

Premiers pas vers le machine learning

*R et Mathématiques pour effectuer une prévision
en apprentissage supervisé.*

Une vue minimaliste

D'après l'ouvrage [Data Scientist et langage R](#) (Eva Laude et Henri Laude)

Sommaire

Quelques notions fondatrices.....	4
Phénomène aléatoire	4
Probabilité.....	5
Variable aléatoire et distribution.....	5
Calculs simples sur les distributions	6
Conditionnement et probabilité conditionnelle.....	6
Indépendance	8
Un peu de mathématiques : notations et définitions utiles.....	9
Liste de notations très basiques	9
Notations et éléments liés à la théorie des mesures	10
Notations et définitions utiles en probabilité.....	12
Exploration des notations possibles	14
Notions Basiques	16
Moments d'une variable aléatoire discrète X.....	16
Espérance mathématique : moment m_1	16
Espérance mathématique (m_1) : généralisation	20
La variance : moment m_2	20
Moments m_3 et m_4	22
Premières considérations sur les erreurs et estimations	24
Se familiariser avec ses données	25
R Commander	27
Rattle.....	29
Matrices et vecteurs	34
Conventions, notations, utilisations basiques	34
Fabriquons une matrice.....	34
Utilisons notre matrice	36
Extraire des vecteurs	39
Différentes notations pour les vecteurs et leur transposée.....	40
Vecteur ou vector ?	41
apply() cas 1 : vector vers vector (issu de vecteur vers vector).....	43

apply() cas 2 : vector vers vecteur (issu de vecteur vers vecteur).....	44
apply() cas 3 : matrice vers matrice	44
apply() : autres cas utiles avec fonctions génériques R.....	44
Matrice.....	46
Plus loin dans la manipulation des matrices avec R	47
Opérations basiques	47
Quelques savoir-faire utiles sur les matrices de R.....	48
Produit de Hadamard	49
Produit matriciel "classique"	50
Somme directe entre vecteurs	51
Somme directe entre matrices	51
Produit de Kronecker.....	52
Normes de vecteurs et normes de matrices	53
Normes de vecteurs.....	53
Distances.....	55
Normes de matrices.....	56
Matrices et vecteurs : diverses syntaxes utiles	57
Transformer une matrice en vector (au sens R) ou vecteur.....	57
Transposition d'une matrice, d'un tableau, d'un data.frame.....	57
Diagonale d'une matrice, identité, triangles	58
Déterminants, valeurs propres, inversion de matrices	60
Diagonalisation de matrices, valeurs propres, vecteurs propres	61
Estimations	63
Positionnement du problème d'estimation.....	63
Formulation générale du problème.....	63
Application et reformulation du problème d'estimation	66
Introduction syntaxique de la fonction hypothèse.....	66
Théorème "No free lunch"	67
Régression linéaire.....	68
Le cas d'un modèle de régression linéaire simple	68
Application au cas d'un modèle de régression linéaire multiple.....	69
Régression linéaire multiple et méthode des moindres carrés.....	71
Matrice Hat	73
Distance de Cook	75

Les indicateurs d'écart utilisés en machine learning	77
MSE, RMSE, SSE, SST.....	77
RMSE, l'indicateur "roi"	77
À quoi servent les indicateurs d'écart ?.....	78
MAE, ME	78
NRMSE/NRMSD, CV_RMSE.....	80
SDR.....	81
Accuracy, R2.....	82
Accuracy.....	82
R2, coefficient de détermination.....	82
Conclusion intermédiaire.....	84
Mise en pratique : apprentissage supervisé	85
Préparation	85
Tester des hypothèses, p_value	88
Analyse graphique interactive avec iplots	88
Test de Breush-Pagan et zoom sur p_value	90
Création d'un modèle (régression linéaire multiple).....	92
Établissement d'une prédiction.....	92
Étude des résultats et représentations graphiques.....	93
Indicateurs courants - calculs	97
Étude du modèle linéaire généré	98
Conclusion sur le modèle linéaire.....	104
Utilisation d'un modèle "Random Forest"	105

Quelques notions fondatrices

Phénomène aléatoire

Vous vous trouvez face à un **phénomène aléatoire** si, quand vous effectuez une **expérience** plusieurs fois dans des conditions identiques, vous obtenez des **résultats** différents (et donc d'une certaine manière... imprévisibles).

Remarquez que dans cette définition, le fait qu'un phénomène soit identifié comme aléatoire dépend de l'incapacité de l'observateur à le prévoir complètement, ce qui lui fera déclarer de façon péremptoire que le phénomène est intrinsèquement hasardeux.

Pour autant, quand on répète l'expérience un grand nombre de fois on voit les résultats se répartir selon une loi stable, c'est-à-dire avec **des fréquences d'apparition des résultats qui dépendent de la valeur du résultat**. Par exemple, si vous observez les notes au baccalauréat d'un grand nombre d'élèves vous trouverez une fréquence de notes moyennes plus élevée que la fréquence des mauvaises ou des bonnes notes. C'est parce que nous disposons de cette **loi des grands nombres** que nous pouvons nous permettre de faire quelques mathématiques pour étudier le phénomène.

Un **événement, qui se définit comme une partie de tous les résultats possibles**, est d'autant plus probable qu'il recouvre une grande part des résultats possibles.

Par exemple, l'événement avoir une note supérieure à 10 a une probabilité plus forte que l'événement avoir une note supérieure à 16.

Quand le nombre de résultats possibles est infini, le nombre d'événements possibles est lui aussi infini et certaines difficultés mathématiques apparaissent. Dans ce cas, les mathématiciens travaillent sur un sous-ensemble de l'ensemble des événements possibles, qu'ils nomment **tribu**. Une tribu est stable par intersection et union dénombrables d'événements (que l'on peut compter), c'est-à-dire que ces intersections ou ces unions appartiennent à la tribu. Une tribu comprend l'événement vide et l'événement "tous les résultats possibles", le complémentaire (c'est-à-dire le contraire) d'un événement d'une tribu fait partie de la tribu. Dans ce cas on ne saura définir une probabilité que sur les membres (événements) de la tribu.

Probabilité

Une **probabilité est une application** allant d'une tribu d'évènements, vers des valeurs comprises entre 0 et 1 (incluses) et telle que :

- La probabilité de l'évènement comprenant tous les résultats possibles soit égale à 1.
- La probabilité de toute union dénombrable d'évènements disjoints de la tribu (c'est-à-dire sans résultat commun) soit égale à la somme des probabilités de ces évènements.

Par exemple, la probabilité d'avoir une note comprise entre 0 et 20 vaut 1, et la probabilité d'avoir une note entre 16 et 18 **ou** entre 18 et 20 est égale à la probabilité d'avoir une note entre 16 et 20.

Par ailleurs la probabilité d'un évènement impossible, par exemple d'avoir une note inférieure à 5 et à la fois supérieure à 18, est évidemment nulle.

Variable aléatoire et distribution

Intéressons-nous maintenant à la notion de variable aléatoire. Cette notion peut prêter à confusion, en effet une **variable aléatoire n'est pas une variable mais une fonction** allant des résultats d'expérience d'un espace mesurable vers un ensemble quelconque et qui nous permet de caractériser l'expérience.

Imaginons que vous vous intéressiez aux livres choisis dans une bibliothèque et que la variable aléatoire x représente le nombre de pages. Vous disposez bien d'une fonction qui, pour chaque résultat (c'est-à-dire chaque livre choisi au hasard), donne le nombre de pages. Évidemment, d'autres variables aléatoires sont possibles, par exemple une variable y , qui exprimera dans notre exemple si la couverture est cartonnée ou souple.

On se dote alors d'une probabilité p_x qui est appelée **loi de la variable x** (ou **distribution de x** , si on se réfère aux statistiques) qui procure une probabilité pour chaque valeur de x (ici le nombre de pages) et pour chaque évènement issu de la tribu des sous-ensembles possibles des valeurs de x (par exemple la probabilité de choisir un ouvrage entre 100 et 200 pages). On conçoit aisément que l'aspect de distribution de x n'a rien à voir avec celle de la distribution de y (couverture cartonnée ou souple).

Calculs simples sur les distributions

Voici quelques propriétés que l'on doit absolument connaître sur les distributions.

On a : $p_{x \text{ et } y} + p_{x \text{ ou } y} = p_x + p_y$

Si x et y sont disjointes, à savoir s'excluent mutuellement on a : $p_{x \text{ et } y} = 0$ et donc $p_{x \text{ ou } y} = p_x + p_y$.

Ces dernières lignes se comprennent facilement si l'on prend un point de vue ensembliste où l'expression **x et y** signifie **l'intersection** des événements correspondants, et où l'expression **x ou y** signifie **l'union** des événements correspondants.

L'exemple de la section suivante décrit une façon de déterminer $p_{x \text{ et } y}$ et vous permettra de mieux saisir cette notion.

Conditionnement et probabilité conditionnelle

Quand on a fixé la ou les valeurs d'une variable aléatoire dont on connaît la distribution et si les variables sont dépendantes, alors cette information conditionne la probabilité de l'autre variable.

Pour simplifier les explications suivantes, nous allons maintenant discréteriser la variable aléatoire x , à savoir la transformer en un nombre fini de valeurs. Ici on considérera qu'un livre de moins de 100 pages est "petit" et "gros" sinon.

Considérons que l'on nous ait expliqué qu'en général, quand le livre est "gros", alors dans 90 % des cas on utilise des couvertures cartonnées et que dans le cas contraire on ne les utilise que dans 15 % des cas. Nous disposons maintenant d'une distribution de **probabilité conditionnelle** que l'on peut noter $p_{y|x}$ et qui s'énonce de la façon suivante : **p_de_y_sachant_x**.

Voyons dans le détail l'aspect de cette distribution de probabilité conditionnelle. Exprimons que la probabilité que le livre soit cartonné **sachant que le livre est gros** est de 90 % :

```
py|x(cartonné , gros_livre ) = 90 % # puis de même :  
py|x(souple   , gros_livre ) = 10 %  
py|x(cartonné , petit_livre ) = 15 %  
py|x(souple   , petit_livre ) = 85 %
```

Il apparaît alors que si l'on connaît la distribution p_x de gros et petits livres dans la bibliothèque on pourra en déduire la distribution des couvertures cartonnées par taille de livre dans cette bibliothèque. Imaginons que l'on ait 80 % de gros livres, alors :

```
px(gros_livre ) = 80 %      # et:  
px(petit_livre ) = 20 %
```

Ce qui nous permet de calculer la distribution de chaque type de livre, par rapport aux deux variables aléatoires ensemble et sur tous les livres de la bibliothèque : p_y et x . La probabilité qu'un livre soit cartonné et gros étant :

$$p_{y \text{ et } x}(\text{cartonné, gros_livre}) = \\ p_{y|x}(\text{cartonné, gros_livre}) * p_x(\text{gros_livre}) = \\ 90\% * 80\% = 72\% = 0.72$$

puis de même :

$$p_{y \text{ et } x}(\text{souple, gros_livre}) = \\ p_{y|x}(\text{souple, gros_livre}) * p_x(\text{gros_livre}) = \\ 10\% * 80\% = 8\% = 0.08$$

$$p_{y \text{ et } x}(\text{cartonné, petit_livre}) = \\ p_{y|x}(\text{cartonné, petit_livre}) * p_x(\text{petit_livre}) = \\ 15\% * 20\% = 3\% = 0.03$$

$$p_{y \text{ et } x}(\text{souple, petit_livre}) = \\ p_{y|x}(\text{souple, petit_livre}) * p_x(\text{petit_livre}) = \\ 85\% * 20\% = 17\% = 0.17$$

On constate que la somme des $p_{y \text{ et } x}$ est égale à 1, ce qui correspond effectivement à la valeur de la probabilité lorsque l'on considère un événement comprenant tous les résultats possibles.

Il est démontré que l'on peut généraliser de la façon suivante, pour les variables aléatoires discrètes (dénombrables) mais aussi pour des distributions sur des variables aléatoires prenant leurs valeurs dans des ensembles non dénombrables, comme les réels ou des vecteurs de réels (ou même des fonctions !) :

$$p_{y \text{ et } x} = p_{y|x} \cdot p_x = p_{x|y} \cdot p_y = p_{x \text{ et } y}$$

Cette formule est valable en supposant les distributions de x et y non nulles.

Dans le même ordre d'idée, on a, avec trois variables aléatoires :

$$p_{x \text{ et } y \text{ et } z} = p_x \cdot p_{y|x} \cdot p_{z|x \text{ et } y}$$

Et ainsi de suite... On peut généraliser à un nombre quelconque de variables aléatoires.

Indépendance

La notion d'indépendance entre variables est très utile, c'est souvent la stricte condition pour pouvoir utiliser certains algorithmes. On perçoit intuitivement qu'il est plus facile d'interpréter un phénomène quand on a la chance de le voir lié à des variables aléatoires indépendantes. L'idée générale est que disposer d'une information sur une des variables indépendantes ne donne aucune information sur la distribution de l'autre variable aléatoire.

Cela se traduit évidemment par (en supposant les distributions non nulles) :

$p_{x|y} = p_x$ ce qui s'avère équivalent à :

$p_{y|x} = p_y$ ce qui s'avère équivalent à :

$p_{x \text{ et } y} = p_x \cdot p_y$

Attention, cette notion d'indépendance ne doit pas être confondue avec la notion d'évènements disjoints, à savoir ne partageant aucune valeur de résultat. Par exemple, imaginez que dans notre bibliothèque les couleurs au dos des livres dépendent de la nationalité des auteurs, qui ne pourraient n'être que français ou allemands. L'évènement { bleu, blanc, rouge } signifie que cet ouvrage a au moins un auteur français et l'évènement { noir, rouge, jaune } signifie que cet ouvrage a au moins un auteur allemand. Ces deux évènements ont une intersection non nulle, à savoir la couleur { rouge }. Pour autant le fait d'identifier qu'un auteur est allemand au travers des couleurs { noir, rouge, jaune } n'apporte aucune information sur le fait qu'il y ait ou non un autre auteur de l'ouvrage qui soit français et identifiable par ses couleurs { bleu, blanc, rouge } et vice versa. Les deux variables aléatoires sont indépendantes et pourtant non disjointes !

Un peu de mathématiques : notations et définitions utiles

Depuis le début du chapitre, nous avons été économies en notation afin de ne pas surcharger l'exposé, voyons maintenant les notations et définitions auxquelles nous pourrions être confrontés.

Liste de notations très basiques

La définition d'un ensemble utilise les accolades ; un ensemble A qui serait l'ensemble des entiers pairs pourrait s'écrire ainsi :

$$\begin{aligned} A &= \{ \text{les entiers pairs} \} \\ &= \{ n \in \mathbb{N} \mid n \text{ pair} \} \# \text{ la barre se lit "tels que"} \\ &= \{0, 2, 4, \dots\} \\ &= \{n \in \mathbb{N} \mid \exists k \in \mathbb{N}, (n = 2k)\} \end{aligned}$$

Ce qui peut se lire "ensemble des éléments de l'ensemble des entiers naturels, que nous nommerons ici n, tels qu'il existe un entier naturel que nous nommerons ici k et que l'on ait n égal à 2 fois k".

