Supervised Machine Learning

Master 1 - MIASHS

Guillaume Metzler

Université Lumière Lyon 2
Laboratoire ERIC, UR 3083, Lyon

guillaume.metzler@univ-lyon2.fr

Automne 2022

Quelques algorithmes

Sommaire

On va rapidement présenter quelques algorithmes assez simples :

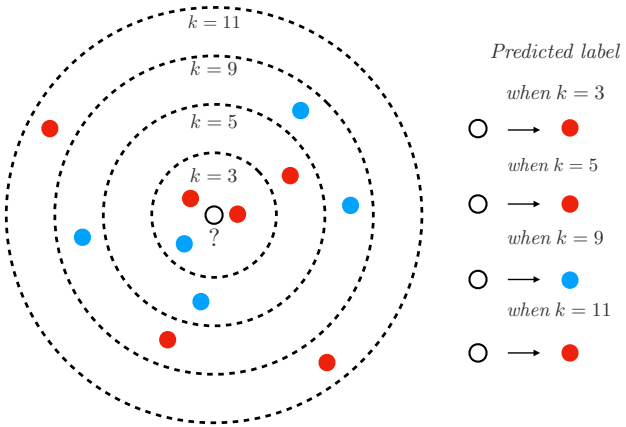
- l'algorithme du plus proche voisin
- la régression linéaire (simple et multiple)
- la régression logistique
- les séparateurs à vaste marge
- réseaux de neurones
- arbres de décisions

Certains de ces algorithmes et parfois d'autres, non mentionnés ici, seront abordés/traités en TD.

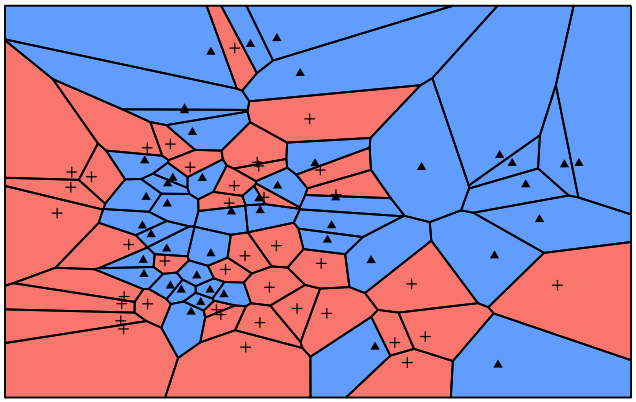
k-plus proches voisins

$k-$NN I

Il s'agit d'un algorithme non paramétrique utilisé pour effectuer des tâches de classification (mais aussi de régression) [Cover and Hart, 1967].



Exemples $k-$NN

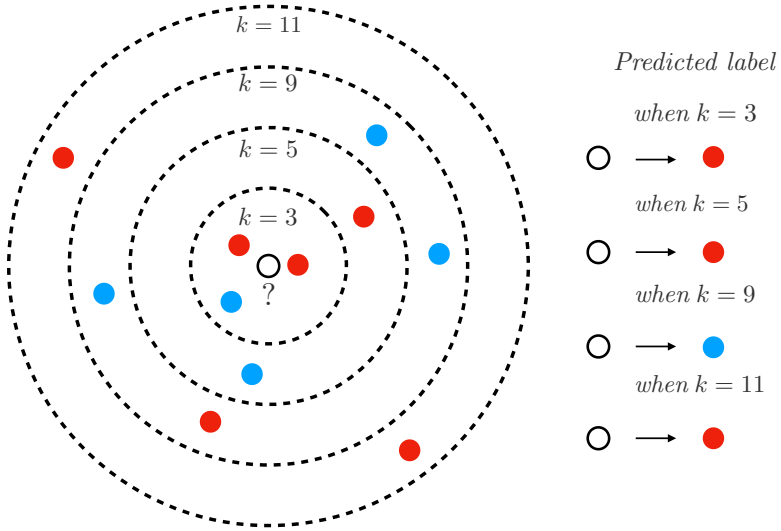


Régions de Voronoï, $k = 1$

$k-$NN II

Il s'agit, en outre, d'un algorithme de classification basé sur la règle de Bayes, i.e. une donnée est prédite comme appartenant à la classe $c^\star$ si

$$c^\star = \arg\max_{c \in \mathcal{Y}} p_k(y = c \mid \mathbf{x}).$$



$k-$NN III

Définition 1.1: Règle de classification

La classe assignée à un nouvel exemple $\mathbf{x}_i$ va dépendre de son positionnement dans l'espace des données par rapport aux données de l'ensemble d'entraînement. Plus précisément, on lui assigne la même étiquette que l'étiquette majoritaire dans son $k-$voisinage

$$c^\star = \arg\max_{c \in \mathcal{Y}} \frac{k_c}{k},$$

ou k_c désigne le nombre de voisins appartenant à la classe c.

$k-$NN IV

- Apprentissage : nécessite de garder en mémoire tous les exemples d'entraînement (mémoire en $\mathcal{O}(m)$)

- Prédiction : nécessite de calculer la distance à tous les exemples d'apprentissages (complexité $\mathcal{O}(md)$) puis d'ordonner les plus proches voisins

$k-$NN V

Cet algorithme repose donc sur la notion de **distance**. On utilise couramment la distance euclidienne $\|\mathbf{x} - \mathbf{x}'\|_2 = \sqrt{\sum_{j=1}^{d}(x_j - x'_j)^2}$, mais nous pourrions utiliser n'importe quelle norme ℓ^p.
Mieux encore ! On pourrait définir sa propre distance $\rightarrow$ **Metric Learning**

Bien que très simple d'utilisation, cette méthode présente rapidement quelques limites :

- en grande dimension, tous les exemples seront rapidement éloignés (l'espace est très rapidement vide)
- un temps de calcul qui augmente linéairement avec le nombre d'exemples

$k-$NN VI

Il existe des méthodes de pré-traitement qui permettent de réduire le temps de calcul de cet algorithme, en supprimant des données en ou en créant des groupes ou encore en réduisant la dimension.

Condensed Nearest Neighnor

Input: Echantillon d'apprentissage S
Output: Un échantillon S' plus petit que S
begin
  Séparer S en deux ensembles aléatoires S_1 et S_2
  while S_1 *et* S_2 *sont modifés* **do**
    Retirer de S_1 tous les exemples mal classifiés à l'aide de S_2 et d'un $1-$NN
    Retirer de S_2 tous les exemples mal classifiés à l'aide de S_1 et d'un $1-$NN
  $S' = S_1 \cup S_2$;
  return S'

On peut aussi faire clustering (créer des modèles locaux) ou encore de l'ACP (réduction de dimension).

$k-$NN VII

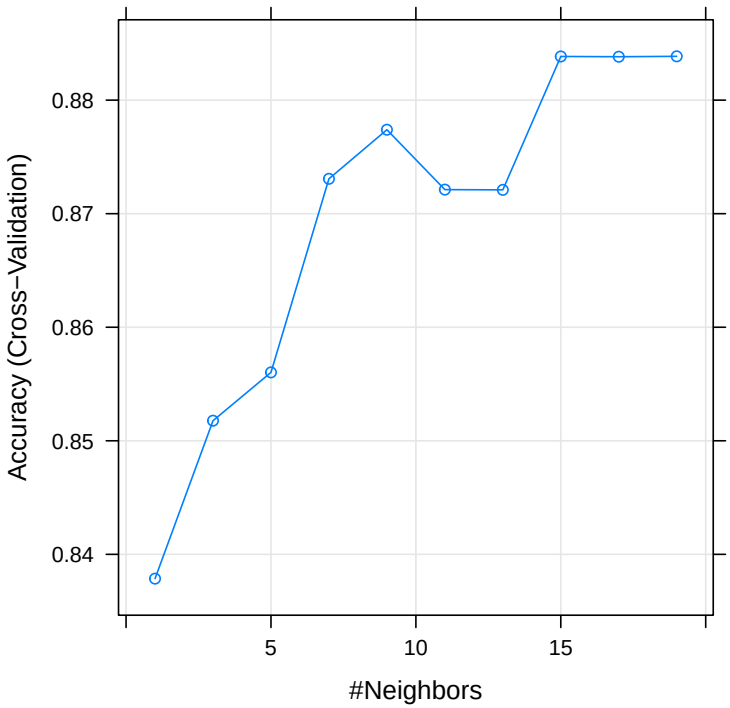
En pratique

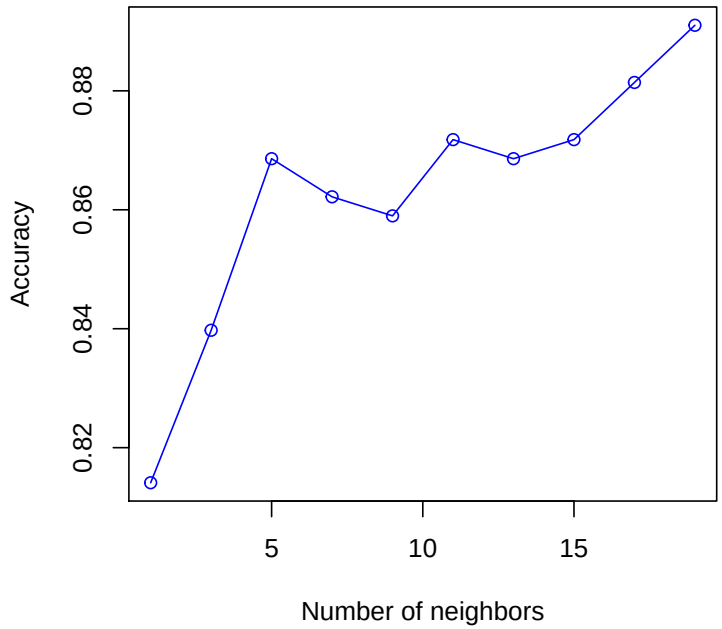
C'est un algorithme non paramétrique, *i.e.* il n'y aucun paramètre à apprendre ! Il y a simplement un hyper-paramètre à tuner : k, le nombre de voisins.

On pourra regarder les packages **caret** de R ou encore la librairie **model_selection** de Python pour effectuer cette cross-validaton.

$k-$NN VIII

Un exemple en application de la cross-validation





$k-$NN IX

Pour finir

C'est un algorithme simple à implémenter et qui peut parfois fournir de très bons résultats.

Il est à la base de plusieurs autres algorithmes plus complexes basées sur le Metric Learning (apprentissage de représentation) mais peut aussi servir à faire du pré-traitement sur les données (*e.g.* des méthodes d'échantillonnage comme on l'a vu avec CNN par exemple).

En revanche, brute, il ne sera que très peu utile/efficace lorsque l'on travaille avec une quantité importante de données.

Régression Linéaire

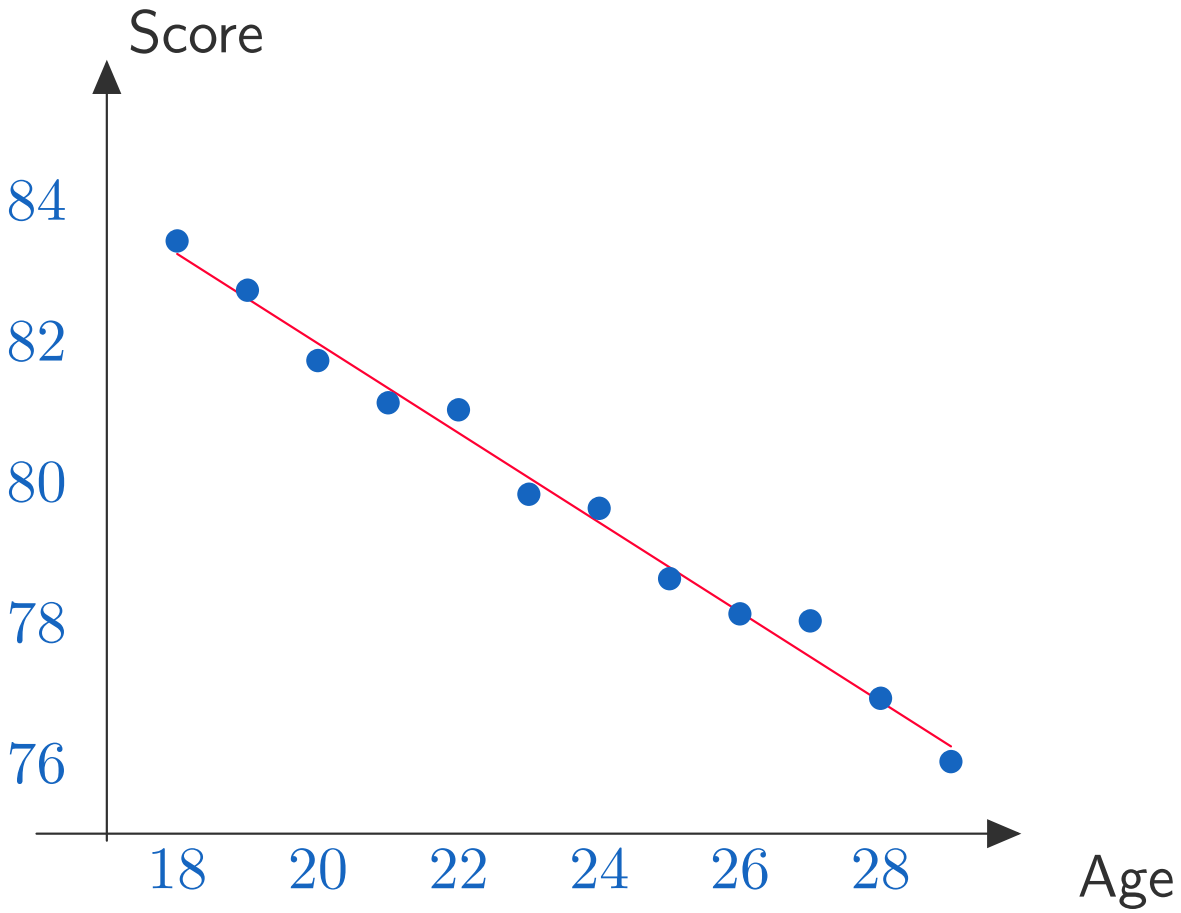
Régression linéaire I

On considère ensuite un ensemble de données dont les descripteurs sont entièrement numériques, *i.e.* on dispose de données de $(x_i)_{i=1}^m \in \mathbb{R}^d$ et **la variable à prédire y_i est un nombre réel**, *i.e.* $y_i \in \mathbb{R}$.

Exemple : lorsque l'on cherche à prédire le score obtenu par une personne à un test donné en fonction de son âge.

On peut utiliser un ou plusieurs prédicteur(s)/variable(s) pour essayer d'apprendre ce modèle (proche des statistiques).

Régression linéaire II



Régression linéaire III

Définition 1.2: Modèle Linéaire Gaussien

Le modèle linéaire **gaussien** simple est le cas particulier où l'on cherche à prédire une variable Y en fonction de la seule variable X, pour cela, on suppose que notre modèle s'écrit sous la forme :

$$Y = X\beta + \varepsilon, \quad \text{où} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

où $X = (1, \mathbf{x})$ pour prendre en compte la constante dans le modèle, ce que l'on peut réécrire

$$y = \beta_0 + \beta_1 x + \varepsilon.$$

On retrouve donc l'équation d'une droite dans le plan.

ε est un bruit gaussien et représente les perturbations du modèle ou l'erreur présente dans les données.

Régression linéaire : formalisation I

Objectif : Minimiser l'erreur de prédiction, *i.e.* l'écart entre la valeur réelle et la valeur prédite.

Nous avons vu que nous avons besoin pour cela d'une fonction objective (ou loss) que l'on cherche à optimiser. Mais pour le moment nous n'avons vu que des loss pour des algorithmes de classification. Mais on peut aussi rencontrer des loss dans un contexte de régression, par exemple, pour un **modèle linéaire**, on va chercher à minimiser **l'erreur quadratique (moyenne)**

$$\ell(y, \hat{y}) = (y - \hat{y})^2 = (y - (\beta_0 + \beta_1 x))^2$$

et donc le risque empirique sur un échantillon S à minimiser serait de la forme

Régression linéaire : formalisation II

$$\mathcal{R}_S(h) = \frac{1}{m}\sum_{i=1}^{m}(y_i - (\beta_0 + \beta_1 x_i))^2.$$

Régression linéaire : formalisation III

L'estimation des paramètres du modèle se fait par la **méthodes des moindres carrés**.

Il s'agit de trouver les valeurs de β_0 et β_1 qui vont minimiser l'erreur quadratique moyenne :

$$\mathcal{R}_S(h) = \frac{1}{m}\sum_{i=1}^{m}(y_i - (\beta_0 + \beta_1 x_i))^2.$$

Comme il s'agit d'une fonction convexe et que le problème est relativement simple, on peut aisément trouver une solution sous forme close, *i.e.* une expression des solutions.

On peut montrer que le coefficients β_1 s'exprime uniquement en fonction de la **variance** de X et de la **covariance** entre X et Y et que β_0 s'expriment en fonction des moyennes de X et Y et de β_1.

Exercice

- Rappeler les hypothèses du modèle linéaires gaussien.
- Montrer que le problème d'optimisation est bien convexe.
- Résoudre le problème de la méthode des moindres carrés dans le cas où notre modèle ne comprend que deux paramètres β_0 et β_1.
- Rappeler comment tester la significativité des paramètres de la droite de régression.

Régression linéaire multiple I

Le problème est exactement le même que précédemment, à la seule différence que nous utilisons non plus un mais plusieurs descripteurs (variables) X pour essayer d'expliquer les observations Y.
Notre modèle s'écrit toujours sous la forme :

$$Y = X\beta + \varepsilon \quad \text{où} \quad \varepsilon \sim \mathcal{N}(0, \sigma^2)$$

où

- $Y = (y_1, ..., y_m)$ est un vecteur de $\mathbb{R}^m$ est **la variable à expliquer**
- $X = (1, \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_d)$ est une matrice de $\mathbb{R}^{m \times (d+1)}$est **la variable explicative** et $\mathbf{x}_j \in \mathbb{R}^m$ et correspond à un descripteur en particulier
- $\beta = (\beta_0, \beta_1, ..., \beta_d)$ est un vecteur de $\mathbb{R}^{d+1}$ qui représente les paramètres du modèle.

Notons h le modèle associé pour la suite.

Régression linéaire multiple II

La résolution peut également se faire par la méthode des moindres carrés.

