

# TER Machine Learning

Fernandes Magalhaes Tiago

Janvier 2022

## Table des matières

<b>I</b>	<b>Régression</b>	<b>4</b>
<b>1</b>	<b>Régression Linéaire simple</b>	<b>6</b>
1.1	Présentation . . . . .	6
1.2	La méthode d'estimation par moindres carrés . . . . .	8
1.3	Résolution . . . . .	11
1.3.1	Méthode de la descente de gradient . . . . .	12
1.3.2	Méthode analytique . . . . .	13
<b>2</b>	<b>Régression Linéaire multiple</b>	<b>14</b>
2.1	Cadre de la regression linéaire multiple . . . . .	14
2.2	Formes matricielles et première résolution . . . . .	15
2.3	La descente de gradient pour la régression linéaire multiple . .	17
<b>3</b>	<b>Régression Logistique</b>	<b>18</b>
3.1	Fonction de modélisation . . . . .	19
3.2	Fonction de coût . . . . .	21
3.3	Résolution . . . . .	22
3.4	Classification à $n > 2$ classes . . . . .	23
<b>II</b>	<b>Réseaux de neurones</b>	<b>25</b>

<b>4</b>	<b>Fonctionnement d'un réseau de neurones</b>	<b>26</b>
4.1	Introduction . . . . .	26
4.2	Propagation en avant . . . . .	27
4.2.1	La matrice de passage . . . . .	27
4.2.2	Le calcul de propagation en avant . . . . .	28
4.3	Rétro-propagation . . . . .	29
4.3.1	Présentation et notation . . . . .	30
4.3.2	Calcul du gradient de la fonction de coût . . . . .	30
<b>5</b>	<b>Pourquoi les réseaux de neurones fonctionnent-ils ?</b>	<b>32</b>
5.1	Intuition . . . . .	32
5.2	Exemple d'un réseau de neurone entraîné . . . . .	32
<b>III</b>	<b>Utilisations numériques des méthodes</b>	<b>34</b>
<b>6</b>	<b>La régression linéaire</b>	<b>34</b>
6.1	Algorithmes pour la fonction de coût et la descente de gradient	35
6.1.1	Fonction de coût . . . . .	35
6.1.2	Fonction de descente de gradient . . . . .	35
6.1.3	Fonction de calcul de la méthode analytique . . . . .	36
6.2	Résolution complète . . . . .	37
6.3	Approfondissement . . . . .	40
6.3.1	Méthode analytique ou descente de gradient ? . . . . .	40
6.3.2	Le choix du pas pour la descente de gradient . . . . .	40
6.3.3	Alternatives informatiques à la méthode du gradient . . . . .	41
<b>7</b>	<b>La régression logistique</b>	<b>42</b>
7.1	Fonctions à préparer . . . . .	43
7.1.1	La fonction mapFeature . . . . .	43
7.1.2	La fonction de calcul du coût et du gradient . . . . .	44
7.2	Résolution . . . . .	45
7.3	Au sujet du sur-ajustement/sous-ajustement . . . . .	48
7.3.1	Le problème . . . . .	48
7.3.2	Quel remède pour un sur-ajustement/sous-ajustement ?	48

# Introduction

Le Machine Learning est défini par *Arthur Samuel*, un de ses précurseurs, de la manière suivante :

*"Le Machine Learning est le domaine d'étude qui donne à l'ordinateur la capacité d'apprendre sans avoir été explicitement programmé."*

On associe au machine learning différentes techniques qui permettent, à partir de données connues, de tirer des conclusions sur d'autres paramètres, à priori inconnus.

Une définition plus formelle énoncée un peu plus tard par *Tom Mitchell*, décrit le machine learning de la façon suivante :

*"On dit qu'un ordinateur apprend par nombre d'expériences  $E$  une tâche  $T$  avec une performance  $P$ , si sa performance à la tâche  $T$ , mesurée par  $P$ , augmente avec l'expérience  $E$ ."*

Lesdites méthodes considérées comme du machine learning s'appuient principalement sur le domaine mathématique des statistiques.

Avec l'explosion de la ressource en données, couplé au progrès calculatoire informatique, le machine learning est actuellement en pleine expansion à la fois scientifiquement et en popularité.

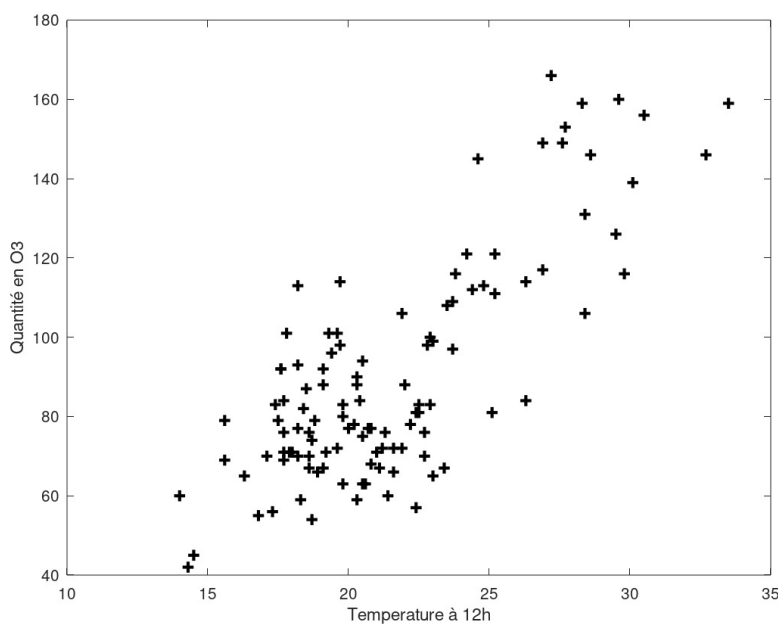
Dans ce TER, nous introduirons des techniques très utilisées et fondamentales que sont la regression linéaire, la régression logistique ainsi que les réseaux de neurones. Elles seront, pour les deux premières, accompagnées d'exemples programmés en Octave dans la troisième partie du document.

## Première partie

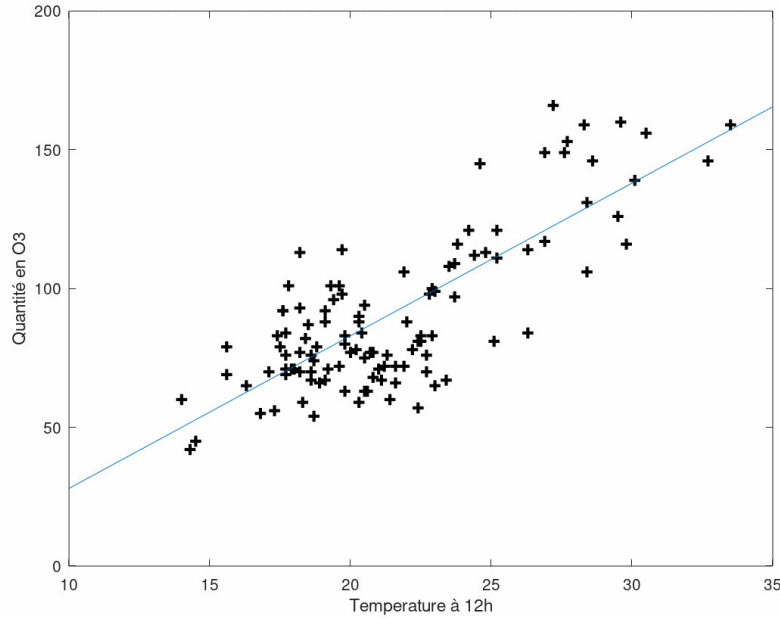
# Régression

Supposons que nous souhaitons vérifier si, en fonction de certains paramètres de l'air, il est possible d'en déduire sa teneur en ozone (O3). Rappelons que l'ozone est toxique lorsqu'ingéré par la faune ou la flore. Pour appuyer nos recherches, nous trouvons un jeu de données sur les statistiques de qualité d'air de la ville de Rennes durant l'été 2001[5].

Visualisons un graphique où l'on va associer, par des points, la température à 12h avec la quantité maximale d'O3 présente dans l'air durant le même jour.



Ce nuage de points fait penser à un alignement selon une forme qui n'est pas très loin d'une droite. Nous pouvons alors essayer de tracer une droite, ayant pour équation  $y = 5x - 27$ , qui suit l'alignement de notre nuage de points :



C'est essentiellement ce que nous essaierons de faire lors d'une régression linéaire. On souhaite obtenir la "meilleure" courbe possible sous la forme d'une fonction reliant la température à la quantité maximale d'O3. Nous expliquerons ce que l'on entend par "meilleure courbe possible" dans la partie sur la régression linéaire simple.

Plus généralement la régression linéaire consiste à considérer un ensemble de  $m$  valeurs dites *valeurs explicatives*  $(x^{(i)})_{i \in \{1, \dots, m\}}$  qui sont associées une à une à des valeurs résultats dites *valeurs expliquées* que l'on note  $(y^{(i)})_{i \in \{1, \dots, m\}}$ .

$$x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}) \in \mathbb{R}^n$$

$$y^{(i)} \in \mathbb{R}$$

Les valeurs explicatives  $x^{(i)}$  sont des vecteurs de taille  $n \geq 1$ .  
Soit la fonction  $h_\theta : \mathbb{R}^n \rightarrow \mathbb{R}$  telle que :

$$h_\theta(x) = \theta_0 + \sum_{k=1}^n \theta_k x_k; \quad \text{où } \theta = (\theta_0, \dots, \theta_n) \in \mathbb{R}^{n+1} \text{ et } (x_1, \dots, x_n) \in \mathbb{R}^n.$$

Avec la régression on souhaite trouver les paramètres  $\theta$  de la fonction  $h_\theta$  qui nous permettrait d'avoir

$$y^{(i)} \approx h_\theta(x^{(i)}), \quad \forall i \in \{1, \dots, m\}.$$

L'intérêt de  $h_\theta$  serait qu'elle puisse prédire  $y^{(i)}$  à partir de nouvelles valeurs  $x^{(i)}$ , avec  $i \in \llbracket m+1 ; p \rrbracket$  pour  $p > m$ . Ce qui signifierait considérer des données supplémentaires.

Nous allons accompagner la présentation de la régression linéaire, par la résolution du problème de régression montré en exemple.

# 1 Régression Linéaire simple

## 1.1 Présentation

Pour la régression linéaire simple on va considérer le vecteur  $\mathbf{x}$  de taille 1, c'est à dire que l'on va essayer de prédire la valeur expliquée  $y \in \mathbb{R}$  à partir d'une variable de dimension 1, ou contenant une seule information dite "explicative".

Dans ce cas, et de la même manière que pour l'exemple présenté plus haut, où "la température explique l'ozone", on peut assez bien visualiser et représenter un jeu de données de départ.

Soit  $h_\theta : \mathbb{R} \rightarrow \mathbb{R}$  la fonction qui modélise la relation liant les variables explicatives et expliquées, et  $\theta := (\theta_0, \theta_1) \in \mathbb{R}^2$  le vecteur des paramètres de la fonction  $h_\theta$ .

On a donc :

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Le principe de la régression linéaire est de déterminer les valeurs de  $\theta_0$  et  $\theta_1$ .

On peut aussi écrire l'expression de  $h_\theta$  sous forme vectorielle :

$$h_\theta(X) = \theta^T X; \tag{1}$$

Avec, pour la régression linéaire simple,  $X = (1, x)^T$ ;  $\theta = (\theta_0, \theta_1)^T$ . La forme vectorielle de  $h_\theta$  s'applique à toute régression linéaire, y compris la régression linéaire multiple.

Puisque nous avons **déjà choisi** la forme de notre fonction  $h_\theta$ , notre objectif se réduit maintenant à trouver le meilleur vecteur  $\theta = (\theta_0, \theta_1)^T$  possible. Pour cela nous allons utiliser la liste de données de départ.

C'est à dire que l'on souhaite minimiser la différence entre les prédictions  $h_\theta(x^{(i)})$  et les résultats  $y^{(i)}$  respectifs.

Soit  $l : \mathbb{R}^2 \mapsto \mathbb{R}$  une fonction fixée. Pour tout  $i$ ,  $l(y^{(i)}, h_\theta(x^{(i)}))$  est un réel qui représente la différence entre le résultat prédit et le résultat réel. Appelons  $J_l(\theta)$  la fonction à minimiser.