Pour désigner une expérience aléatoire, on emploie souvent le symbole Σ .

L'espace d'état associé à l'expérience (c'est-à-dire ensemble de tous les résultats possibles de l'expérience aléatoire Σ), se note oméga en majuscule : Ω .

Un des résultats de cet ensemble s'écrit (oméga en minuscule) : ω .

Le fait que ce résultat appartienne à Ω se notera donc $\omega \in \Omega$.

Si Ω est dénombrable, on peut l'écrire sous la forme de l'ensemble discret des résultats possibles $\{\omega_1, \omega_2, \dots\}$ (ici nous avons indicé de 1 à l'infini, d'où \mathbb{N}^*) :

$$\Omega = \{ \omega_i \}_{i \in \mathbb{N}^*}.$$

Si Ω est fini avec un cardinal (c'est-à-dire un effectif) de n éléments, on peut l'écrire sous la forme de l'ensemble discret des résultats possibles $\{\omega_1, \omega_2, \dots, \omega_n\}$:
 $\Omega = \{\omega_i\}_{i \leq n, i \in \mathbb{N}^*}$.

L'ensemble des parties de Ω , à savoir tous les ensembles que l'on peut construire avec des éléments de Ω auquel on adjoint l'ensemble vide \emptyset s'écrit souvent $\mathcal{P}(\Omega)$.

Notations et éléments liés à la théorie des mesures

Derrière le vocabulaire "mesure", on comprend intuitivement que l'on aborde une théorie qui nous permettra de maîtriser la façon dont on peut appréhender des quantités (nombres réels positifs ou nuls) qui correspondront à des ensembles de "quelque chose".

Il faut avoir conscience de ce que tout n'est pas mesurable et que donc le besoin de concevoir une théorie précise de la mesure s'est imposé aux mathématiciens. Les éléments suivants introduisent le vocabulaire qui permet de lire des textes manipulant cette notion de mesure. Nous ne pensions pas raisonnable d'éviter ces concepts, un peu abstraits, dans un ouvrage sur les data sciences car les data scientists manipulent des probabilités et que ces probabilités sont en fait des mesures au sens mathématique du terme.

Commençons par la définition d'un terme étrange dans ce contexte, le terme "tribu".

Une tribu \mathcal{A} sur Ω est un ensemble de sous-ensembles issu des parties de Ω : $\mathcal{P}(\Omega)$, qui comprend l'ensemble vide \emptyset , qui est stable par complémentarité (le complémentaire dans Ω de toute partie appartient aussi à Ω) et qui est stable par union dénombrable (toute union finie d'ensembles appartenant à la tribu appartient également à la tribu). Comme le complémentaire de l'ensemble vide \emptyset est Ω lui-même, d'après la règle précédente toute tribu comprend au moins les deux éléments de la tribu triviale suivante : $\{\emptyset, \Omega\}$.

Il est utile de savoir qu'une union ou l'intersection finie de tribus de Ω est elle-même une tribu de Ω .

Nous définirons la notion de mesure sur ces tribus. C'est un point important car cela signifie que ce qui n'est pas une tribu n'est sans doute pas mesurable au sens de la théorie de la mesure. Dans votre pratique de data scientist, vous serez souvent tenté de concevoir des mesures diverses ou de probabiliser un ensemble. Dans ce cas il faudra vous poser la question de savoir si l'ensemble que vous voulez mesurer a bien les caractéristiques d'une tribu, dans le cas contraire cet ensemble est souvent incomplet ou infini et mal défini.

Une mesure μ sur une tribu \mathcal{A} est une fonction telle que $\mu(\emptyset)=0$, et telle que la mesure d'une union dénombrable d'ensembles deux à deux disjoints de \mathcal{A} (pas d'élément commun) est égale à la somme des mesures de ces ensembles.

Autrement dit, quand vous mesurez le vide vous obtenez 0 et les différentes mesures possèdent une certaine cohérence quand on les additionne.

Soit \mathcal{A} une tribu extraite de $\mathcal{P}(\Omega)$,

soit p une mesure telle que $p(\Omega) = 1$,

p est alors appelée **probabilité** et les éléments de cette tribu sont des **événements** (ce sont des ensembles mesurables par une probabilité). L'espace mesurable est noté (Ω, \mathcal{A}) et cet espace muni de cette probabilité est appelé **espace probabilisé** (Ω, \mathcal{A}, p) .

Par construction, l'application d'une mesure (et donc a fortiori d'une probabilité), est telle que la mesure (ou probabilité) d'une union dénombrable d'événements deux à deux disjoints de \mathcal{A} est égale à la somme des mesures (ou probabilités) de ces événements.

Notons qu'une mesure est un réel positif. Comme $p(\emptyset) = 0$ on comprend donc mieux pourquoi une probabilité a une valeur comprise entre 0 et 1 inclus, c'est-à-dire $[0,1]$.

Quand Ω est un ensemble fini, la tribu naturellement utilisée en probabilité est $\mathcal{P}(\Omega)$.

Attention, ce n'est plus vrai en dimension infinie où l'on prend une tribu plus petite (c'est d'ailleurs ce pour quoi la notion de tribu nous est très utile pour pouvoir manipuler des probabilités sur des ensembles Ω infinis).

Notations et définitions utiles en probabilité

Au début de ce chapitre, nous utilisions la notation suivante pour le terme probabilité d'une variable aléatoire x : p_x . Cette notation est très agréable et très compacte et s'avère très rapide à manipuler lorsque vous voulez commenter des calculs simples et dans des contextes non ambigus comme ceux que nous avons abordés précédemment. Mais cette notation ne recouvre pas tous nos besoins ce qui fait que les auteurs utiliseront une autre notation plus souple, plus exacte, plus complexe, mais plus verbeuse et que nous allons explorer ici.

Dans les textes très mathématiques, désigner les variables aléatoires par des lettres majuscules peut être une bonne habitude. Cela rappelle que ce sont des fonctions $X, Y, Z\dots$ et cela les différencie des variables habituelles $x, y, z\dots$

Le symbole utilisé dans ce chapitre pour désigner une probabilité est plus "riche", ceci pour bien le distinguer d'une fonction habituelle en analyse, c'est le symbole \mathbb{P} .

La notation que nous allons aborder maintenant, faussement similaire à la notation que nous avons utilisée plus haut, en découle naturellement pour exprimer la probabilité d'un résultat donné, ici nommé ω .

La probabilité d'un évènement constitué du singleton $\{\omega\}$ se note naturellement :

$$\mathbb{P}(\{\omega\})$$

Un singleton est un ensemble comportant un et un seul élément.

Ce qui s'écrit parfois $\mathbb{P}(\omega)$ dans une notation abrégée (relativement impropre mais courante) ou, encore plus abrégée, p_Ω . Notation qui ne doit pas être confondue avec p_x où le x désigne la variable aléatoire et pas un évènement ! La notation mathématique, et en particulier la notation en statistiques, en économie et en physique est une histoire d'usage, de contexte, de style, d'auteur, de sens, de pays et de discipline. Les Anglo-Saxons et les Français ne partagent pas exactement les mêmes notations et la notation des informaticiens se limite à l'usage des caractères ASCII dans beaucoup de cas. **Soyez donc très attentif à l'éventualité d'un contresens quand vous abordez un texte mathématique et vérifiez attentivement le sens des symboles qui vous sont proposés.**

Explorons plus avant le sens de l'expression $\mathbb{P}(\{\omega\})$.

Quand Ω est infini, le nombre d'évènements singleton est infini et on constate que $\mathbb{P}(\{\omega\})$ est très souvent nulle, ce qui se conçoit bien car la probabilité d'un résultat donné versus une infinité de résultats possibles semble converger vers 0.

Quand Ω est infini, les ω tels que $\mathbb{P}(\{\omega\})$ n'est pas nulle sont appelés des **atomes**, ces atomes correspondent à des lieux particuliers de la fonction de la loi de probabilité, et on ne peut en trouver que si celle-ci n'est pas absolument continue.

À l'inverse, dans le cas où Ω est fini, on a de nombreux singletons ayant des probabilités non nulles. Par exemple le lancement d'une pièce non truquée nous donne :

$$\Omega = \{\text{pile, face}\}$$

$$\mathbb{P}(\{\text{pile}\}) = 1/2$$

$$\mathbb{P}(\{\text{face}\}) = 1/2$$

$$\mathbb{P}(\{\text{pile, face}\}) = 1 = \mathbb{P}(\Omega)$$

Ces dernières notations peuvent se décliner de diverses façons en fonction des divers auteurs, ou suivant les usages liés à l'expression classique d'un théorème ou d'un autre. Nous vous proposons maintenant d'explorer ces déclinaisons pour que vous puissiez avoir l'assurance de déchiffrer facilement les différents textes que vous trouverez dans votre pratique de data scientist.

Exploration des notations possibles

Remarquons tout d'abord que la probabilité d'un évènement représenté par un élément A de la tribu \mathcal{A} s'écrit naturellement :

$$\mathbb{P}(A)$$

Cette notation simple n'introduit pas la notion de variable aléatoire.

Les difficultés de notation apparaissent justement dans le cas où l'on veut décrire une probabilité faisant référence à une variable aléatoire. Pour comprendre ce qui peut créer de la confusion dans les différentes notations que vous allons aborder maintenant, il vous faut avoir à l'esprit, comme cela a été évoqué plus haut, qu'une **variable aléatoire n'est pas une variable**, mais une fonction.

Considérons la variable aléatoire X, l'ensemble des évènements d'une tribu \mathcal{A} dont l'image par X possède la valeur v s'écritra : $\{ a \in \mathcal{A} | X(a) = v \}$.

Si vous voulez maîtriser votre sujet, méditez cette dernière expression.

La probabilité de cet ensemble d'évènements, tels que la variable aléatoire appliquée à chaque évènement donne une valeur v s'écritra donc $\mathbb{P}(\{ a \in \mathcal{A} | X(a) = v \})$ et en écriture simplifiée $\mathbb{P}(X = v)$.

Dans une écriture encore plus simplifiée nous obtenons $p_X(v)$.

Ne perdez jamais de vue le sens profond caché derrière ces écritures simplifiées !

Imaginez une collection de valeurs : $v_1 v_2 \dots v_n$ que l'on pourrait décrire en utilisant un indice muet (c'est-à-dire générique, rien que pour se souvenir qu'il y a une collection de valeurs). Les probabilités des différentes valeurs peuvent alors s'écrire de la façon suivante :

$$p_X(v_i).$$

Si l'on n'a pas d'ambiguïté sur X, cette notation devient :

$$p(v_i)$$

Si on n'a qu'une seule collection de valeurs et qu'une seule variable aléatoire alors la notation se simplifie encore et devient :

$$p_i$$

Pour mémoire, en utilisant la notation précédente, et en considérant que les i valeurs correspondent à i évènements indépendants dont l'union représente tous les évènements possibles on constate : $\sum_i p_i = 1$

Considérons maintenant la variable aléatoire X , l'ensemble des évènements d'une tribu \mathcal{A} dont l'image par X appartient à l'ensemble V s'écrit $\{ a \in \mathcal{A} \mid X(a) \in V \}$, ce qui peut s'exprimer également de la façon suivante :

$$X^{-1}(V)$$

Dans ce cas la notation simplifiée de :

$$\mathbb{P}(\{ a \in \mathcal{A} \mid X(a) \in V \}),$$

qui vaut également :

$$\mathbb{P}(X^{-1}(V)), \text{ sera très abusive, mais très courante, et donnera } \mathbb{P}(X \in V).$$

Pour exprimer la même quantité on peut également invoquer la notion de loi de probabilité de la variable aléatoire X , que l'on écrit $\mathbb{P}_X \dots$ comme nous l'avions fait en début de ce chapitre en introduisant la notation très simple suivante : \mathbf{p}_x .

Cette loi est une mesure de probabilité, qui est l'image de \mathbb{P} par X (et oui X est une fonction !).

En conclusion on a donc :

$$\mathbb{P}(\{ a \in \mathcal{A} \mid X(a) \in V \}) = \mathbb{P}(X^{-1}(V)) = \mathbb{P}_X(V) = \mathbb{P}(X \in V)$$

Souvent on nomme les valeurs avec la lettre minuscule de la variable aléatoire : les valeurs possibles retournées par la fonction variable aléatoire X seront des x , les valeurs retournées par Y seront des y ...

Voyons maintenant ce que nous pouvons faire simplement à partir de ces notions.

Notions Basiques

Moments d'une variable aléatoire discrète X

On a vu plus haut que l'on peut écrire les probabilités de différentes valeurs de la façon suivante (quand il n'y a pas d'ambiguïté sur la variable aléatoire et sur les évènements considérés) : p_i .

Par ailleurs on a vu que si les i valeurs correspondent à i évènements indépendants dont l'union représente tous les évènements possibles on pourra écrire $\sum_i p_i = 1$.

Imaginez qu'un professeur incompétent note ses élèves de 0 à 20 sans utiliser de valeurs intermédiaires ($n = 21$ notes possibles; $x_1 = 0, x_2 = 1 \dots x_{21} = 20$).

La probabilité d'avoir une note entre 0 et 20 est évidemment $\sum_i p_i = 1$ (100 % de chance d'avoir une note quelconque !).

Du fait de son incompétence, le professeur tire ses notes à partir d'un chapeau dans lequel il a mis 21 petits papiers avec les 21 notes. Il prend soin de remettre les papiers dans le chapeau après chaque tirage pour laisser la même chance à chaque note d'être donnée à un élève (c'est-à-dire que chaque p_i vaut $1/21$).

En quelque sorte, ce professeur a créé une machine à fabriquer des distributions uniformes de notes.

Espérance mathématique : moment m_1

On perçoit que si le proviseur de l'établissement regarde la moyenne des notes de la classe, il n'y verra pas d'anomalie... car cette moyenne a toutes les chances d'être autour de 10. C'est en effet la **moyenne pondérée par leurs probabilités** respectives des différentes notes possibles.

C'est ce que nous appellerons espérance mathématique de la variable aléatoire discrète X et nous la noterons $\mathbb{E}(X)$.

L'expression en est la suivante : $\mathbb{E}(x) = \sum_i p_i x_i$.

Faisons le calcul en R :

```
## calcul naïf E(X), distribution uniforme ##  
  
x <- 0:20      # 21 valeurs de v[1] ...v[21], notes de 0 ... 20  
n <- length(v) # on a bien 21 notes  
  
p <- 1:21       # création des 21 probabilités
```

```

p[1:21] <- 1/n    # affectation d'une probabilité uniforme

# calcul de E(X) = sigma des pi.vi
E <- 0            # initialisation de E(X)
for (i in 1:21) {E <- E + x[i] * p[i]}
E                  # espérance de X

```

On obtient $\mathbb{E}(x) = 10$.