On cherche à résoudre le problème d'optimisation suivant :

$$\underset{\beta \in \mathbb{R}^{d+1}}{\arg\min} \|Y - X\beta\|_2^2 = \sum_{i=1}^{m} (y_i - h(\mathbf{x}_i))^2.$$

On montre alors qu'une estimation de β est donnée par la relation

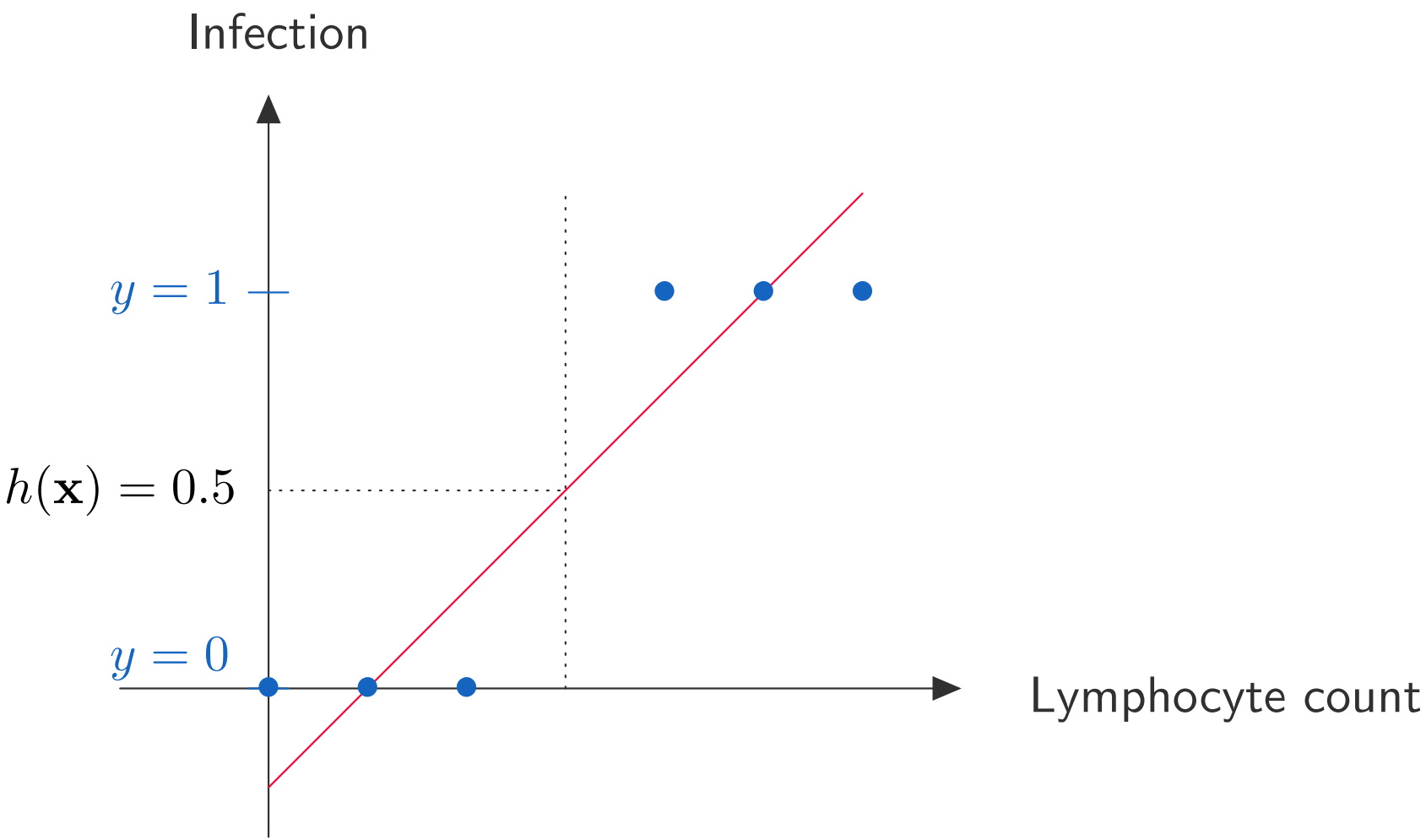
$$\boldsymbol{\beta} = (X^T X)^{-1} X^T Y,$$

à la condition que la matrice $X^T X$ soit bien inversible, *i.e.* que les variables ne soient pas corrélées.

Exercice

- Retrouver l'expression du paramètre β donnée dans le slide précédent, en résolvant le problème des MCO.
- Expliquer pourquoi et justifier (par le calcul) que ce problème de minimisation admet une seule et unique solution.
- Montrer que l'expression de β ainsi obtenue est aussi celle de l'estimateur du maximum de vraisemblance.
-

De la régression vers la classification I



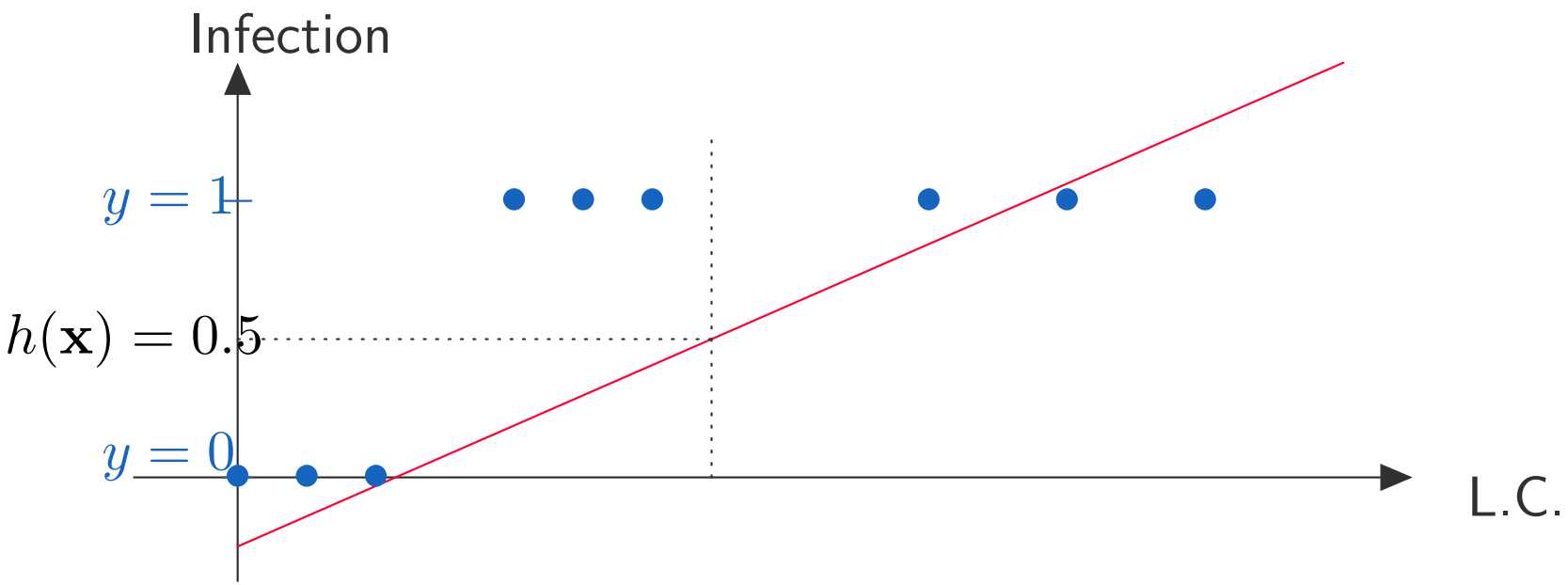
De la régression vers la classification II

Dans cet exemple, on souhaite déterminer si une personne présente une infection en fonction de sa numération en lymphocytes.

Le problème que l'on considère devient alors un problème de classification, mais qu'est-ce qui nous empêche de le voir a priori comme un problème de régression ?

Si on reprend le jeu de données présentés précédemment, tout semble se passer pour le mieux. Mais est-ce toujours le cas si on modifie un peu ces données ?

De la régression vers la classification III



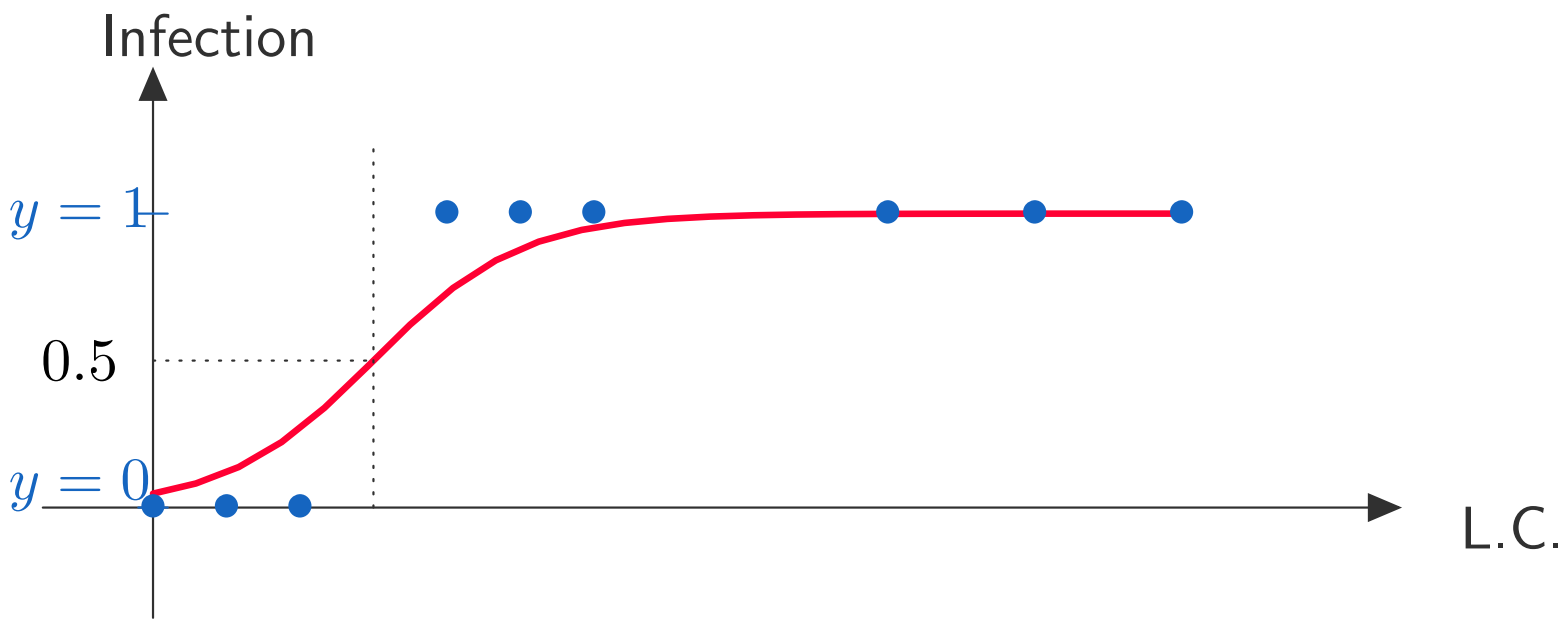
De la régression vers la classification IV

Cet exemple très simple montre que le modèle de régression employé n'est pas approprié et qu'il va certainement falloir voir la valeur que l'on cherche à prédire y non pas comme un nombre réel, plutôt comme un identifiant.

Problème : le modèle actuel renvoie des valeurs réelles alors que ce que l'on cherche à prédire prend ses valeurs dans l'ensemble $\{0, 1\}$.

Solution : il faudrait trouver une fonction qui permet de modéliser le problème et qui prend ses valeurs dans l'intervalle $[0, 1]$, *i.e.* transformer les valeurs de notre modèle linéaire pour qu'elles se retrouvent dans l'intervalle $[0, 1]$.

De la régression vers la classification V



De la régression vers la classification VI

Ces valeurs comprises dans l'intervalle $[0, 1]$ peuvent s'interpréter comme des probabilités !
Ce que l'on va donc faire, c'est estimer la probabilité qu'un patient soit malade en fonction de sa numération en lymphocytes. Pour cela on va utiliser une transformation à l'aide d'une fonction logistique

$$\frac{1}{1 + \exp\left(-h(\mathbf{x})\right)}.$$

On parle alors de **Régression Logistique**

Régression Logistique

Régression Logistique I

Le modèle de Régression Logistique, également appelé *modèle logit* est un modèle qui date du 20ème siècle [Cox, 1958]. Ce sont des modèles spécifiquement dédiés au cas où la variable à prédire prend **une valeur discrète**.
Pour simplifier la présentation, on va se placer dans le cas où nous n'avons que deux classes à prédire, *i.e.* y ne prend que deux valeurs -1 et 1.

Idée Ce type de modèle peut également être vu comme un modèle linéaire mais ... qui ne cherche pas directement à prédire la valeur de y.
C'est un modèle qui va chercher à prédire la probabilité qu'un exemple appartienne a la classe positive, *i.e.* que $y = 1$.

Régression Logistique II

On va donc chercher un modèle qui va approximer la probabilité suivante

$$\eta = \mathbb{P}[Y = 1 \mid X]$$

Pour faire cela, on va chercher un modèle linéaire qui va approximer l'*odd ratio* entre la classe positive et la classe négative.

Principe Apprendre un modèle linéaire de la forme $\boldsymbol{\beta}^T \mathbf{x}$ tel que

$$\ln\left(\frac{\mathbb{P}[y = 1 \mid \mathbf{x}]}{\mathbb{P}[y = 0 \mid \mathbf{x}]}\right) = h(\mathbf{x}) = \boldsymbol{\beta}^T \mathbf{x},$$

où β sont donc les paramètres du modèle à apprendre. Une fois le modèle appris, on détermine la probabilité qu'une donnée appartienne à la classe positive en calculant :

Régression Logistique III

$$g(\mathbf{x}) = \mathbb{P}[y = 1 \mid \mathbf{x}] = \frac{1}{1 + \exp(-h(\mathbf{x}))}.$$

Est-ce que vous pourriez montrer que cette fonction n'est ni convexe, ni concave.

Régression Logistique IV

Apprentissage

Il est plus complexe que pour l'apprentissage d'un modèle linéaire multiple mais peut se faire de la même façon, c'est-à-dire par **maximum de vraisemblance**.

$$\mathfrak{L}(\mathbf{w}, b, S) = \prod_{i=1}^{m} \mathbb{P}(Y = y_i \mid X = x_i),$$

$$\downarrow \text{ separer } y_i = 0 \text{ and } y_i = 1$$

$$= \prod_{i=1, y_i=1}^{m} \mathbb{P}(Y = y_i \mid X = x_i) \times \prod_{i=1, y_i=0}^{m} \mathbb{P}(Y = y_i \mid X = x_i),$$

$$\downarrow \text{ Utiliser la loi de Bernoulli}$$

Régression Logistique V

$$= \prod_{i=1}^{m} \left(\frac{1}{1+\exp(-\boldsymbol{\beta}^T \mathbf{x})} \right)^{y_i} \times \left(\frac{1}{1+\exp(\boldsymbol{\beta}^T \mathbf{x})} \right)^{(1-y_i)} .$$

Plutôt que de chercher à maximiser la vraisemblance, on parfois chercher à minimiser l'opposé de la log-vraisemblance des données.

$$\begin{aligned} \ell(\boldsymbol{\beta}, S) = & -\ln\left(\mathfrak{L}(\boldsymbol{\beta}, S)\right), \\ = & -\sum_{i=1}^{m} y_i \ln\left(\frac{1}{1+\exp\left(-\boldsymbol{\beta}^T \mathbf{x}\right)}\right) \\ & + (1-y_i) \ln\left(1 - \frac{1}{1+\exp\left(-\boldsymbol{\beta}^T \mathbf{x}\right)}\right). \end{aligned}$$

Régression Logistique VI

Contrairement à ce que nous avons vu le pour le modèle linéaire simple, dans le cas présent, il n'existe pas de solution explicite du vecteur $\boldsymbol{\beta}$, nous devons obligatoirement trouver son expression en passant par un algorithme d'optimisation, comme une *descente de gradient*. En effet, notre problème d'optimisation reste un problème d'optimisation **convexe** (exercice).

On va faire cela, et on va plus précisément appliquer une méthode de **Newton-Raphson** pour résoudre ce problème

$$\boldsymbol{\beta}^{(k+1)} \leftarrow \boldsymbol{\beta}^{(k)} - \left(\nabla^2 \ell(\boldsymbol{\beta}^{(k)})\right)^{-1} \cdot \nabla \ell(\boldsymbol{\beta}^{(k)}).$$

Régression Logistique VII

On va donc commence par calculer le gradient de ℓ par rapport au vecteur $\boldsymbol{\beta}$, ce qui nous donne :

$$\begin{aligned}
\nabla \ell(\boldsymbol{\beta}, S) &= \begin{bmatrix} \frac{\partial \ell}{\partial \beta_1}(\boldsymbol{\beta}, S) \\ \frac{\partial \ell}{\partial \beta_2}(\boldsymbol{\beta}, S) \\ \vdots \\ \frac{\partial \ell}{\partial \beta_{d+1}}(\boldsymbol{\beta}, S) \end{bmatrix}, \\
&= -\sum_{i=1}^{m} -y_i(1 - g(\boldsymbol{\beta}, \mathbf{x}_i))\mathbf{x}_i + (1 - y_i)g(\boldsymbol{\beta}, \mathbf{x}_i)\mathbf{x}_i, \\
&= -\sum_{i=1}^{n} \left(y_i - \frac{1}{1 + \exp(-\langle \boldsymbol{\beta}, \mathbf{x}_i \rangle)} \right) \mathbf{x}_i.
\end{aligned}$$

Régression Logistique VIII

On pourrait s'arrêter là et appliquer notre descente de gradient "classique" à l'aide de cette dérivée

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} - \eta \, \nabla \ell(\boldsymbol{\beta}^{(k)}, S),$$

$k = 1, 2, \ldots$ and η is the learning rate.

La plupart du temps, on préfère employer la descente de **Newton-Raphson**, cela nécessite de déterminer la matrice hessienne de ℓ, *i.e.* il va falloir déterminer la dérivée seconde de notre objectif

Régression Logistique IX

$$\nabla^2 \ell(\boldsymbol{\beta}, S) = \begin{bmatrix} \dfrac{\partial^2 \ell}{\partial \beta_1^2}(\boldsymbol{\beta}, S) & \cdots & \dfrac{\partial^2 \ell}{\partial \beta_1 \partial \beta_{d+1}}(\boldsymbol{\beta}, S) \\ \dfrac{\partial^2 \ell}{\partial \beta_2 \partial \beta_1}(\boldsymbol{\beta}, S) & \cdots & \dfrac{\partial^2 \ell}{\partial \beta_1 \partial \beta_{d+1}}(\boldsymbol{\beta}, S) \\ \vdots & & \vdots \\ \dfrac{\partial^2 \ell}{\partial \boldsymbol{\beta}_{d+1} \partial \beta_1}(\boldsymbol{\beta}, S) & \cdots & \dfrac{\partial^2 \ell}{\partial \beta_{d+1}^2}(\boldsymbol{\beta}, S) \end{bmatrix},$$

$$= \sum_{i=1}^{m} g(\boldsymbol{\beta}, \mathbf{x}_i) \left(1 - g(\boldsymbol{\beta}, \mathbf{x}_i)\right) \mathbf{x}_i \mathbf{x}_i^T.$$

Régression Logistique X

On peut également employer une formulation beaucoup plus compacte

$$\nabla^2 \ell(\boldsymbol{\beta}, \mathbf{X}) = \mathbf{X}^T \mathbf{G} \mathbf{X},$$

où $\mathbf{X} \in \mathbb{R}^{m \times (d+1)}$ est la matrice de design,(*i.e.* la matrice des données) et $\mathbf{G} \in \mathbb{R}^{m \times m}$ est la matrice définie par :

$$\mathsf{G} = \begin{bmatrix} g(\boldsymbol{\beta}, \mathbf{x}_1)\,(1 - g(\boldsymbol{\beta}, \mathbf{x}_1)) & 0 & \cdots & \cdots & 0 \\ 0 & g(\boldsymbol{\beta}, \mathbf{x}_2)\,(1 - g(\boldsymbol{\beta}, \mathbf{x}_2)) & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ & & & & 0 \\ 0 & \cdots & \cdots & 0 & g(\boldsymbol{\beta}, \mathbf{x}_m)\,(1 - g(\boldsymbol{\beta}, \mathbf{x}_m)) \end{bmatrix}.$$

Régression Logistique XI

Règle de prédiction

Une fois que l'on dispose de notre modèle, nous sommes donc capable d'évaluer la probabilité qu'une nouvelle donnée $\mathbf{x}$ appartienne à la classe positive.
Pour prédire son étiquette, on utilise couramment la valeur 0.5 comme valeur critique ou comme valeur seuil (*threshold* en anglais) de telle sorte que :

$$y = \begin{cases} 1 & \text{si } \mathbb{P}[Y \mid X = \mathbf{x}] > 0.5, \\ 0 & \text{sinon.} \end{cases}$$

Mais c'est bien évidemment loin d'être la seule règle et il est parfois intéressant de bouger ce seuil dans certains contextes de classification

Quelques mots sur la régularisation I

Regularisation

Pour éviter les problèmes de sur-apprentissage, il est courant d'ajouter un terme régularisation

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^{d+1}} \frac{1}{m}\sum_{i=1}^{m} \ell(\boldsymbol{\beta},\boldsymbol{\xi}) + \lambda\|\boldsymbol{\beta}\|,$$

où ℓ est la loss précédemment introduite.

Dans le modèle linéaire gaussien, nous aurions

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \|\mathbf{y} - h(\boldsymbol{\theta},\mathbf{X})\|_2^2 + \lambda\|\boldsymbol{\theta}\|^2 = \min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \sum_{i=1}^{m} (y_i - h(\boldsymbol{\theta},\mathbf{x}_i))^2 + \lambda\|\boldsymbol{\theta}\|^2$$

Quelques mots sur la régularisation II

Deux régularisation classiques

- La norme L_1 est notamment utilisée pour induire de la sparsité dans l'hypothèse apprise, c'est-à-dire quand on veut que l'hypothèse apprise dépende d'un minimum de paramètres/variables. Lorsque nous utilisons ce terme de régularisation, nous parlons de *Régression Lasso*. Ce type de régularisation est particulièrement utilisé dans les modèles à haute dimension, lorsque les variables sont nombreuses, comme cela peut être le cas en génétique.