Nous souhaitons résoudre le problème de minimisation suivant :

$$\operatorname{argmin}_{\theta \in \mathbb{R}^2} J_l(\theta) = \operatorname{argmin}_{\theta \in \mathbb{R}^2} \frac{1}{2m} \sum_{i=1}^m l(y^{(i)}, h_\theta(x^{(i)})). \quad (2)$$

La fonction  $l$  est appelée la fonction de coût. Elle est celle qui va déterminer la façon dont on évalue la différence entre  $y^{(i)}$  et  $h_\theta(x^{(i)})$ .

On pourrait se demander par exemple, si le choix  $l(x, y) = |x - y|$  ne serait pas le plus pertinent. En effet,  $l(y^{(i)}, h_\theta(x^{(i)})) = |y^{(i)} - h_\theta(x^{(i)})|$  est la distance qui sépare  $y^{(i)}$  de  $h_\theta(x^{(i)})$ .

**La fonction de coût des moindres carrés**, développée au début du XIX<sup>e</sup> siècle par Gauss et Legendre, de façon indépendante, est celle qui s'est imposée. Elle permet notamment à  $J$ , la fonction à minimiser, d'être strictement convexe, notion que l'on va expliquer dans la section 1.3. Le  $\theta$  solution, par cette fonction de coût, possède beaucoup de bonnes propriétés statistiques qui témoignent de sa qualité. Aussi, la fonction de coût des moindres carrés offre un accès relativement aisé vers l'utilisation d'outils statistiques qui permettent de donner "une valeur de vérité" au  $\theta$  trouvé (intervalles de confiance, tests d'hypothèse). Le but de la prochaine section sera d'énoncer le *théorème de Gauss-Markov* qui justifiera l'utilisation de la fonction de coût des moindres carrés.

## 1.2 La méthode d'estimation par moindres carrés

Formalisons le problème dans le cadre probabiliste afin de pouvoir utiliser les outils statistiques.

Pour tout  $i \in \llbracket 1 ; m \rrbracket$ , considérons le couple de variable aléatoire  $(X^{(i)}, Y^{(i)})$ . La famille  $(X^{(i)}, Y^{(i)})_{i \in \{1, \dots, m\}}$  est un  $m$ -échantillon indépendant de loi  $\mathbb{P}_\theta$ , que l'on va introduire maintenant :

$$Y^{(i)} = \theta_1 X^{(i)} + \theta_0 + \varepsilon$$

avec  $\varepsilon \sim \mathcal{N}(0, \sigma^2)$  et  $Y^{(i)} | X^{(i)} \sim \mathcal{N}(\theta_0 + \theta_1 X^{(i)}, \sigma^2)$

On appelle  $\varepsilon$  la variable aléatoire de l'erreur(ou du bruit) pour tout couple d'indice  $(X^{(i)}, Y^{(i)})_{i \in \{1, \dots, m\}}$  en supposant alors que **la variable aléatoire de l'erreur  $\varepsilon$  est la même pour tous les  $i$ .**

*Remarque :*  $\mathcal{N}(\mu, \sigma^2)$  symbolise une loi normale(ou gaussienne) d'espérance  $\mu$  et de variance  $\sigma^2$ .

Soit  $(\mathbb{R}^{2m}, \xi, (\mathbb{P}_\theta)_{\theta \in \mathbb{R}^2})$  le modèle statistique dans lequel nous évoluons. Avec  $(\mathbb{R}^2)^m$  l'ensemble des valeurs possibles pour le  $m$ -échantillon et  $\xi$  la tribu contenant les ensembles mesurables par la mesure de probabilité  $\mathbb{P}_\theta$ .

Nous souhaitons obtenir, ou du moins nous approcher au mieux, des valeurs de  $\theta_0$  et  $\theta_1$  à partir des informations que nous possédons, c'est à dire les valeurs d'exemple  $(x^{(i)}, y^{(i)})_{i \in \{1, \dots, m\}}$  qui sont respectivement des réalisations de  $(X^{(i)}, Y^{(i)})_{i \in \{1, \dots, m\}}$ .

*Autrement dit :*

- Soit  $\theta = (\theta_0, \theta_1) \in \mathbb{R}^2$  fixé (bien qu'inconnu).
- Soit  $(X_1, \dots, X_m)$  des variables aléatoires indépendantes et identiquement distribuées de loi  $\mathbb{P}_\theta$ .
- On souhaite estimer les  $\theta_i$ .

### Définition(Estimateur)

Un estimateur de  $\theta_i$  est une fonction  $\hat{\theta}_i := h_m(X_1, \dots, X_m)$  où  $h_m : \mathbb{R}^m \mapsto \mathbb{R}$ .

*Exemple :* Soit  $(X_1, \dots, X_k)$  des variables aléatoires indépendantes et identiquement distribuées de loi  $\mathbb{P}_\mu$ . Alors un estimateur de l'espérance  $\mu$  des  $X_i$



respectifs est  $\hat{\mu} = \frac{1}{k} \sum_{i=1}^k X_i$ .

*Remarque* : Un estimateur est aussi une variable aléatoire. On peut calculer son espérance et sa variance.

### Définition(Estimation)

Soient  $x^{(i)}$  des valeurs expérimentales pour chaque  $X^{(i)}$ .

Alors  $h_m(x_1, \dots, x_m)$  est une estimation de  $\theta_i$ .

*Exemple* : Soit  $(X_1, \dots, X_k)$  des variables aléatoires indépendantes et identiquement distribuées de loi  $P_\mu$  ayant pour réalisation  $(x_1, \dots, x_k)$  respectivement. Alors une estimation de l'espérance  $\mu$  des  $X_i$  est  $\frac{1}{k} \sum_{i=1}^k x_i$ .

En statistique, pour évaluer la "qualité" d'un estimateur, on utilise une fonction appelée *l'erreur quadratique moyenne*, que l'on note MSE.

Soit  $\Theta_i$  l'ensemble des estimateurs possibles de  $\theta_i$ .  $\text{MSE}_i : \Theta_i \mapsto \mathbb{R}$  tel que :

$$\text{MSE}_i(\hat{\theta}_i) := \mathbb{E}((\hat{\theta}_i - \theta_i)^2) = \text{Bias}(\hat{\theta}_i)^2 + \text{Var}(\hat{\theta}_i).$$

En effet, plus la valeur de l'erreur quadratique est petite, *meilleur* est l'estimateur.

Définissons le biais d'un estimateur.

### Définition(Biais d'un estimateur)

On appelle *bias* d'un estimateur  $\theta_i$  la fonction, notée *Biais*, définie par :

$$\text{Biais}(\hat{\theta}_i) = \mathbb{E}(\hat{\theta}_i) - \theta_i, \text{ avec } \text{Biais} : \Theta_i \mapsto \mathbb{R}.$$

On utilise aussi la notation  $b_{\theta_i}$ .

*Exemple* : Considérons les variables aléatoires et l'estimateur de  $\mu$  donné en exemple plus haut  $\hat{\mu} = \frac{1}{k} \sum_{i=1}^k X_i$ . Calculons l'espérance de  $\hat{\mu}$ ,  $\mathbb{E}(\hat{\mu}) = \frac{1}{k} \sum_{i=1}^k \mathbb{E}(X_i) = \frac{1}{k} k\mu = \mu$ . On se rend compte que l'estimateur  $\hat{\mu}$  est sans-biais.

*Remarque* : Un estimateur  $\hat{\theta}_i$  sans-biais est un estimateur dont le biais est nul, en particulier  $\mathbb{E}(\hat{\theta}_i) = \theta_i$ . Auquel cas, l'erreur quadratique moyenne devient  $\text{MSE}_i(\hat{\theta}_i) = \text{Var}(\hat{\theta}_i)$ .

Revenons maintenant à notre problème de régression linéaire.

**Définition** [1, p13]

On appelle estimateurs des moindres carrés de  $\theta_0$  et  $\theta_1$  les estimateurs  $\hat{\theta}_0, \hat{\theta}_1$  obtenus par la minimisation de la quantité :

$$(\hat{\theta}_0, \hat{\theta}_1) = \underset{(\theta_0, \theta_1) \in \mathbb{R}^2}{\operatorname{argmin}} \sum_{i=1}^m (Y^{(i)} - \theta_0 - \theta_1 X^{(i)})^2. \quad (\text{E})$$

Cela correspond au problème d'optimisation à résoudre pour la régression linéaire simple, avec l'utilisation de la fonction de coût des moindres carrés. C'est à dire  $l(x, y) := (x - y)^2$ .

Le théorème de *Gauss-Markov* donné ci-dessous confirme que l'estimateur de nos paramètres, par la méthode des moindres carrés, est le meilleur parmi les estimateurs sans-biais.

**Théorème** (Gauss-Markov) [1, p13]

Parmi les estimateurs sans biais linéaires en  $Y$ , les estimateurs  $\hat{\theta}_i$ , obtenus par (E), sont de variance minimale.

Pour  $i \in \{0, 1\}$ , soit  $\hat{\theta}_i$  l'estimateur des moindres carrés pour  $\theta_i$  et  $\hat{g}_m \in \Theta_i$  un estimateur sans biais quelconque de  $\theta_i$ . Alors le *théorème de Gauss-Markov* explique que  $\text{MSE}(\hat{\theta}_i) \leq \text{MSE}(\hat{g}_m)$ .

Avec le *théorème de Gauss-Markov*, nous possédons maintenant un fort argument en faveur de l'utilisation de la fonction de coût des moindres carrés pour nos régression linéaires.

A présent, résolvons les calculs des estimateurs  $\hat{\theta}_0$  et  $\hat{\theta}_1$ , par la méthode des moindres carrés.

*Notation* : Pour  $M$  une matrice, on note  $M'$  sa transposée.

Donnons ici une première formule de résolution qui sera expliquée dans la section sur la régression linéaire multiple :

$$\hat{\theta} = \begin{pmatrix} \hat{\theta}_0 \\ \hat{\theta}_1 \end{pmatrix} = (X'X)^{-1}X'Y, \quad (3)$$

$$\text{où } X'X \in \text{GL}_n(\mathbb{R}), \quad X = \begin{pmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ \ddots & \ddots \\ 1 & x^{(m)} \end{pmatrix} \in \text{M}_{m,2}(\mathbb{R}), \quad Y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \ddots \\ y^{(m)} \end{pmatrix} \in \mathbb{R}^m.$$

### 1.3 Résolution

Avec les conclusions de la section 1.2, la fonction de coup  $l$  choisie est celle des moindres carrées, et la fonction à minimiser  $J_l$  dans le but d'obtenir le paramètre  $\theta = (\theta_0, \theta_1)$  devient alors  $J$  telle que :

$$\operatorname{argmin}_{\theta \in \mathbb{R}^2} J(\theta) = \operatorname{argmin}_{\theta \in \mathbb{R}^2} \frac{1}{2m} \sum_{i=1}^m l(y^{(i)}, h_{\theta}(x^{(i)})) = \operatorname{argmin}_{\theta \in \mathbb{R}^2} \frac{1}{2m} \sum_{i=1}^m (y^{(i)} - h_{\theta}(x^{(i)}))^2$$

Dans le but d'obtenir ce minimum, deux méthodes sont disponibles.

Ces méthodes sont :

- la descente de gradient,
- la méthode analytique.

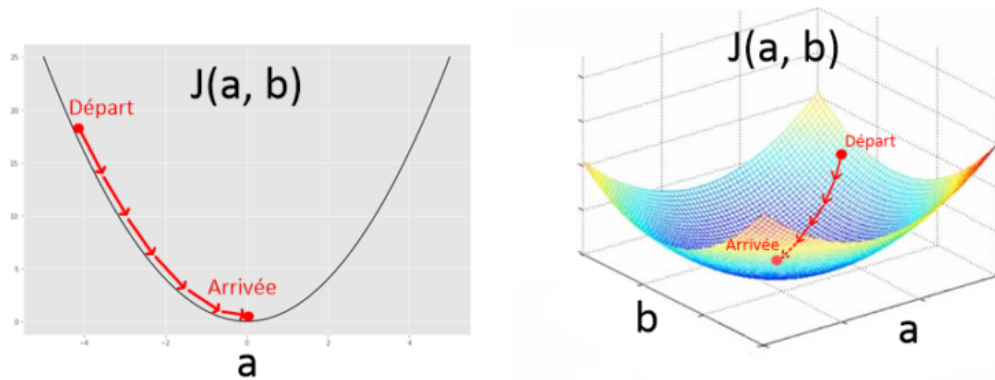
Pour le problème de minimisation, il est important de s'assurer que notre fonction est une fonction convexe. C'est bien le cas du problème de régression linéaire lorsque l'on utilise la fonction de coût des moindres carrés.