Comparons ce calcul à la moyenne des notes possibles :

```

## re-calcule E(X) et comparaison à la moyenne m          ##
E <- sum(p*x)      # même calcul en plus rapide
                    # c'est le produit scalaire des vecteurs
                    # p et v
E

m <- sum(x)/n      # calcul de la moyenne
(m == E)           # test si espérance et moyennes sont égales

```

L'expression ($m == E$) donne le résultat TRUE, l'espérance de notre distribution uniforme est bien sa moyenne.

L'espérance mathématique est le premier *moment* caractéristique d'une distribution. Elle exprime la tendance centrale de la distribution, parfois on l'appelle tout simplement *moyenne de la variable*.

Pour plus de compacité, il est commun de noter l'espérance mathématique μ ou m_1 (le 1 signifiant *premier moment*).

Comme notre professeur a vraiment tiré au hasard les notes, il y a sans doute une différence entre la valeur estimée de l'espérance mathématique (ici 10), et la valeur constatée. Nous allons demander à R de nous fabriquer une distribution uniforme discrète aléatoire et allons procéder au calcul précédent pour avoir une idée de la différence. Puis, ce calcul sera répété 10000 fois et on comparera la valeur de l'espérance aléatoire moyenne de ces différentes itérations avec la valeur théorique (10).

```

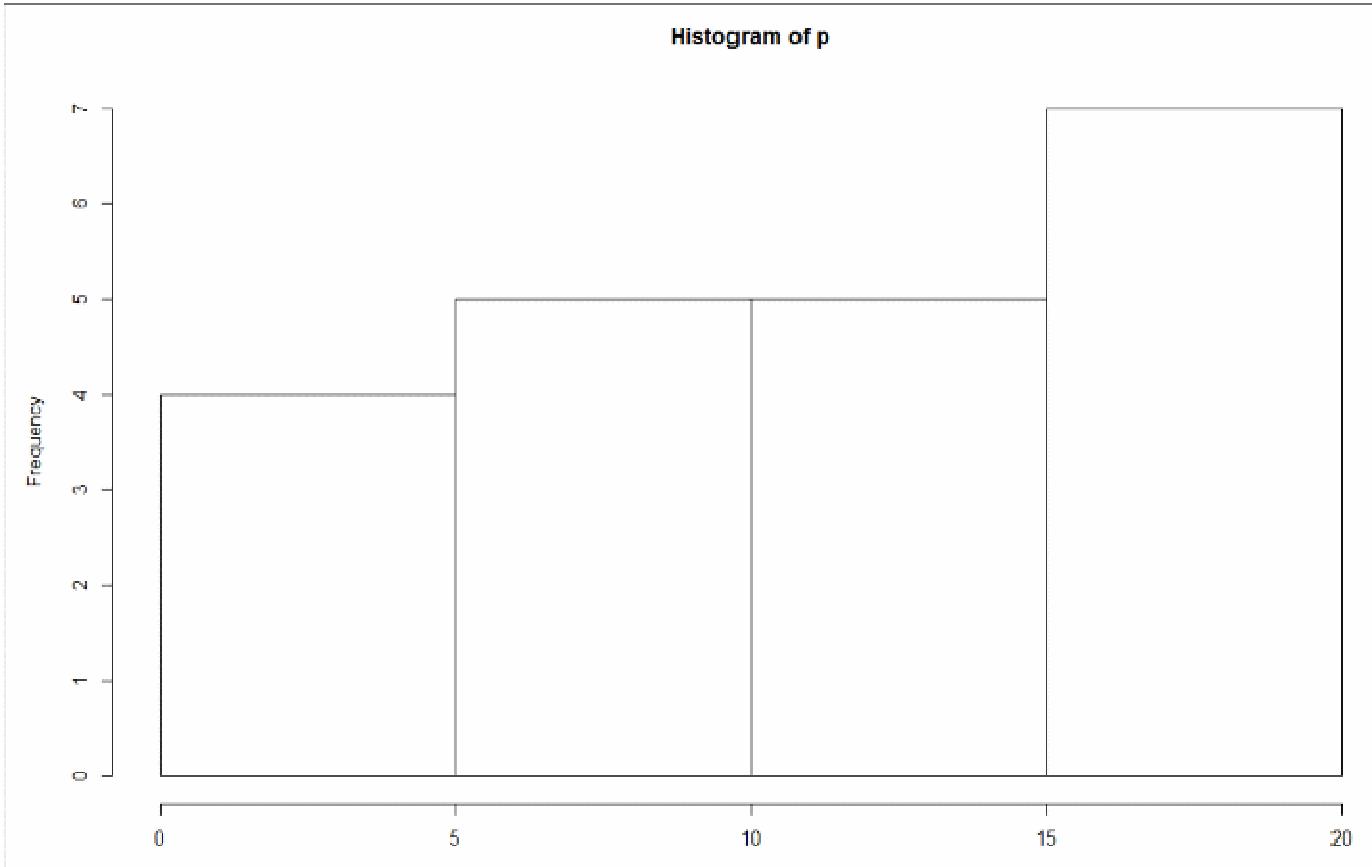
## avec une distribution uniforme comportant des aléas          ##
## calcul de la moyenne des E sur 10000 essais                ##

E <- 0            # initialisation du cumul des E
nb <- 0            # initialisation du comptage du nombre de boucles

for (s in 1:10000) { # on va faire 10000 aléas
  set.seed(s) # réinitialise le générateur de nombres aléatoires
  nb <- nb + 1
  p <- sample(1:20, 21, replace=T)      # distribution uniforme
  if (nb == 1) hist(p)    # dessinons une distribution sur 10000
  p <- p/sum(p)
  x <- 0:20        # 21 valeurs de v[1] ...v[21], notes de 0 ... 20
  E <- E + sum(p*x)
}
E <- E/nb # calcul de la moyenne des 10000 espérances
E

```

Ce qui nous donne l'histogramme suivant pour la première itération, où l'on constate que le tirage aléatoire est bien moins uniforme que dans le calcul d'origine.



Distribution réelle des notes obtenues à la première itération

Pourtant, après les 10000 itérations on constate que l'espérance mathématique moyenne est de 10.00566.

Tout le monde a entendu parler de la loi normale en forme de cloche, voyons avec un petit programme R similaire ce que cela donnerait avec une loi normale.

```
## avec une distribution loi normale comportant des aléas      ##
## calcul de la moyenne des E sur 10000 essais               ##

fn <- function (s){# répartition notes - loi normale de 0 a 20
  set.seed(s)
  p <- rnorm(21, mean = 10, sd = 5) # loi normale
  p <- p - min(p)
  d <- max(p) - min(p)
  p <- (p/d)*20
}

E <- 0      # initialisation du cumul des E
nb <- 0      # initialisation du comptage du nombre de boucles

for (s in 1:10000) { # on va faire 10000 aléas
```

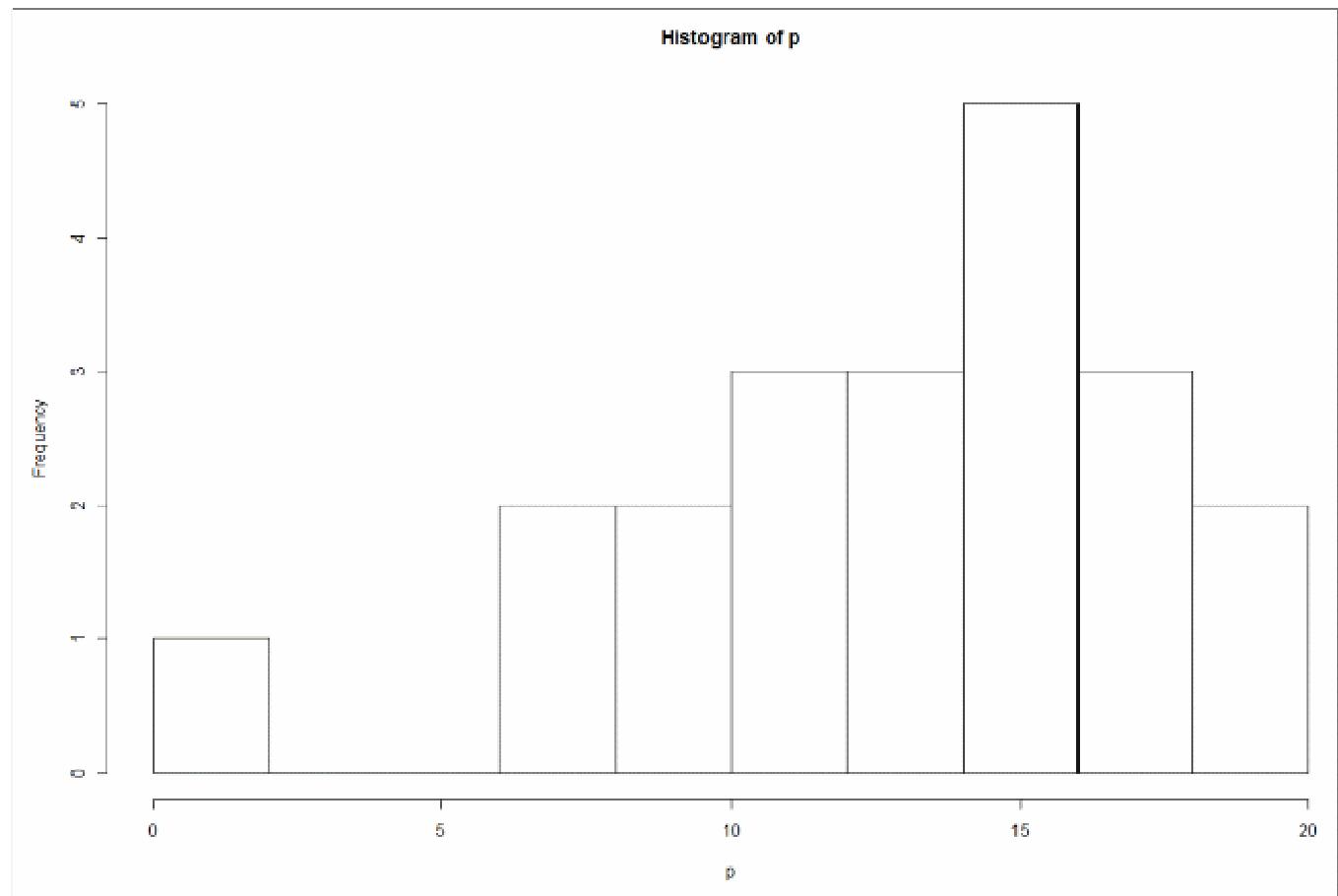
```

nb <- nb + 1
p <- fn(s)           # appel de la distribution
                      # dessinons une distribution sur 10000
if (nb == 1) hist(p, breaks = 10)

p <- p/sum(p)
x <- 0:20      # 21 valeurs de v[1] ...v[21], notes de 0 ... 20
E <- E + sum(p*x)
}
E <- E/nb # calcul de la moyenne des 10000 espérances
E

```

Ce qui nous donne l'histogramme suivant pour la première itération, où l'on constate sur ce premier tirage que la note de 10 n'est sans doute pas la moyenne.



Distribution réelle des notes obtenues à la première itération

Pourtant après les 10000 itérations on constate que l'espérance mathématique moyenne est de 10.00579.

Notre professeur aurait donc pu créer assez facilement une distribution de notes centrée vers la moyenne mais avec des notes plus vraisemblables (c'est-à-dire moins de notes extrêmes) s'il avait utilisé un mécanisme générant des distributions normales.

Espérance mathématique (m_1) : généralisation

La formule de moyenne utilisée plus haut ne fonctionne que sur les distributions discrètes, l'expression suivante est plus générale :

$$\mathbb{E}(x) = \int_{\mathbb{R}} x \mathbb{P}_X(dx)$$

Cette dernière expression se simplifie dans le cas où la variable aléatoire est continue et possède une densité de probabilité f , telle que la probabilité pour tout intervalle réel s'exprime de la façon suivante en fonction de f :

$$\mathbb{P}_X([a; b]) = \int_a^b f(x) dx$$

Dans ce cas on obtient :

$$\mathbb{E}(x) = \int_{-\infty}^{+\infty} x f(x) dx$$

En regardant attentivement cette formulation on constate que ce n'est toujours que la moyenne des x ; en effet $f(x)$ est une densité comprise entre 0 et 1.

Attention : on trouve d'autres notations, que nous utiliserons parfois lorsqu'elles sont plus pratiques ou traditionnelles dans un contexte donné, ou lorsque l'on veut souligner une nuance :

$$\mathbb{E}(x) = \langle x \rangle = \bar{x} = \mu_x$$

Le plus souvent, \bar{x} désigne la moyenne de la distribution (empirique), alors que μ_x désigne la moyenne de la loi.

La variance : moment m_2

Maintenant que nous disposons d'une représentation permettant de percevoir la tendance centrale d'une distribution, nous allons construire une représentation de sa dispersion.

En effet, en toute première approximation, si vous avez une idée du "centre" de la distribution et de son étalement, vous avez déjà une certaine perception de votre distribution

On construit donc une différence entre la variable et sa moyenne, mise au carré pour éviter qu'une dispersion à droite ne compense une dispersion à gauche et on en calcule la moyenne au sens statistique, à savoir son espérance mathématique.

L'expression résultante est donc :

$$m_2 = \mathbb{E} [(X - \mu)^2]$$

Ce moment a pour nom *variance*.

Pour mieux appréhender la dispersion sur une distribution opérationnelle, il faut pouvoir comparer des valeurs comparables, c'est-à-dire pouvant porter la même unité, on dit aussi la *même dimension* (mètre, kilo, âge...) que la variable aléatoire. Hors la variance est homogène au carré de l'unité de la variable aléatoire. On en prend donc la racine, que l'on appelle écart-type (*standard deviation* en anglais : stdv).

De ce fait on écrit souvent la variance comme étant le carré de l'écart-type : σ_x^2 .

Dans le cas discret cette expression devient évidemment :

$$\sigma_x^2 = \sum_i p_i (x_i - \mu)^2$$

et dans le cas continu :

$$\sigma_x^2 = \int_{-\infty}^{+\infty} (x_i - \mu)^2 f(x) dx$$

Au quotidien, l'écart-type est très souvent utilisé directement pour appréhender la dispersion. Pourtant, imaginez une distribution avec une moyenne de 10 et un écart-type de 1 et une autre avec une moyenne de 1000 et un écart-type de 2, vous pourriez légitimement penser que la dispersion de la première est plus conséquente que celle de la deuxième alors que son écart-type est de moitié. Cette considération amène à l'utilisation d'autres indicateurs de dispersion comme le *coefficient de variation* (à ne pas confondre avec son homonyme en analyse mathématique). Le coefficient de variation est tout simplement l'écart-type divisé par la moyenne, ce qui rend les comparaisons possibles. Notez toutefois que lorsqu'il y a des valeurs négatives la moyenne peut être nulle et que le coefficient de variation n'a pas de sens.