- La norme L_2 permet d'éviter que certains paramètres prennent un poids trop important dans la décision. On parle, dans ce cas, de *Régression Ridge*.

Les Séparateurs à Vaste Marge (SVM)

A propos des SVM I

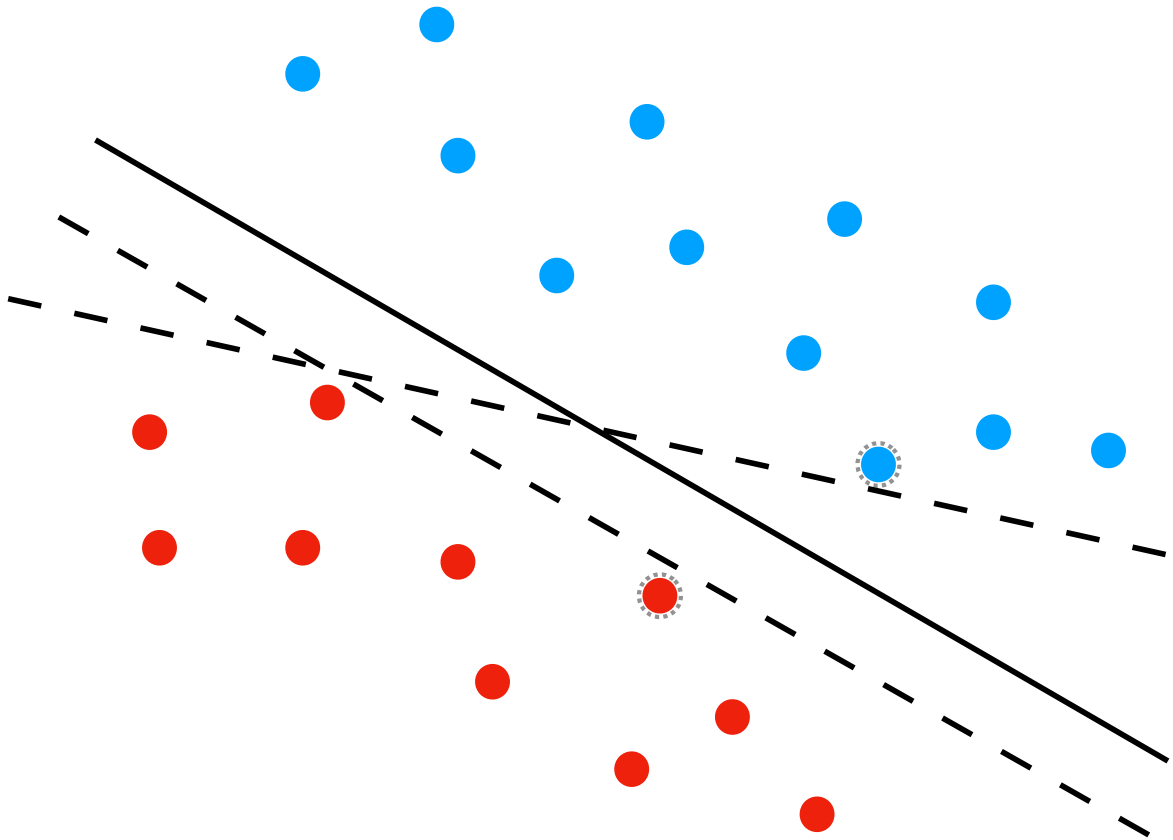
Certainement l'un des algorithmes les plus répandues en apprentissage machine [Vapnik and Cortes, 1995] pour effectuer des tâches de classification binaires.

Idée : apprendre une hypothèse h dont le but est de prédire l'étiquette d'une donnée. Elle se présente sous la forme d'un hyperplan (affine) qui va séparer l'espace en deux : $\{-1, +1\}$:

$$h(\mathbf{x}) = sign[\langle \mathbf{w}, \mathbf{x} \rangle + b] = \begin{cases} -1 & \text{if} \quad \langle \mathbf{w}, \mathbf{x} \rangle + b < 0, \\ +1 & \text{if} \quad \langle \mathbf{w}, \mathbf{x} \rangle + b > 0. \end{cases}$$

Problème : plusieurs plans peuvent êtres solutions et séparer parfaitement nos données.

A propos des SVM II



Quel séparateur choisir et pourquoi ?

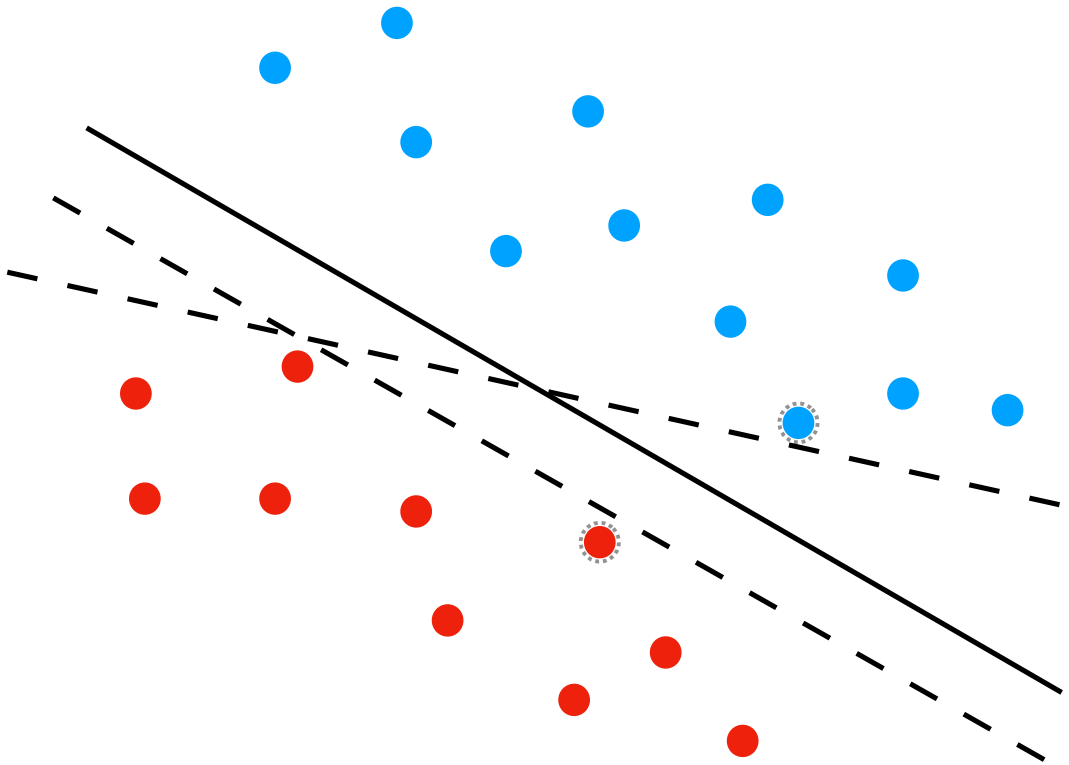
A propos des SVM III

L'idée présentée par [Boser et al., 1992] est de choisir le modèle qui se trouve à la plus grande distance entre les deux classes.

Mais pourquoi choisir un tel modèle ?

Il suffit de se rappeler de notre objectif en Machine Learning, *on souhaite avoir un modèle qui est performant et avec de bonnes capacités en généralisation*

A propos des SVM IV



Cela est vérifié pour la droite en trait plein, mais pour celles en trait pointillé. On retient un modèle qui présente **la plus grande marge**.

A propos des SVM V

Comment est-ce que l'on peut définir cette notion de marge ? Nous avons dit que c'est la distance qui sépare les exemples de classe opposée.

Si on reprend les équations de notre SVM, on pourra définir la marge comme la distance entre les deux droites d'équations :

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = \pm \rho.$$

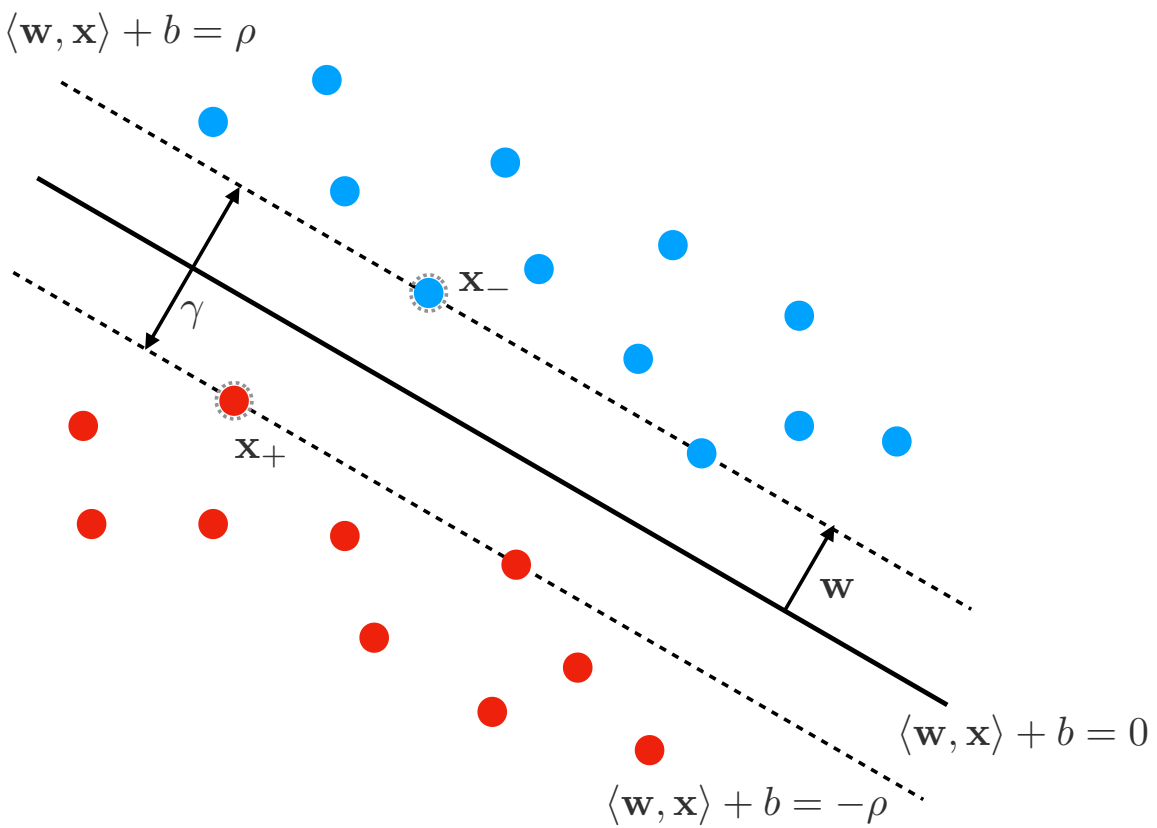
Ce que l'on peut également réécrire

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = \pm 1,$$

en normalisant les paramètres.

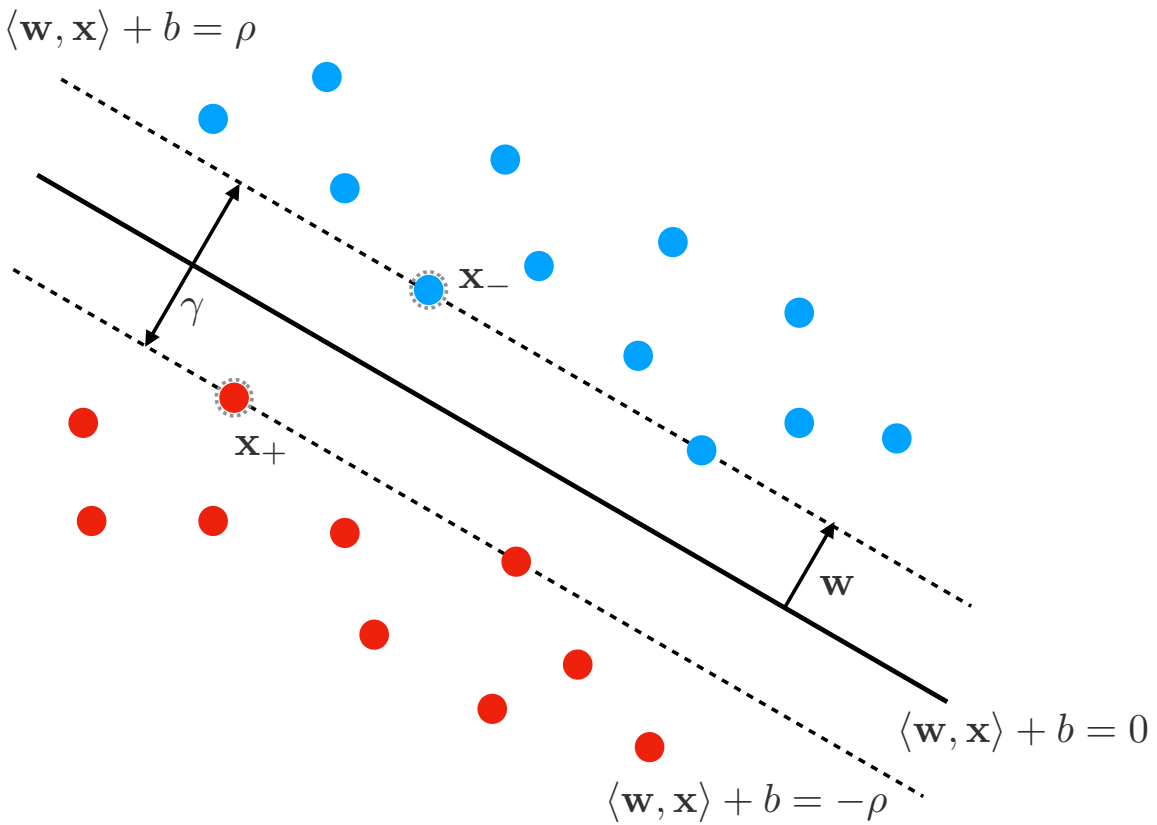
A propos des SVM VI

On considère maintenant deux points $\mathbf{x}_-$ et $\mathbf{x}_+$ qui se trouvent sur les deux droites (ou plus généralement hyperplans) représentés ci-dessous



A propos des SVM VII

Sur ce même dessin, on définit également la marge γ de notre séparateur linéaire.



A propos des SVM VIII

Chacun des vecteurs $\mathbf{x}_+$ et $\mathbf{x}_-$ peuvent se décomposer comme la somme d'un vecteur qui se *colinéaire* à $\mathbf{w}$ (noté $\mathbf{x}^{\mathbf{w}}$) et d'un vecteur qui sera orthogonal à $\mathbf{w}^{\mathbf{w}^\perp}$.

Ce qui nous amène aux relations suivantes :

$$\begin{aligned}
h(\mathbf{x}_+) - h(\mathbf{x}_-) &= 2, \\
\langle \mathbf{w}, \mathbf{x}_+ \rangle + b - (\langle \mathbf{w}, \mathbf{x}_- \rangle + b) &= 2, \\
\langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}} \rangle + \underbrace{\langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}^\perp} \rangle}_{=0} + b - (\langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}} \rangle + \underbrace{\langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}^\perp} \rangle}_{=0} + b) &= 2, \\
\langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}} \rangle - \langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}} \rangle &= 2. \qquad (1)
\end{aligned}$$

A propos des SVM IX

De plus, la marge γ, *i.e.* la distance entre nos deux hyperplans, est définie comme le coefficient de la projection du vecteur $\mathbf{x}_+ - \mathbf{x}_-$ sur le vecteur unitaire $\frac{\mathbf{w}}{\|\mathbf{w}\|_2}$, *i.e.*

$$\gamma = \frac{\langle \mathbf{x}_+ - \mathbf{x}_-, \mathbf{w} \rangle}{\|\mathbf{w}\|_2}.$$

On va maintenant réécrire ce produit scalaire et réutiliser la définition de nos deux hyperplans.

A propos des SVM X

Ainsi, nous avons :

$$\gamma = \frac{\langle \mathbf{x}_+ - \mathbf{x}_-, \mathbf{w} \rangle}{\|\mathbf{w}\|_2},$$

$$= \frac{1}{\|\mathbf{w}\|_2} \underbrace{\langle \mathbf{x}_+ - \mathbf{x}_-, \mathbf{w} \rangle}_{},$$

$\downarrow$ en développant et en conservant uniquement la partie colinéaire

$$= \frac{1}{\|\mathbf{w}\|} \underbrace{\langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}} \rangle - \langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}} \rangle}_{},$$

$\downarrow$ en utilisant l'équation (1)

$$\gamma = \frac{2}{\|\mathbf{w}\|_2}.$$

A propos des SVM XI

Ainsi, maximiser notre marge γ revient a minimiser la quantité $\frac{\|\mathbf{w}\|_2}{2}$ ou encore $\frac{\|\mathbf{w}\|_2^2}{2}$.

L'ajout d'un carré sur la norme est plus pour des raisons "pratiques" et permet de s'affranchir de la racine carré dans la résolution du problème d'optimisation.

Le problème de minimisation associé est appelé *hard margin SVM* et est défini par :

A propos des SVM XII

Définition 1.3: Hard Margin SVM

Soit $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ un ensemble de m exemples (individus). Alors le meilleure *Séparateur à Vaste Marge* que l'on puisse obtenir est la solution du problème d'optimisation suivant :

$$\begin{aligned} \min_{(\mathbf{w},b)\in\mathbb{R}^{d+1}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ s.t. \quad & y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1, \quad \text{for all } i = 1, ..., m. \end{aligned}$$

où la contrainte d'inégalité n'est rien d'autre qu'une écriture "synthétique" de celle présentée en équation (46).

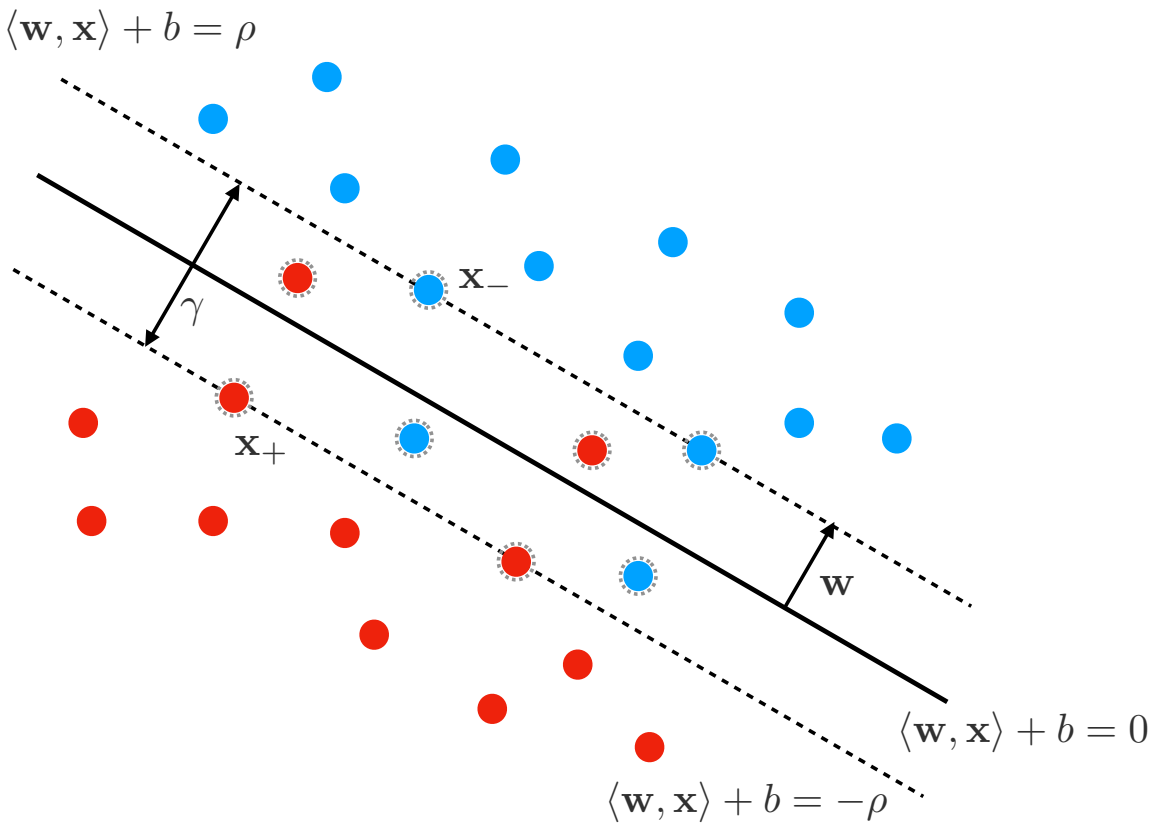
A propos des SVM XIII

Ce problème peut facilement être résolu par des procédés classiques et cela sans grande difficultées.

