ soit de la forme ci-dessous, que nous prenions deux points (θ_a, J_a), (θ_b, J_b) avec $J_a = J(\theta_a)$, $J_b = J(\theta_b)$, $\theta_b > \theta_a$. Nous voulons savoir si en allant de θ_a à θ_b nous grimpons la courbe de J ou nous la descendons ?

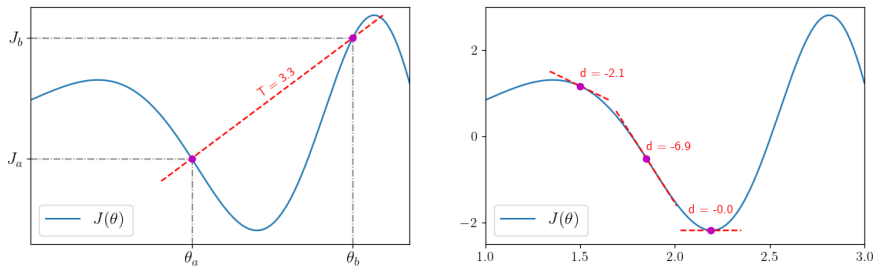


FIGURE 4. Exemple du calcul d'un taux d'accroissement (graphique de gauche) et de la valeur de la dérivée sur plusieurs points (graphique de droite).

Pour répondre à cette question, nous pouvons tracer une droite entre ces deux points et observer si la pente de cette droite est positive ou négative. Cette pente est appelée le taux d'accroissement et est défini comme suit :

$$T = \frac{J_b - J_a}{\theta_b - \theta_a}$$

Si T est positif, cela signifie que nous avons grimpé, alors que si T est négatif, cela indique que nous avons descendu (exactement comme pour le dénivelé d'une route). Cependant, le problème ici est que nous avons ignoré tout ce qui s'est passé entre θ_a et θ_b. Il serait donc plus pratique de savoir si, à chaque point de la courbe, nous montons ou descendons lorsque nous nous déplaçons vers la gauche ou vers la droite.
Pour cela, on peut réduire l'écart entre θ_a et θ_b de sorte qu'il soit à peine discernable (pour qu'il n'y ai pas de variation entre ces deux points).
Redéfinissons $\theta_b = \theta_a + d\theta$ où $d\theta$ est une distance infinitésimal entre θ_a et θ_b, nous avons :

$$T = \frac{J(\theta_a + d\theta) - J(\theta_a)}{d\theta} \coloneqq \frac{dJ}{d\theta}(\theta_a)$$

Nous venons de définir la dérivée. Pour reformuler, avec $d\theta$ très petit, c'est comme si nous avions suffisamment zoomé pour que la courbe ressemble à une droite. Si la pente de cette droite est positive, alors en nous déplaçant vers la droite, on monte. Si on se déplace vers la gauche, on descend et on attendra un minimum [7] (inversement si la dérivé est négative). Le Gradient Descent utilise justement ce concept :

1. On commence par se placer aléatoirement sur la courbe à θ^0
2. On calcul la dérivée de J à θ^0. Si elle est positive alors il faut reculer (i.e. $\theta^1 < \theta^0$), sinon il faut avancer (i.e. $\theta^0 < \theta^1$)

7. Il peut exister plusieurs minimums pour une fonction, nous nous cherchons le minimum global, c'est-à-dire le plus "bas". Si le modèle admet un unique minimum, on dit que le modèle est convexe.

3. Ce qui nous donne la mise à jour :

$$\theta^{k+1} = \theta^k - \eta \frac{dJ}{d\theta}(\theta^k)$$

où le learning rate η définit simplement si le pas entre θ^{k+1} et θ^k est plus ou moins grand.

4. Quand on atteint le minimum, la pente devient nulle (en effet au minimum, on stagne : $J(\theta + d\theta) = J(\theta)$). Ainsi, $\theta^{k+1} = \theta^k$ et l'on dit que l'on a convergé.

A.1.2 Dimension $n > 1$

Le cas pour $\theta \in \mathbb{R}^n$ avec $n > 1$ est très similaire. Restreignons nous à $n = 2 : \theta = (\theta_1, \theta_2)^T \implies J(\theta) = J(\theta_1, \theta_2)$. Savoir comment varie J, revient à savoir comment varie J selon θ_1 **et** selon θ_2. Autrement dit, la variation de J, noté dJ comme précédemment, est la somme des variation de J selon les deux axes θ_1, θ_2 :

$$dJ = \frac{\partial J}{\partial \theta_1} d\theta_1 + \frac{\partial J}{\partial \theta_2} d\theta_2$$