S'il n'y a pas d'ambiguïté sur le nom de la variable aléatoire, son expression est la suivante :

$$C_V = \frac{\sigma}{\mu}$$

En anglais, on parle parfois de RSD pour *Relative Standard Deviation*, ce qui correspond au coefficient de variation en valeur absolue et est souvent exprimé en pourcentage.

Une autre façon de mesurer la dispersion, particulièrement robuste, est tout simplement de mesurer l'écart de valeur qui correspond à un certain pourcentage de la population étudiée (typiquement 50 %), centré sur la médiane telle que 50 % de la population est en dessous et le reste au-dessus de celle-ci. On parle d'écart interquartile, puisque cela correspond de fait à la différence entre les quartiles Q3 et Q1. En anglais on le nomme IQR pour *Interquartile Range* ou parfois *Midspread*.

Moments m_3 et m_4

Leurs noms respectifs sont les suivants : *Skewness* (coefficient de dissymétrie) et *Kurtosis* (coefficient d'aplatissement).

Ce sont des paramètres de forme.

Quand vous utiliserez leur implémentation en R il est souvent préférable d'utiliser des formulations dites moins "naïves" que les formulations de base de ces moments, nous verrons cela dans le code R plus bas.

Le Kurtosis d'une loi normale normalisée est égal à 3.

Comme la loi normale est une référence, on calcule souvent la valeur *kurtosis - 3*, que l'on nomme *excess_kurtosis*. L'*excess_kurtosis* de la loi normale vaut donc 0.

Une loi plus pointue possède un *excess_kurtosis* positif qui peut être de 3, et une loi très aplatie comme la loi uniforme possède un *excess_kurtosis* négatif inférieur à -1.

C'est un risque courant que d'appliquer des algorithmes qui presupposent la normalité des lois alors que leur coefficient d'aplatissement est très différent d'une loi normale ou qu'il existe une déformation ("longue queue"), typiquement à droite de la moyenne de la distribution.

Dans certains cas, on peut utiliser ces deux moments pour créer de nouvelles variables (features) à exploiter dans nos modèles de prédiction.

Voici le code R permettant de créer un petit tableau récapitulant les moyennes, écarts-types, skewness et excess_kurtosis sur trois distributions classiques : Uniforme, Normale et Poisson.

```

## calcul moyenne, écart-type, skewness, kurtosis de trois distributions ##
## création d'un petit tableau récapitulatif ##

library(e1071) # la boîte à outils
set.seed(10) # initialisation du générateur de nombres aléatoires
l_uniforme <- runif(10000) # 10000 valeurs sur loi uniforme
l_normale <- rnorm(10000, mean = 1) # sur loi normale
l_poisson <- rpois(10000, lambda = 1) # sur loi de Poisson

# combinons les colonnes
all <- cbind(l_uniforme,l_normale,l_poisson)

# application des fonctions
moyenne <- apply(all,2,mean)
ecart_type <- apply(all,2,sd)
skewness <- apply(all,2,function(x){skewness(x, type = 3)})
ekurtosis <- apply(all,2,function(x){kurtosis(x, type = 3)})

# combinons les lignes
all <- rbind(moyenne,ecart_type,skewness,kurtosis)
all <- round(all, digits = 1) # arrondi
all

```

Ce qui donne le tableau :

	l_uniforme	l_normale	l_poisson
moyenne	0.5	1	1.0
ecart_type	0.3	1	1.0
skewness	0.0	0	1.0
ekurtosis	-1.2	0	0.9

On constate que la loi de Poisson est dissymétrique avec une queue à droite (skewness positif et vaut 1) et que la loi uniforme est très plate (excess_kurtosis = -1.2).

Remarquez :

- 1) L'utilisation de `type = 3` dans les moments m_3 et m_4 , pour stipuler de ne pas utiliser les formes naïves de ces fonctions.
- 2) L'utilisation de `apply` pour appliquer une fonction sur des colonnes via le paramètre 2.
- 3) L'utilisation de fonctions génériques non nommées dans les `apply`.

Premières considérations sur les erreurs et estimations

Supposons que vous ayez effectué un calcul de moyenne. Si ce calcul a été effectué sur un échantillon d'une population plus importante, vous allez sans doute considérer que cette moyenne représente une estimation de la moyenne réelle de la distribution. Vous vous poserez donc alors légitimement la question de savoir quel est l'ordre de grandeur de l'erreur que vous vous apprêtez peut-être à commettre en considérant cette moyenne comme étant celle de toute la distribution.

La réponse à cette préoccupation en ce qui concerne l'estimation de la moyenne se nomme *erreur type de la moyenne* (ou en anglais SEM pour *Standard Error of the Mean*) et son expression est la suivante :

$$\frac{\sigma}{\sqrt{n}}$$

On constate que quand cet échantillon est très grand l'erreur type de la moyenne tend vers 0, ce qui va de soi puisque l'on calcule alors la moyenne sur un échantillon qui se rapproche de la population toute entière.

En termes de notation, l'usage "data science" est souvent de noter \bar{x} la moyenne calculée sur l'échantillon et μ la moyenne de la distribution de référence. De même on note souvent s l'écart-type calculé sur l'échantillon alors que σ désigne l'écart-type de la distribution de référence.

Pour signaler que l'on parle d'une estimation, l'usage est de mettre un petit chapeau sur le nom de l'objet considéré. Avec cette notation l'estimation de la moyenne μ s'écrit donc $\hat{\mu}$.

Avec cette notation, pour un **échantillon donné** l'erreur réelle commise sur le calcul de la moyenne est donc :

$$e = \hat{\mu} - \mu$$

Se familiariser avec ses données

Fort de nos nouvelles connaissances, nous allons utiliser quelques outils interactifs pour nous familiariser rapidement avec notre jeu de données. Ici nous allons observer rapidement un petit jeu de données statistiques (Longley) sur le chômage aux US que nous avons francisé pour vous. Ce jeu de données est inclus dans R nativement dans le package **stats**.

```
require(stats)
longley
```

Avec ce cours est fournie la version francisée de ce jeu de données accessible en invoquant la fonction décrite dans le code suivant. De plus, ce code vous présente une représentation des données par pairs.

```
# LECTURE ET FRANCISATION DATA SET LONGLEY SUR PIB US ET CHOMAGE
# usage en mode bavard (default): Z <- f_longley()
# usage en mode muet : Z <- f_longley(FALSE)
f_longley <- function(verbose = TRUE){
  require(stats)      # statistiques de base
  require(graphics)   # graphiques de base
  require(dplyr)       # librairie de manipulation de données
  source("pair_panels.R")
  Z <- data.frame(longley[, 1:7], row.names = NULL) # lecture
  Z <- rename(Z,
    an = Year,
    indice_PIB = GNP.deflator,
    PIB = GNP,
    chomeurs = Unemployed,
    militaires = Armed.Forces,
    population = Population,
    travailleurs = Employed)

  ordre <- sort(names(Z))      # obtenir les colonnes par ordre alphabétique
  Z <- data.frame(Z[,ordre])  # reclasser les colonnes

  if (verbose){
    glimpse(Z)           # voir ce qu'il y a dans les données
    pairs(Z,diag.panel = panel.hist,
      upper.panel = panel.cor,
      lower.panel = panel.smooth,
      main ="visualisation rapide : data longley")
    y <- data.frame(travailleur = Z$travailleur) # régression
    X <- select(Z, -travailleur)                  # features
    print(summary(lm(y[,1]~ .,X)))             # REGRESSION TEST
  }
  Z                                # data_frame
}

df1 <- f_longley()      # notre data frame se nommera df1
```

Dans RStudio, si vous cliquez sur **df1** dans la fenêtre en haut à droite vous obtenez ceci :

Visualisation data.frame "Longley" (francisé) sous RStudio

Voyons maintenant comment appréhender nos données de façon interactive.

R Commander

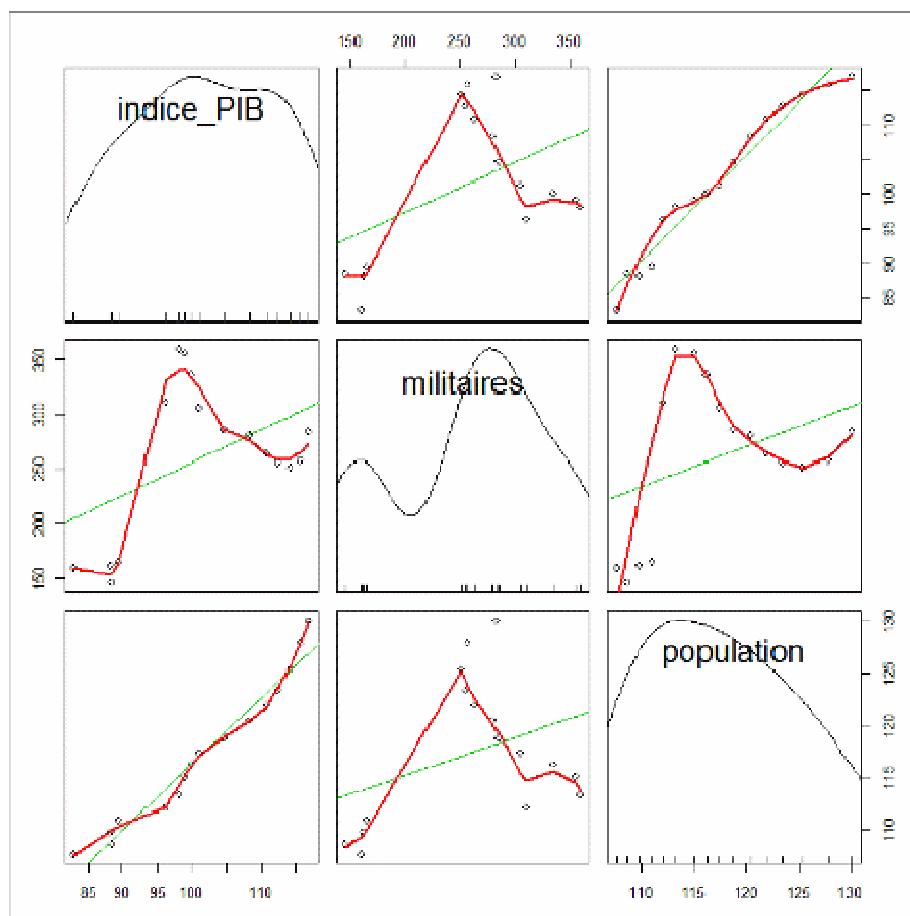
L'outil **R Commander** permet de visualiser un certain nombre d'informations sur vos données.

Pour l'invoquer il suffit d'invoquer le package.

```
library(Rcmdr)
```

Nous vous laissons explorer ce petit outil, mais pour pouvoir commencer, sachez qu'il vous faut lui signaler les données que vous allez utiliser via les menus **Données - Jeu de données actif - Sélectionner le jeu de données actif**.

Après avoir choisi le **data.frame df1** vous pouvez sélectionner les données que vous voulez croiser ensemble et visualiser les densités approximées des différentes distributions, ce qui est très utile pour comprendre son échantillon via les menus **Graphes - Matrice de points**. En sélectionnant des variables avec la touche [Ctrl] clic gauche et enfin en appuyant sur [Enter], vous obtenez des graphiques par paires.



Diagrammes par paires (régressions) et distributions

On constate que les distributions approximées (dans la diagonale) n'ont pas l'air très gaussiennes et que la distribution **militaire** a même l'air bimodale (deux sommets),

ce qui pourrait nous inciter à une certaine attention quant à l'emploi d'algorithmes qui auraient comme hypothèse la normalité des distributions.

Notre attention est attirée par la corrélation visuelle entre **population** et **indice_PIB**.

Cette linéarité est peut-être covariante, on est amené à regarder des graphiques en 3D pour mieux appréhender nos données. Le menu **Graphes - 3Dgraphes** nous le permet aisément et nous donne la capacité de faire tourner le graphe en utilisant notre souris.

Trois variables proches d'un plan

Vous constaterez avec plaisir que toutes les instructions R qui ont servi à vous façonner ces graphiques sont visibles et manipulables dans l'interface de R Commander.

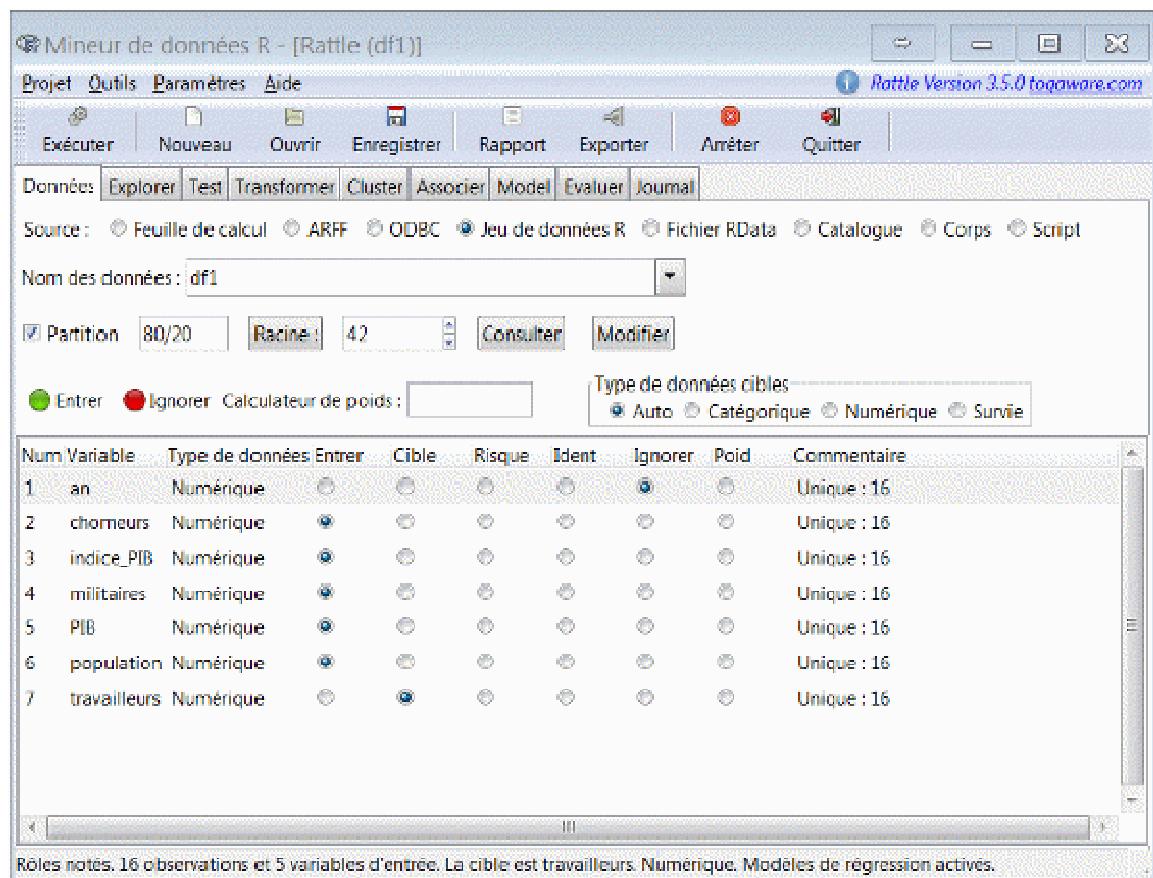
Un autre outil va nous apporter un complément d'information ; lui aussi est interactif, il se nomme **Rattle**.