La stricte convexité va permettre la mise en place de *la méthode de descente de gradient* et de *la méthode analytique* en s'assurant que le résultat trouvé soit bien le minimum global de la fonction à minimiser  $J$ .

#### Propriété 1

*Une fonction continue strictement convexe sur un ouvert  $\mathcal{O}$  de  $\mathbb{R}^n$ , possède un unique minimum local sur  $\mathcal{O}$ .*

Illustrons la méthode de descente de gradient pour une fonction strictement convexe :



On observe dans le graphique deux représentations de la méthode de descente de gradient pour deux fonctions strictement convexes à 1 et 2 dimensions de départ.

Ces graphes illustrent la forme "parabolique" d'une fonction strictement convexe, on y remarque l'unicité des minimum locaux, donc l'existence d'un minimum global.

Aussi, en observant que le minimum d'une fonction strictement convexe se retrouve à l'unique endroit où la *différentielle/ dérivée est nulle*, on obtient l'intuition principale pour la méthode analytique.

### 1.3.1 Méthode de la descente de gradient

Comme illustré et expliqué ci-dessus, la méthode de la descente de gradient évolue vers son minimum local pas à pas. Or la taille de ce pas doit être définie. Posons  $\alpha \in \mathbb{R}_+$  la taille du pas.

Nous allons implémenter les approximations successives effectuées par  $\theta_0$  et  $\theta_1$  **en même temps** lors du passage de l'itération d'un  $\theta$  vers le suivant. Cette approche itérative est préférable puisqu'elle permet d'effectuer une descente "*directe*" vers le minimum pour en même temps simplifier la récurrence algorithmique jusqu'au résultat. C'est ce qui nous permet d'évoluer sur les deux dimensions dans le graphe, et non pas en "zig-zag".

Les dérivées partielles de  $J$  sont :

$$\begin{cases} \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^m h_{\theta}(x^{(i)}) - y^{(i)} \\ \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x^{(i)} \end{cases}$$

L'algorithme de descente de gradient va donc devoir répéter l'itération suivante jusqu'à arriver au minimum :

$$\begin{cases} \theta_0 = \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m h_{\theta}(x^{(i)}) - y^{(i)} \\ \theta_1 = \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x^{(i)} \end{cases}$$

*Remarque :* Dans la dernière partie du rapport où nous présenterons un exemple de régression, nous aborderons aussi la question du *choix du pas*.

### 1.3.2 Méthode analytique

Ayant observé et énoncé certaines des propriétés d'une fonction strictement convexe, on se rend compte que le point minimum souhaité de  $J(\theta)$  se situe à l'endroit où les dérivées partielles sont égales à zéro.

Il suffit alors de résoudre les équations :

$$\frac{\partial J(\theta_0, \theta_1)}{\partial \theta_0} = 0 \quad \text{et} \quad \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_1} = 0$$

C'est à dire résoudre :

$$\frac{1}{m} \sum_{i=1}^m h_{\theta}(x^{(i)}) - y^{(i)} = 0 \quad \text{et} \quad \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x^{(i)} = 0$$

La résolution de la régression linéaire par l'équation (3) se démontre par le même raisonnement analytique qu'ici utilisé, à l'exception près que l'on considère le problème sous forme matricielle.

## 2 Régression Linéaire multiple

Reprenons l'exemple de l'explication de la quantité maximale d'O3 dans l'air à Rennes. On a d'abord voulu expliquer la quantité maximale d'O3 par un seul paramètre, celui la température à 12h. Si on se décide à vouloir l'expliquer par **la température, la nébulosité** ainsi que **la quantité maximale d'O3 la veille**, alors on utilise *trois* paramètres et notre modélisation rentre maintenant dans le cadre de la régression linéaire multiple.

Ce problème de régression linéaire sera celui traité dans la troisième partie du rapport sur l' "*Utilisation numériques des méthodes*".

### 2.1 Cadre de la regression linéaire multiple

Nous possédons toujours une famille de couple de valeurs,  $(x^{(i)}, y^{(i)})_{i \in \{1, \dots, m\}}$ , avec pour tout  $i \in \{1, \dots, m\}$ ,  $x^{(i)} \in \mathbb{R}^n$  les valeurs explicatives,  $y^{(i)} \in \mathbb{R}$  les valeurs expliquées et  $m \in \mathbb{N}$  le nombre de couples dans la famille.

Les valeurs explicatives contiennent alors plusieurs paramètres.

Concrètement,  $x^{(i)} = \begin{pmatrix} x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{pmatrix} \in \mathbb{R}^n$  avec  $n \geq 2$ .

Soit  $h_\theta : \mathbb{R}^n \mapsto \mathbb{R}$  la fonction qui modélise une régression linéaire multiple à  $n \geq 2$  paramètres :

$$h_\theta(x^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \dots + \theta_n x_n^{(i)},$$
$$\theta = \begin{pmatrix} \theta_0 \\ \vdots \\ \theta_n \end{pmatrix} \in \mathbb{R}^{n+1}.$$

Comme pour toute régression, l'objectif est que  $h_\theta$  puisse prédire au mieux une valeur expliquée, à partir d'une valeur explicative.

Pour tout  $i \in \llbracket 1 ; m \rrbracket$ , considérons le couple de variable aléatoire  $(X^{(i)}, Y^{(i)})$

avec  $X^{(i)} = \begin{pmatrix} X_1^{(i)} \\ \vdots \\ X_n^{(i)} \end{pmatrix}$ .

La famille  $(X^{(i)}, Y^{(i)})_{i \in \{1, \dots, m\}}$  est un  $m$ -échantillon indépendant de loi  $\mathbb{P}_\theta$ , que l'on va introduire maintenant :

$$Y^{(i)} = \sum_{k=1}^n \theta_k X_k^{(i)} + \theta_0 + \varepsilon$$

avec  $\varepsilon \sim \mathcal{N}(0, \sigma^2)$  et  $Y^{(i)} | X^{(i)} \sim \mathcal{N}(\theta_0 + \sum_{k=1}^n \theta_k X_k^{(i)}, \sigma^2)$

Ici encore, la variable aléatoire de l'erreur  $\varepsilon$  est la même pour tous les  $i$ .

Pour tout  $i \in \llbracket 1 ; m \rrbracket$ , on considère que les  $(x^{(i)}, y^{(i)})$  en notre possession sont une réalisations du couple  $(X^{(i)}, Y^{(i)})$ .

De la même manière que pour la régression linéaire simple, on va alors pouvoir construire des estimateurs de paramètres pour les  $\theta_j$ , où  $j \in \llbracket 0 ; n+1 \rrbracket$ .

En reprenant les notations précédemment utilisées, nommons finalement  $J_l : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$  la fonction que l'on souhaite minimiser :

$$\min_{\theta \in \mathbb{R}^{n+1}} J(\theta) = \min_{\theta \in \mathbb{R}^{n+1}} \frac{1}{2m} \sum_{i=1}^m l(y^{(i)}, h_\theta(x^{(i)})).$$

## 2.2 Formes matricielles et première résolution

Puisque la fonction qui modélise les rapports entre  $x$  et  $y$  est linéaire, l'algèbre linéaire permettra de simplifier les notations. Initons les matrices  $X$ ,  $\theta$  et  $Y$  telles que :

$$X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix}, \quad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} \quad \text{et} \quad Y = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

pour  $X \in \mathcal{M}_{m, n+1}(\mathbb{R})$ ,  $\theta \in \mathbb{R}^{n+1}$  et  $Y \in \mathbb{R}^m$  et remarquons que

$$X\theta = \begin{pmatrix} h_\theta(x^{(1)}) \\ h_\theta(x^{(2)}) \\ \vdots \\ h_\theta(x^{(m)}) \end{pmatrix}.$$

Pour des raisons ergonomiques de calculs, considérons à présent que pour tous  $i$ ,  $x_0^{(i)}$  existe tel que  $x_0^{(i)} = 1$ .

*Le théorème de Markov énoncé plus haut s'applique aussi pour la régression linéaire multiple.*

Alors, on va ici encore utiliser la fonction de coût des moindres carrés.

Pour notre régression linéaire multiple à  $m$  entraînement et  $n$  paramètres explicatifs, il nous faut donc résoudre la minimisation sur  $J(\theta)$  suivante :

$$\min_{\theta \in \mathbb{R}^{n+1}} J(\theta) = \min_{\theta \in \mathbb{R}^{n+1}} \frac{1}{2m} \sum_{i=1}^m \left( y^{(i)} - \sum_{j=0}^n \theta_j x_j^{(i)} \right)^2$$

Soit  $M$  une matrice, on va noter  $M'$  sa transposée.

**Définition** (Estimateurs des Moindres-Carrés)[1, 2.2, p38]

On appelle estimateur des moindres carrés de  $\theta$  l'estimateur  $\hat{\theta}$  obtenu par la minimisation de la quantité :

$$\hat{\theta} = \underset{\theta_0, \dots, \theta_n}{\operatorname{argmin}} \sum_{i=1}^m \left( y^{(i)} - \sum_{j=0}^n \theta_j x_j^{(i)} \right)^2 = \underset{\theta \in \mathbb{R}^{n+1}}{\operatorname{argmin}} (Y - X\theta)'(Y - X\theta)$$

Comme annoncé précédemment, il existe une formule qui permet d'obtenir le vecteur solution de l'estimateur des moindres carrés  $\hat{\theta}$ , à partir des matrices  $X$  et  $Y$ .

**Théorème** (Expression de l'estimateur des Moindres Carrés)[1, 2.2, p38]

*Supposons que les vecteurs colonnes de  $X$  soient indépendants.*

*L'estimateur des moindres carrés  $\hat{\theta}$  de  $\theta$  vaut :*

$$\hat{\theta} = (X'X)^{-1}X'Y$$



On suppose que les vecteurs colonnes de  $X$  sont linéairement indépendants les uns des autres afin que  $X'X$ , de taille  $n^2$ , puisse être inversible.

L'indépendance linéaire des colonnes de  $X$  signifie que l'on possède plus de valeurs d'entraînement que de paramètres explicatifs ( $m > n$ ) et, sans redondance d'information dans nos valeurs d'entraînement.

Pour trouver le résultat, nous procédons dans la même idée que pour la méthode analytique, c'est à dire en calculant la matrice gradient de  $J$  par rapport à  $\theta$ , de  $J(\theta) = (Y - X\theta)'(Y - X\theta)$  tel que  $\nabla_{\theta}J(\theta) = 0$ .

Or, s'en suit un calcul de gradient matriciel avec pour grandes étapes :

$$\nabla_{\theta}J(\theta) = 0 \Leftrightarrow X'X\theta - X'Y = 0 \Leftrightarrow \theta = (X'X)^{-1}X'Y.$$

*Remarque :* Le théorème peut aussi s'expliquer par une considération géométrique en partant du sous-espace vectoriel  $\mathcal{F} = Vect\{\mathbf{1}, (x^{(i)})_{i \in \{1, \dots, m\}}\}$  dans  $\mathbb{R}^n$ .

## 2.3 La descente de gradient pour la régression linéaire multiple

Puisque la fonction à minimiser  $J(\theta)$  est strictement convexe, on peut aussi utiliser la méthode de descente de gradient pour la régression linéaire multiple.

Le procédé est sensiblement le même que pour la régression linéaire simple. Nous allons répéter la mise à jour des valeurs de  $\theta$  en fonction de ses dérivées partielles.

$$\forall 1 \leq j \leq n, \quad \frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Alors, avant chaque correction de  $J(\theta)$ ,  $\theta$  se met à jour de façon à ce que pour tous  $j$  de  $\llbracket 1, m \rrbracket$  :

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}.$$

En appliquant cet algorithme on trouve le  $\theta$  qui minimise le coût quadratique et qui correspond à notre estimation des moindres carrés.

### 3 Régression Logistique

Il existe d'autres types de régression, notamment la régression logistique. La régression logistique est utilisée pour les problèmes dits de "classification".