On pourra par exemple regarder les packages **e1071** de  ou encore la librairire **libsvm** sous Python.

Mais le problème présenté ici est un cas idyllique où l'on a supposé que nos données sont linéairement séparables, mais cela n'arrive jamais en pratique.

A propos des SVM XIV



A propos des SVM XV

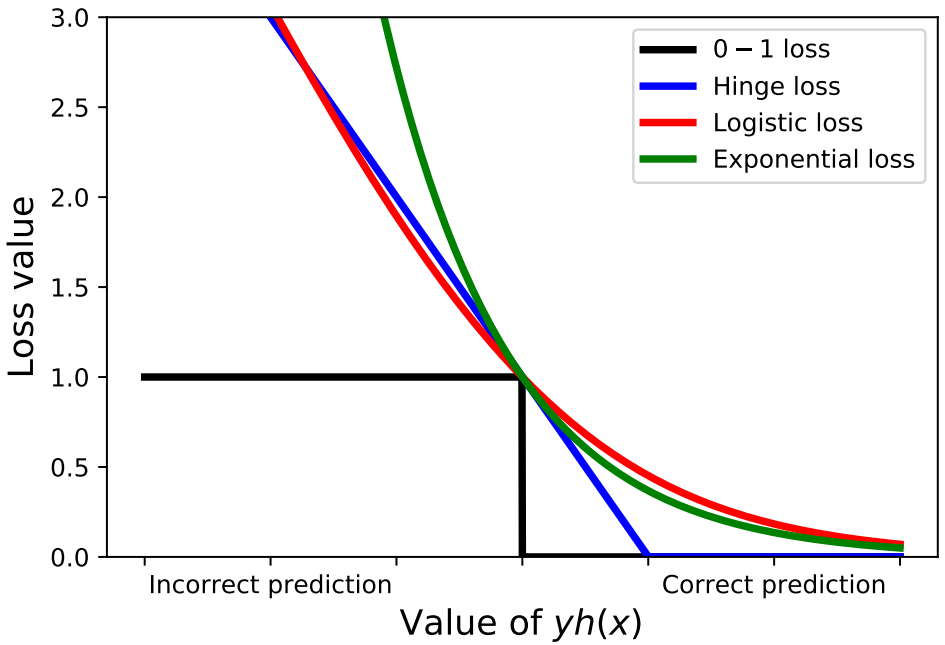
On voit que certains points violent la contrainte

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1.$$

Ce qui peut signifier deux choses :

- votre point se trouve du mauvais côté de l'hyperplan.
- votre point se trouve du bon côté de l'hyperplan mais à l'intérieur de la marge.

A propos des SVM XVI



A propos des SVM XVII

Ainsi, dans cette situation, le problème d'optimisation que nous avions précédemment présenté n'admettrait pas de solutions car il est impossible de mettre tous les points du bon côté

$$\begin{array}{ll} \min\limits_{(\mathbf{w},b)\in\mathbb{R}^{d+1}} & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ s.t. & y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1, \quad \text{for all } i = 1, ..., m. \end{array}$$

On doit donc **relaxer** ou **relâcher** notre problème de façon à lui autoriser à faire des erreurs.

A propos des SVM XVIII

Ces dernières vont prendre la forme de variables dites *slacks* $\boldsymbol{\xi} = (\xi_1, ..., \xi_m)$ que l'on va inclure dans le problème d'optimisation. Ces variables prennent des valeurs positives si les contraintes ne sont pas respectées et nulle dans le cas contraire.

Ces variables vont directement injectées dans le problème d'optimisation de façon à autoriser assouplir la contrainte et l'objectif sera alors de trouver un **compromis** entre la *maximisation de la marge du modèle* et la *minimisation du taux d'erreur*.

A propos des SVM XIX

Définition 1.4: Soft Margin SVM

Soit $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ un échantillon de taille m et notons $\boldsymbol{\xi} = (\xi_1, ..., \xi_m)$ le vecteur des variables dites *slack*. Alors le meilleur séparateur linéaire que l'on puisse trouver au sens de l'algorithme SVM est solution du problème d'optimisation suivant :

$$\begin{aligned} \min_{\xi \in \mathbb{R}^m,\ (\mathbf{w},b)\in\mathbb{R}^{d+1}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{m}\sum_{i=1}^m \xi_i \\ s.t. \quad & y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b) \geq 1 - \xi_i, \quad \text{for all } i = 1, ..., m, \\ & \xi_i \geq 0, \quad \text{for all } i = 1, ..., m. \end{aligned} \tag{2}$$

A propos des SVM XX

Comparée à la Définition 1.3, un terme en $\frac{C}{m}\sum_{i=1}^{m}\xi_i$ a été ajouté et il représente le coût de violation des contraintes.

Le juste équilibre entre le contrôle de l'erreur et la maximisation de la marge va se faire avec un hyper-paramètres C qui devra alors être tuné lors du processus d'apprentissage.

Travailler avec cette formulation peut se révéler cependant difficile, et il n'est pas rare de réécrire le problème.

A propos des SVM XXI

Proposition 1.1: Formulation équivalente du Soft Margin SVM

Soit $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ un ensemble de données de taille m et soit $\boldsymbol{\xi} = (\xi_1, ..., \xi_m)$ le vecteur des variables slacks. Si les contraintes sont prises en compte et directement injectées dans le problème d'optimisation, alors le problème 2 peut se réécrire :

$$\min_{(\mathbf{w},b)\in\mathbb{R}^{d+1}} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{m}\sum_{i=1}^{m}\left[1 - y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b)\right]_+ ,$$

où $[x]_+ = \max(0, x)$, *i.e.* représente la hinge loss.

A propos des SVM XXII

Démonstration

Pour montrer ce résultat, on a uniquement besoin de se focaliser sur les variables *slacks*, qui, rappelons le, sont des variables positives d'après la deuxième contrainte de notre définition des *Soft SVM*.

Si on se concentre sur la première contrainte, on remarque ξ_i est nulle si et seulement si $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$, sinon elle est égale à la différence, *i.e.*

$$\forall\, i = 1, \ldots, m,\ \xi_i = \begin{cases} 0 & \text{si } 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \leq 0, \\ 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) & \text{sinon.} \end{cases}$$

Ce que l'on peut synthétiser par

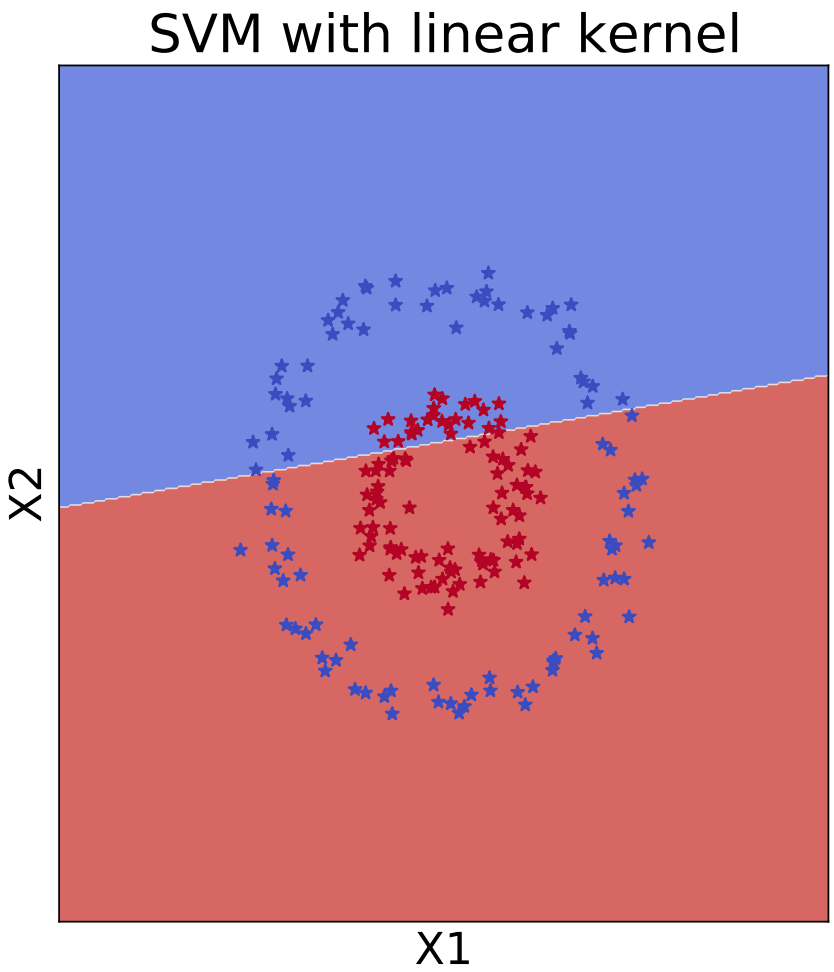
$$\forall\, i = 1, \ldots, m,\ \xi_i = [1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle) + b]_+$$

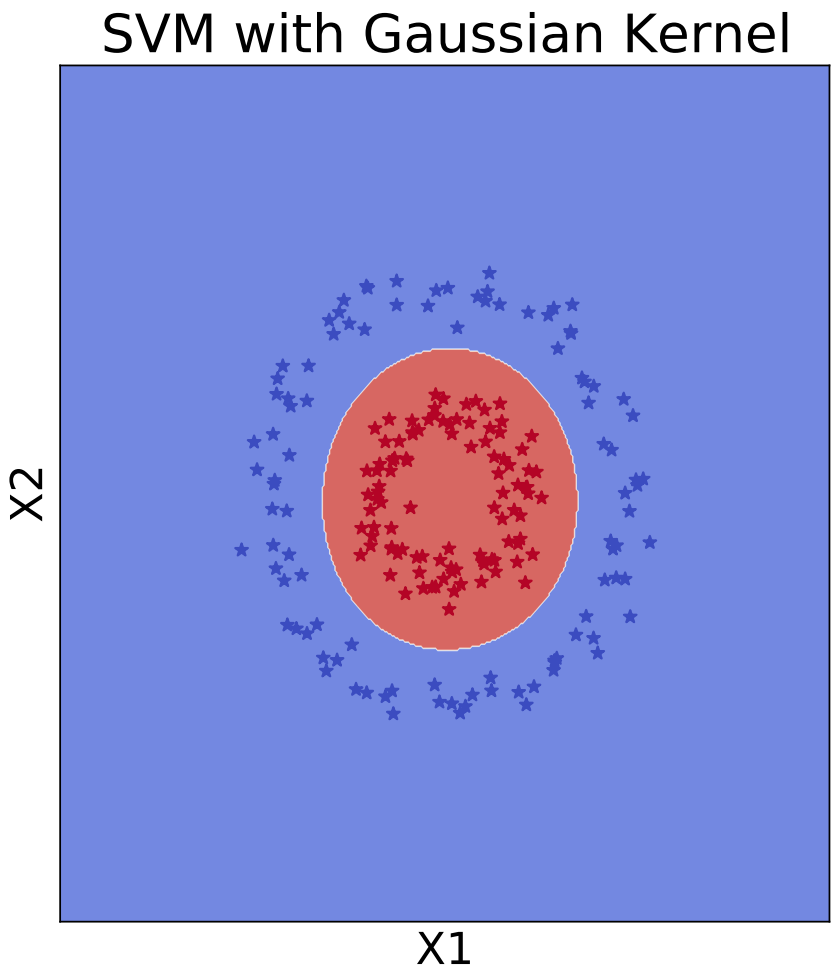
A propos des SVM XXIII

Bien que ce problème soit convexe, il reste très compliqué à optimiser car la fonction dont on cherche le minimum n'est pas lisse.

De plus, la résolution du problème est d'autant plus complexe que nos données possèdent de descripteurs. Cette complexité est en $\mathcal{O}(d^3)$, ce qui En outre, cela reste un séparateur qui est **linéaire**, son expressivité ou son **pouvoir discriminant** reste donc limité quand notre jeu de données est plus complexe.

A propos des SVM XXIV





SVM non linéaire I

L'objectif serait d'apprendre un séparateur qui *a priori* n'est plus forcément linéaire qui serait linéaire mais dans un autre espace ...

Dit autrement, il faudrait être en mesure de trouver une transformation des données qui permette de pouvoir résoudre le problème avec un algorithme de séparation linéaire.

Pour cela on a besoin de considérer le problème "dual" de notre problème d'optimisation initial.

SVM non linéaire II

Proposition 1.2: Problème Dual SVM

Soit $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ un échantillon de m exemples. La formulation duale du problème de Soft Margin SVM présenté en Définition 1.4 est :

$$\begin{array}{ll} \max\limits_{\boldsymbol{\alpha}} & -\frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \sum_{i=1}^m \alpha_i, \\ s.t. & 0 \leq \alpha_i \leq \dfrac{C}{m} \qquad \text{for all } i = 1, ..., m, \\ & \sum_{i=1}^m y_i \alpha_i = 0. \end{array} \tag{3}$$

Le vecteur $\boldsymbol{\alpha}$ est le vecteur des variables lagrangiennes (ou duales).

SVM non linéaire III

Démonstration

On repart de la définition de départ, afin d'obtenir la version duale de ce problème, nous allons considérer une seule fonction objectif. Cette dernière va regrouper à la fois la quantité que l'on va initialement minimiser mais on y ajoute les contrainte.

Cette nouvelle fonction s'appelle le Lagrangien

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{m} \alpha_i \left(1 - \xi_i - y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right)\right) - \sum_{i=1}^{m} \beta_i \xi_i.$$

ou $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ sont les varaiables duales associées aux deux contraintes du problème (primal).

SVM non linéaire IV

On rappelle que si le problème primal est convexe (de même que les contraites) alors les solutions du problème vérifient les égalités suivantes :

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0, \quad \frac{\partial \mathcal{L}}{\partial b}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0,$$

et

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0.$$

Elles vont nous permettre de trouver uen expression des variables primales (*i.e.* $\mathbf{w}$, b et $\boldsymbol{\xi}$) comme une fonction des variables duales (*i.e.* $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$). C'est ce que l'on appelle les conditions de KKT (Karush-Kuhn-Tucker)

SVM non linéaire V

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0, \quad \Longleftrightarrow \quad \mathbf{w} - \sum_{i=1}^{m} y_i \alpha_i \mathbf{x}_i = 0 \iff \mathbf{w} = \sum_{i=1}^{m} y_i \alpha_i \mathbf{x}_i, \tag{4}$$

$$\frac{\partial \mathcal{L}}{\partial b}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0, \quad \Longleftrightarrow \quad \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\xi}_i}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0 \quad \Longleftrightarrow \quad \frac{C}{m} - \alpha_i - \beta_i = 0, \iff \alpha_i, \beta_i \geq 0. \tag{6}$$

On utilise enfin ces résultats et on injecte tout dans l'expression du Lagrangien

SVM non linéaire VI

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta})$$

$$= \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{m} \alpha_i \left(1 - \xi_i - y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right)\right) - \sum_{i=1}^{m} \beta_i \xi_i,$$

↓ using Equation (4)

$$= \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \sum_{i=1}^{m} \xi_i \left(\frac{C}{m} - \alpha_i - \beta_i\right) + \sum_{i=1}^{m} \alpha_i$$

$$- \sum_{i=1}^{m} \alpha_i y_i \langle \sum_{j=1}^{m} y_j \alpha_j \mathbf{x}_j, \mathbf{x}_i \rangle + b \sum_{i=1}^{n} \alpha_i y_i,$$

↓ using Equations (5) and (6)

SVM non linéaire VII

$$= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \sum_{i=1}^{m} \xi_i \times 0 + \sum_{i=1}^{m} \alpha_i$$
$$- \sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_j, \mathbf{x}_i \rangle + b \times 0,$$

$\downarrow$ computing the difference

$$= -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle + \sum_{i=1}^{m} \alpha_i.$$

On doit maximiser le Lagragien par rapport aux variables duales. Enfin les équations(5) and (6) conduisent aux deux contraintes du problème dual.

SVM non linéaire VIII

Garder à l'esprit que le problème dual est un problème strictement concave par rapport aux variables duales. Ainsi, il n'admet qu'une seule et unique solution.

On peut réécrire le problème précédent un peu plus simplement en définissant une matrice dite de *Gram*.

$$\max_{\boldsymbol{\alpha}} \; -\frac{1}{2}\boldsymbol{\alpha}^T \mathbf{G}\boldsymbol{\alpha} + \sum_{i=1}^{m} \alpha_i,$$

où $\mathbf{G}$ est la matrice définie par $G_{ij} = y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$. Elle est semi-définie positive, donc notre problème va admettre une seule et unique solution.

SVM non linéaire IX

Remarques

- La résolution du problème dual conduit à trouver une solution dans un espace différent : *espace des individus au lieu de l'espace des variables*.
- La solution fournie par le problème dual est la même que celle donnée par le problème primal ! (on parle de dualité forte)
- Si on connaît la solution du problème dual, il est possible de retrouver les valeurs de $\mathbf{w}$ et b correspondantes à l'aide des relation de KKT.

Cette approche sera intéressante intéressante lorsque l'on dispose d'un nombre relativement raisonnable de données par rapport au nombre de variables. Mais cela ne résout pas le problème des données non linéairement séparables. On va devoir employer ce que l'on appel **l'astuce du noyau**.

SVM non linéaire X

Au lieu d'utiliser le produit scalaire standard entre deux exemples, on va définir une fonction $K(\cdot, \cdot)$ qui prendra en entrée deux vecteurs et qui va retourner une valeur réelle.

Une telle fonction s'appelle un noyau et la matrice $\mathbf{K}$ associée doit vérifier les propriétés suivantes : (i) symétrique et (ii) semi-définie positive, *i.e.*

(i) $\forall\, (\mathbf{x}, \mathbf{x}') \in \mathbb{R}^d \times \mathbb{R}^d$, nous avons : $K(\mathbf{x}, \mathbf{x}') = K(\mathbf{x}', \mathbf{x})$,

(ii) $\forall\, (\mathbf{x}_i, \mathbf{x}_j) \in \mathbb{R}^d \times \mathbb{R}^d$ et $\forall\, \mathbf{c} \in \mathbb{R}^d$, nous avons :
$c^T \mathbf{K} c = \sum_{i=1}^{m} \sum_{j=1}^{m} c_i c_j K(\mathbf{x}_i, \mathbf{x}_j) \geq 0.$

SVM non linéaire XI

Ces propriétés de la matrice $\mathbf{K}$ (ou de la fonction K) sont importantes et conduisent au résultat suivant [Mercer, 1909].

Théorème 1.1: Theorème de Mercer

Soit $\mathcal{X}$ un compact de $\mathbb{R}^d$ et soit K une forme bilinéaire symétrique semi-définie positive, i.e. un noyau. Alors il existe une base orthogonale $(\Phi_j)_{j\in\mathbb{N}}$ et une suite $(\lambda_j)_{j\in\mathbb{N}}$, où $\lambda_j \geq 0$ pour tout j, telle que :

$$K(\mathbf{x},\mathbf{x}') = \sum_{j=1}^{\infty} \lambda_j \Phi_j(\mathbf{x})\Phi_j(\mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle,$$

où $\Phi(\mathbf{x}) = (\sqrt{\lambda_1}\Phi_1(\mathbf{x}), ..., \sqrt{\lambda_j}\Phi_j(\mathbf{x}), ...)$ est la représentation du vecteur $\mathbf{x}$ dans un nouvel espace.

SVM non linéaire XII

En introduisant K, dans la formulation duale 3, on obtient une formulation généralisée du problème :

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j) + \sum_{i=1}^{m}\alpha_i, \\ s.t. \quad & 0 \leq \alpha_i \leq \frac{C}{m}, \qquad \text{for all } i = 1, ..., m, \\ & \sum_{i=1}^{m}\alpha_i = 0, \end{aligned}$$

Il existe une grande série de noyaux que l'on pourrait employer pour nos tâches de classification (ou même d'estimation de densité),
voir [Genton, 2002] pour une liste non exhaustive de tels noyaux.