Les ∂ ("d rond") signifient dérivées partielles. La dérivée partielle $\frac{\partial J}{\partial \theta_1}$ correspond simplement à la dérivée de J par rapport à θ_1 en laissant θ_2 constant [8]. Autrement dit, on regarde comment varie J quand on se déplace sur l'axe de θ_1, sans se déplacer sur l'axe θ_2.
C'est précisement ce qui nous intéresse pour le Gradient Descent. On souhaite mettre à jour θ_1 pour atteindre le minimum selon son axe puis faire de même avec θ_2. D'où la mise à jour :

$$\theta_i^{k+1} = \theta_i^k - \eta \frac{\partial J}{\partial \theta_i}(\theta_i^k) \quad \text{avec} \quad i = 1, 2$$

A.2 Régression Logistique

Illustrons la section 2.3 avec le "hello world" de la classification : la régression logistique. Commençons par poser le problème :
Supposons un jeux de données très simple avec 2 features et 2 classes présenté dans la figure 5.

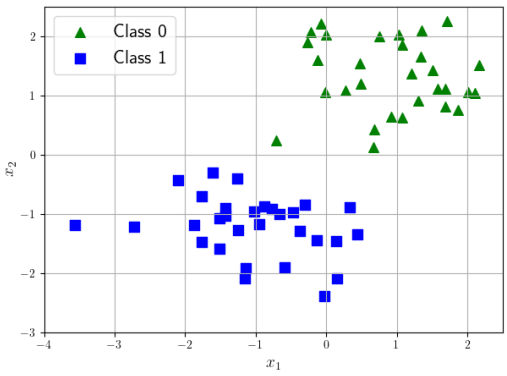


FIGURE 5. Exemple de données 2D linéairement séparable

La première étape est de regarder les données. Ce que nous voyons tout de suite c'est que les données de chaque classes sont bien distinctes et qu'il est possible de les séparer linéairement (i.e. on peut tracer une droite [9] pour séparer les 2 classes). Ce problème est donc idéal pour la régression logistique, qui est une classification linéaire [10].

8. Le vecteur dont l'élément selon l'axe i est la dérivée partielle selon θ_i de J est appelé **gradient** de J d'où le nom "Gradient Descent". Il est généralement noté avec un nabla : $\nabla_\theta J = (\partial_{\theta_1} J, \partial_{\theta_2} J)^T$

9. Dans un cas général où il y a $n \geq 1$ features, on ne parle pas de droite mais d'hyperplan. Pour $n = 1$, la séparation se fait au niveau d'un point, pour $n = 2$, d'une droite, pour $n = 3$, d'un plan, etc. Pour plus de détails, on peut se référer l'article sur Wikipédia [5]

10. Si les données ne sont pas linéairement séparables, on peut quand même utiliser la régression logistique via le rajout de dimensions. On peut donner au modèle de nouvelles features ou des combinaisons des premières tel que $x_3 = x_1^2$, $x_4 = x_1.x_2$, etc.

Pour rappel, une droite correspond à la représentation graphique d'une fonction linéaire f. Ainsi dans notre exemple, on a :

$$f(x_1) = \alpha x_1 + \beta \quad \text{avec} \quad f(x_1) = x_2$$

On peut transformer cela pour nous donner l'équation de la droite :

$$\theta_1.x_1 + \theta_2.x_2 + \theta_3.1 = 0$$

Si l'on prend un point (x'_1, x'_2) et qu'il vérifie l'équation précédente alors ce point appartient la droite. Si $\theta_1.x'_1 + \theta_2.x'_2 + \theta_3.1 < 0$, le point est "en-dessous" de la droite et "au-dessus" si le résultat est strictement positif. On peut donc utiliser cette équation pour obtenir notre classification. Il nous faut juste trouver les bons paramètres $(\theta_1, \theta_2, \theta_3)$ pour que la droite soit située entre les 2 classes.

On peut noter qu'il y a $2 + 1$ paramètres. Si l'on note cette équation d'un point de vue vectoriel, on a :

$$\theta.x = 0 \quad \text{avec} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

La composante 1 dans x est appelé *bias term*. Ainsi, on passe simplement de $x \in \mathbb{R}^n$ à $x \in \mathbb{R}^{n+1}$.

Avec ce qu'on vient de voir, on pourrait utiliser la fonction $g_\theta(x) = \theta.x$ comme fonction d'hypothèse mais cette fonction peut prendre n'importe quelle valeur entre $-\infty$ et $+\infty$ alors que l'on a seulement deux classes (une de valeur 0 et l'autre de valeur 1). C'est pourquoi, on enveloppe cette fonction avec la fonction sigmoïde. C'est fonction nous renvoie uniquement des valeurs entre 0 et 1.