Rattle

Nous allons utiliser cet autre outil R très pratique pour explorer nos données, le très connu "dataminer" Rattle. Rien de plus simple que de l'invoquer.

```
# rattle  
library(rattle)  
rattle()
```

Vous obtenez alors une interface que je vous propose de paramétriser en indiquant que vous voulez travailler avec **df1** comme jeu de données R, en stipulant que votre cible sera **travailleurs** et que vous allez ignorer **an**. La partition de vos données étant **80 %** pour l'entraînement, **20 %** pour le test/validation (sans séparer test de validation ici). Attention, il faut cliquer sur **Exécuter** après chaque changement dans l'interface graphique, sinon Rattle n'exécute rien.



Interface de Rattle

Rattle va vous permettre d'explorer vos données. Dans le fichier **Journal** vous trouverez l'ensemble des commandes R générées par Rattle, c'est très utile pour progresser en les étudiant. Rattle utilise de nombreux autres packages, il vous incitera donc à les installer au fur et à mesure que vous allez l'utiliser.

Sans doute aurez-vous du mal à résister au fait d'aller jeter un œil sur les corrélations linéaires entre variables numériques.

The screenshot shows the Rattle software interface. The menu bar includes Projets, Outils, Paramètres, and Aide. The toolbar has icons for Exécuter, Nouveau, Ouvrir, Enregistrer, Rapport, Exporter, Arrêter, and Quitter. The main menu bar is set to Données, Explorer, Test, Transformer, Cluster, Associer, Model, Evaluer, and Journal. The 'Cluster' option is currently selected. The status bar indicates 'Rattle Version 3.5.0 togaware.com'.

Under the 'Cluster' menu, the 'Type:' dropdown is set to 'Corrélation'. Other options include Résumé, Distributions, Composantes principales, and Interactif. There are checkboxes for 'Ordonnée' (checked), 'Explorer les valeurs manquantes' (unchecked), and 'Hiérarchique' (checked). The 'Méthode:' dropdown is set to 'Pearson'.

The main pane displays a correlation matrix:

```

militaires      PIB indice_PIB population   chomeurs
militaires  1.0000000  0.4118257  0.4393722  0.3399045 -0.1383884
PIB          0.4118257  1.0000000  0.9928679  0.9918176  0.6121132
indice_PIB   0.4393722  0.9928679  1.0000000  0.9801487  0.6196313
population    0.3399045  0.9918176  0.9801487  1.0000000  0.6926895
chomeurs     -0.1383884  0.6121132  0.6196313  0.6926895  1.0000000

```

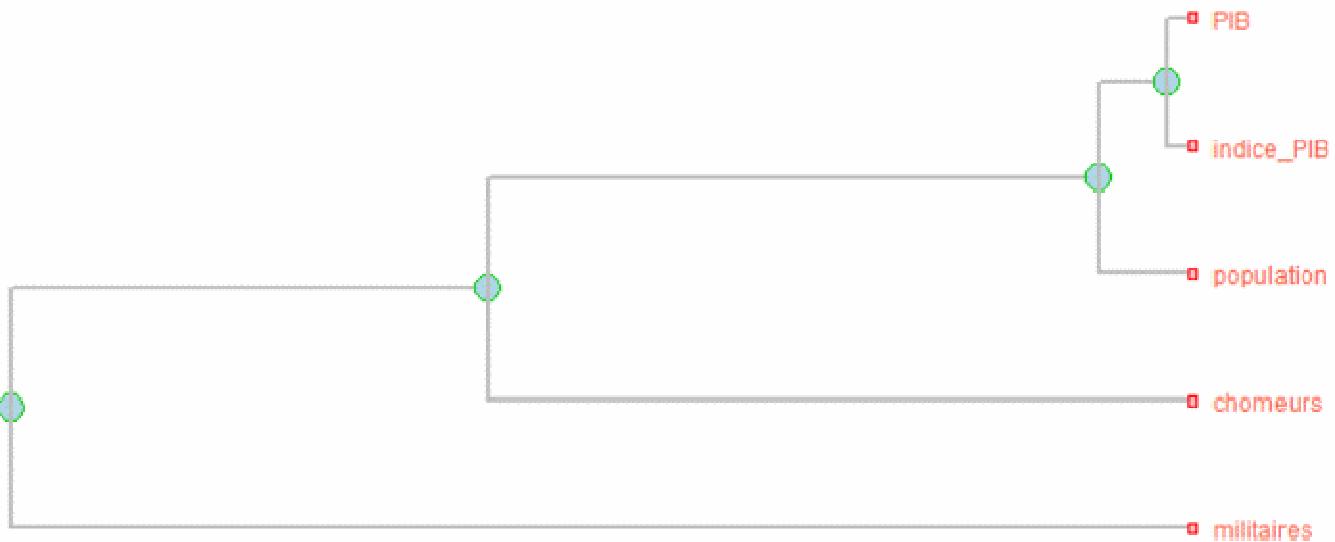
Below the matrix, a message states: "Notez que seules les corrélations entre les variables numériques sont prises en charge."

The bottom pane shows a dendrogram titled "Cluster hiérarchique de corrélations tracées".

Corrélations entre variables numériques

L'exécution de ce paramétrage vous a créé un **dendrogramme** des corrélations dans RStudio.

Clusters de corrélation de variables df1 utilisation de Pearson



Hiérarchie des corrélations

On commence à percevoir la structure des features ; évidemment il ne faut pas se ruer sur une quelconque conclusion, mais ces représentations participent à la compréhension de vos données.

Nous allons maintenant regarder si cela se "clusterise" un peu, à savoir si on trouve des groupes d'observations. Nous allons choisir une mécanique assez résistante au fait que notre connaissance des données soit faible. Nous allons effectuer un **clustering hiérarchique ascendant**, dont la mécanique est un peu simpliste, mais n'introduisant pas de complexités qui nuiraient à notre compréhension des données.

On se dote d'une distance, nous avons choisi la **distance de Canberra**. La distance de Canberra est une distance de Manhattan que l'on a pondérée. Autrement dit c'est une distance basée sur la valeur absolue, qui fonctionne par "pas" successifs, mais que l'on a pondérée en fonction des valeurs des variables. Ce qui fait que les "*pas*" dans les différentes directions ont un impact similaire.

Considérons deux lignes quelconques d'observation et p features, la distance de Canberra entre les deux observations est la suivante :

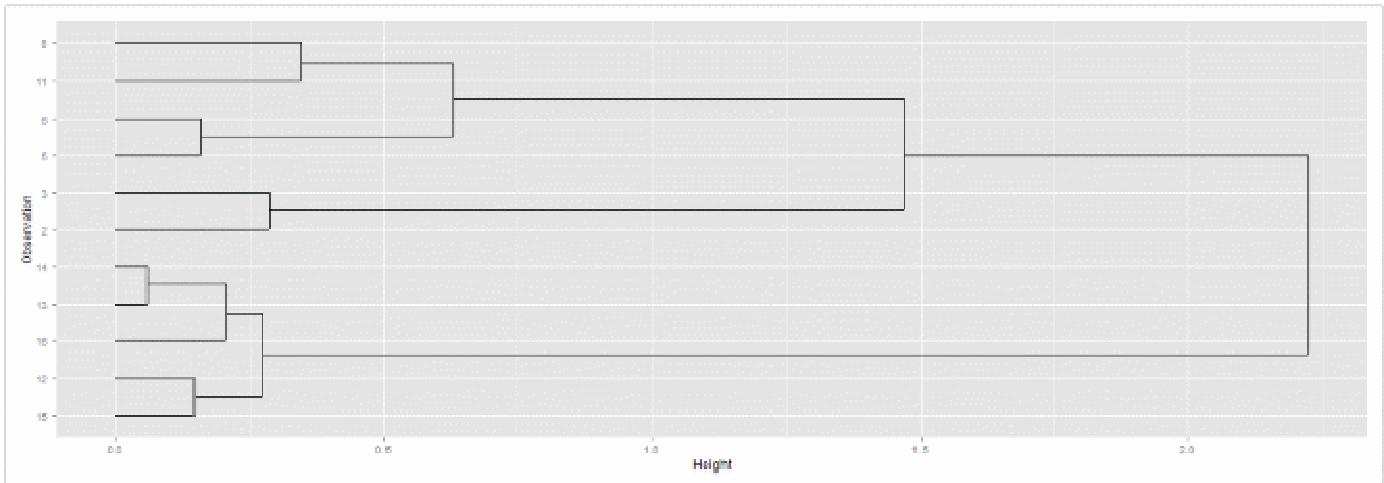
$$d(x_{i1}, x_{i2}) = \sum_{j=1}^{j=p} \frac{|x_{i1,j} - x_{i2,j}|}{|x_{i1,j}| + |x_{i2,j}|}$$

En bref : l'algorithme commence par chercher les deux points les plus proches et forme le premier cluster. Il calcule le centre des deux points, ce qui donne un nouveau point (factice) puis on itère. Les points sont remplacés par leur centre et on recommence, chaque point factice devient un niveau dans le dendrogramme correspondant. Si bien que l'on obtient une classe unique ! C'est cette classe vue comme un dendrogramme que l'on peut ensuite explorer. Le dendrogramme de construction sera scindé en clusters, on le coupe à l'endroit de la meilleure répartition.

	chomeurs	indice_PIB	militaires	PIB	population
[1.]	424.760	114.040	262.040	500.5836	125.7234
[2.]	263.625	100.675	321.025	370.4638	115.5023
[3.]	300.350	88.350	153.600	258.7400	109.2025

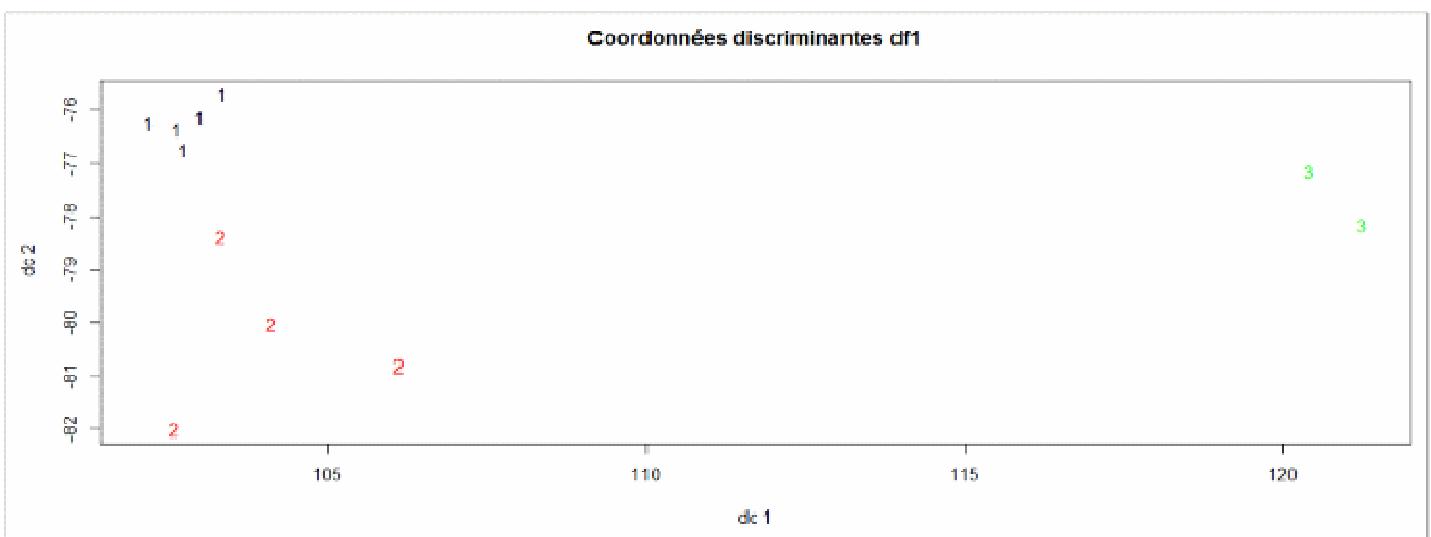
Le menu Cluster

On voit nettement les trois clusters.



Trois classes dans le dendrogramme

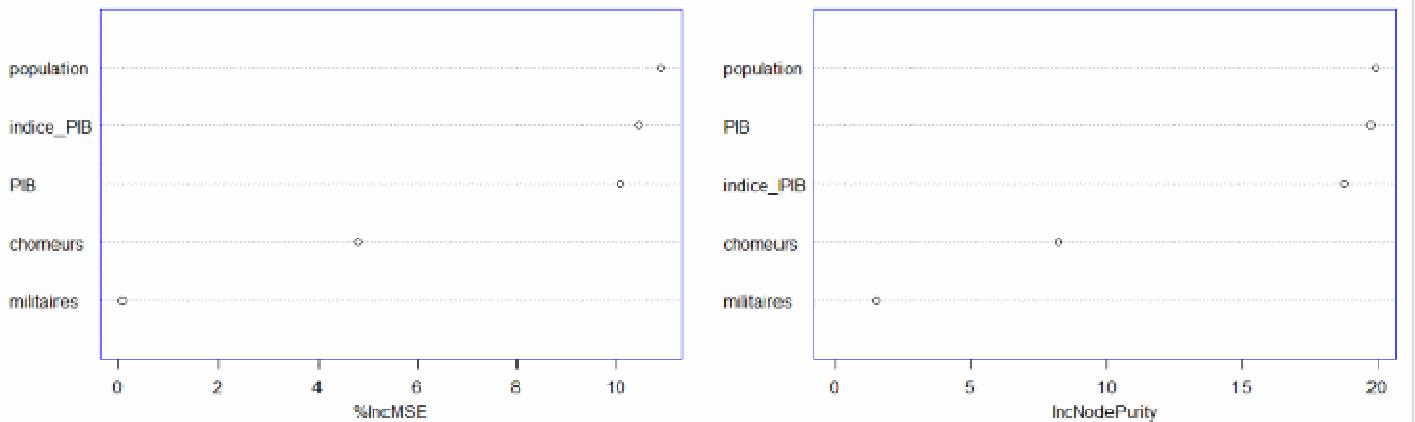
Ce qui apparaît également dans le diagramme suivant, qui montre que le cluster 3 est le plus marqué.



Clusters en deux dimensions

Dans le menu **Model** de Rattle, choisissez **Random forest**, et cliquez sur le bouton **Importance**. Vous obtenez un des sous-produits très utiles de l'application de ce modèle qui vous donne une certaine idée de l'influence des différentes variables sur une éventuelle prédiction.