SVM non linéaire XIII

Exemples de noyaux

- **Noyau linéaire :** $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$, c'est le noyau le plus classique mais aux capacités limitées.
- **Noyau polynomial :** $K(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^p$, où c est une constante
- **Noyau Gaussien :** $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\sigma^2}\right)$, où σ est un hyper-paramètre que l'on doit tuner.
  Il contrôle l'importance que l'on va donner à la similarité entre deux exemples. Plus cette valeur est grande, moins on accorde d'importance à la distance entre les points, on aura tendance à lisser la frontière de décision.

SVM non linéaire XIV

Quelques exemples

On considère deux vecteurs $\mathbf{x}$ et $\mathbf{z}$ de $\mathbb{R}^2$ et noyau polynomial de degré 2.
On va essayer de réécrire ce noyau comme un produit scalaire entre deux vecteurs *i.e.* comme une fonction Φ, telle que $K(\mathbf{x}, \mathbf{z}) = \Phi(\mathbf{x})^T \Phi(\mathbf{z})$.

$$
\begin{aligned}
K(\mathbf{x}, \mathbf{z}) &= (x_1 z_1 + x_2 z_2 + c)^2, \\
&= c^2 + x_1^2 z_1^2 + x_2^2 z_2^2 + 2c x_1 z_1 + 2c x_2 z_2 + 2 x_1 z_1 x_2 z_2, \\
&= \left(c, x_1^2, \sqrt{2c} x_1, \sqrt{2c} x_2, \sqrt{2c} x_1 x_2\right)^T \left(c, z_1^2, \sqrt{2c} z_1, \sqrt{2c} z_2, \sqrt{2c} z_1 z_2\right) \\
&= \Phi(\mathbf{x})^T \Phi(\mathbf{z}),
\end{aligned}
$$

où $\Phi : \mathbf{x} \mapsto \left(c, x_1^2, \sqrt{2c} x_1, \sqrt{2c} x_2, \sqrt{2c} x_1 x_2\right)$.
En utilisant ce noyau, on projette implicitement nos données dans un espace de dimension 5.

SVM non linéaire XV

Phase de prédiction

Avec le SVM linéaire (hard or soft) dans sa version primale, la prédiction du label d'un nouvel exemple $\mathbf{x}'$ se fait via le calcul suivant :

$$h(\mathbf{x}') = \text{sign}\left(\langle \mathbf{w}, \mathbf{x}' \rangle + b\right).$$

Si on emploie a version dite "kernelisée", *i.e.* en passant par la résolution du problème duale, notre prédicteur va prendra la forme suivante :

$$h(\mathbf{x}') = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}', \mathbf{x}_i)\right).$$

Noter que dans ce dernier cas, il est nécessaire de calculer la similarité du nouveau point à l'ensemble des points d'apprentissage !

Les réseaux de Neurones

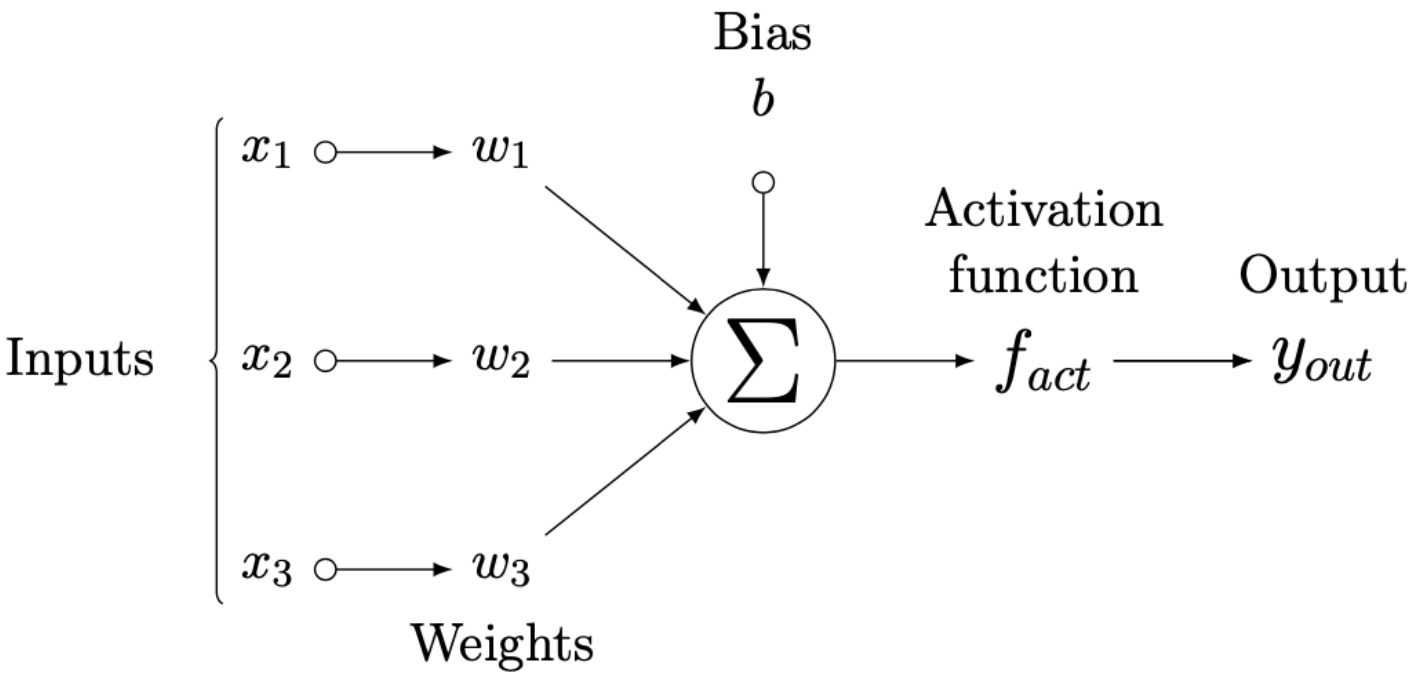
Réseaux de Neurones I

Cet algorithme est librement inspiré des neurosciences et peut se voir comme une traduction des modèles synaptiques (imaginer un neurone avec son axone).

Les premières traces de ces modèles peuvent se trouvent dans des travaux qui ont été publiés au milieu du 20ème siècle et non pas par des informaticiens, mais plutôt par des biologistes/ chercheurs en neurosciences [McCulloch and Pitts, 1943].

On va ici s'intéresser à un premier modèle (mathématique) de réseau de neurones qui est connu sous le nom de perceptron [Rosenblatt, 1958],

Réseaux de Neurones II



Réseaux de Neurones III

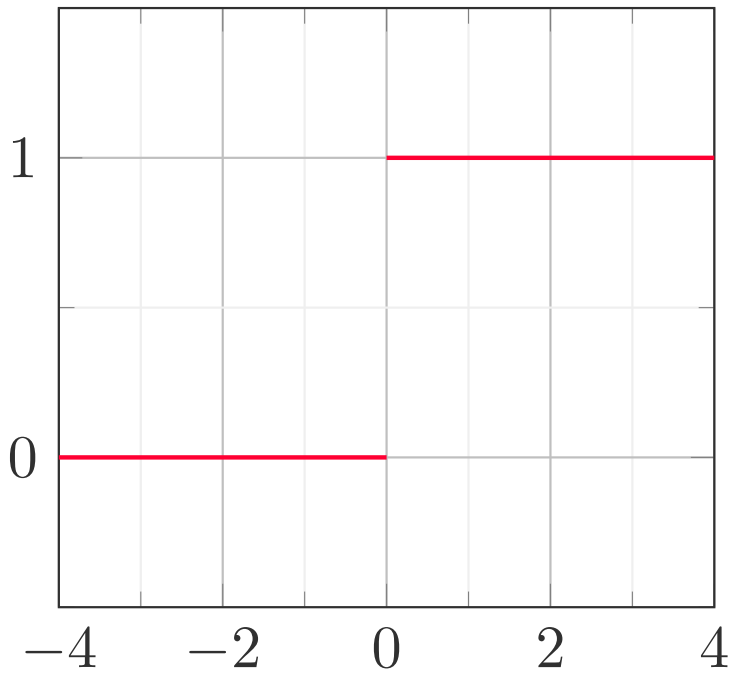
Ces modèles prennent en entrée un vecteur $\mathbf{x} \in \mathbb{R}^d$ et ils dépendent d'un paramètre $(\mathbf{w}, b) \in \mathbb{R}^{d+1}$ (noter que cela est très proche des SVM !)
L'output $h(\mathbf{x})$ est historiquement calculé comme étant la valeur du produit scalaire entre $\mathbf{x}$ et $\mathbf{w}$ à laquelle on ajoute b, *i.e.*

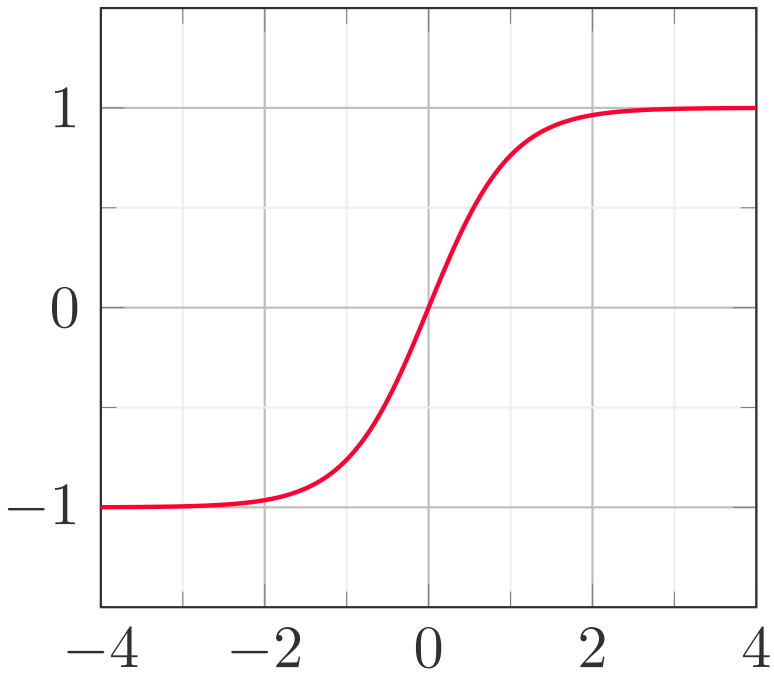
$$h(\mathbf{x}) = sign\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) = sign\left(\sum_{j=1}^{d} w_j x_j + b\right).$$

Ce premier modèle était donc destiné à faire de la classification binaire et il est dit *affine*. Une *fonction d'activation* ici la fonction *sign* est appliquée afin de retourner l'étiquette de la donnée.

$$h(\mathbf{x}) = \begin{cases} 0 & \text{si} \langle \mathbf{w}, \mathbf{x} \rangle + b < 0, \\ 1 & \text{sinon.} \end{cases}$$

Réseaux de Neurones IV





Le premier algorithme d'apprentissage des rédeaux de neurones (*i.e.* utilisé pour apprendre les paramètres $\mathbf{w}$ et b) est connu sous le nom de *Hebb algorithm*. Un algorithme simple mais qui converge uniquement dans le cas où les données sont linéairement séparables.

Réseaux de Neurones V

Règle de mise à jour des paramètres : soit $\mathcal{I}$ l'ensemble des exemples mal classés Alors, pour tout $(\mathbf{x}_i, y_i)$ tel que $i \in I$, on calcule

$$\mathbf{w} = \mathbf{w} + \alpha y_i \mathbf{x}_i \quad \text{and} \quad b = b + \alpha y_i,$$

où α est un pas d'apprentissage.
Une autre règle employée et est la *loi de Widrow-Hoff*, qui fonctionne de façon semblable mais qui prend aussi en compte l'erreur à l'état courant :

$$\mathbf{w} = \mathbf{w} + (y_i - h(\mathbf{x}_i))\mathbf{x}_i \quad \text{and} \quad b = b + y_i - h(\mathbf{x}_i).$$

Réseaux de Neurones VI

Cette seconde mise à jour pourrait se faire avec la même fonction d'activation heavyside.

Il n'est cependant pas rare d'utiliser des fonctions plus lisses pour effectuer ces mises à jour comme la fonction $tanh$ ou encore toute autre forme de sigmoïdes.

Ces dernières ont l'avantage d'être beaucoup plus simples à manipuler pour des algorithmes de descente de gradient (qui sont à la base de l'apprentissage des réseaux de neurones).

Réseaux de Neurones VII

On rencontre à nouveau le même problème que précédemment ... cette version très simple ne semble être en mesure de ne résoudre que des problèmes qui sont linéaires.

Par exemple, en logique, ce perceptron est connu pour résoudre très facilement le problème **OR** ou encore **AND** mais un simple perceptron n'est en revanche pas capavle de résoudre le problème **XOR**.

Regardons ces problème là dans un espace à deux dimensions

Réseaux de Neurones VIII

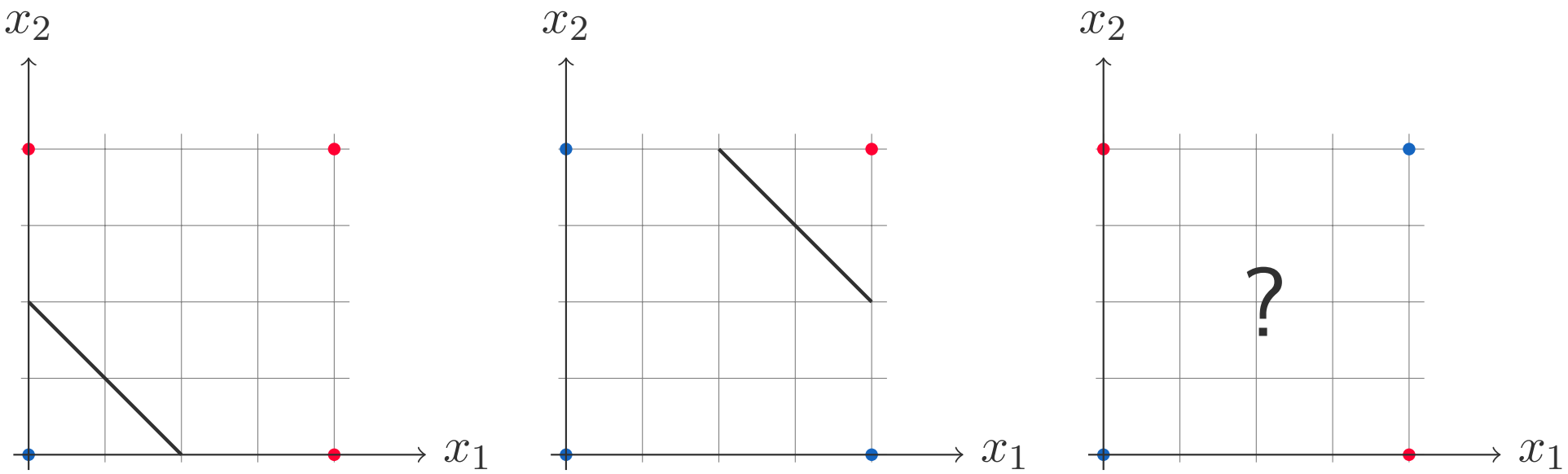


Figure – Représentation, dans un espace bi-dimensionnel, des problèmes **OR**, **AND** et **XOR**

Réseaux de Neurones IX

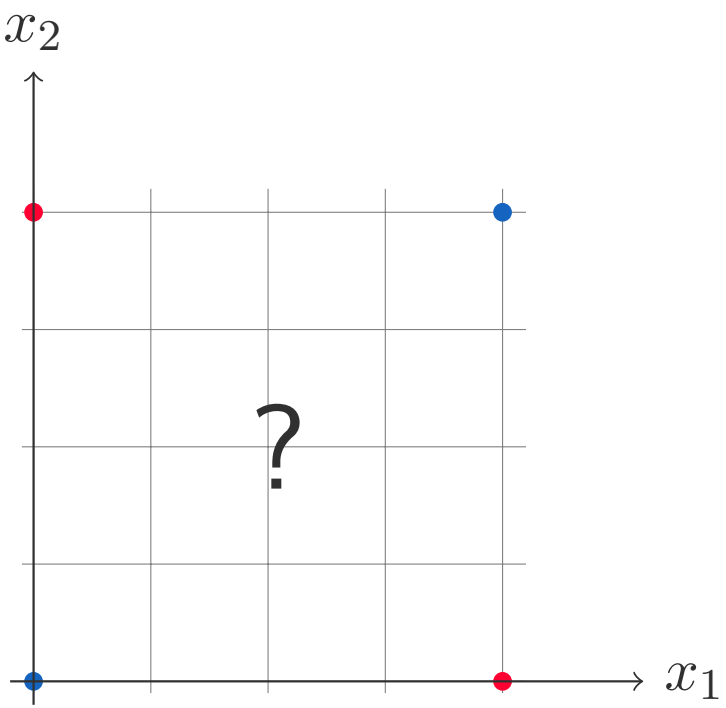
Pour résoudre ce problème, on garde notre inspiration des neurosciences et de la structure d'un cerveau par exemple, ou plusieurs neurones sont connectés entre eux.
On obtient alors ce que l'on appelle un **réseaux de neurones**.

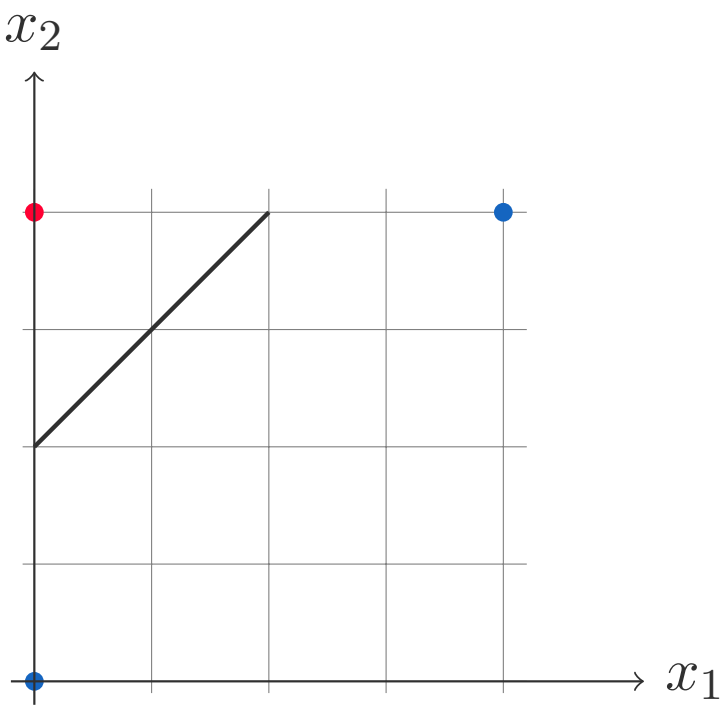
Mais avant de fournir des détails sur leur architecture, on va retourner sur notre problème **XOR**.