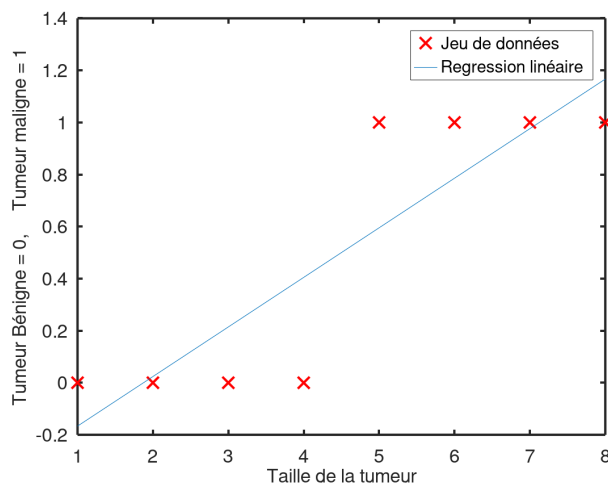
Dans un problème de classification on souhaite connaître la probabilité des valeurs expliquées  $y^{(i)} \in \mathcal{I} \subset \mathbb{N}$ , en fonction de valeurs explicatives  $x^{(i)}$ . En effet, les valeurs expliquées symbolisent une "catégorie" plus qu'un chiffre. On dit alors que la valeur expliquée est représentée par une variable aléatoire *qualitative*.

*Exemple :* Considérons la classification qui, en fonction de la taille d'une tumeur, prédit si la tumeur est bénigne (i.e non mortelle) ou bien maligne. Les valeurs expliquées ne peuvent choisir que parmi deux catégories lesquelles on peut assigner à  $y = 0$  pour la première et à  $y = 1$  pour la seconde.

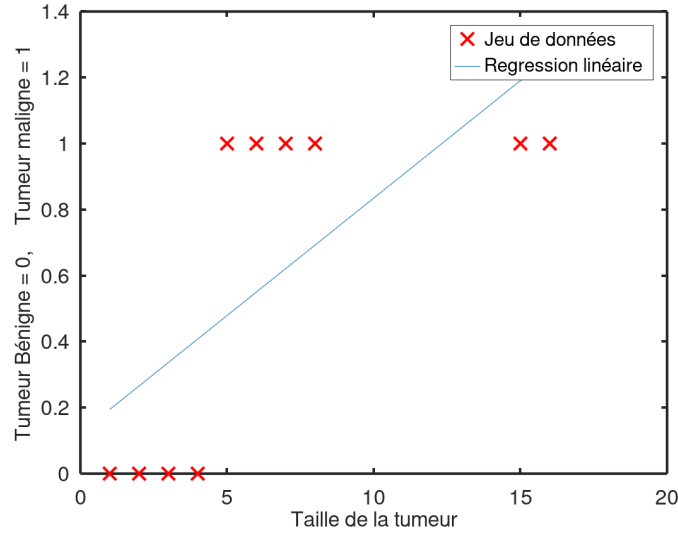
Montrons pourquoi, dans ces problèmes, la régression linéaire n'est pas adaptée.

Imaginons alors un jeu de données, à priori cohérent, qui a pour valeurs explicatives la taille d'une tumeur et pour valeurs expliquées la nature "bénigne" ou "maligne" de la tumeur.

Pour un jeu de données équilibré, la régression linéaire à l'air de plutôt bien s'en sortir avec  $\theta_{\text{solution}} = (0.36, 0.19)^T$  :



Par contre, si l'on rajoute des valeurs un peu plus déséquilibrantes, la régression linéaire en est facilement perturbée avec  $\theta_{\text{solution}} = (0.12, 0.07)^T$  :



C'est dans le but de remédier à de tels défauts pour la régression linéaire que l'on va recourir à la régression logistique qui a, pour les deux jeux de données offerts en exemple ci-dessus, le même  $\theta$  solution au centième près.

Nous allons d'abord étudier le cas de classification où seulement deux classes peuvent être résultat. On peut alors poser  $Y \in \{0, 1\}$  en considérant  $Y$  comme une variable aléatoire de loi de Bernoulli.

### 3.1 Fonction de modélisation

Considérons un problème de classification avec un jeu de données  $(x^{(i)}, y^{(i)})_{i \in \{1, \dots, m\}}$  où  $\forall 1 \leq i \leq m$ ,  $x^{(i)} \in \mathbb{R}^n$ ,  $y^{(i)} \in \{0, 1\}$ , avec  $n \in \mathbb{N}^*$  le nombre de paramètres explicatifs.

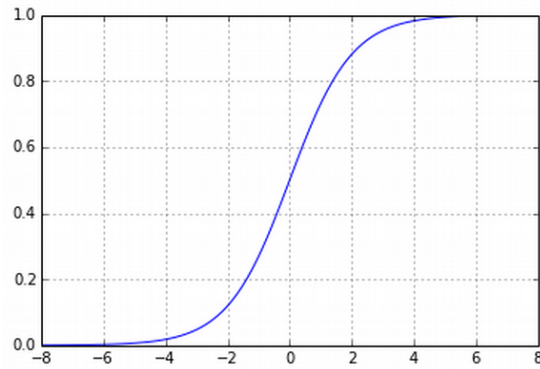
Pour la régression logistique dans une classification à deux classes, on souhaiterait une fonction  $h_\theta$  qui ne rend pas de valeurs en dehors de l'intervalle  $[0, 1]$  tel que  $h_\theta : \mathbb{R}^n \mapsto [0, 1]$ .

On souhaiterait aussi que la fonction tende rapidement vers ses bornes  $Y = 0$  d'un côté et  $Y = 1$  de l'autre de son point  $x \in \mathbb{R}^{n+1}$  tel que  $h_\theta(x) = 0,5$ . Avec pour objectif une prédiction claire de la classe.

Une fonction qui correspond à cette idée est la fonction logistique. Notons-la  $g : \mathbb{R} \mapsto ]0, 1[$ , cette fonction a pour expression :

$$g(u) = \frac{1}{1 + e^u}.$$

Son graphe, qui illustre bien les propriétés voulues, est le suivant :



Cette fonction donne l'appellation régression logistique. La fonction logistique est strictement monotone et tend vers 0 quand  $u \rightarrow -\infty$  ou vers 1 quand  $u \rightarrow +\infty$ . Surtout, la fonction logistique "accélère" vers 1 à droite de  $u = 0$  et vers 0 à gauche de  $u = 0$ .

Soient  $g$  la fonction logistique,  $X \in \mathbb{R}^n$  un vecteur explicatif,  $X_1 \in \mathbb{R}^{n+1}$  correspond à  $X$  auquel on a rajouter 1 comme première coordonnée, et  $\theta \in \mathbb{R}^{n+1}$  les paramètres que l'on souhaite trouver. Présentons alors notre fonction de modélisation de régression logistique  $h_\theta : \mathbb{R}^n \mapsto ]0, 1[$  :

$$h_\theta(X) = g(\theta^T X_1).$$

Avec

$$\begin{aligned} \mathbb{P}(Y^{(i)} = 1 | X^{(i)} = x^{(i)}) &= h_\theta(x^{(i)}), \\ \mathbb{P}(Y^{(i)} = 0 | X^{(i)} = x^{(i)}) &= 1 - h_\theta(x^{(i)}). \end{aligned}$$

## 3.2 Fonction de coût

La fonction de coût pour la régression logistique ne peut pas être celle des moindres carrés, en effet, elle ne rend même pas convexe la fonction à minimiser. Comme pour la régression linéaire, nous souhaitons trouver une fonction de coût issue d'un "bon" estimateur.

Pour trouver ce bon estimateur, on utilise *la méthode de l'estimateur de maximum de vraisemblance*, développée en 1922, par *Ronald Aymler Fisher*, une vingtaine/trentaine d'années avant l'apparition de la régression logistique par *Joseph Berkson*.

Définissons d'abord ce qu'est la fonction de vraisemblance :

### Définition (Fonction de vraisemblance)

Soit  $f_\theta$  la fonction loi de densité de la loi de probabilité  $\mathbb{P}_\theta$ .

La fonction de vraisemblance est la densité de  $(X^{(1)}, \dots, X^{(m)})$  sous  $\mathbb{P}_\theta$  pour leur réalisation  $(x^{(1)}, \dots, x^{(m)}) \mapsto L_\theta(x^{(1)}, \dots, x^{(m)})$  avec :

$$L_\theta(x^{(1)}, \dots, x^{(m)}) = f_\theta(x^{(1)})f_\theta(x^{(2)})\dots f_\theta(x^{(m)}).$$

On appelle "*estimateur du maximum de vraisemblance*", l'estimateur d'un paramètre  $\theta$  obtenu en calculant le maximum de la vraisemblance par rapport au paramètre  $\theta$ . Appelons  $\hat{\theta}$  cet estimateur, ayant pour équation :

$$\hat{\theta} = \operatorname{argmax}_{\theta \in \mathbb{R}^{(n+1)}} L_\theta(x^{(1)}, \dots, x^{(m)}).$$

*Remarque* : L'estimateur de vraisemblance est le type d'estimateur le plus utilisé en statistiques. Ses multiples bonnes propriétés en sont la raison. Le lecteur est renvoyé vers le chapitre des estimateurs de la référence [2], livre de Michel Lejeune, pour plus de détails à ce sujet.

*Remarque* : Pour la régression linéaire, l'estimateur des moindres carrés est aussi l'estimateur maximal de vraisemblance.

Revenons à notre problème de départ et calculons alors l'estimateur du maximum de vraisemblance.

Pour la régression logistique, nous avons comme fonction de densité  $f_\theta$  :

$$f_\theta(x^{(i)}) = \begin{cases} 1 - h_\theta(x^{(i)}) & \text{pour } y^{(i)} = 0 \\ h_\theta(x^{(i)}) & \text{pour } y^{(i)} = 1 \end{cases}, \text{ d'où}$$

$$\hat{\theta} = \operatorname{argmax}_{\theta \in \mathbb{R}^{(n+1)}} \prod_{i=1}^m h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}.$$

En cherchant l'*argmax* de  $\log \circ L_\theta$  au lieu de  $L_\theta$ , l'équation se simplifie alors que les solutions restent inchangées. Si, de surcroît, nous inversons le signe de l'équation, il nous reste à résoudre  $J(\theta)$  sous forme de problème d'optimisation :

$$\min_{\theta \in \mathbb{R}^{n+1}} J(\theta) = \min_{\theta \in \mathbb{R}^{n+1}} -\frac{1}{m} \sum_{i=1}^m y^{(i)} (\log(h_\theta(x^{(i)}))) + (1 - y^{(i)}) (\log(1 - h_\theta(x^{(i)}))). \quad (4)$$

La fonction à minimiser  $J(\theta)$  obtenue est strictement convexe, on peut donc utiliser la méthode de descente de gradient pour trouver nos estimations de  $\theta$ .

### 3.3 Résolution

Nous allons résoudre le problème de minimisation par la méthode de descente de gradient.

En comparaison avec la régression linéaire, on retrouve la même formule des gradients malgré la différence entre les fonctions qui modélisent les régressions ainsi qu'entre les fonction de coût :

$$\forall 1 \leq j \leq n, \quad \frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}.$$

Et de la même façon on progresse en répétant, jusqu'à atteindre la solution, la mise à jour de  $\theta$  à chaque pas :

$$\forall 1 \leq j \leq n, \quad \theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}.$$

Une fois les paramètres  $\theta$  trouvés, notre fonction de modélisation de régression logistique  $h_\theta$  est prête à être utilisée.

Pour finir, la prédiction d'une classe  $Y = 0$  ou  $1$ , en fonction d'une valeur explicative  $x$  se fait de la façon suivante :

$$\begin{cases} \text{Si } h_\theta(x) < 0,5 & \text{alors } x \text{ prédit la classe } y = 0 \\ \text{Si } h_\theta(x) > 0,5 & \text{alors } x \text{ prédit la classe } y = 1 \end{cases}.$$

### 3.4 Classification à $n > 2$ classes

Pour une régression logistique, lorsque l'on se retrouve face à un problème de classification avec plus de 2 classes, ils nous faut utiliser la méthode dite du "un contre tous".

Il faut développer, en faveur de chaque classe  $j$  et contre toutes les autres classes, une fonction qui modélise la régression logistique  $h_\theta^{(j)}$ . On obtient alors un vecteur contenant ces fonctions en faveur de chaque classe.