Importance des variables Forêt aléatoire df1



Nous verrons au chapitre Feature Engineering, qu'il faut beaucoup nuancer votre interprétation d'un tel diagramme quant au fait d'éliminer telle ou telle feature. À l'inverse, il est probable qu'il vous faille composer un certain panel de features importantes et moins importantes pour effectuer une excellente prédiction.

Maintenant que vous avez eu l'occasion d'appréhender les données d'une façon plutôt descriptive, nous allons évoluer vers un traitement plus analytique et prédictif. Pour en comprendre les mécanismes, Il va nous falloir aborder les matrices et les vecteurs.

Matrices et vecteurs

Conventions, notations, utilisations basiques

Cette section n'a pas d'autre prétention que de vous familiariser avec l'utilisation pratique des matrices et des vecteurs, telle qu'elle est le plus souvent présentée dans ce texte. L'aspect mathématique sous-jacent n'est pas pris en compte (ou peu pris en compte).

Fabriquons une matrice

Supposez un dataset composé de lignes (par exemple représentant des évènements ou des individus) et dont les colonnes représentent les valeurs pour chaque attribut (en anglais feature). Une ligne décrivant un individu nommé Robert, d'âge 40 ans, de statut marié et habitant dans la ville Paris pourrait être encodée de la façon suivante dans un tableau avec d'autres individus :

```
!nom      ! âge ! statut ! ville    !
!          !     !           !
!ROBERT   !  40!      M ! PARIS    !
!PHILIPPE !  30!      C ! LYON    !
!RODOLF   !  20!      C ! LYON    !
!MARIE    !  35!      M ! PARIS    !
```

Si l'on mémorise le numéro de ligne de chaque individu :

```
!nom      ! ligne !
!          !       !
!ROBERT   !  1   !
!PHILIPPE !  2   !
!RODOLF   !  3   !
!MARIE    !  4   !
```

Si on encode le statut de la façon suivante : "M" vaut 1 et "C" vaut 0, puis si on remplace la ville par son code postal, on peut obtenir un tableau de chiffres ayant la même signification :

```
! ligne ! âge ! statut ! ville    !
!          !     !           !
! 1      !  40!      1 ! 75000    !
! 2      !  30!      0 ! 69000    !
! 3      !  20!      0 ! 69000    !
! 4      !  35!      1 ! 75000    !
```

Si par ailleurs on se souvient de la signification et de l'ordre de chaque colonne, il devient donc inutile de les nommer. On obtient :

```

! 1      ! 40!      1 ! 75000 !
! 2      ! 30!      0 ! 69000 !
! 3      ! 20!      0 ! 69000 !
! 4      ! 35!      1 ! 75000 !

```

Comme on voit que 1 est à la ligne 1, 2 à la ligne 2... Il devient évident que l'on peut être encore plus économique en notation et ne plus mentionner le numéro de ligne, sans perdre la moindre information :

```

! 40! 1 ! 75000 !
! 30! 0 ! 69000 !
! 20! 0 ! 69000 !
! 35! 1 ! 75000 !

```

Cette représentation comporte quatre lignes et trois colonnes d'éléments **homogènes en termes de nature**. Nous dirons que c'est une **matrice**. On peut avoir des matrices de nombres entiers ou réels ou complexes, de booléens (vrai - faux). On peut même imaginer des matrices de matrices !

La condition d'homogénéité en termes de nature n'implique aucune homogénéité en termes de sémantique. Chaque élément peut représenter quelque chose de complètement différent (par exemple l'âge n'a rien à voir sémantiquement avec le code postal !). Dans les matrices que nous manipulerons en data sciences on aura toujours une homogénéité sémantique en ligne ou en colonne ; la plupart du temps en colonne (on a ici trois colonnes homogènes sémantiquement, la première représente des âges, la deuxième le statut, la troisième la ville).

On aime bien noter les matrices par une lettre majuscule (parfois en gras), mais rien ne nous y oblige. Il est d'usage de mettre les éléments entre parenthèses, si bien que notre matrice que nous pouvons noter M s'écrira alors de la façon suivante :

$$M = \begin{pmatrix} 40 & 1 & 75000 \\ 30 & 0 & 69000 \\ 20 & 0 & 69000 \\ 35 & 1 & 45000 \end{pmatrix}$$

Utilisons notre matrice

Si l'on veut évoquer cette matrice, il suffira de parler de M. si l'on veut se souvenir que c'est une matrice de quatre lignes et trois colonnes on peut noter : $M_{4,3}$.

Notez bien ce "motto" : **ligne par colonne** (et non pas l'inverse !). Ici nous avons 4×3 éléments dans la matrice.

Chaque élément de la matrice peut être désigné par son positionnement dans la matrice, par exemple l'élément en deuxième ligne et troisième colonne vaut 69000. C'est la ville de Philippe. Quand on a noté la matrice via une lettre majuscule, il est courant d'utiliser la même lettre en minuscule pour désigner un élément. Évidemment, il faut repérer celui-ci par son numéro de ligne et son numéro de colonne que l'on met en indices. La ville de Philippe se notera donc : $m_{2,3}$.

On a $m_{2,3} = 69000$.

Dans le même esprit on a :

$$M = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \\ m_{4,1} & m_{4,2} & m_{4,3} \end{pmatrix} = M_{4,3}$$

Pour définir une matrice avec R, il faut lui indiquer le nombre de lignes (ou de colonnes), et lui dire si l'on veut qu'il la remplisse ligne par ligne ou colonne par colonne, ce qui donne le code suivant pour notre matrice M :

```
#> ## saisir une matrice, par ligne, 4 lignes ##

M <- matrix (byrow = TRUE,           # par ligne
             nrow = 4,            # nombre de lignes
             c( 40, 1 , 75000 ,
                30, 0 , 69000 ,
                20, 0 , 69000 ,
                35, 1 , 75000 )
)
M
```

Ce qui nous donne effectivement :

```
[,1] [,2] [,3]
[1,] 40    1 75000
[2,] 30    0 69000
[3,] 20    0 69000
[4,] 35    1 75000
```

Pour obtenir $m_{2,3}$ il suffit d'écrire (notez les crochets au lieu de parenthèses) :

```
M[2,3]          # deuxième ligne
              # et troisième colonne
```

On obtient 69000 comme escompté.

Pour obtenir toute la deuxième ligne il suffit de ne pas stipuler le numéro de colonne (notez la virgule suivie de... rien!) :

```
M[2, ]          # deuxième ligne
is.matrix(M[2, ]) # ce n'est pas une matrice !
```

La deuxième ligne est bien extraite (notez bien que le résultat est monoligne, donc R indique [1]), c'est-à-dire une ligne de résultat :

```
[1] 30    0 69000
[1] FALSE
```

Mais le résultat n'est malheureusement pas une matrice d'une ligne comme l'a montré le résultat **FALSE** donné par la fonction **is.matrix()**. Pour obtenir une matrice il faut l'imposer, ce point est très important.

```
m2_ <- matrix(byrow = TRUE,      # une matrice d'une ligne
               nrow = 1,
               M[2, ])
```

On peut en vérifier les dimensions :

```
m2_
is.matrix(m2_)          # c'est une matrice
dim(m2_)                # dimension d'une ligne, 3 colonnes
```

C'est une matrice d'une ligne et de trois colonnes :

```
[,1] [,2] [,3]
[1,]    30     0 69000
[1] TRUE
[1] 1 3
```

Dans le même esprit, nous allons extraire une colonne de la matrice, la colonne 3 :

```
m_3 <- matrix(byrow = TRUE,           # une matrice d'une colonne
               nrow = 4,
               M[, 3])
m_3
is.matrix(m_3)                      # c'est une matrice
dim(m_3)                            # une colonne de 3 lignes
```

On obtient une matrice d'une colonne :

```
[,1]
[1,] 75000
[2,] 69000
[3,] 69000
[4,] 75000
[1] TRUE
[1] 4 1
```

Extraire des vecteurs

Dans notre pratique quotidienne nous appellerons **vecteur** une matrice comportant **une seule colonne**.

La constitution de vecteurs est aisée, en effet la fonction **matrix()** a un comportement par défaut qui consiste à fabriquer un vecteur colonne si on ne lui stipule aucun paramètre.

On peut simplifier la création du vecteur colonne **m_3** de la façon suivante :

```
m_3 <- matrix(M[, 3])          # colonne par defaut  
m_3  
  
[,1]  
[1,] 75000  
[2,] 69000  
[3,] 69000  
[4,] 75000
```

Mais ce n'est pas le résultat attendu pour **m2_**, car nous désirons une ligne, il faut donc transposer la colonne obtenue pour obtenir une matrice ligne. Cela s'effectue en utilisant la fonction de transposition **t()**.

```
m2_ <- matrix(M[2, ])          # ! colonne par defaut  
m2_                                         # Problème : on obtient une colonne !  
  
m2_ <- t(matrix(M[2, ]))          # transposition  
m2_
```

Le premier résultat ne nous convenait pas :

```
[,1]  
[1,] 30  
[2,] 0  
[3,] 69000
```

Après transposition on obtient bien une matrice ligne :

```
[,1] [,2] [,3]  
[1,] 30    0 69000
```

Différentes notations pour les vecteurs et leur transposée

En France on aime bien affubler les vecteurs d'une petite flèche, de la façon suivante (ici pour un vecteur "x") :

$$\overrightarrow{x}$$

Cette notation est utile lorsqu'il peut y avoir une ambiguïté, ce qui est rarement le cas dans notre discipline. Ce n'est donc pas la notation que nous utiliserons couramment. Nous nous contenterons de nommer x , le vecteur en question !

Avec cette notation, la matrice ligne obtenue par la transposition du vecteur x , se notera :

$$\overrightarrow{x}^T$$

Le T signifiant "transposée de" et n'étant pas un exposant (attention !).

D'autres préfèrent écrire :

$${}^T\overrightarrow{x}$$

Pour signifier "transposée de", certains auteurs utilisent aussi le signe "prime" :

$$\overrightarrow{x}'$$

Le signe "prime" peut avoir de nombreuses autres significations (dérivée, prime vs second...), vérifiez toujours le sens des notations utilisées par un auteur.

Parfois les auteurs nomment "vecteur-ligne" la transposée d'un vecteur "normal", que nous appelons souvent "vecteur-colonne".

Vecteur ou vector ?

Dans R, vous aurez souvent l'occasion de manipuler des données de type **vector**. Attention, ce ne sont pas à strictement parler des vecteurs, au sens où vous ne pouvez pas effectuer naturellement les opérations que l'on effectue entre vecteurs et matrices.

Le type **vector** est fondamental dans R, il représente la combinaison de plusieurs valeurs en un agrégat manipulable simplement par de nombreuses fonctions. L'utilisation de ce type de structure de données est caractéristique d'un langage vectorisé. Un tel langage vectorisé permet l'économie de nombreuses boucles, et simplifie la compacité du code. Unitairement ces vectors se comportent comme des vecteurs normaux pour quelques opérations simples : l'addition de deux vectors additionne leurs éléments, la multiplication d'un vector par un scalaire (c'est-à-dire un nombre) multiplie chaque élément par ce nombre. Certaines fonctions appliquées au vector s'appliquent unitairement à chaque élément.

Créons un **vector** par la fonction de combinaison **c()** et étudions le résultat.

```
v <- c(1,2,7,10)          # création d'un vector
v
dim(v)                      # pas de dimension
length(v)                   # sa longueur
str(v)                      # en détail

[1] 1 2 7 10
NULL
[1] 4
num [1:4] 1 2 7 10
```

Au vu de ces résultats, le **vector** a été créé. Il n'a pas de dimension (**NULL**) et ce n'est donc pas un vecteur. Sa longueur est de quatre éléments. Il est composé de nombres (**num**).

Appliquons-lui quelques opérations élémentaires.

```
2*v                      # produit sur chaque composant
2*v + 100                 # encore mieux
v^2                      # carré sur chaque composant

[1] 2 4 14 20
[1] 102 104 114 120
[1] 1 4 49 100
```

Ce sont bien les résultats espérés, voyons ce qui se passe avec deux **vectors**.

```
w <- c(-10,-20,-30,-100)  # un autre vector
v+w                      # addition membre à membre
v*w                      # produit membre à membre! attention
```

```
[1] -9 -18 -23 -90
[1] -10 -40 -210 -1000
```

L'addition de deux **vectors** a bien fonctionné.

La multiplication de deux **vectors** n'est pas du tout une multiplication vectorielle classique, mais une multiplication des éléments membre à membre (cela se nomme produit de Hadamard).

On peut faire des opérations entre les éléments d'un **vector**, ici somme et moyenne :

```
sum(v)                      # somme des éléments
mean(v)                     # moyenne des éléments

[1] 20
[1] 5
```

Souvent on a besoin du **produit scalaire** de deux vecteurs. Voyons si on peut l'obtenir avec deux vectors. Nous verrons plus loin divers usages du produit scalaire. Voici deux façons de calculer ce produit scalaire. Ne soyez pas étonné de la formulation `%*%`, en effet R permet de définir des opérations spécifiques à des types d'objets. Dans ce cas on encadre le nom de l'opération par des pourcentages. Ici l'opération `%*%` appliquée aux **vectors** a été définie par les concepteurs de R comme une autre forme de multiplication (le produit scalaire) :

```
sum(v*w)                    # produit scalaire v, w
v%*%w                       # produit scalaire v, w

[1] -1260
[1,] -1260
```

On constate une petite différence dans le résultat (la virgule). Voyons ce à quoi nous avons affaire.

```
is.vector(sum(v*w))        # c'est un vector
is.vector(1789)             # un scalaire simple est un vector
is.vector(v%*%w)            # ce n'est pas un vector
is.matrix(v%*%w)            # mais une matrice !

[1] TRUE
[1] TRUE
[1] FALSE
[1] TRUE
```

Le premier calcul donne le produit scalaire sous forme d'un **vector**, notons qu'un nombre quelconque est un vector. Le deuxième n'est pas un **vector**, mais une matrice avec un seul élément qui contient le produit scalaire.

Nous allons maintenant regarder comment appliquer nos propres fonctions sur des **vectors** ou des matrices.

apply() cas 1 : vector vers vector (issu de vecteur vers vector)

Nous allons maintenant appliquer une fonction quelconque à chaque élément d'un vector. **Soyez très attentif à cette syntaxe** qui peut étonner. La fonction **apply()** utilisée ici est au cœur du langage R.

Tout d'abord créons notre fonction quelconque, ici une fonction qui crée quelques perturbations sur des valeurs.

```
f <- function(x) {x^(1.001)-0.001} # une fonction quelconque  
f(10)                                # test de la fonction  
  
[1] 10.02205
```

Cette fonction transforme un petit nombre en une valeur très proche, ce qui va nous faciliter le repérage de son application dans les résultats qui vont suivre.