Pour le résoudre, on va appliquer les transformations suivantes sur les axes x_1 et x_2

- $x_1 \leftarrow x_1 \wedge x_2$
- $x_2 \leftarrow x_1 \vee x_2$

Réseaux de Neurones X





$x_1 \leftarrow x_1 \wedge x_2$ et $x_2 \leftarrow x_1 \vee x_2$

Réseaux de Neurones XI

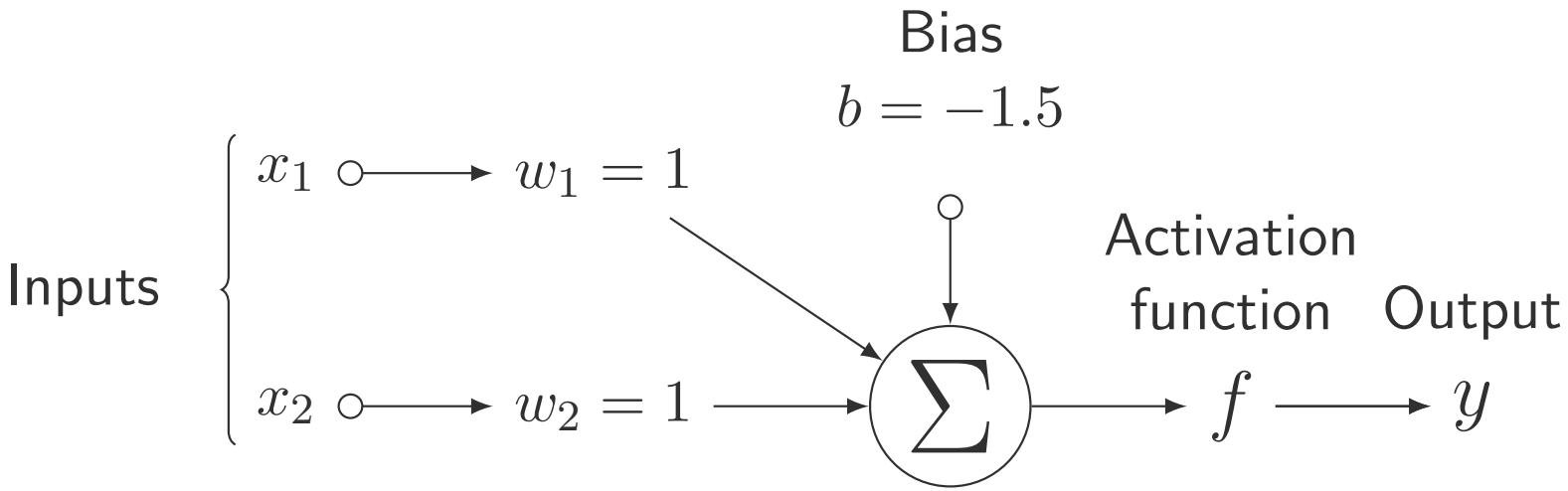
Pour trouver l'architecture du réseaux de neurones qui est capable de résoudre le problème du **XOR**, il faut garder à l'esprit que la sortie doit être *VRAIE* si et seulement si exactement une des entrées est *VRAIE*, *i.e.*

$$XOR(x_1, x_2) = (x_1 \vee x_2) \wedge (\overline{x_1 \wedge x_2}) \, .$$

Il suffira ensuite de traduire cette expression à l'aide de plusieurs perceptron

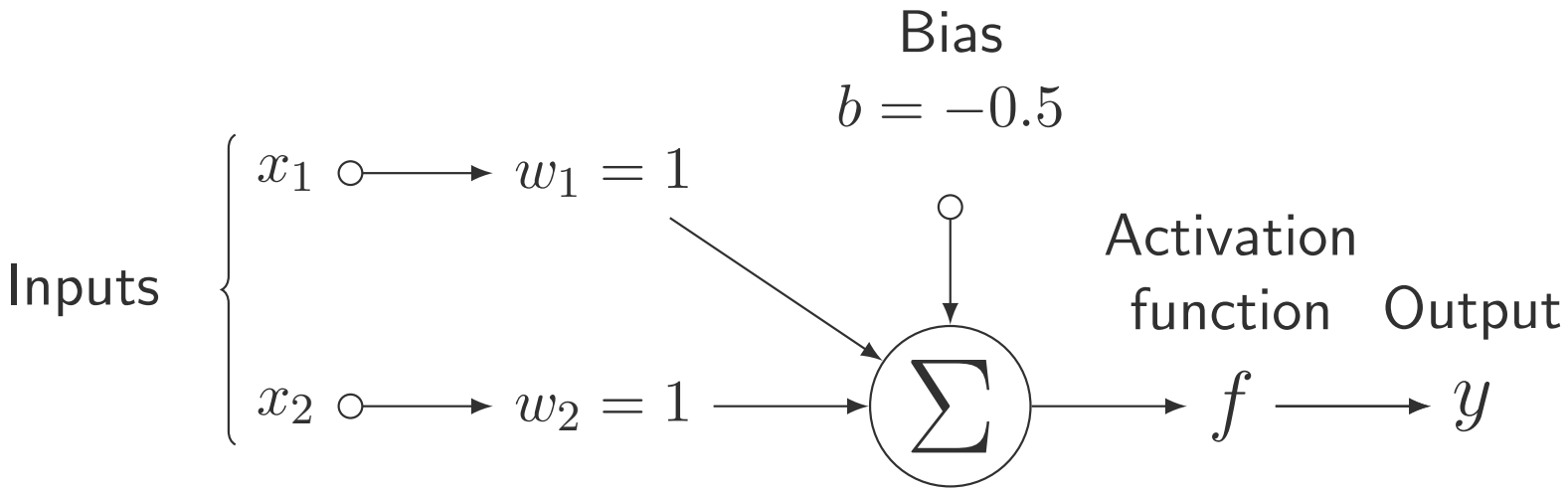
Réseaux de Neurones XII

The AND perceptron



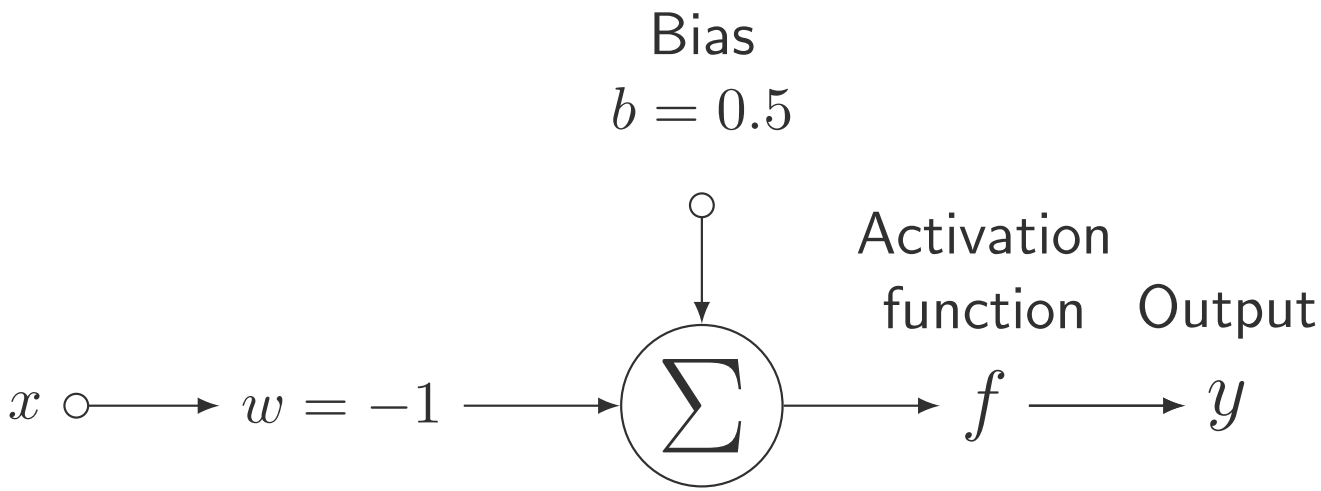
Réseaux de Neurones XIII

The **OR** perceptron :



Réseaux de Neurones XIV

The **NOT** perceptron :



Réseaux de Neurones XV

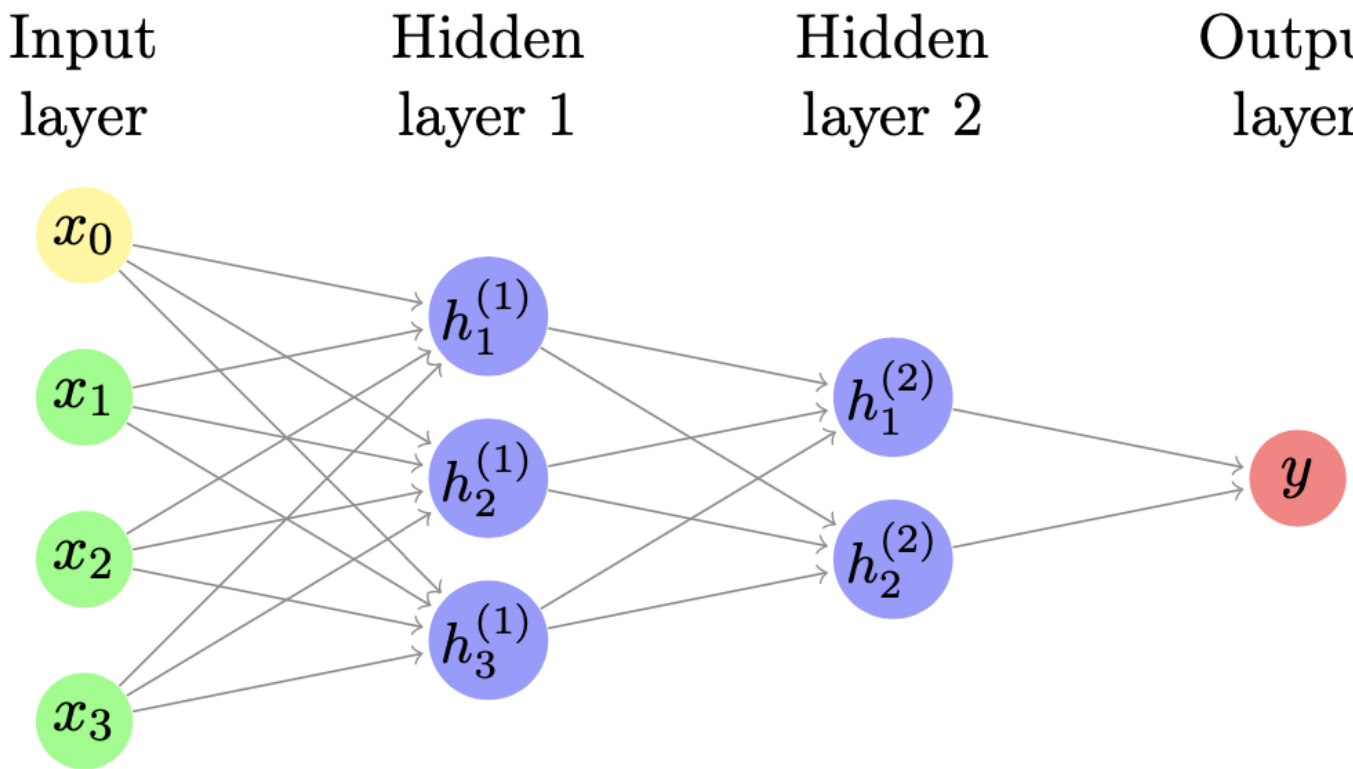
On peut être amené à diverses fonctions d'activations selon le contexte et l'application :

- Rampe : $f(x) = x$
- ReLU : $f(x) = \max(0, x)$
- ParReLU : $f(x) = \alpha x \mathbb{1}_{x<0} + x \mathbb{1}_{x>0}$
- SoftReLu : $f(x) = \ln(1 + e^x)$
- Heaviside : $f(x) = \mathbb{1}_{x>0}$
- $f(x) = \tanh(x)$
- ...

L'usage des fonctions d'activation va dépendre de votre contexte (classification ou regression) mais aussi des propriétés attendues : *monotonie* - *dérivabilité* - *smoothness* ou encore de l'endroit dans le réseau.

Réseaux de Neurones XVI

Si vous essayez de représenter le réseau de neurones associé, vous obtiendrez ce que l'on appelle un **réseau de neurones multi-couches**.



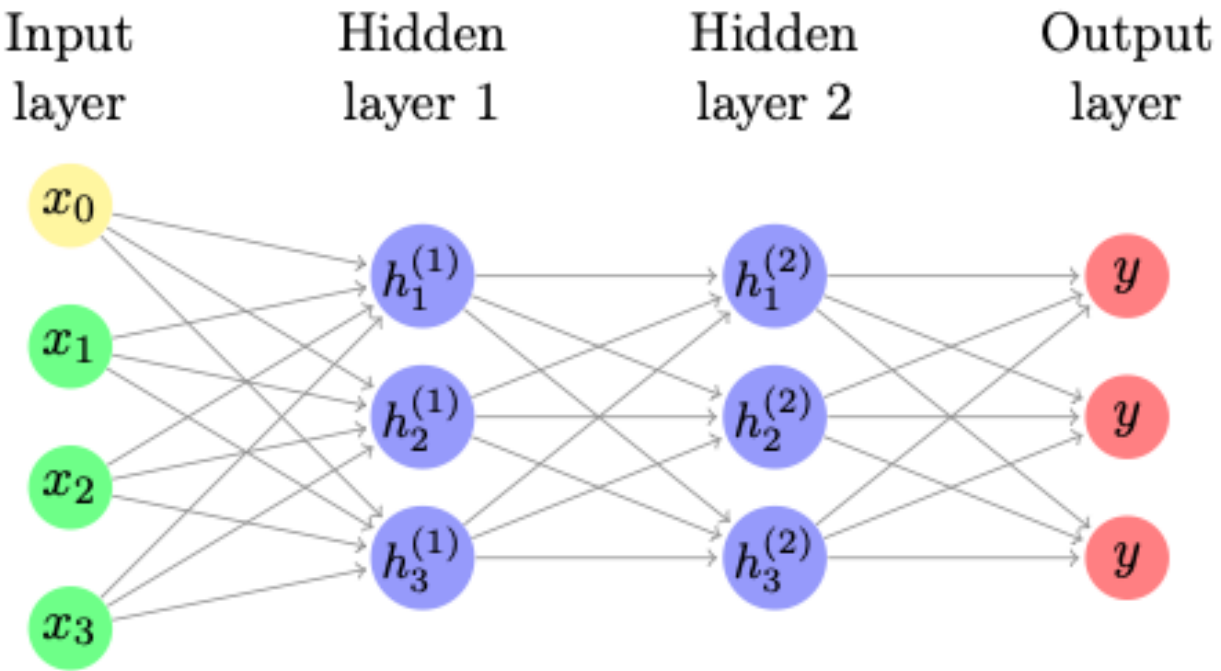
Réseaux de Neurones XVII

Un réseau de neurones est une succession de neurones qui est caractérisé par :

- le nombre de couches du réseau
- le nombre de neurones à chaque couche
- la dimension de l'output du réseau
- les fonctions d'activations utilisées
- les opérations effectuées (convolution ou somme pondérée)
- la tâche à effectuer (classification, calcul score, régression, ...)

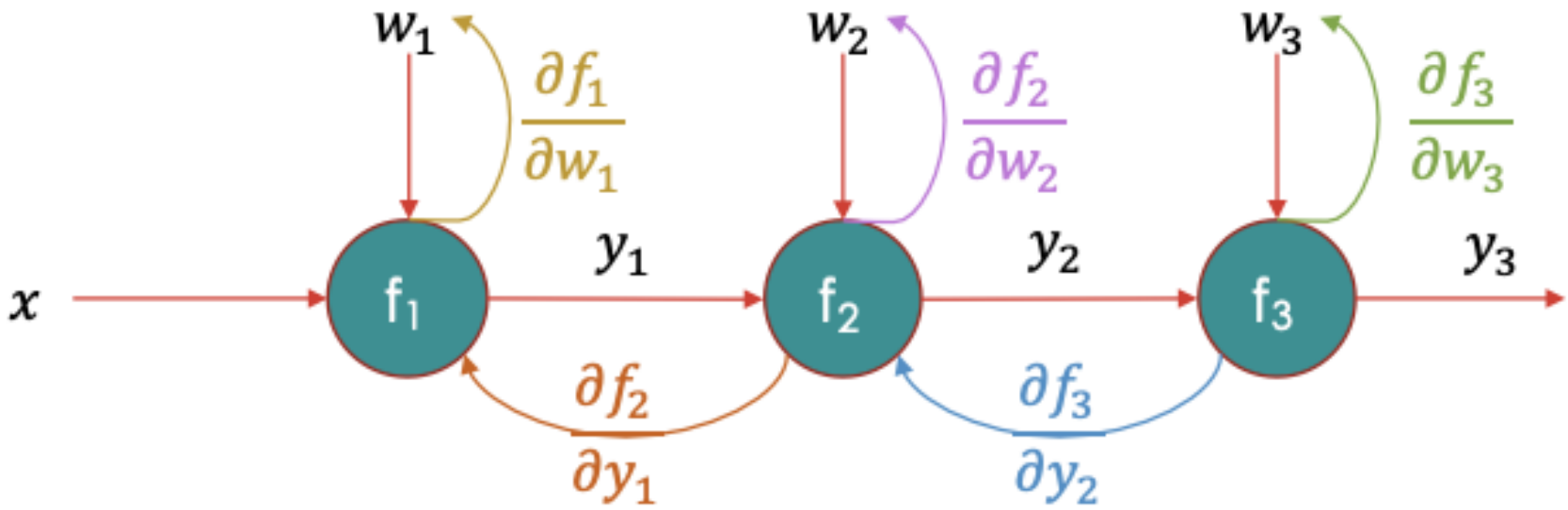
Réseaux de Neurones XVIII

Ci-dessous un autre réseau de neurones qui peut cette fois-ci être utilisé pour faire de la classification multi-classe.



Réseaux de Neurones XIX

Training and losses



Il s'agit d'alimenter un réseaux qui va se mettre à jour via un process dit *Forward - Backward*

Réseaux de Neurones XX

Les loss couramment employés dans le cadre des réseaux de neurones sont :

- La loss quadratique :

$$\ell(h(\mathbf{x}), y) = (h(\mathbf{x}) - y)^2.$$

- la cross entropy (notamment pour de la classification multi-classe avec C classes)

$$\ell(h(\mathbf{x}), y) = \sum_{c=1}^{C} y_c \ln(h(\mathbf{x})_c),$$

où $y_c = 1$ si $\mathbf{x}$ appartient à la classe c et 0 sinon. La sortie du réseau est un vecteur de probabilité d'appartenance à la classe c, *i.e.* $h(\mathbf{x})_c \in [0, 1]$

Réseaux de Neurones XXI

Sur le plan pratique

Il n'est pas nécessaire de tout coder pour implémenter son propre réseau de neurones ! Des librairies permettent de faire cela efficacement

- le package *neuralnet* sous R. On pourra consulter l'aide sous R du dit package pour voir son implémentation
- les librairies *network*, *fc_layer*, *activations* et *activation_layers* sous Python. On pourra voir un exemple à l'adresse suivante :

Tutoriel Réseaux de Neurones

Tensorflow - Keras - Pytorch

Quelques réseaux I

Nous venons de voir les réseaux profonds appelés plus particulièrement les *Feed Forward Neural Networks*.

Ces réseaux sont employés pour faire aussi bien de la régression que la classification.

Cette architecture reste cependant très simple en pratique et ne permet de traiter qu'un type de données en particulier, *i.e.* les données vectorielles. Alors comment faire pour traiter les données qui se présentent comme des images ?