*Exemple :* Soit une régression logistique à  $p \in \mathbb{N}^*$  classes, et  $x \in \mathbb{R}^n$  une valeur explicative. Alors :

$$h_\theta(x) = \begin{pmatrix} h_\theta^{(1)}(x) \\ \vdots \\ h_\theta^{(j)}(x) \\ \vdots \\ h_\theta^{(p)}(x) \end{pmatrix} \in [0, 1]^p.$$

En effet, ce que l'on va faire c'est associer la classe soutenue à  $Y = 1$ , et puis, associer toutes les autres classes à la valeur  $Y = 0$ . Cela va nous permettre de rester dans le cadre d'une classification à 2 classes.

*Exemple :* Soit une régression logistique à 3 classes.

Pour  $h_\theta^{(j)}$ , nous changeons le jeu de donnée de départ de façon à ce que :

$$\begin{cases} Y = 1 & \text{pour } y^{(i)} = j \\ Y = 0 & \text{pour } y^{(i)} \neq j \end{cases}.$$

Supposons  $x$  une valeur explicative, nous souhaitons pouvoir prédire à quelle classe  $j \in \{1, \dots, p\}$ , la valeur explicative  $x$  doit être associée. La sélection se fait en comparant les valeurs  $(h_\theta^{(j)}(x))_{j \in \{1, \dots, p\}}$ .

Remarquons que  $h_\theta^{(j)}(x) = \mathbb{P}(Y^{(i)} = j | X^{(i)} = x)$ .

L'indice  $j \in \{1, \dots, p\}$  auquel  $h_\theta^{(j)}(x)$  aura la valeur la plus élevée sera celle sélectionnée.

*Exemple* : Soit  $x$  une valeur explicative et  $i = \operatorname{argmax}_j h_\theta^{(j)}(x)$ .

La donnée explicative  $x$  prédit  $y = i$  et donc, on prédit que la valeur explicative  $x$  est associée à la classe  $i$ .



## Deuxième partie

# Réseaux de neurones

Pour le problème de classification, la régression logistique fonctionne très bien dans la pratique pour des variables explicatives  $x^{(i)} \in \mathbb{R}^n$  de petite dimension  $n$ . A chaque aggrandissement  $n + 1$  du nombre de paramètre des variables explicatives, la quantité de calcul nécessaire via la régression logistique augmente beaucoup trop (cf fonction `mapFeature` III.2.2). Face à ce défaut, les réseaux de neurones apparaîtront comme la solution et encore plus.

Les réseaux de neurones tel que l'on les présentera ont été inventés dans les années 80. Avant cela, est apparu en 1943 le concept de neurone formel ayant pour but de formaliser le fonctionnement d'un neurone biologique.

On qualifie les réseaux de neurones de *boite noire* pour la raison qu'il est difficile de comprendre ce qu'il s'y passe réellement, notamment l'impact de chaque neurone sur les valeur explicatives.

Cette partie sur les réseau de neurones prend comme source les cours d'Andrew Ng, sur son cours à Stanford (la troisième référence [4]) et son cours en ligne *Machine Learning* sur *Coursera*.

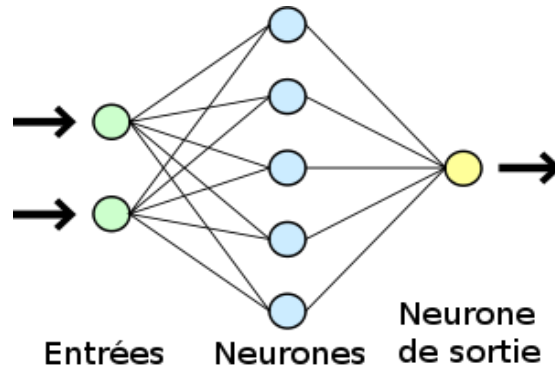
Nous allons présenter dans une première section la forme d'un réseaux de neurones, ensuite nous expliquerons le fonctionnement des réseaux de neurones pour finir en essayant d'offrir une intuition sur la raison de leur fonctionnement.

Cette partie n'approfondira pas le raisonnement mathématique ainsi que les calculs permettant de trouver ce qui y est présenté. Elle a pour but d'introduire la notion de réseau de neurones et présenter ses mécanismes de fonctionnement, une fois cela fait il sera possible de s'attaquer à la programmation. Pour plus de détail sur les réseaux de neurones, consulter la référence [6][Chapitre 11].

## 4 Fonctionnement d'un réseau de neurones

### 4.1 Introduction

Voici une représentation schématique d'un réseau de neurones :



Comme représenté sur le schéma, on divise un réseaux de neurones en trois régions distinctes :

- (1) La couche de neurones des variables d'entrée
- (2) Les couches de neurones cachées
- (3) La couche de neurones de sortie

(1) La couche des variables d'entrée est celle qui concerne les valeurs explicatives, elle aura un nombre égal de "neurones" à celui du nombres de ses paramètres d'entrée, c'est à dire la dimension  $n$  de ses valeurs explicatives.

(2) Les couches de neurones cachées vont contenir un nombre de neurones déterminés par le programmeur. Elles sont celles qui suggère l'appellation "boite noire" des réseaux de neurones, en effet il est difficile de décrire précisément qui fait quoi tant cela diffère pour chaque problème. C'est aussi le coeur du fonctionnement des réseaux de neurones.

(3) La couche de neurones de sortie a le même nombres de neurones que de classe dans le problème de classification. Ses neurones vont contenir les valeurs de probabilités assignées à chaque classe pour la variable explicative en

entrée. En outre, le resultat de notre prédiction par notre réseau de neurones.

Un réseau de neurones progresse en allant de la couche des variables d'entrée ou explicatives vers les variables de sorties. Excepté la couche d'entrée, on va calculer une couche en fonction des valeurs de la couche précédente.

Il est *important* de noter qu'à chaque couche, exepté celle des neurones de sortie, on rajoute un neurone de biais(= 1), non calculé par la couche de neurone précédente mais qui participe uniquement au calcul de la couche suivante. **Expliqué plus bas ?** Elles n'ont donc pas été représentés dans le schéma plus haut.

Détaillons ensuite le processus de calcul effectué par un réseau de neurones, de ses valeurs d'entrée(explicatives) jusqu'à arriver à ses valeurs sortantes(expliquées).

## 4.2 Propagation en avant

Comme énoncé dans l'introduction, un réseaux de neurones se propage en allant des variables d'entrées vers les variables de sorties. En partant de nos données de départ, on "transforme" ces données de départ étape par étape, en passant de couche en couche de neurones.

Une étape désigne alors le passage d'une couche de neurones à la suivante.

Une fois le réseau de neurones "entraîné", à partir d'un vecteur explicatif, on évalue une solution en effectuant cette "propagation en avant" que l'on va maintenant décrire.

### 4.2.1 La matrice de passage

En effet, pour le passage d'une couche à une autre, on utilisera une matrice. Appelons cette matrice la matrice de passage  $\Theta^{(i)}$  que l'on numérote en fonction de sa position.

*Exemple :* La matrice de passage de la couche de neurones 1 vers la couche de neurones 2 est la matrice  $\Theta^{(1)}$ , ainsi que la matrice de passage de la couche  $l$  vers la couche  $l + 1$  est la matrice  $\Theta^{(l)}$ .

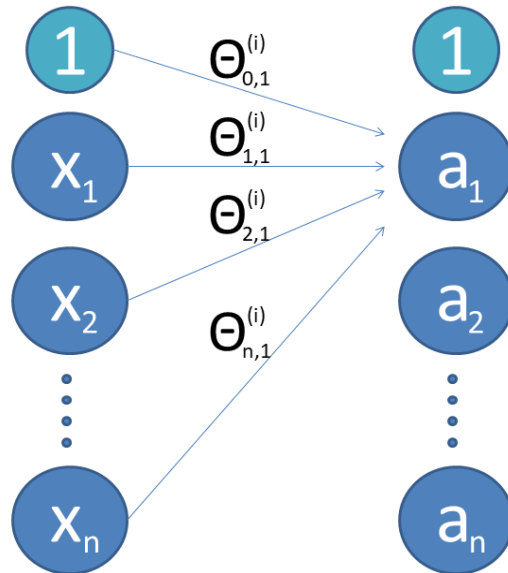
Soit  $s_l$  le nombre de neurones dans la couche  $l$  en comptant ici le neurone du biais. La matrice  $\Theta^{(l)}$  de passage de la  $l^{\text{ème}}$  couche vers la  $l + 1^{\text{ème}}$  couche est de taille  $(s_{l+1} - 1)$  lignes et  $s_l$  colonnes.

Une exception existe néanmoins, soit  $l_{\max}$  la dernière couche du réseau de neurones, alors la dernière matrice de passage  $\Theta^{(l_{\max}-1)}$  est de taille  $s_{l_{\max}}$  lignes et  $s_l$  colonnes. On le remarque en se rappelant que la couche de sortie n'a pas de neurone de biais.

#### 4.2.2 Le calcul de propagation en avant

Pour décrire la propagation en avant, il suffit de décrire une seule étape, le passage d'une couche à une autre.

Prenons appui sur l'image suivante :



Pour avancer de la  $i^{\text{ème}}$  couche de taille  $n$  vers la suivante de taille  $n$ , la matrice de passage est alors de taille  $n + 1$  lignes et  $n$  colonnes.

Le calcul effectué par un réseau de neurones lors d'une propagation en avant pour obtenir les valeurs après passage à la couche de neurones suivante sont :

$$\begin{aligned} a_1 &= g(\Theta_{1,0}^{(i)}x_0 + \Theta_{1,1}^{(i)}x_1 + \dots + \Theta_{1,n}^{(i)}x_n) = g(\Theta_1^{(i)}x), \\ a_2 &= g(\Theta_{2,0}^{(i)}x_0 + \Theta_{2,1}^{(i)}x_1 + \dots + \Theta_{2,n}^{(i)}x_n), \\ &\vdots \\ a_n &= g(\Theta_{n,0}^{(i)}x_0 + \Theta_{n,1}^{(i)}x_1 + \dots + \Theta_{n,n}^{(i)}x_n) \end{aligned}$$

Avec  $g$  la fonction logistique,  $\Theta_j^{(i)}$  le  $j$ -vecteur colonne de  $\Theta^{(i)}$  et  $x = (1, x_1, \dots, x_n)$ .

Le calcul explique finalement la taille des matrices de passage.

Appelons  $a^{(i)}$  le vecteur ligne des neurones à la  $i^{\text{ème}}$  couche et soit  $n_{(i)} + 1$  le nombre de neurones de la couche suivante. Considérons aussi la valeur d'entraînement de départ  $x^{(i)} = a^{(1)}$ .

On peut donc généraliser la formule de passage d'une couche  $i$  à la couche  $i + 1$  sous la forme suivante :

$$\forall 1 \leq j \leq n_{(i)}, \quad a_j^{(i+1)} = g(\Theta_j^{(i)}a^{(i)}).$$

### 4.3 Rétro-propagation

Essentiellement, les matrices de passage sont les paramètres à trouver pour que notre réseau de neurones fonctionne au mieux.

Il nous faut "entraîner" notre réseau de neurones pour qu'il "apprenne" des valeurs d'entraînement (ou jeu de donnée) qui lui sont fournies.

La rétro-propagation est la méthode utilisée pour effectuer la descente de gradient qui permet de trouver les "meilleures" matrices de passage.

Pour la rétro-propagation, de la même façon qu'avec notre fonction de coût dans la partie régression, nous aurons une fonction de coût qui va permettre de trouver les "meilleures" matrice de passage via une descente de gradient.

#### 4.3.1 Présentation et notation

Soit  $K \in \mathbb{N}$  le nombre de classes de la valeur expliquée.  
Soit  $h_{\Theta}(x^{(i)})_k$  la valeur prédite pour la  $k$ -ème classe/paramètre après propagation en avant avec  $\Theta$  le symbole de l'ensemble des matrices de passage utilisées.

La fonction de coût  $J$  du réseau de neurones :

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\Theta}(x^{(i)})_k) + (1 - y_k^{(i)}) (\log(1 - h_{\Theta}(x^{(i)})_k)).$$

Puisque l'on a plusieurs matrices de paramètres à trouver, contrairement à un vecteur pour les régression linéaires et logistique, le calcul du gradient sera aussi plus difficile.