Pour appliquer **f()** sur chaque élément de **v**, qui est un **vector**, et obtenir un nouveau **vector**, utilisons la syntaxe suivante :

```
u <- apply(matrix(v), 1, f) # application sur un vector  
u  
is.vector(u)                # c'est un vector  
  
[1] 0.999000 2.000387 7.012635 10.022052  
[1] TRUE
```

Voyons comment cela fonctionne :

- La fonction **apply()** possède trois paramètres.
- Ici on a choisi de l'appliquer sur une matrice, car c'est plus simple à maîtriser.
- Le premier paramètre représente ce sur quoi on va poser la fonction à appliquer. Nous avons décidé d'appliquer la fonction sur une matrice colonne issue de notre vector **v**, en effet **apply()** est efficace sur des structures possédant une ou des dimensions, comme les matrices et non pas comme les vectors. Dans notre fonction, **apply()** va être appliquée au vecteur **matrix(v)**. **C'est donc une application d'un cas : vecteur vers vector.**
- Le deuxième paramètre stipule la dimension qui peut valoir **1**, **2** ou **c(1, 2)** suivant que l'on travaille la première, la deuxième ou les deux dimensions d'une matrice.
- Le troisième paramètre représente la fonction à appliquer, notez que l'on ne stipule pas les parenthèses.

En fait la fonction **apply()** est très riche et puissante, nous n'effleurons ici qu'une petite part de son utilisation.

apply() cas 2 : vector vers vecteur (issu de vecteur vers vecteur)

Seul le deuxième paramètre change.

```
u <- apply(matrix(v), 2, f) # application sur un vector
u
is.vector(u)                # ce n'est pas un vector
is.matrix(u)                # mais une matrice

[,1]
[1,] 0.999000
[2,] 2.000387
[3,] 7.012635
[4,] 10.022052

[1] FALSE
[1] TRUE
```

On a obtenu un vecteur colonne. On est en fait dans un cas **vecteur vers vecteur** appliqué au vecteur **matrix(v)** qui est le cas le plus "normal".

apply() cas 3 : matrice vers matrice

Traitons ici l'application sur une matrice qui n'est pas un vecteur, ni une transposée de vecteur, c'est-à-dire dont chaque dimension est strictement supérieure à 1. Appliquons notre fonction **f()** membre à membre sur notre matrice **M**.

```
N <- apply(M, c(1,2), f)      # application sur une matrice
N
is.matrix(N)                  # c'est une matrice

[,1]    [,2]    [,3]
[1,] 40.14683  0.999 75846.64
[2,] 30.10121 -0.001 69773.09
[3,] 20.05900 -0.001 69773.09
[4,] 35.12366  0.999 75846.64

[1] TRUE
```

apply() : autres cas utiles avec fonctions génériques R

Souvent, en bas d'un tableau on tire une ligne et on fait une addition, une moyenne...

Voyons ce type de comportement en utilisant **apply()** sur une matrice.

Effectuons la moyenne de chaque colonne de **M** :

```
t(matrix(apply(M, 2, mean))) # moyenne par colonne

[,1] [,2]  [,3]
[1,] 31.25 0.5 72000
```

Effectuons la moyenne de chaque ligne d'une matrice **T** (nous avons choisi **T** comme la transposée de **M**) :

```
T <- t(M)
T
matrix(apply(T,1,mean))      # moyenne par ligne
```

La matrice **T** :

```
[,1]   [,2]   [,3]   [,4]
[1,]     40     30     20     35
[2,]     1      0      0      1
[3,] 75000 69000 69000 75000
```

Moyenne par ligne de **T** :

```
[,1]
[1,]    31.25
[2,]    0.50
[3,] 72000.00
```

Nous pouvons utiliser cette méthode pour diverses fonctions génériques :

- sum : somme de plusieurs nombres.
- min : minimum de plusieurs nombres.
- max : maximum de plusieurs nombres.
- stdev : écart-type de plusieurs nombres.
- prod : produit de plusieurs nombres.

Ce qui correspond au code suivant :

```
t(matrix(apply(M,2,sum)))  # somme de chaque colonne
t(matrix(apply(M,2,min)))  # min de chaque colonne
t(matrix(apply(M,2,max)))  # max de chaque colonne
t(matrix(apply(M,2,stdev)))# écart-type de chaque colonne
t(matrix(apply(M,2,prod))) # produit des termes par colonne

matrix(apply(T,1,sum))  # somme de chaque ligne
matrix(apply(T,1,min))  # min de chaque ligne
matrix(apply(T,1,max))  # max de chaque ligne
matrix(apply(T,1,stdev))# écart-type de chaque ligne
matrix(apply(T,1,prod)) # produit des termes par ligne
```

Matrice

Une courte introduction

Dans cette section, nous appellerons **n** le nombre de lignes de notre matrice d'observations (nombre de faits observables) et **p** son nombre de colonnes (nombre d'attributs du problème). Du fait de la force de l'habitude, nous appellerons **X** la matrice des observations. Cette matrice est donc une matrice de dimension **n x p**.

La forme de cette matrice X est :

$$\begin{pmatrix} x_{1,1} & \cdots & x_{1,p} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,p} \end{pmatrix}$$

L'indice de ligne varie entre 1 et n, nous allons appeler i cet indice.

L'indice de colonne varie entre 1 et p, nous allons appeler j cet indice.

Ce qui peut s'exprimer en déclarant que chaque élément de la matrice a la forme $x_{i,j}$ avec :

$$1 \leq i \leq n \text{ et } 1 \leq j \leq p$$

D'une façon plus compacte, on écrira :

$$X = [x_{i,j}]_{\substack{1 \leq i \leq n \\ 1 \leq j \leq p}}$$

La transposée de la matrice X, qui s'écrit en permutant les rôles respectifs des lignes et des colonnes s'écrirait alors :

$$X^T = [x_{j,i}]_{\substack{1 \leq i \leq n \\ 1 \leq j \leq p}}$$

Comme les indices sont conventionnels, on dit aussi "muets", cette matrice transposée peut également s'écrire :

$$X^T = [x_{i,j}]_{\substack{1 \leq i \leq p \\ 1 \leq j \leq n}}$$

La première colonne de la matrice X est le vecteur :

$$\begin{pmatrix} x_{1,1} \\ \vdots \\ x_{n,1} \end{pmatrix} = x_1$$

La jème colonne de la matrice X est le vecteur :

$$\begin{pmatrix} x_{1,j} \\ \vdots \\ x_{n,j} \end{pmatrix} = x_j$$

La dernière colonne de la matrice X est le vecteur :

$$\begin{pmatrix} x_{1,p} \\ \vdots \\ x_{n,p} \end{pmatrix} = x_p$$

On peut alors écrire :

$$X = (x_1, \dots, x_j, \dots, x_p)$$

comme étant composée des vecteurs colonnes représentant les colonnes d'attributs de nos observations. Souvent on considère une $p+1^{\text{ème}}$ colonne nommée par exemple y qui représente **la colonne sur laquelle on veut effectuer une prédiction**.

Dans la suite, on constatera que l'idée générale de l'apprentissage supervisé est de trouver **une estimation** de l'application f , le "modèle de prédiction", telle que l'on puisse déduire le vecteur y en connaissant la matrice X : $y = f(X) + \text{erreur}(X)$ (telle que l'erreur soit la plus petite possible, avec une moyenne à zéro par exemple).

Souvent on se place dans l'hypothèse que l'erreur ne dépend pas de X , c'est donc une erreur irréductible. On exprime cela sous la forme (à retenir) : $y = f(X) + \varepsilon$

Notons qu'en faisant cette hypothèse on considère maintenant que le vecteur y est une fonction de deux variables : la matrice X et le vecteur ε .

Plus loin dans la manipulation des matrices avec R

Opérations basiques

Additionner ou soustraire, comme multiplier par un scalaire (c'est-à-dire un nombre) est trivial :

```
M-N                                # soustraction membre à membre
[,1]   [,2]      [,3]
[1,] -0.14682767 0.001 -846.6352
[2,] -0.10120964 0.001 -773.0863
[3,] -0.05900448 0.001 -773.0863
[4,] -0.12365865 0.001 -846.6352

2*M                                # multiplication par 2 des membres
```

```
[,1] [,2] [,3]
[1,] 80    2 150000
[2,] 60    0 138000
[3,] 40    0 138000
[4,] 70    2 150000
```

On a aussi la possibilité d'extraire une sous-matrice :

```
M[2:3,2:3] # extraction sous-matrice
```

```
[,1] [,2]
[1,] 0 69000
[2,] 0 69000
```

Et on a la possibilité de changer les valeurs d'un "rectangle" de la matrice :

```
M[2:3,2:3] <- 2 * M[2:3,2:3] # remplacement de valeurs
M # dans une partie de la matrice
```

```
[,1] [,2] [,3]
[1,] 40    1 75000
[2,] 30    0 138000
[3,] 20    0 138000
[4,] 35    1 75000
```

Voyons maintenant comment procéder pour les travaux un peu plus lourds sur les matrices et les vecteurs.

Quelques savoir-faire utiles sur les matrices de R

Pour les calculs suivants, dotons-nous de deux vecteurs (séquences de nombres) :

```
x <- matrix( seq( 0, 4 ) ) # les 5 premiers entiers
print(x)
y <- matrix( seq( 5, 9 ) ) # les 5 suivants
print(y)

[,1]
[1,] 0
[2,] 1
[3,] 2
[4,] 3
[5,] 4

[,1]
[1,] 5
[2,] 6
[3,] 7
[4,] 8
[5,] 9
```

Pour aller plus loin, l'utilisation de la librairie **matrixcalc** est bien utile.

```
library(matrixcalc) # pour faire des calculs matriciels
```

Cette librairie nous donne la possibilité d'accéder à des matrices classiques déjà composées, dont on peut avoir l'usage dans divers cas de figure.

Nous allons créer deux matrices pour nos exemples de calculs ultérieurs.

Ici une matrice carrée **H** avec une sous-diagonale de nombres de 1 à 4 :

```
H <- creation.matrix( 5 )          # sous-diagonale 1 to n-1
print( H )
is.matrix(H)                      # c'est bien une matrice

[,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    1    0    0    0    0
[3,]    0    2    0    0    0
[4,]    0    0    3    0    0
[5,]    0    0    0    4    0

[1] TRUE
```

Ici une matrice de Vandermonde, où chaque vecteur colonne comporte une des puissances successives, en partant de 0, d'un vecteur donné (ici nous utilisons le vecteur des cinq premiers entiers non nuls, la première colonne est faite de 1 car un nombre puissance 0 vaut toujours 1).

```
V <- vandermonde.matrix(c(seq(1,5)),5) # matrice de Vandermonde
print( V )
```

```
[,1] [,2] [,3] [,4] [,5]
[1,]    1    1    1    1    1
[2,]    1    2    4    8   16
[3,]    1    3    9   27   81
[4,]    1    4   16   64  256
[5,]    1    5   25  125  625
```

Produit de Hadamard

Le produit de Hadamard de deux matrices est peu employé. Ce n'est pas le produit habituel utilisé entre matrices, mais il peut être très utile pour faire un filtre ou un masque sur une matrice (par exemple sur une matrice de pixels ; ces filtres seront détaillés en fin d'ouvrage).

Ce produit entre les matrices A et B est souvent noté A o B

```
# produit de Hadamard
# produit membre * membre : z_ij = v_ij * h_ij
z <- hadamard.prod (V,H)
print(z)

# c'est pareil que
z <- V * H                                # produit de Hadamard
print(z)

[,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
```

```
[2,] 1 0 0 0
[3,] 0 6 0 0
[4,] 0 0 48 0
[5,] 0 0 0 500
```

Produit matriciel "classique"

Le "vrai" produit matriciel est celui-ci, notez bien que cet opérateur produit est noté `%*` en R et est souvent noté par un point (ou rien) en mathématiques.

Pour multiplier deux matrices A et B via le produit matriciel classique, il faut que leurs dimensions soient compatibles : si A est de dimension n,p et B de dimension m,q il faut que p soit égal à m pour pouvoir effectuer la multiplication. Cette multiplication matricielle consiste à obtenir pour chaque position de la nouvelle matrice la somme des produits membre à membre de la ligne et de la colonne correspondantes des matrices à multiplier :

$$AB = [a_{i,k}]_{\substack{1 \leq i \leq n \\ 1 \leq k \leq p}} \cdot [b_{k,j}]_{\substack{1 \leq k \leq p \\ 1 \leq j \leq q}} = \left[\sum_{k=1}^p a_{i,k} b_{k,j} \right]_{\substack{1 \leq i \leq n \\ 1 \leq j \leq q}}$$

```
# produit z_ij = somme_sur_k(v_ik * h_kj) #  
Z <- V %*% H # lignes * colonnes  
is.matrix(Z) # c'est bien une matrice  
print(Z)
```

```
[,1] [,2] [,3] [,4] [,5]  
[1,] 1 2 3 4 0  
[2,] 2 8 24 64 0  
[3,] 3 18 81 324 0  
[4,] 4 32 192 1024 0  
[5,] 5 50 375 2500 0
```

Si A est la matrice d'une application linéaire f et x un vecteur appartenant à l'ensemble de définition de f et donc de dimension compatible : A.x = f(x).

Si A est la matrice d'une application linéaire f, et B la matrice d'une application linéaire g, alors AB est la matrice d'une application linéaire f \circ g c'est-à-dire de l'application successive de g puis de f à un vecteur. Avec les bonnes conditions en termes d'ensemble de définition on a : AB.x = f(g(x)) = f \circ g(x).

Attention : dans certains cas les matrices peuvent représenter autre chose que des applications linéaires habituelles et correspondre à de l'algèbre multilinéaire.

Ce type de produit fonctionne également entre vecteurs, et entre matrices et vecteurs, si tant est que les dimensions de ceux-ci soient compatibles.

Le produit d'un vecteur colonne par un vecteur ligne donne une matrice :

```
x%*%t(y) # on obtient une matrice
```

```
[,1] [,2] [,3] [,4] [,5]  
[1,] 0 0 0 0 0  
[2,] 5 6 7 8 9  
[3,] 10 12 14 16 18  
[4,] 15 18 21 24 27  
[5,] 20 24 28 32 36
```

Le produit de la transposée d'un vecteur par un autre se nomme aussi produit scalaire, il a la propriété d'être nul quand les deux vecteurs sont perpendiculaires.

```
t(x)%*%y # produit scalaire  
t(y)%*%x # idem
```

```
[,1]  
[1,] 80
```

```
[,1]  
[1,] 80
```

La racine carrée du produit scalaire d'un vecteur par lui-même est égale à sa norme euclidienne notée $\|x\|_2$ (voir fin de la section).

```
sqrt(t(x)%*%x) # norme euclidienne
```

```
[,1]  
[1,] 5.477226
```

Notez que l'on a obtenu des matrices 1x1 composées d'un scalaire unique, c'est donc en fait le contenu de cette mini matrice qui était le vrai résultat escompté lors de l'opération (concédons que dans la pratique cela fait rarement de différence !).