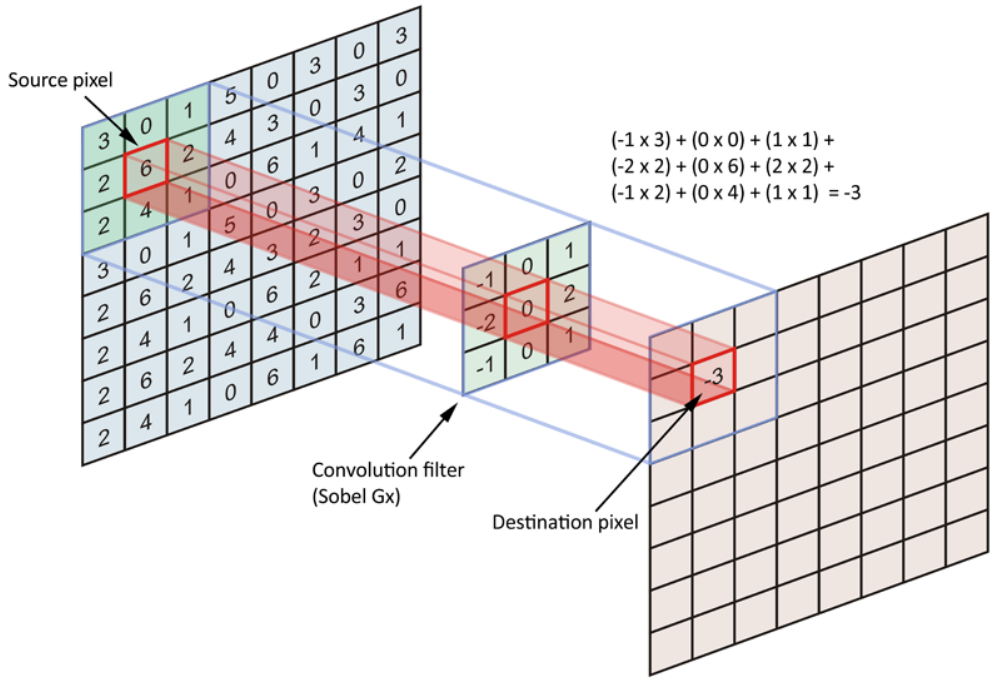
Quelques réseaux II

Réseaux convolutifs

Des convolutions (filtres) : pour apprendre à reconnaitre des caractéristiques dans l'image + réduction de dimension

Du pooling : pour condenser l'information et réduire la dimension des données (max - mean - sum)

Quelques réseaux III



Quelques réseaux IV

Des applications très variées :

- Détection d'anomalies [Bascol et al., 2017]
- Reconnaissance de motifs (images et vidéos)
- Segmentation sémantique [Fourure et al., 2017] (voir image ci-dessous)
- Analyse d'images médicales

Quelques réseaux V

Quelques réseaux VI

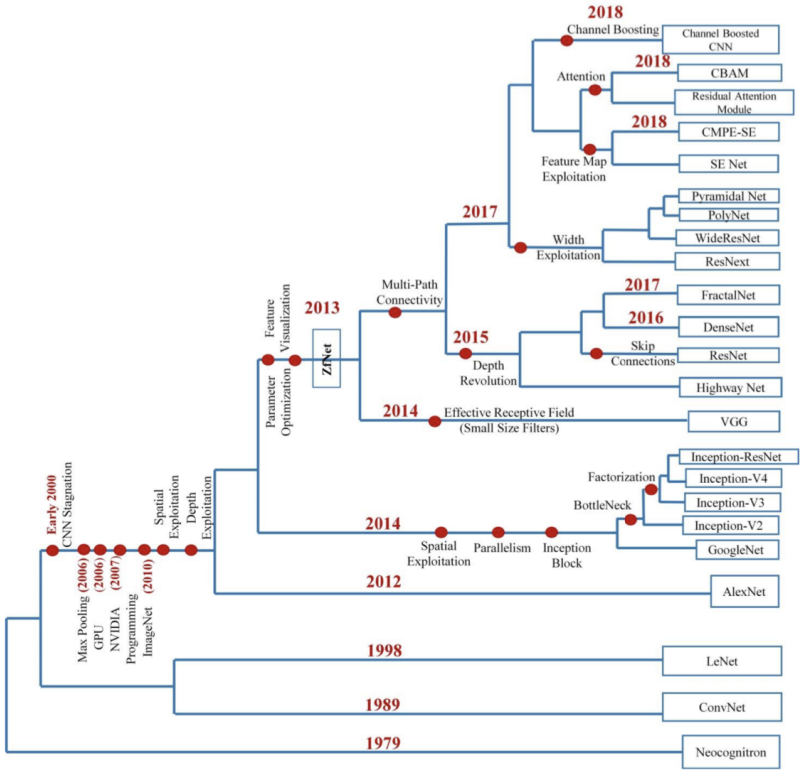


Image extraite de [Khan et al., 2020]

Quelques réseaux VII

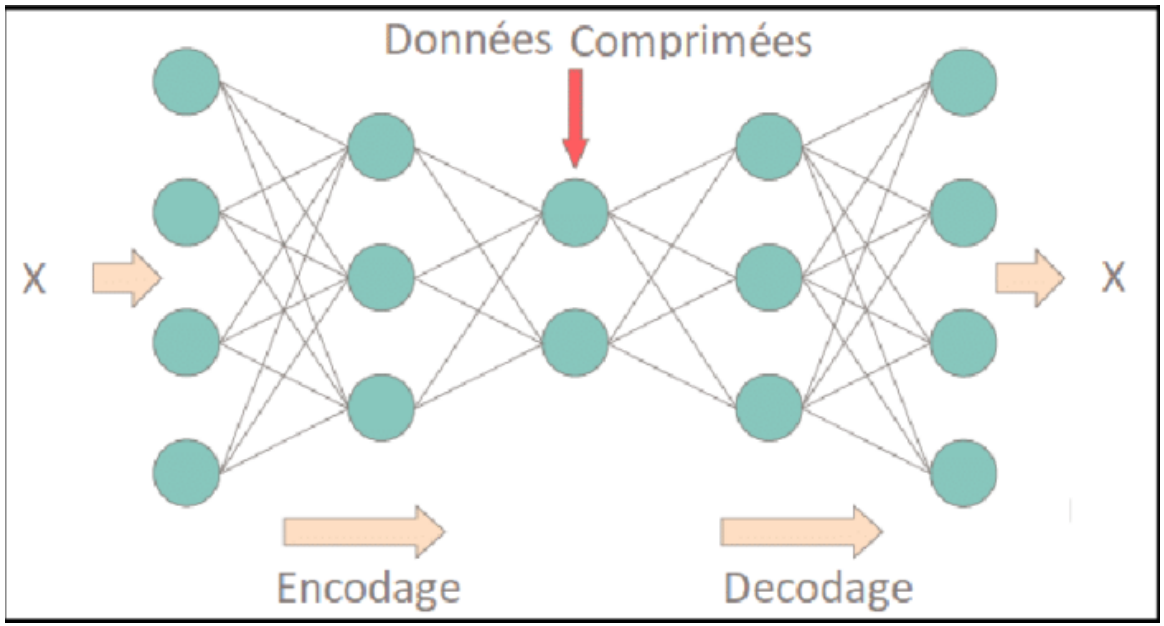
- Ce genre de réseaux présente une grande complexité, un nombre important de couches (*layers*) et donc un très grand nombres de paramètres à apprendre.
- Ces modèles présentant un très grand nombre de paramètres nécessitent donc un nombre important de données pendant la phase d'apprentissage ! Des images qui peuvent être de tailles conséquentes.
- Il est donc parfois intéressant de partir de réseaux connus et pré-entrainés sur des datasets proche de la du dataset utilisé (comme *ImageNet*) et de simplement apprendre les poids du réseaux qui concernent la tâche à effectuer (*e.g.* de la classification).

Quelques réseaux VIII

Auto-encodeurs

Un auto-encodeur se décompose de deux parties : une partie encodage ϕ et une patie décodage ψ. L'objectif de ce type de réseau (non supervisé !) est d'apprendre une représentation dans un espace de dimension plus petites de vos données. Cette représentation doit cependant permettre de conserver l'information présente dans vos données, i.e. permettre de retrouver les features originelles.

Quelques réseaux IX



Quelques réseaux X

Etant donné l'objectif principal des auto-encodeurs, une loss que l'on utilise naturellement est l'erreur quadratique :

$$\ell(\mathbf{x}, \psi \circ \phi(\mathbf{x})) = \|\mathbf{x} - \psi \circ \phi(\mathbf{x})\|^2.$$

Applications

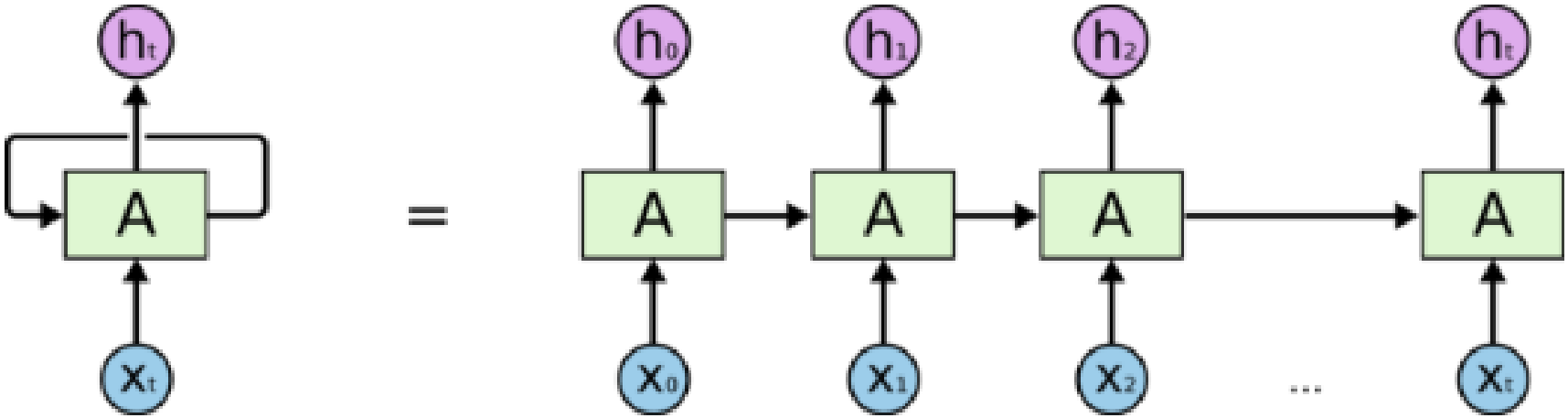
- Réduction de dimension
- Détection d'anomalies
- Débruitage de données

Variantes : auto-encodeurs variationnels pour l'apprentissage de la distribution des variables latentes (usage de la KL-divergence) utilisés pour des applications médicales ou encore la génération de données.

Quelques réseaux XI

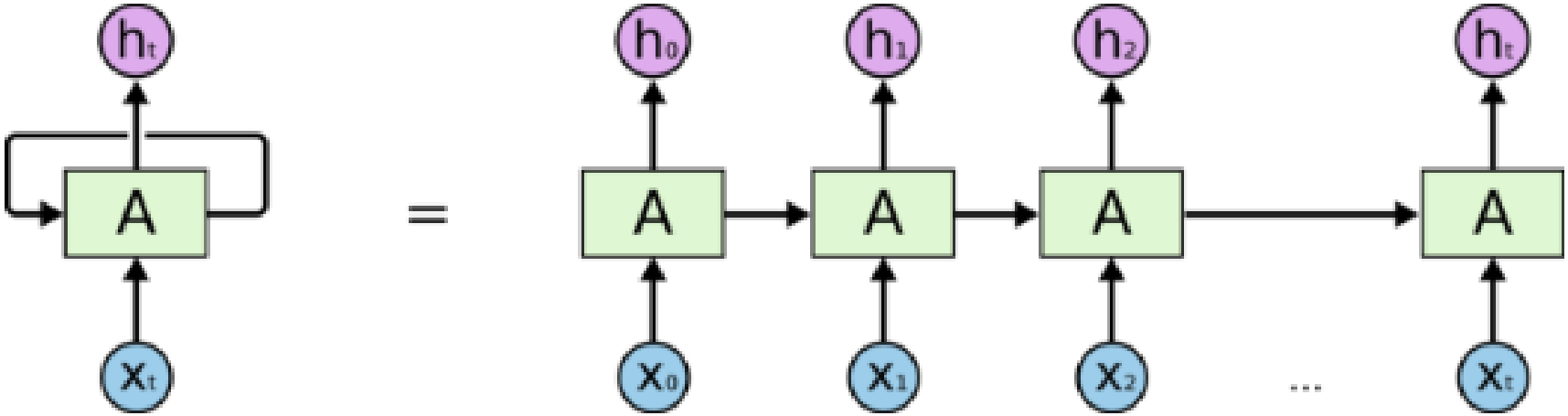
Réseaux-recurrents

Un réseau de neurones récurrents [Rumelhart et al., 1986] est un réseau avec une structure "cyclique". Il peut se représenter comme un même schéma de réseau réseau de neurones juxtaposé les uns à la suite des autres.



An unrolled recurrent neural network.

Quelques réseaux XII



An unrolled recurrent neural network.

Leur structure sous forme de graphe cyclique orienté les rends particulièrement intéressant pour l'étude ou l'analyse de séries temporelles, l'étude de texte (du langage) $\rightarrow$ étude de données séquentielles.
En effet ces réseaux ont la particularité de prendre en compte la sortie du "modèle précédent" dans l'estimation de la sortie courante.

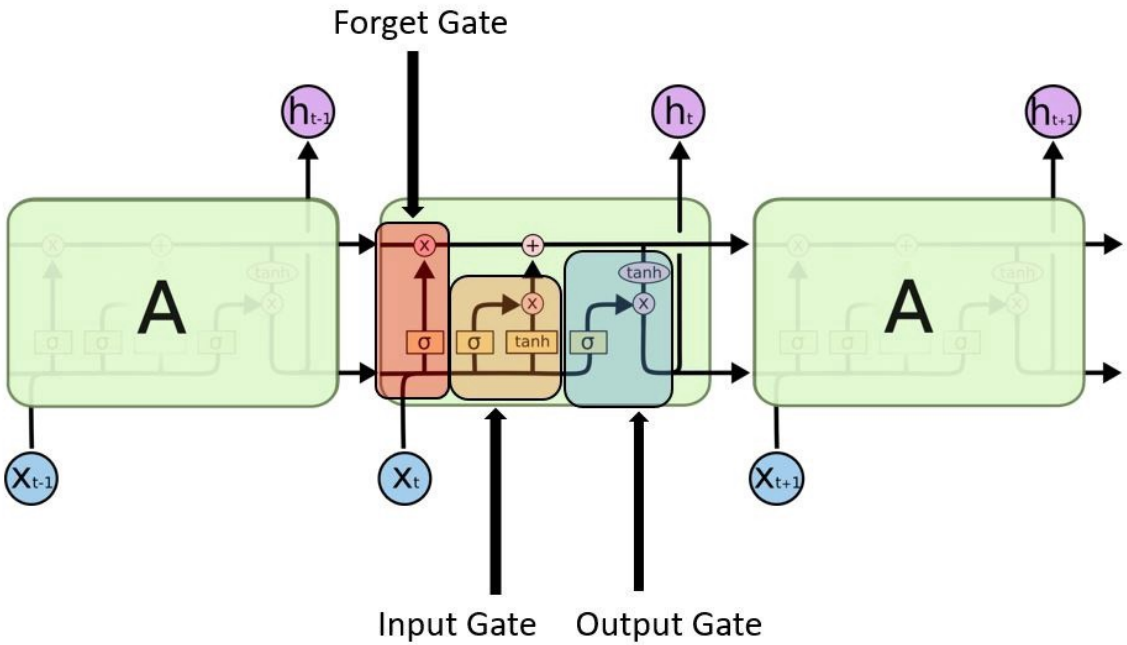
Quelques réseaux XIII

Inconvénients : l'usage d'une structure simple est cependant à proscrire car ce type de réseau présente rapidement un problème de transmission de la *mémoire*.
Plus précisément, plus la séquence d'un texte à étudier est longue, plus l'information se trouvant au début de la séquence aura du mal à se retrouver lors de l'étude de la fin de la séquence.
Ce problème est du à un mauvais "transport' du gradient qui aura tendance à tendre vers 0 plus la séquence à analyser est longue (problème de remontée du gradient le long du réseau.

$\rightarrow$ **plusieurs structures existent pour palier à se problème d'annulation du gradient**

Quelques réseaux XIV

LSTM : Long Short Term Memory



Quelques réseaux XV

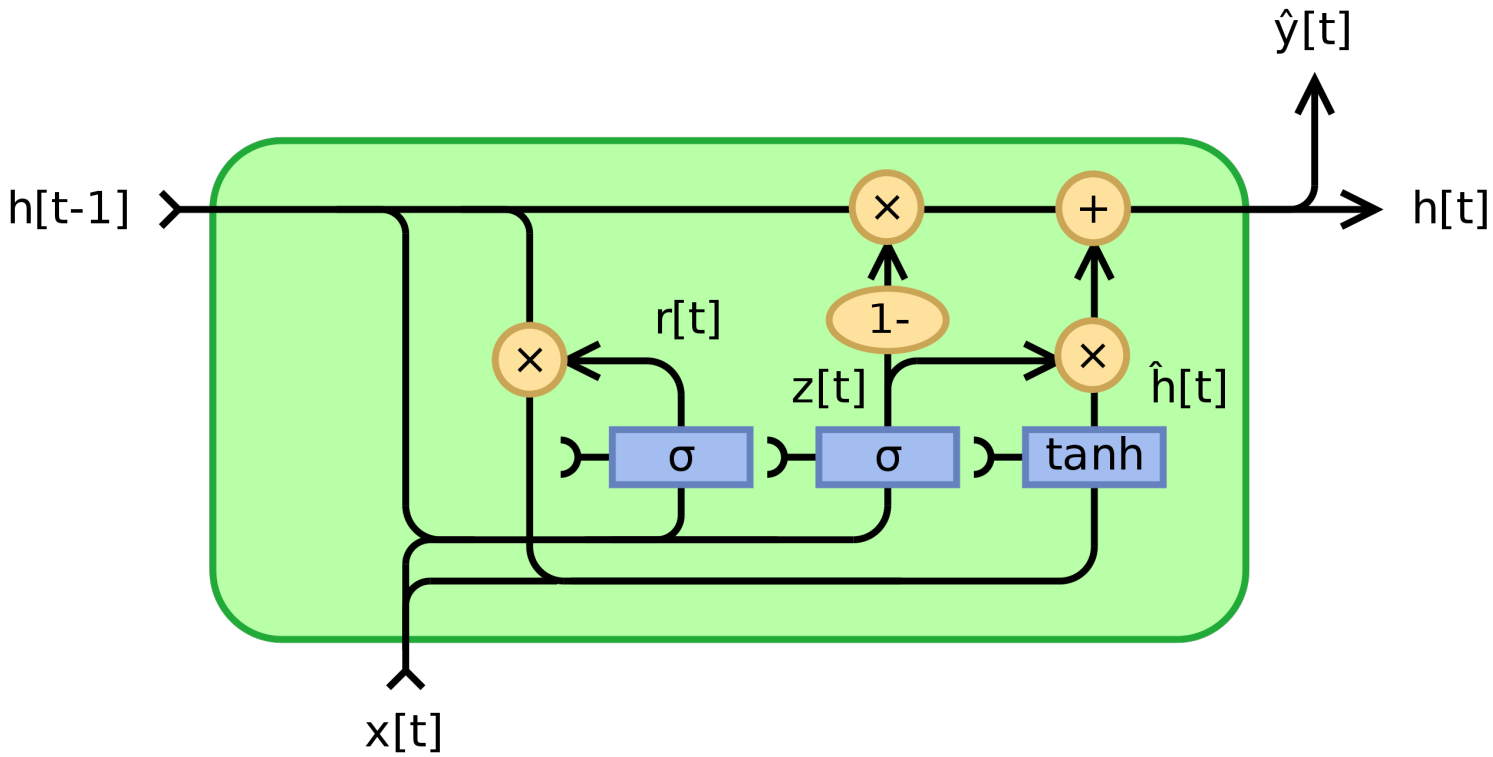
Trois composantes :

- *Input Gate* : quelles informations et quel poids données à l'information courante que l'on souhaite garder en mémoire
- *Forget Gate* : on regarde ce que l'on conserve en mémoire de l'étape précédente par rapport à la données qui arrive
- *Output Gate* : calcul de la sortie en courante en fonction de l'état de la mémoire et la donnée courante

Quelques exemples : Systèmes de recommandations avec des *Time LSTM* [Zhu et al., 2017] ou encore de la classification de signaux avec des *Phased LSTM* [Neil et al., 2016].