#### 4.3.2 Calcul du gradient de la fonction de coût

Soit  $x^{(i)}$ , associée à la valeur expliquée  $y^{(i)}$ , la valeur d'entraînement de départ que nous utilisons comme exemple. Posons  $x^{(1)} = a^{(1)}$ .

Soit  $g$  la fonction logistique, reprenons les notations de propagation en avant. Considérons, pour tout passage d'une couche à la suivante dans le sens "en avant", comme le passage d'une couche de taille  $n \in \mathbb{N}^*$  vers une autre de taille  $n' \in \mathbb{N}^*$ .

$$\text{Avec, pour } l \neq 1, z^{(l)} = \begin{pmatrix} z_1^{(l)} = \Theta_1^{(l-1)} a^{(l-1)} \\ \vdots \\ z_{n'}^{(l)} = \Theta_{n'}^{(l-1)} a^{(l-1)} \end{pmatrix} \text{ et } a^{(l)} = \begin{pmatrix} a_1^{(l)} = g(z_1^{(l)}) \\ \vdots \\ a_{n'}^{(l)} \end{pmatrix}.$$

Calculons les vecteurs d'erreur par couche, avec en indice supérieur leur couche  $\delta^{(\text{couche})}$ . On sépare deux cas, le vecteur d'erreur  $l_{\max}$  de la dernière couche, qui s'avère être le premier que l'on calcule, et tous les autres pour  $2 \leq j \leq l_{\max} - 1$ .

Soit  $\langle ., . \rangle$  le produit scalaire usuel euclidien,  $g'$  la fonction dérivée de la fonction logistique, et  $l_{\max}$  la dernière couche du réseau de neurone.

$$\begin{cases} \delta^{(l_{\max})} = a^{(l_{\max})} - y^{(i)}, \\ \delta^{(l)} = \langle (\Theta^{(l)})^T \delta^{(l+1)}, g'(z^{(3)}) \rangle, \quad \text{pour } 2 \leq l \leq l_{\max} - 1 \end{cases}.$$

Dans l'idée, on effectue le calcul de gradient en traversant le réseau de neurones dans le sens inverse.

En partant des neurones les plus proches de l'algorithme, nous calculerons leurs erreur, qui nous permettront de calculer celle des neurones précédent et ainsi de suite jusqu'à remonter aux derniers neurones.

Pour compléter l'entraînement du réseau de neurones, il va falloir effectuer la rétro propagation pour toutes les valeurs d'entraînement.

En effet, il faut ensuite répéter pour toute couche  $1 \leq l \leq l_{\max} - 1$ , et  $i, j$  qui recouvrent toutes les coordonnées d'une matrice de même taille que  $\Theta^{(l)}$ .

$$\Delta_{i,j}^{(l)} = \sum_{i=1}^m a_j^{(l)} \delta_i^{(l+1)}$$

Enfin, pour tout  $1 \leq l \leq l_{\max} - 1$ , soit  $D^{(l)}$  une matrice de la même taille que  $\Delta^{(l)}$  et  $\Theta$  telle que :

$$D_{i,j}^{(l)} := \frac{1}{m} \Delta_{i,j}^{(l)} \quad | \quad \frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta) = D_{i,j}^{(l)}$$

*Remarque :* Dans la pratique, on rajoute un terme de régularisation (expliqué dans la partie III.2) aux termes des matrices  $D^{(l)}$ .

L'entraînement d'un réseau de neurones se fait alors en réitérant l'algorithme de rétro-propagation pour toute les valeurs d'entraînement jusqu'à satisfaction.

## 5 Pourquoi les réseaux de neurones fonctionnent-ils ?

Je souhaiterais, dans cette section, pouvoir expliquer l'intuition de la transformation par laquelle passe une valeur d'entraînement via les couches cachées de neurones. A l'image des graphiques présentés en introduction pour la régression linéaire simple et la régression logistique, il est bien plus difficile de trouver d'exemples clairs pour les réseaux de neurones.

### 5.1 Intuition

*Une explication vulgarisée offre un parallèle avec la physique.*

Le monde physique tel que l'on le connaît aujourd'hui est considéré comme ayant "plusieurs profondeurs", plusieurs prismes par lesquels on peut analyser un objet dans l'univers.

*Exemple :* Différentes régions d'onde électromagnétiques (radio, visible, gamma, X, ...).

Les réseaux de neurones vont alors établir une "organisation hiérarchique" des informations avec chaque couche qui représentera **sa** profondeur.

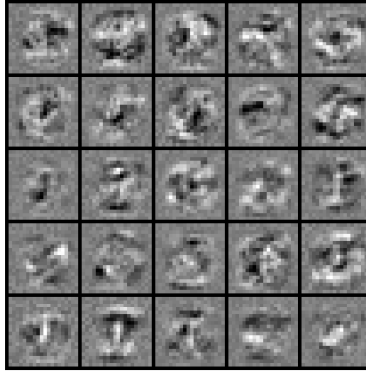
*Exemple :* Ondes radio pour la première couche, ondes visibles pour la seconde, ...

### 5.2 Exemple d'un réseau de neurone entraîné

Présentons un exemple concret d'un réseau de neurones entraîné à la reconnaissance des chiffres.

Observons l'aspect de données passées par la seule couche cachée du réseau de neurones





On interprète l'image d'un chiffre passé par la couche cachée comme une identification des *traits et des formes*.

Cela paraît être en accord avec l'idée de la décomposition à toute échelle et/ou caractéristique par chaque couche du réseau de neurones.

## Troisième partie

# Utilisations numériques des méthodes

Dans cette dernière partie, nous allons montrer des cas concrets d'utilisation des méthodes introduites dans la première partie du devoir. Nous présenterons un exemple pour chacun des procédés qui y sont détaillés, sur la régression linéaire ainsi que sur la régression logistique.

Chaque algorithme présenté sera accompagné de commentaires expliquant ligne par ligne les instructions codées en Octave.

## 6 La régression linéaire

Dans cette section sur la régression linéaire, nous allons utiliser les deux processus de résolution présentés dans la partie 1, la résolution par la descente de gradient et la résolution par la méthode analytique.

Munissons nous du problème de régression linéaire introduit dans la partie *Régression*.

Nous considérons ici une régression linéaire multiple puisque l'algorithmique de résolution présenté fonctionne aussi pour une régression linéaire simple.

La fonction de coût et la fonction de descente de gradient seront dans un premier temps présentées. Elles seront importantes pour la résolution du problème, ainsi que pour analyser les résultats obtenus.

## 6.1 Algorithmes pour la fonction de coût et la descente de gradient

### 6.1.1 Fonction de coût

Rappelons la fonction de coût des moindres carrés  $J(\theta)$  énoncée dans la partie 1 :

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m \left( y^{(i)} - \sum_{j=0}^n \theta_j x_j^{(i)} \right)^2 = \frac{1}{2m} (Y - X\theta)'(Y - X\theta).$$

Voici alors une fonction codée qui calcul le coût d'une regression linéaire :

```
1 function J = computeCost(X, y, theta)
2 % J = COMPUTECOST(X, y, theta) calcul le coût d'utiliser theta
3 % comme le parametre pour la regression lineaire entre X et y
4
5 m = length(y); % nombre d'exemples d'entrainement
6
7 J = 0; % initialisation de la variable du cout
8
9 A = X*theta - y; % Vecteur colonne contenant à chaque lignes un élément
10 % de la somme de J
11 A = A.^2; % Les termes du vecteur A mis aux carré
12
13 J = sum(A)/(2*m); % sommation finale
14
15 end
```

### 6.1.2 Fonction de descente de gradient

Rappelons le procédé de descente de gradient énoncé dans la partie 1.

Nous allons répéter la mise à jour des valeurs de  $\theta$  en fonction de ses dérivées partielles :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Alors, avant chaque correction de  $J(\theta)$ ,  $\theta$  se met à jour de façon à ce que pour tous  $j$  :

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Voici alors une fonction codée qui effectue une descente de gradient pour une régression entre  $X$  et  $y$  avec  $alpha$  le pas de la descente et  $num\_iter$  le nombre de pas effectués :

```

1  function [theta, J_history] = gradientDescent(X, y, theta, alpha, num_iters)
2  %GRADIENTDESCENT effectue une descente de gradient pour connaitre theta
3  %   actualise theta en prenant num_iters nombre de pas, de taille alpha
4
5  m = length(y); % nombre de valeurs d'entraînement
6  J_history = zeros(num_iters, 1); % initialisation
7
8  for iter = 1:num_iters
9
10     M = X.*(X*theta - y); % obtention de la matrice m*(n+1) contenant dans
11     % sa j-ème colonne le j-ème terme de la somme pour toute les expériences
12
13     somme = sum(M); % addition des colonnes de la matrice et obtention du
14     % vecteur ligne des résultats des sommes pour chaque paramètre de theta
15
16     sous = alpha.*somme.*(1/m); % completion de la formule de gradient connue,
17     % avec alpha le pas, et 1/m la redimension
18
19     theta = theta - sous'; % soustraction du dernier theta par le gradient
20     % redimensionné et mesuré par son pas
21
22     J_history(iter) = computeCost(X, y, theta); % à chaque itération de theta,
23     % on va stocker le coût pour cette valeur de theta
24 end
25 end

```

Aussi,  $J\_history$  nous permet de vérifier le coût à chaque  $\theta$  en cas de bug. On évite normalement de l'inclure dans l'algorithme en raison du coût en calcul.

### 6.1.3 Fonction de calcul de la méthode analytique

Par la méthode analytique on a pour solution  $\theta$  :

$$\theta = (X'X)^{-1}X'Y \quad \text{avec}$$

$$X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix}, \quad \theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{pmatrix} \quad \text{et} \quad Y = \begin{pmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{pmatrix}$$

pour  $X \in \mathcal{M}_{m,n+1}(\mathbb{R})$ ,  $\theta \in \mathbb{R}^{n+1}$  et  $Y \in \mathbb{R}^m$ .

Un algorithme qui donne en sortie un tel  $\theta$  en fonction de données explicatives matriciellement  $X$  et de données expliquées  $y$  est le suivant :

```

1  function [theta] = normalEqn(X, y)
2  %  NORMALEQN(X,y) calcul theta selon la méthode analytique
3  %  pour une regression linéaire.
4
5
6  theta = zeros(size(X, 2), 1); % Initialisation du vecteur theta
7  inverse = pinv(X'*X); % Inverse du produit de X et sa transposée
8  temp = X'*y; % Calcul des deux termes restants
9  theta = inverse*temp; % Produit total
10
11
12 end

```

## 6.2 Résolution complète

Soit 'ozone.txt' le fichier contenant les données du problème de l'ozone présenté en introduction dans la partie de la régression linéaire. Chaque ligne représente des valeurs météorologiques prélevées durant la même journée.

Nous supposons qu'il existe une relation entre la température à 12h, la nébulosité à 9h et la quantité maximale d'ozone(O3) la veille pour prédire la quantité maximale d'ozone(O3) dans l'air durant la même journée, à Rennes durant l'été 2001. On trouve ces valeurs explicatives respectivement sur les colonnes 4, 6 et 11 du fichier 'ozone.txt'.