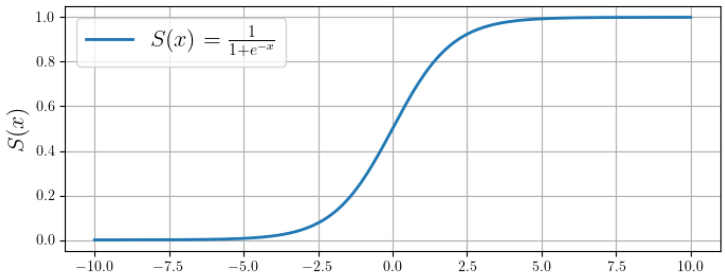


FIGURE 6. Représentation graphique de la fonction sigmoïde

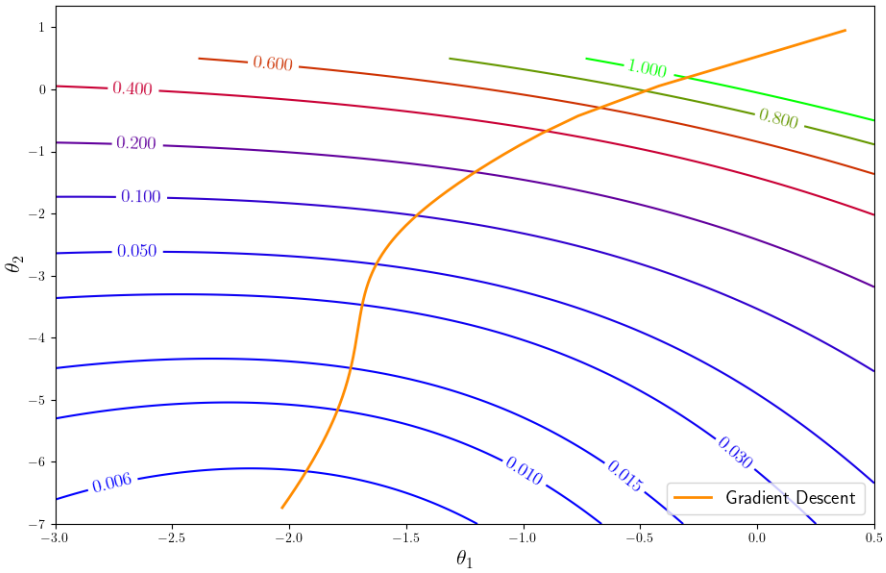
Ce qui nous donne la fonction d'hypothèse :

$$h_\theta(x) = \frac{1}{1 + e^{-\theta.x}}$$

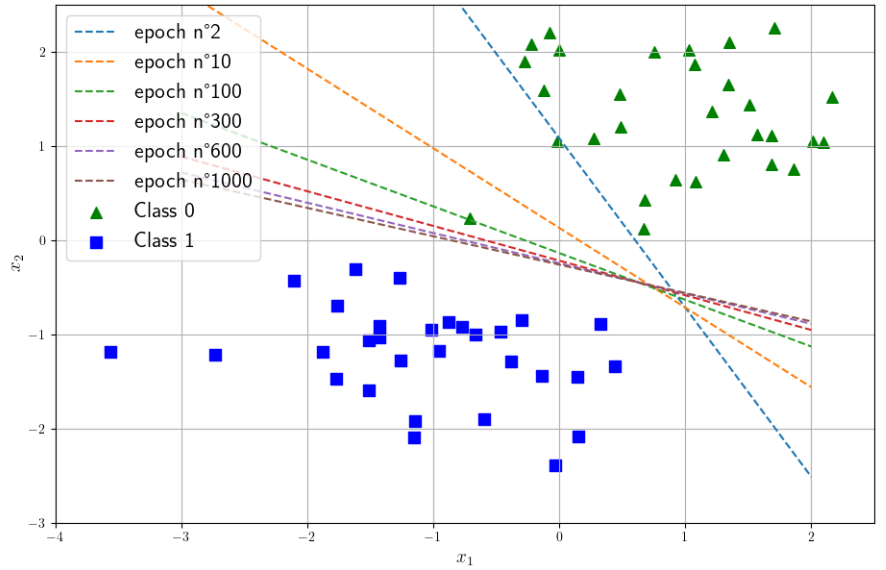
Dans le cas d'une classification, la fonction de perte la plus utilisée est l'entropie de Shanon [2] [4] qui donne le risque suivant :

$$J(\theta) = \frac{1}{m} \sum_i^m -y_i \log(h_\theta(x_i)) + (1 - y_i) \log(1 - h_\theta(x_i))$$

On peut maintenant minimiser ce risque avec le gradient Descent pour obtenir $\tilde{\theta}$.
Le droite séparant les deux classes (appelé frontière de décision) obtenue après différentes mises à jour de θ est présentée dans la figure 7.



(A) Gradient Descent sur $J(\theta)$ par rapport à θ_1 et θ_2



(B) Evolution des frontières de décision après plusieurs mises à jour de θ

FIGURE 7. Gradient Descent illustré sur notre jeu de donnée (1 epoch correspond à une mise à jour de θ)