Quelques réseaux XVI

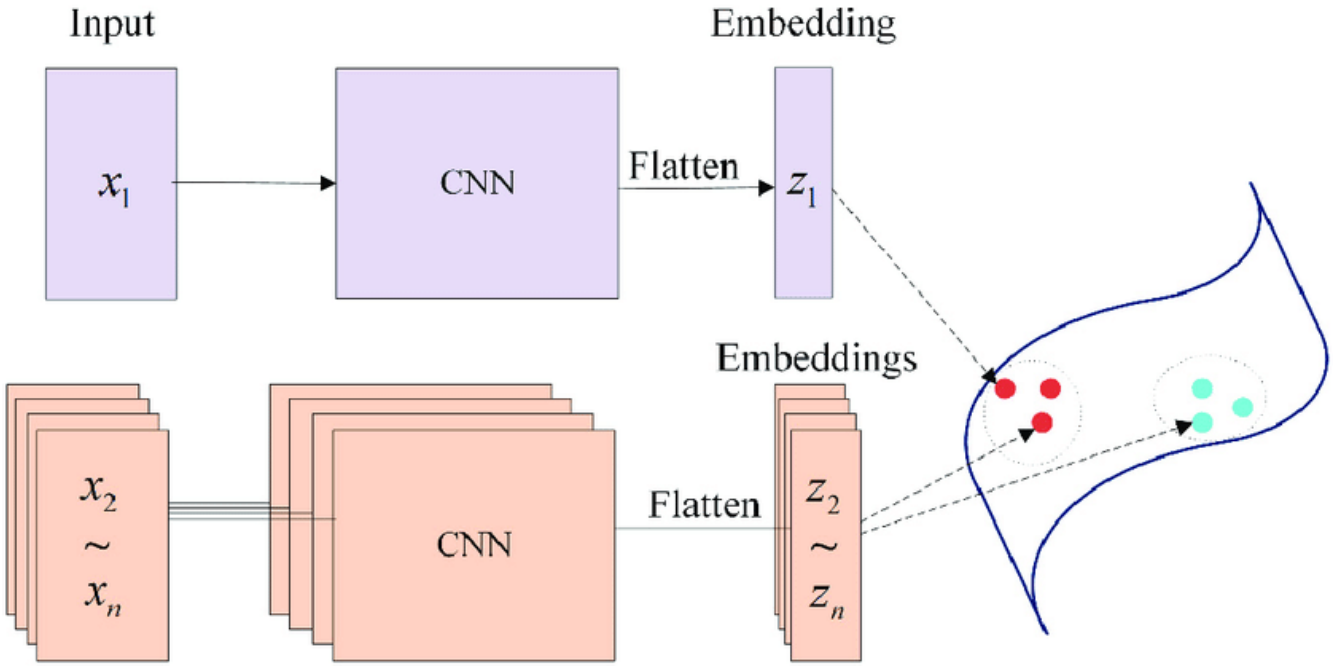
GRU : Gated Recurrent Unit



Quelques réseaux XVII

Réseaux Siamois

Structure de réseau développée dans les années 90 [Bromley et al., 1994, Chopra et al., 2005]. Il se compose de deux réseaux qui partagent **les mêmes poids** et d'une fonction de "similarité" qui va permettre de déterminer si deux exemples sont proches ou non.



Quelques réseaux XVIII

Loss

- $\ell(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2} y_{ij} d(\mathbf{x}_i, \mathbf{x}_j)^2 + \frac{1}{2}(1 - y_{ij}) \max(0, m - d(\mathbf{x}_i, \mathbf{x}_j)^2)$
- $\ell(d_S(\mathbf{x}, \mathbf{x}_S), d(\mathbf{x}, \mathbf{x}_D)) = \max(0, d_S^2 - d_D^2 + m)$

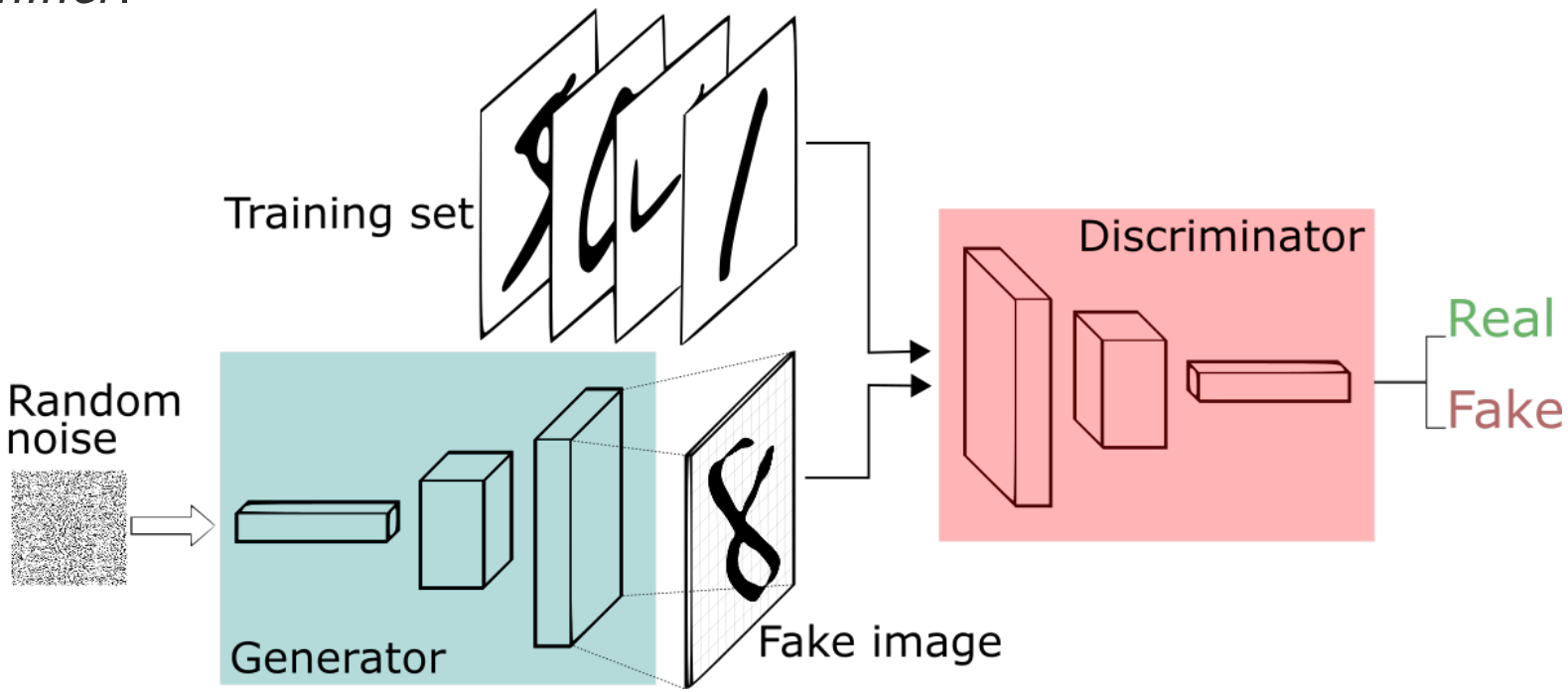
Applications

- Apprentissage de représentations
- Zero or one (or few) shot learning
- Réduction de dimension
- Détection d'anomalies

Quelques réseaux XIX

GAN : Generative Adversarial Networks

Il s'agit d'un modèle un peu particulier qui met en jeu deux réseaux qui vont se confronter [Goodfellow et al., 2014]. Un premier réseau G va *générer* des données et un deuxième réseau D aura pour rôle de *discriminer*.



Quelques réseaux XX

La fonction permettant d'entraîner ce type de modèle se présente sous la forme d'un problème $\min - \max$.

$$\min_G \max_D \ell(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}}[\ln(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z} \sim p_{gen}}[\ln(1 - D(G(\mathbf{z})))]$$

Quelques applications

- Faire de la génération de données : texte - audio - photos
- Deepfake
- Il existe aussi des applications dans le domaine de la santé ou climat, mais elles sont mineures
- Détection d'anomalies [Bashar and Nayak, 2020]

Les arbres de décisions

Arbre de Décision I

Les arbres de décisions ont été introduits au milieu des données 80 et sont une catégorie d'algorithme d'apprentissage supervisé qui permet de faire à la fois de la classification et de la régression [Breiman et al., 1984].

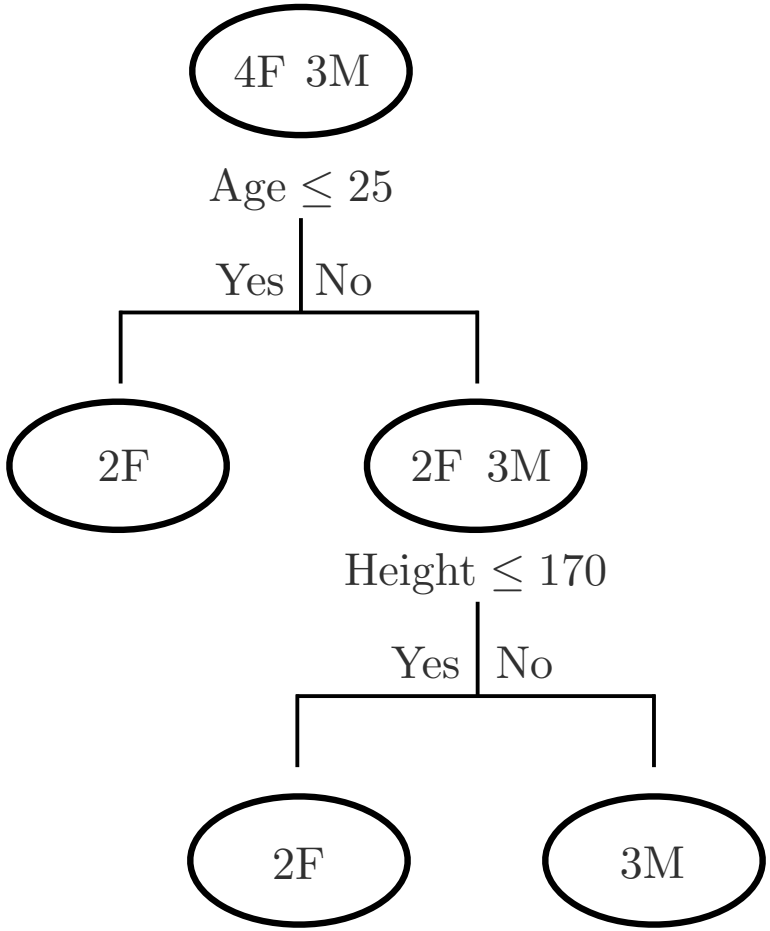
Appelés également *arbres de décision*, ils consistent en une série de règles (souvent binaires) qui vont permettre de successivement séparer le jeu de données en deux.
La nature de l'arbre dépend de la nature de l'espace $\mathcal{Y}$

- quand $\mathcal{Y} \subset \mathbb{R}$, on parle d'arbre de régression
- quand $\mathcal{Y} = \{0, 1, \ldots, C\}$, on parle d'arbre de classification

Arbre de Décision II

Toy dataset

Age	Height	Sex
20	175	F
32	180	M
40	175	M
28	172	M
22	165	F
40	169	F
70	170	F



Arbre de Décision III

Un tel algorithme est capable d'apprendre des frontières de décision non linéaires et donc de résoudre des problèmes complexes.
Comme les autres algorithmes d'apprentissage, nous avons besoin de définir un critère à optimiser.

Nous avons donc besoin de définir ce que l'on appelle une *métrique* qui va permettre d'évaluer la qualité d'une séparation (*i.e.* un split) que l'on appelle le
gain [Safavian and Landgrebe, 1991, Rokach and Maimon, 2005].

Bien évidemment, la métrique employée dépend de la nature de l'arbre (régression, classification) [Rokach and Maimon, 2005]

Arbre de Décision IV

- la *Variance*, pour les arbres de régression

$$\frac{1}{n}\sum_{i=1}^{m_N}(y_i - \bar{y})^2,$$

où m_N représente le nombre d'individus dans le noeud N et $\bar{y}$ la valeur moyenne des y_i dans le noeud.

- *l'impureté de Gini* pour les arbres de classification. Cela mesure l'impureté d'un noeud en considérant la représentation de chaque classe dans un noeud (en terme de proportion).

Arbre de Décision V

Par exemple, en classification binaire l'impureté de Gini G_N d'un noeud N est définie par :

$$G_N = \sum_{j=1,-1} p_j(1-p_j) = 2p_1(1-p_1), \tag{7}$$

où p_j représente la proportion d'exemples dans la classe j.

La définition peut être étendue dans le cas multi-classe en considérant la quantité

$$G_N = \sum_{j=1}^{L} p_j(1-p_j)$$

où p_j représente la proportion d'exemples dans la classe j.

Arbre de Décision VI

En classification binaire, l'impureté de Gini est une valeur comprise entre $[0, 0.25]$.

Une valeur proche de 0 signifie que le noeud est pur, au contraire si la valeur est proche de 0.25, cela signifie que le désordre est maximal, *i.e.* il y le même nombre d'exemples de chaque classe dans le noeud considéré.

Arbre de Décision VII

Dans le précédent exemple, l'impureté de Gini à la racine était égale à $G_{\text{root}} = 2 \times \frac{4}{7}\left(1 - \frac{4}{7}\right) = \frac{24}{49}$ alors qu'elle est égale à 0 et $\frac{12}{25}$ respectivement pour chaque noeud après la première séparation.

Le noeud de gauche ainsi obtenu est un noeud pur, son impureté de Gini est nul et ne peut donc plus être améliorée.

On va alors se concentrer sur le noeud de droite et on va essayer de déterminer la meilleure séparation qui conduit à la plus petite impureté après séparation de ce noeud.

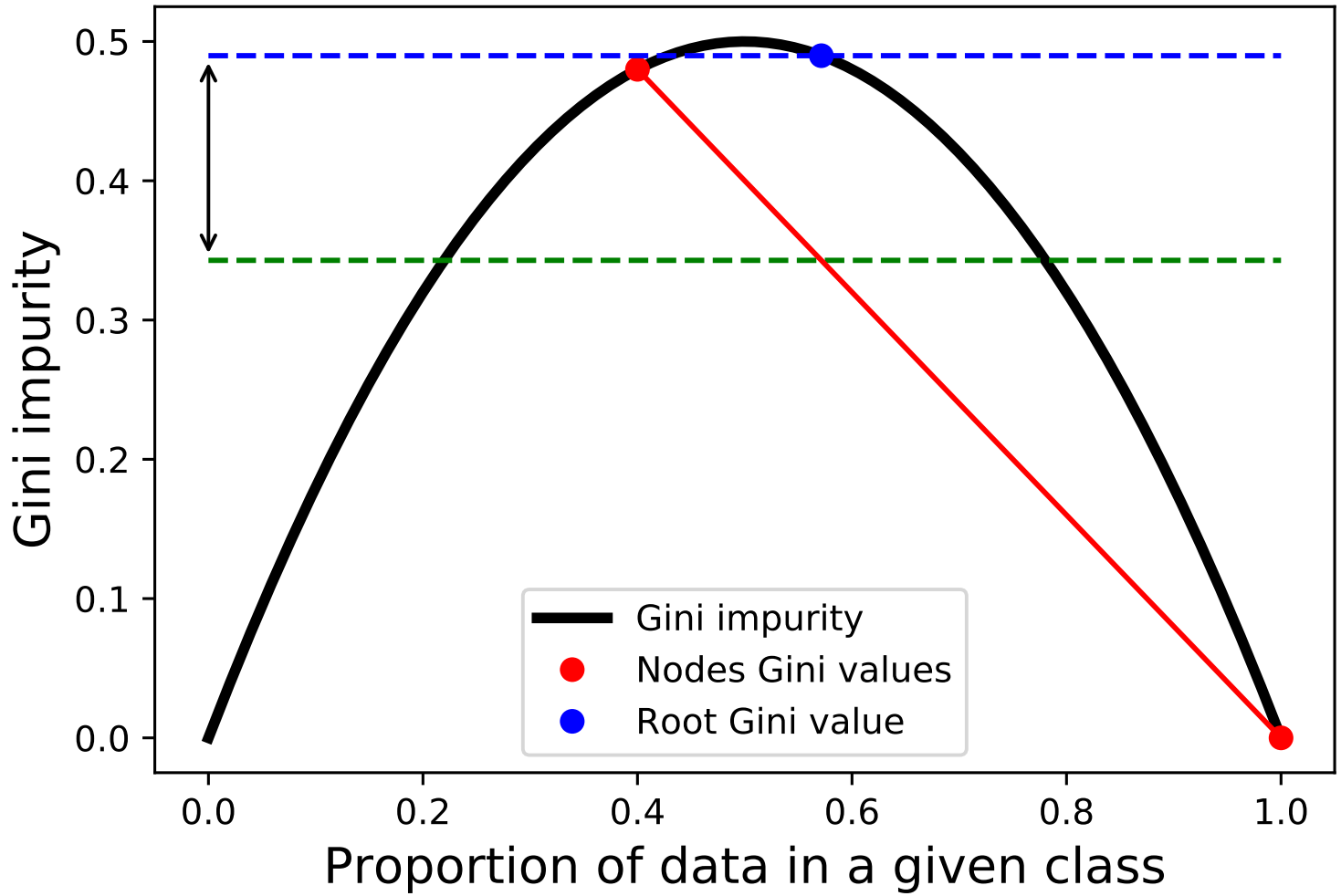
Arbre de Décision VIII

Pour faire cela, on définit ce que l'on appelle le *gain de Gini* :

$$\Gamma = G_{\text{root}} - \left(\frac{|N_L|}{|N_L + N_R|} G_{N_L} + \frac{|N_R|}{|N_L + N_R|} G_{N_R} \right),$$

où G_{N_L} et G_{N_R} désignent l'impureté de Gini pour les noeuds de gauche et droite respectivement.

Arbre de Décision IX



Arbre de Décision X

La concavité de la fonction employée assure la positivité du gain (inégalité de Jensen) et ainsi que chaque split conduit bien à une minimisation de l'erreur de classification.

En outre, à chaque étape, on va choisir le séparateur optimal qui va permettre de maximiser le gain de Gini et le processus est répété jusqu'à l'obtention de feuilles pures.

Arbre de Décision XI

En pratique, il est toujours possible de se ramener à des feuilles pures en construisant un arbre suffisamment profond. Mais ces arbres sont sujets au sur-apprentissage et vont donc être moins vorie peu performants sur de nouvelles données.

Comme pour les autres algorithmes, il est donc nécessaire de jouer sur plusieurs hyper-paramètres des arbre comme

- la *taille/profondeur* de l'arbre,
- *la taille du noeud* : le nombre minimal d'exemples qui doit être contenu dans un noeud avant séparation,
- *la taille d'une feuille* : le nombre minimum d'exemples que l'on doit retrouver dans une feuille après séparation,
- *un seuil sur le gain* : la valeur minimal du gain que l'on doit atteindre pour que le split soit effectué.

Références I


Bascol, K., Emonet, R., Fromont, E., and Debusschere, R. (2017).
Improving chairlift security with deep learning.
In *International Symposium on Intelligent Data Analysis*, pages 1–13. Springer.


Bashar, M. A. and Nayak, R. (2020).
Tanogan : Time series anomaly detection with generative adversarial networks.


Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992).
A training algorithm for optimal margin classifiers.
In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152. ACM.

Références II


Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*. The Wadsworth statistics/probability series. Wadsworth and Brooks/Cole Advanced Books and Software, Monterey, CA.


Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1994). Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744.


Chopra, S., Hadsell, R., and LeCun, Y. (2005). Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE.

Références III


Cover, T. and Hart, P. (1967).
Nearest neighbor pattern classification.
IEEE Transactions on Information Theory, 13(1) :21–27.


Cox, D. R. (1958).
The regression analysis of binary sequences (with discussion).
Journal of the Royal Statistical Society, 20 :215–242.


Fourure, D., Emonet, R., Fromont, E., Muselet, D., Trémeau, A., and Wolf, C. (2017).
Residual conv-deconv grid network for semantic segmentation.
In *Proceedings of the British Machine Vision Conference, 2017*.


Genton, M. G. (2002).
Classes of kernels for machine learning : A statistics perspective.
Journal of Machine Learning Research, 2 :299–312.

Références IV


Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014).
Generative adversarial nets.
In *Advances in neural information processing systems*, pages 2672–2680.


Khan, A., Sohail, A., Zahoora, U., and Qureshi, A. S. (2020).
A survey of the recent architectures of deep convolutional neural networks.
Artificial Intelligence Review.


McCulloch, W. S. and Pitts, W. (1943).
A logical calculus of the ideas immanent in nervous activity.
The bulletin of mathematical biophysics, 5(4) :115–133.

Références V


Mercer, J. (1909).
Functions of positive and negative type, and their connection with the theory of integral equations.
Philosophical Transactions of the Royal Society A : Mathematical, Physical and Engineering Sciences, 209(441-458) :415–446.


Neil, D., Pfeiffer, M., and Liu, S.-C. (2016).
Phased lstm : Accelerating recurrent network training for long or event-based sequences.
In *Advances in neural information processing systems*, pages 3882–3890.


Rokach, L. and Maimon, O. (2005).
Top-down induction of decision trees classifiers - a survey.
IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 35(4) :476–487.

Références VI


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


Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986).
Learning representations by back-propagating errors.
nature, 323(6088) :533–536.


Safavian, S. R. and Landgrebe, D. (1991).
A survey of decision tree classifier methodology.
IEEE Transactions on Systems, Man, and Cybernetics, 21(3) :660–674.


Vapnik, V. and Cortes, C. (1995).
Support-vector networks.
Machine Learning, 20 :273–297.

Références VII


Zhu, Y., Li, H., Liao, Y., Wang, B., Guan, Z., Liu, H., and Cai, D. (2017).
What to do next : Modeling user behaviors by time-lstm.
In *IJCAI*, volume 17, pages 3602–3608.