Compiler le code suivant résoudrais alors le problème de regression linéaire multiple de l'ozone :

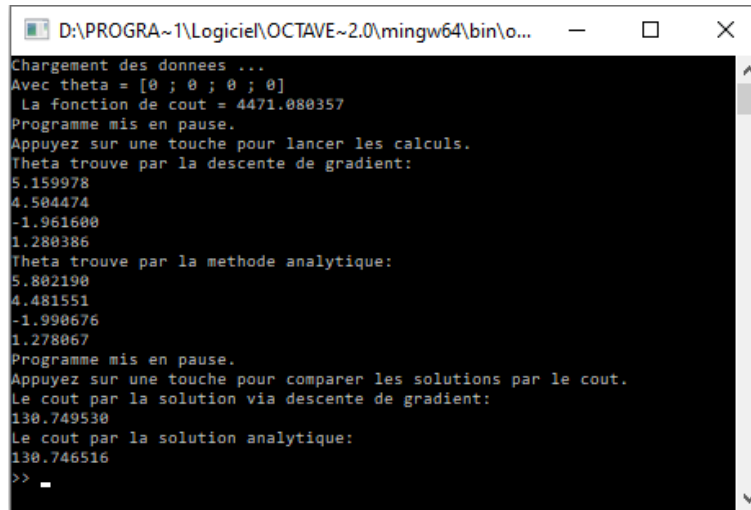
```
1  % Modéliser l'ozone O3 par rapport à la température à 12h,
2  % la nébulosité à 9h et la quantité maximale d'ozone(O3) la veille
3
4  %% ===== Part 1-Initialisation des paramètres =====
5
6  %% Fermeture et effacement de toutes les figures ou variable existantes
7  clear ; close all; clc
8
9  fprintf('Chargement des donnees ...\n');
10
11  %% Chargement des données
12  data = load('ozone.txt');
13  X = data(:, [4, 6, 11]);
14  y = data(:, 2);
15  m = length(y);
16
17  % Ajout des 1 pour theta_0
18  X = [ones(m, 1) X];
19
20  % Paramètres pour la descente de gradient
21  iterations = 50000;
22  alpha = 0.003;
23
24  % Cout et theta de départ pour la descente de gradient
25  theta = [0;0;0;0];
26  J = computeCost(X, y, theta);
27  fprintf('Avec theta = [0 ; 0 ; 0 ; 0]\n La fonction de cout = %f\n', J);
28
29  %% ===== Part 2-Méthode analytique et descente de gradient ==
30
31  % Descent de gradient
32  [gtheta, J_history] = gradientDescent(X, y, theta, alpha, iterations);
33
34  % Affichage à l'écran du theta trouvé
35  fprintf('\n');
```

```

36 fprintf('Theta trouve par la descente de gradient:\n');
37 fprintf('%f\n', gtheta);
38
39 % Méthode analytique
40 atheta = normalEqn(X,y);
41
42 % Affichage à l'écran du theta trouvé
43 fprintf('Theta trouve par la methode analytique:\n');
44 fprintf('%f\n', atheta);
45
46 %% ===== Part 3-Comparaison du cout =====
47
48 fprintf('\n');
49 fprintf('Comparaison des solutions par leur coût.\n');
50
51 % Calcul de cout pour les deux solutions
52 Ja = computeCost(X, y, atheta);
53 Jg = computeCost(X, y, gtheta);
54
55 fprintf('Le cout par la solution via descente de gradient:\n');
56 fprintf('%f\n', Jg);
57 fprintf('Le cout par la solution analytique:\n');
58 fprintf('%f\n', Ja);

```

En compilant le programme dans un fichier contenant toute les fonctions appelées, on obtient en sortie :



```
D:\PROGRA~1\Logiciel\OCTAVE~2.0\mingw64\bin\o...
Chargement des donnees ...
Avec theta = [0 ; 0 ; 0 ; 0]
La fonction de cout = 4471.080357
Programme mis en pause.
Appuyez sur une touche pour lancer les calculs.
Theta trouve par la descente de gradient:
5.159978
4.504474
-1.961600
1.280386
Theta trouve par la methode analytique:
5.802190
4.481551
-1.990676
1.278067
Programme mis en pause.
Appuyez sur une touche pour comparer les solutions par le cout.
Le cout par la solution via descente de gradient:
130.749530
Le cout par la solution analytique:
130.746516
>>
```

On observe que les solutions sont proches et que le coût calculé de ces solutions par la fonction de coût le sont encore plus.

## 6.3 Approfondissement

### 6.3.1 Méthode analytique ou descente de gradient ?

Avec les fonctions utilisables dans les bibliothèques des langages de programmation d'aujourd'hui (Python, R,...), la vitesse à laquelle on programme une utilisation, soit de la méthode analytique soit de la descente de gradient, est à peu près égale.

La *vitesse de résolution* (appelée complexité en informatique) de l'algorithme sera un levier plus clair et pertinent pour départager les deux méthodes.

En effet, à partir d'une certaine taille des données d'expérience explicatives  $(x^{(i)})_{i \in \{1, \dots, m\}}$ , la méthode analytique va consommer trop de puissance de calcul. En ordre de grandeur et pour les ordinateurs d'aujourd'hui on considère que la limite d'utilisation se situera dans les régressions avec  $n > 10^5$  paramètres explicatifs. [4]

### 6.3.2 Le choix du pas pour la descente de gradient

Le choix du pas pour la descente de gradient se fait dans un premier temps en proposant une valeur de départ puis en tatonnant jusqu'à une valeur au



résultat considéré satisfaisant.

Si la descente de gradient diverge, cela signifie que le pas est trop grand, si le calcul est trop long alors on peut essayer d'agrandir la taille du pas.

Un agrandissement ou rétrécissement par un rapport approximatif de 3 est raisonnable.[4]

*Exemple* : Supposons que l'on ait choisi un pas de départ  $\lambda = 1$ .

- Si l'on considère le calcul trop long, alors on peut passer à  $\lambda = 3$ , à  $\lambda = 10$  si l'on poursuit, et ainsi de suite.
- Si le calcul ne s'effectue pas, alors on peut passer à  $\lambda = 0,3$ , à  $\lambda = 0,1$  si l'on poursuit, et ainsi de suite.

### 6.3.3 Alternatives informatique à la méthode du gradient

- Il existe en effet d'autres algorithmes permettant d'implémenter une descente de gradient entre autres le "Conjugate Gradient" ou "BFGS" qui vont permettre de minimiser la fonction de coût plus rapidement et sans nécessité d'indiquer le pas( $=\alpha$ ).
- Une particularité de ces méthodes est que, bien qu'elles soient accessibles à l'utilisation, elles sont néanmoins beaucoup plus complexes à comprendre et décrire (connaissance des langages informatiques et de programmation approfondie requise).

## 7 La régression logistique

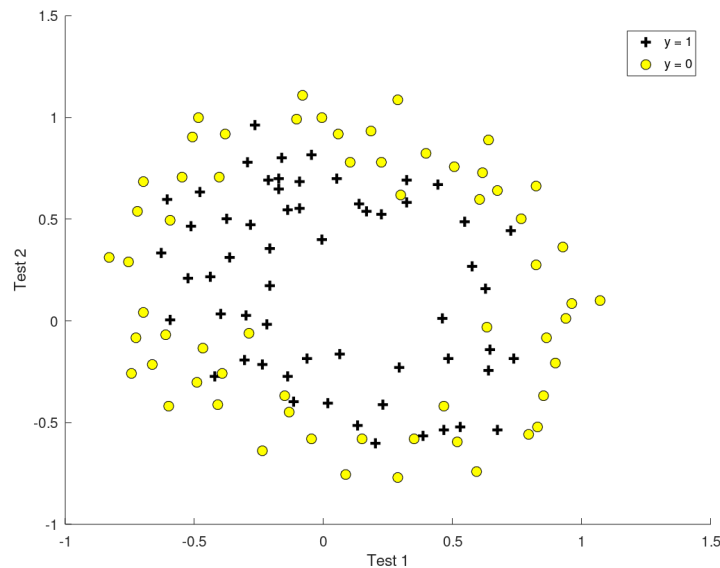
Proposons maintenant un exemple d'utilisation d'une régression logistique.

Supposons qu'une entreprise nous ait recruté en tant que gérant d'usine après le départ à la retraite du précédent gérant. Nous avons la responsabilité de décider parmi notre production quels sont les produits qui peuvent être commercialisés. *Sont effectués, pour chaque produit, deux tests différents qui permettent d'évaluer leur qualité.*

Nous possédons pour nous aider, les données d'anciennes pièces avec les résultats des deux tests différents, accompagnés de l'information sur leur commercialisation/non-commercialisation.[3][Chapitre 3/Exercice]

Nous faisons face à un problème de classification, nous allons alors le résoudre grâce à une régression logistique. Les valeurs explicatives correspondent aux deux tests,  $x^{(i)} = (x_1^{(i)}, x_2^{(i)})$ ; les valeurs expliquées à la commercialisation, notée  $y^{(i)} = 1$ , ou la non commercialisation notée  $y^{(i)} = 0$ .

Nous pouvons dans ce cas représenter graphiquement les données :



## 7.1 Fonctions à préparer

Afin de résoudre notre problème de minimisation, nous allons utiliser une fonction d'optimisation propre à Octave, *fminunc*. Pour cela, il faut fournir une fonction qui renvoi le gradient, la fonction qui renvoi le coût de la fonction à minimiser ainsi que des paramètres de départ, ici un  $\theta$  de départ.

Pour produire une résolution de qualité, nous allons introduire la notion de régularisation. Dans la régularisation, on additionne à la fonction à minimiser la somme  $\sum_{i=1}^n \theta_i^2$ . Remarquons que la valeur correspondant au biais  $\theta_0$  n'est pas incluse dans la somme.

Nous allons aussi faire appel à une fonction "mapFeature" qui va générer de nouvelles valeurs explicatives à partir de celles que nous possédons déjà.

*Exemple* : Nouvelles variable explicative comme  $x_1x_2$  ou  $x_1^2$  ou encore  $x_1^2x_2^2$ .

Leur utilité sera expliquée plus en détail dans les deux parties suivantes.

### 7.1.1 La fonction mapFeature

Voici l'algorithme de la fonction mapFeature que nous allons utiliser :

```
1 function out = mapFeature(X1, X2)
2
3 % Degré max des nouvelles valeurs explicatives à partir de X1,X2
4 degree = 6;
5
6 % Nouvelle matrice des valeurs explicatives avec la colonnes des 1
7 out = ones(size(X1(:,1)));
8
9 % création des nouveaux vecteurs explicatifs
10 for i = 1:degree;
11     for j = 0:i;
12         % à chaque itération, on génère une nouvelle colonne
13         out(:, end+1) = (X1.^(i-j)).*(X2.^j);
14     end end end % Fin des deux boucle et fin de la fonction
```

### 7.1.2 La fonction de calcul du cout et du gradient

Rappelons la forme du gradient :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

En rappelant que  $h_{\theta}(X) = g(X_1\theta)$  avec  $g$  la fonction logistique et  $X_1$  correspond à  $X$  en rajoutant en premier paramètre la valeur 1.

La fonction "costFunctionReg" va nous permettre de renvoyer la fonction de cout ainsi que le gradient en fonction d'un theta pour le problème de régression logistique représenté par X, y (sous leur forme explicitée page.14), ainsi que lambda(multiplicateur de la régularisation).

```
1 function [J, grad] = costFunctionReg(theta, X, y, lambda)
2
3 m = length(y); % nombre d'éléments d'entrainement
4
5 % Nous allons retourner comme résultat les variables suivantes corrigées
6 J = 0;
7 grad = zeros(size(theta));
8
9 % Calcul naif du gradient et de la fonction de cout
10 prod = sigmoid(X*theta);
11 temp = -y.*log(prod) - (1 - y).*log(1 - prod);
12 J = sum(temp)/m;
13 grad = (1/m)*((X')*(prod - y));
14
15 % Rajout de la régularisation
16 temp = theta(1);
17 theta(1) = 0;
18 J = J + (sum(theta.^2))*(lambda/(2*m));
19 grad = grad + theta*(lambda/m);
20
21 end
```

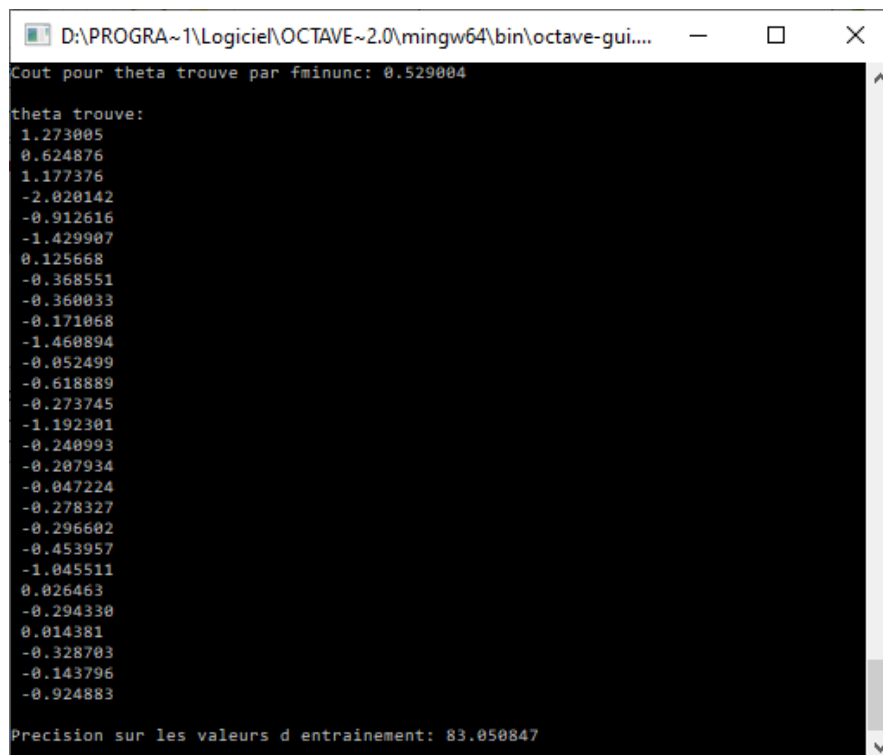
La fonction appelée sigmoid correspond à la fonction logistique, elle doit aussi être codée mais nous n'allons pas l'expliciter.

## 7.2 Résolution

Le code de résolution du problème de régression logistique posé serait alors :

```
1 data = load('ex2data2.txt');
2 X = data(:, [1, 2]); y = data(:, 3);
3
4 % La fonction mapFeature rajoute aussi la première colonne des 1
5 X = mapFeature(X(:,1), X(:,2));
6
7 % Initialisation d'un theta aux bonnes dimensions
8 i_theta = zeros(size(X, 2), 1);
9
10 % Le meilleur lambda trouvé (régularisation)
11 lambda = 1;
12
13 % Etablissement des options pour fminunc
14 options = optimset('GradObj', 'on', 'MaxIter', 400);
15
16 % Optimisation
17 [theta, cout] = ...
18     fminunc(@(t)(costFunctionReg(t, X, y, lambda)), i_theta, options);
19
20
21 % Résultats affichés
22 fprintf('Cout pour theta trouve par fminunc: %f\n', cout);
23 fprintf('\n');
24 fprintf('theta trouve: \n');
25 fprintf(' %f \n', theta);
26
27 % Calcul de la précision sur nos données d'entraînement
28 p = predict(theta, X);
29
30 fprintf('\n');
31 fprintf('Precision sur les valeurs d entrainement:');
32 fprintf(' %f\n', mean(double(p == y)) * 100);
```

En lançant l'algorithme avec les paramètres présentés, on aurait comme résultat :



```
D:\PROGRA~1\Logiciel\OCTAVE~2.0\mingw64\bin\octave-gui....
Cout pour theta trouve par fminunc: 0.529004

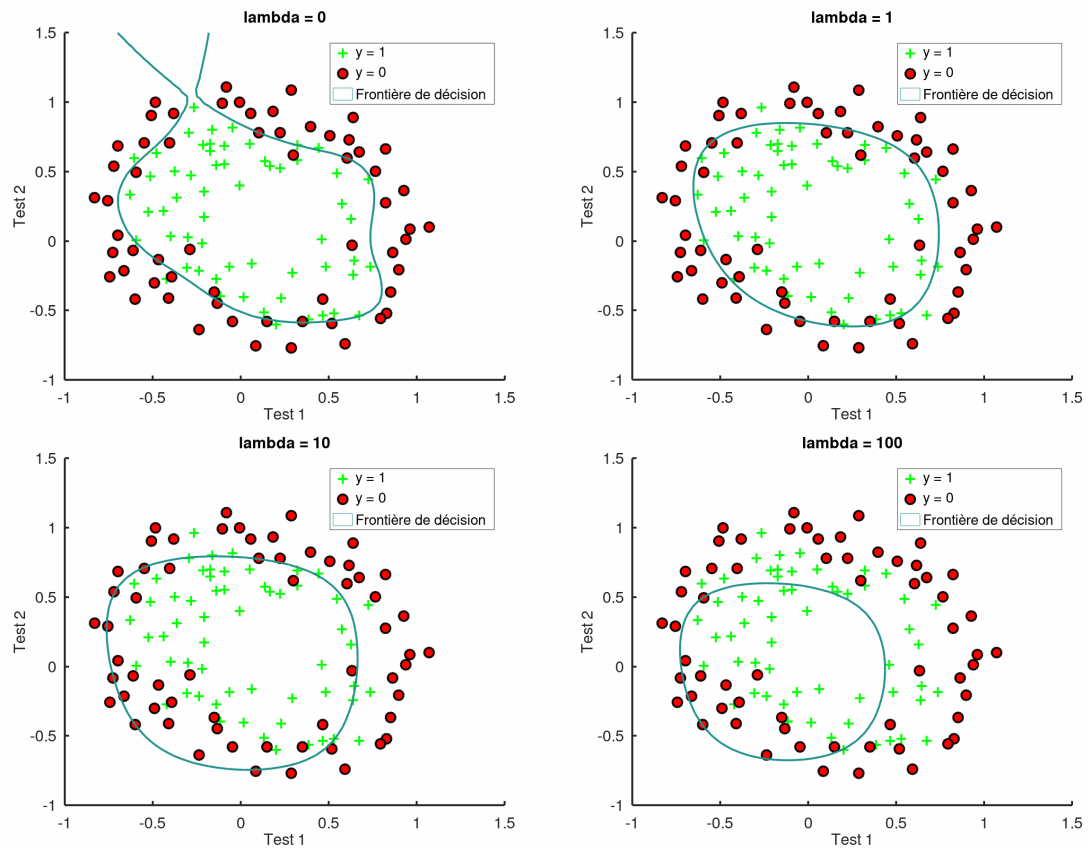
theta trouve:
1.273005
0.624876
1.177376
-2.020142
-0.912616
-1.429907
0.125668
-0.368551
-0.360033
-0.171068
-1.460894
-0.052499
-0.618889
-0.273745
-1.192301
-0.240993
-0.207934
-0.047224
-0.278327
-0.296602
-0.453957
-1.045511
0.026463
-0.294330
0.014381
-0.328703
-0.143796
-0.924883

Precision sur les valeurs d entrainement: 83.050847
```

Sans utilisation de `mapFeature`, on ne dépasse pas les 58% de précision sur les valeur d'entraînement, avec ou sans régularisation.

Pour le theta trouvé sans régularisation, c'est à dire pour  $\lambda = 0$ , le pourcentage de prédiction est à 86% au lieu de 83% trouvé plus haut avec les paramètres considérés comme les meilleurs.

Observons alors, différentes zones de décision selon le multiplicateur de régularisation  $\lambda$ .



La zone d'acceptation est l'intérieur du cercle, celle du refus est l'extérieur du cercle.

On observe que la régularisation a pour effet "d'aplatir" les irrégularités les plus fortes de la forme prise par la frontière de décision.

La frontière de décision la plus cohérente apparaît être celle générée par  $\lambda = 1$ .

Pour  $\lambda = 0$ , nous sommes confrontés au problème du *sur-ajustement* (ou overfit en anglais).

Pour  $\lambda = 10$  ou  $100$ , nous sommes confrontés au problème de *sous-ajustement*.

## 7.3 Au sujet du sur-ajustement/sous-ajustement

### 7.3.1 Le problème

En régression linéaire si l'ensemble donné de points est fini, il nous est possible de construire une fonction polynomiale qui passe très exactement par chacun des points tel que :

$$y^{(i)} = h_{\theta}(x^{(i)})$$

On peut construire cette fonction en utilisant l'interpolation de Lagrange et le degré de la fonction sera  $m - 1$  avec  $m$  le nombre d'entraînement. La fonction de coût/cost-function sera alors nulle.

Le problème avec une telle fonction est qu'elle correspondra à une courbe particulièrement "vacillante", avec une variance trop élevée des estimateurs, qui serait peu fiable pour la prédiction de nouvelles valeurs. C'est un problème de sur-ajustement.

On appelle sur-ajustement ce problème où la fonction de prédiction est *trop "influencée"* par ses points.

*Exemple :* La frontière de décision illustrée pour  $\lambda = 0$  sur la page précédente.

En contre partie, le sous-ajustement désigne une fonction de prédiction qui ne prête pas assez attention aux nuances des données. Une solution qui représente la tendance de manière *trop* grossière.

*Exemple :* La frontière de décision illustrée pour  $\lambda = 10$  ou  $100$  sur la page précédente.

### 7.3.2 Quel remède pour un sur-ajustement/sous-ajustement ?

Une première étape vers la résolution de ce problème est de reconsidérer les valeurs explicatives. En effet, pour un sur-ajustement il est possible que certains des paramètres dans les valeurs explicatives soient redondants, ou bien, sans lien avec la valeur à expliquer.



Pour un sous-ajustement, au contraire, il manquerait alors des informations, des paramètres explicatifs, afin d'obtenir une régression satisfaisante.

Si néanmoins, on considère que toutes les informations offertes sont pertinentes, une solution qui selon les pratiquants est une alternative utile voire obligatoire est celle de la pénalisation.[3][Chapitre 2]

On peut pénaliser un groupe de coordonnées de  $\theta$  ou bien la totalité des coordonnées en rajoutant à la fonction de coût la somme :  $\lambda \sum_{i=1}^n \theta_i^2$ , avec  $\lambda$  le multiplicateur de la régularisation. On appelle cette méthode la *pénalisation*.

*Remarque* : Une régularisation trop forte ( $\lambda$  trop grand) donne naissance à un sous-ajustement.

Le paramètre de  $\theta$  qui correspond au biais,  $\theta_0$  n'est lui pas pénalisé. En effet, à l'image d'une régression linéaire,  $\theta_0$  qui influence le point à l'abscisse n'impacte pas la régularité de la courbe.

C'est la pénalisation que nous avons utilisé pour notre régression logistique.

*Remarque 2* : La régularisation s'applique aussi sur une régression linéaire. Dans le problème de l'ozone les valeur explicatives étaient bien adaptées, c'est pourquoi il n'a pas été nécessaire d'implémenter de régularisation.

## Conclusion

Nous avons expliqué les fondamentaux du fonctionnement de la régression linéaire et la régression logistique, ainsi que décrit le processus de fonctionnement des réseaux de neurones.

Il existe plusieurs autres méthodes qui rentrent dans le cadre du machine learning non présentées dans ce devoir, entre autres "arbre et forêts aléatoires", "machine à vecteur de support" ou encore "apprentissage supervisé".

Aujourd'hui le machine learning continue de se développer rapidement, notamment par le biais de réseaux de neurones pensés pour des tâches de plus en plus ciblées.

On peut citer des exemples connus comme le computer vision qui développe la capacité d'apprentissage par les images et vidéos d'un algorithme ou encore le Natural Language Processing dit NLP qui concerne l'apprentissage algorithmique de tâches linguistique, en considérant comme *données explicatives* des fichiers sonores ou textuels.

Ces développements sont pour beaucoup un approfondissement de la technique des réseaux de neurones.

Beaucoup rêvent les possibilités des réseaux de neurones tandis que d'autres avertissent déjà quant aux limites de celles-ci ; en particulier relativement aux espoirs d'une véritable intelligence artificielle. En effet, un même réseau de neurones n'a jamais été entraîné efficacement pour pouvoir effectuer deux tâches différentes.

## Références

- [1] Pierre-André Cornillon et ERIC MATZNER LOBER. *Régression Théorie et applications*. 2007.
- [2] Michel LEJEUNE. *Statistique La théorie et ses applications - deuxième édition*. 2010.
- [3] Andrew NG. *Coursera - Machine Learning*. 2014. URL : <https://www.coursera.org/learn/machine-learning>.
- [4] Andrew NG. *CS229 Lectures Notes - cours sur le Machine Learning dispensé à Stanford*. 2013.
- [5] Laboratoire de mathématiques appliquées de l'Agrocampus OUEST. *Données météorologiques, été 2001 à Rennes*. URL : [https://s3-eu-west-1.amazonaws.com/static.oc-static.com/prod/courses/files/parcours-data-analyst/Cours\\_realisez\\_des\\_modelisations\\_performantes/NOUVEAU/Dataset\\_ozone.txt](https://s3-eu-west-1.amazonaws.com/static.oc-static.com/prod/courses/files/parcours-data-analyst/Cours_realisez_des_modelisations_performantes/NOUVEAU/Dataset_ozone.txt).
- [6] Jerome Friedman TREVOR HASTIE Robert Tibshirani. *The Elements of statistical learning - Data Mining, Inference and Prediction - (2nd edition)*. 2013.