Il existe d'autres opérateurs parfois utiles entre les matrices. Essayez les trois suivants, dont les résultats sont trop encombrants pour être imprimés dans cet ouvrage.

Somme directe entre vecteurs

La somme directe de deux vecteurs est souvent notée \oplus .

```
# somme directe de matrices #  
# c'est une opération par bloc, à partir de matrices A et B #  
# on obtient une diagonale A-B et le reste étant des 0 #  
Z <- direct.sum( x, y) # somme "directe"  
print(Z) # deux colonnes de dim(10)
```

Somme directe entre matrices

La somme directe de deux matrices est souvent notée \oplus .

```
Z <- direct.sum( V, H) # somme "directe"  
print(Z) # 10 x 10
```

Produit de Kronecker

Le produit de Kronecker est souvent noté \otimes .

```
# produit de Kronecker = produit direct (c'est un p tensoriel) #
# chaque valeur aij de la première matrice A est remplacée      #
# par le produit de cette valeur et de toute la matrice B      #
# soit [aij * B]                                                 #

Z <- direct.prod( x, y)      # produit "direct"
print(Z)                      # 1 colonne de dim(25)
                             # x11 * y
                             # x21 * y ...
                             # ...

Z <- direct.prod( V, H )    # produit "direct"
print(Z)                      # 25 x 25
                             # v11 * H , v12 * H ...
                             # v21 * H , v22 * H ...
                             # v31 * H ...
```

Normes de vecteurs et normes de matrices

Les normes sont des fonctions à valeurs positives qui permettent de comparer des objets entre eux.

Normes de vecteurs

Quand on est face à plusieurs vecteurs, ou à plusieurs matrices sur un même espace, on est légitimement amené à vouloir classer ces objets entre eux, de trouver les extrêmes, d'établir et comparer des proportions...

Par exemple, imaginez avoir créé deux modèles de prédiction de y à partir de X :

$$y = f_1(X) + \varepsilon_1 = f_2(X) + \varepsilon_2$$

Nous aimions choisir entre ces deux modèles. Une première idée pourrait être "prenons celui avec la plus petite erreur". Oui, mais lequel de ε_1 ou ε_2 est le plus petit ?

Pour ce faire on va utiliser une norme $\|\cdot\|$ et on pourra voir si $\|\varepsilon_1\| < \|\varepsilon_2\|$ ou pas. Le choix de la norme peut influencer notre réponse, mais heureusement, concernant ce strict aspect de comparaison il existe une notion d'équivalence de normes qui fait que certaines normes – mais pas toutes – sont équivalentes et que l'on peut dans ce cas d'ailleurs choisir la plus facile à manipuler sans trop de perte de sens.

La plus connue des normes est la "norme euclidienne d'un vecteur", qui correspond à la **longueur de ce vecteur** dans un espace euclidien (par exemple dans l'espace géométrique en trois dimensions dans lequel nous vivons). Quand on dispose d'une base orthonormée (deux vecteurs unitaires tous perpendiculaires les uns aux autres en géométrie plane par exemple), le calcul de cette norme est une application simple du bon vieux théorème de Pythagore.

On note souvent les normes de la façon suivante : $\|x\|$

Mais comme il existe de nombreuses autres normes que la norme euclidienne, on ajoute parfois des indices pour éviter toute ambiguïté.

La norme euclidienne d'un vecteur x est donc également notée $\|x\|_2$ (en référence au carré dans l'expression plus bas).

Pour un vecteur $x=[X_i]$ en n dimensions, dont les coordonnées sont exprimées dans une base orthonormée, la norme euclidienne est la racine carrée de la somme des carrés des n coordonnées du vecteur :

$$\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$$

De nombreuses normes de vecteur sont à notre disposition.

La classe la plus importante de normes, en dimension finie, à laquelle appartient $\|\cdot\|_2$ a pour expression générale : $\|x\|_p = (\sum_i |x_i|^p)^{1/p}$.

Quand p est infini l'expression change et cette norme devient le maximum des $|x_i|$.

Voyons les différentes valeurs de ces normes pour un vecteur diagonal en dimension 3.

```
# normes de vecteurs #  
  
z <- matrix(c(1,1,1))      # vecteur diagonal 3D  
z  
n1  <- entrywise.norm( z, 1 )    # p= 1  
n2  <- entrywise.norm( z, 2 )    # norme euclidienne p= 2  
n10 <- entrywise.norm( z, 10 )   # p= 10  
n_i <- maximum.norm(z)         # norme max p= infini  
print (c(n1,n2,n10,n_i))  
  
[1] 3.000000 1.732051 1.116123 1.000000
```

Comparons avec les valeurs de ces normes pour un vecteur de la base orthonormée.

```
z <- matrix(c(0,1,0))      # vecteur de la base  
z  
n1  <- entrywise.norm( z, 1 )    # p= 1  
n2  <- entrywise.norm( z, 2 )    # norme euclidienne p= 2  
n10 <- entrywise.norm( z, 10 )   # p= 10  
n_i <- maximum.norm(z)         # norme max p= infini  
print (c(n1,n2,n10,n_i))  
  
[1] 1 1 1 1
```

On constate que l'application de ces normes change la topologie de l'espace vectoriel, puisque les proportions entre un vecteur de la base orthonormée et un vecteur quelconque ne sont pas les mêmes suivant la norme !

Par exemple, si je désire obtenir tous les points de l'espace affine situés à une certaine distance d'un point donné, je n'obtiens pas les mêmes points d'une norme à l'autre.

Mais au fait, qu'est-ce qu'une distance ?

Distances

À partir de normes on peut définir des distances. Par exemple, à partir de la norme euclidienne dans l'espace géométrique 3D dans lequel nous sommes plongés, on peut définir la distance entre deux points **a** et **b**. Ces deux points peuvent se définir comme des vecteurs construits en reliant l'origine d'un repère et les points de notre espace affine. En fait, pour être exact, il serait plus correct de stipuler que tout espace vectoriel peut être muni d'une structure d'espace affine par l'opération de soustraction vectorielle.

Dans notre espace on a :

$$\text{distance}(a, b) = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2} = \|b - a\|_2$$

Par analogie, les distances naturellement induites par les normes sont de la forme $\|b - a\|$, quelle que soit la définition de la norme et la nature des objets **a** et **b**.

Dans notre pratique de data science, le choix d'une distance peut être très impactant. L'exemple du **clustering**, à savoir la recherche de classes de fait au travers de la découverte d'amas d'observations, est manifestement sensible à la distance choisie, puisque suivant la nature des distances la topologie de l'espace est différente.

Les normes utilisées depuis longtemps ont des noms, par exemple :

$\|b - a\|_1$ se nomme distance de Manhattan.

$\|b - a\|_2$ se nomme distance euclidienne.

Certaines distances ne se déduisent pas de normes, mais toutes les normes permettent de définir des distances. Certaines quantités sont abusivement nommées distances, mais n'ont pas toutes les caractéristiques d'une distance. On les utilise quand même dans certains algorithmes car elles ont prouvé une certaine efficacité liée à la nature d'un problème particulier. Sans information particulière sur le contexte, utilisez de vraies distances qui conservent l'inégalité triangulaire. C'est-à-dire telles que la distance entre deux objets est toujours plus petite ou égale à la somme des distances correspondant au fait de passer par un troisième objet.

Normes de matrices

On peut facilement définir des normes sur les matrices en considérant une matrice sans ses dimensions et en lui appliquant une définition de norme de vecteur (en effet, si l'on considère les matrices comme des vecteurs colonnes qui seraient construits en tant que liste monodimensionnelle de tous les éléments de ces matrices on peut définir un espace vectoriel qui justifie cette démarche).

Pour calculer une telle norme de la matrice X, nous appliquons donc l'application `vec()` à la matrice pour en faire un vecteur, puis nous appliquons une norme de vecteur :

$$\|X\|_p = \|\text{vec}(X)\|_p$$

La fonction `entrywise.norm()` nous permet de calculer ces normes :

```
# normes de matrices                                         #

# Frobenius (généralisation : Hilbert- Schmidt)
print(frobenius.norm( V ))          # donne : 696.084

# qui est un cas particulier de
print( entrywise.norm( V, 2 ) )    # qui donne également 696.084

# norme maximum
print(maximum.norm(V))           # donne 625

# qui est un cas particulier de
print( entrywise.norm( V, Inf ) ) # qui donne également 625
```

Signalons qu'il existe de nombreuses autres normes de matrices, comme par exemple la norme spectrale :

```
spectral.norm(V)                 # donne 695
```

Attention : certaines normes sont notées de la même façon par différents auteurs ou dans différentes circonstances. Vérifiez toujours de quelle norme il est question avant d'implémenter un algorithme. D'autant que certaines normes sont très coûteuses en temps de calcul.

Matrices et vecteurs : diverses syntaxes utiles

Transformer une matrice en vector (au sens R) ou vecteur

Quand une fonction ne supporte pas le type "matrix" en entrée, il faut souvent transformer une matrice ou un vecteur en vector.

```
c <- as.vector(H)
print(c)  # on perd les dimensions et ce n'est plus une matrice
[1] 0 1 0 0 0 0 0 2 0 0 0 0 0 3 0 0 0 0 0 4 0 0 0 0 0
```

En mathématiques c'est l'application **vec()** qui transforme une matrice en vecteur colonne, triviale à coder en R en ne stipulant pas de dimension :

```
c <- matrix(H)
head(c)          # les cinq premiers éléments du vecteur colonne

[,1]
[1,]    0
[2,]    1
[3,]    0
[4,]    0
[5,]    0
[6,]    0
```

Transposition d'une matrice, d'un tableau, d'un data.frame

Nous avons utilisé plus haut la fonction **t()** pour effectuer une transposition, cette fonction agit aussi sur les data.frames. La fonction **aperm()** est beaucoup plus générale, elle transpose les matrices mais peut aussi intervertir les dimensions de tableaux à plus de deux dimensions.

```
# transposition de matrices, de tableaux, de data.frames
Tv <- aperm(V, c(2, 1)) # transposition de dimensions
```

```

Tv <- t(V)          # idem propre aux matrices ou data.frames
print(Tv)

```

Diagonale d'une matrice, identité, triangles

Considérons une matrice carrée, c'est-à-dire telle que le nombre de colonnes soit égal au nombre de lignes, les termes de la matrice tels que l'indice de ligne est égal à l'indice de colonne forment la diagonale de la matrice.

```

d <- diag(V)      # diagonale de la matrice
print(d)           # vector ligne au sens R
is.matrix(d)       # ce n'est pas une matrice !!!
[1] 1 2 9 64 625
[1] FALSE

```

Façonnons une matrice avec cette diagonale :

```

Dv <- diag(diag(V))  # matrice avec la diagonale de V
print(Dv)             # Enfin !!

[,1] [,2] [,3] [,4] [,5]
[1,] 1 0 0 0 0
[2,] 0 2 0 0 0
[3,] 0 0 9 0 0
[4,] 0 0 0 64 0
[5,] 0 0 0 0 625

```

Ce qui nous permet de remplacer la diagonale de V par des zéros.

```

V0 <- V-Dv          # remplacement de la diag de V par des 0
print(V0)

[,1] [,2] [,3] [,4] [,5]
[1,] 0 1 1 1 1
[2,] 1 0 4 8 16
[3,] 1 3 0 27 81
[4,] 1 4 16 0 256
[5,] 1 5 25 125 0

```

La matrice identité, qui est l'élément neutre de la multiplication de matrice ($AI = IA = A$) s'obtient facilement pour une dimension donnée :

```

# obtenir la matrice identite
I5 <- diag(1,5)      # matrice 5 x 5 avec des 1 sur la diagonale
print(I5)             # en dimension 5

[,1] [,2] [,3] [,4] [,5]
[1,] 1 0 0 0 0
[2,] 0 1 0 0 0
[3,] 0 0 1 0 0
[4,] 0 0 0 1 0
[5,] 0 0 0 0 1

```

On peut également extraire le triangle supérieur ou inférieur :

```
# triangles hauts et bas de la matrice

Vl <- lower.triangle(V) # triangle bas dont diagonale
print(Vl)
Vu <- upper.triangle(V) # triangle haut dont diagonale
print(Vu)
Vbis <- Vl + Vu - Dv # reconstruction de la matrice
```

Triangle bas, avec diagonale :

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1	0	0	0	0
[2,]	1	2	0	0	0
[3,]	1	3	9	0	0
[4,]	1	4	16	64	0
[5,]	1	5	25	125	625

On utilise également souvent la trace (somme des termes diagonaux) :

```
# calcul de la trace
matrix.trace(V) # somme des termes de la diagonale

[1] 701
```

Déterminants, valeurs propres, inversion de matrices

Quand le déterminant d'une matrice A n'est pas nul on peut en calculer son inverse A^{-1} tel que $A \cdot A^{-1} = A^{-1} \cdot A = I$

```
# déterminant d'une matrice  
print(det(V)) # non nul
```

[1] 288

Pour déterminer si une matrice est singulière, il existe une fonction très rapide :

```
# cette matrice est-elle singulière ?  
is.singular.matrix( H )      # oui  
is.singular.matrix( V )      # non
```

Calculons l'inverse d'une matrice.

```
# inverse d'une matrice non singulière  
Vi <- matrix.inverse(V)  
print(Vi)  
solve(Vi) # si on n'utilise pas le package matrixcalc  
           # on dispose quand même d'une fonction équivalente  
           # réputée moins fiable ici : solve
```

On peut vérifier que le produit de la matrice et de son inverse donne l'identité, mais le résultat est approxatif, donc on va arrondir et vérifier que la norme de la matrice à laquelle on soustrait l'identité est proche de zéro. **Cette méthode est à retenir**, car on ne peut pas toujours affirmer qu'une matrice est égale à une autre, mais on peut facilement tester qu'elles sont plus ou moins égales.

```
print(V %*% Vi) # proche de l'identité  
# la norme de l'écart avec matrice identité est-elle nulle ?  
print(round(frobenius.norm( V %*% Vi - I5), digits = 11)) # 0
```

	[,1]	[,2]	[,3]	[,4]	[,5]
[1,]	1.000000e+00	-1.387779e-16	0	5.273559e-16	-2.081668e-17
[2,]	2.331468e-15	1.000000e+00	0	3.108624e-15	1.443290e-15
[3,]	1.776357e-15	1.421085e-14	1	0.000000e+00	3.552714e-15
[4,]	5.329071e-15	2.131628e-14	0	1.000000e+00	5.329071e-15
[5,]	7.105427e-15	2.842171e-14	0	1.421085e-14	1.000000e+00

[1] 0

Par ce calcul, on a bien vérifié que les valeurs prises globalement tendent "collectivement vers 0" (remarque : dans le langage R, une valeur comme **1.42 e-15** signifie $1,42 \cdot 10^{-15}$, ce qui est infiniment petit, et même non interprétable du fait des limites d'encodage de votre ordinateur ; par ailleurs **round()** est la fonction d'arrondi de R).

Diagonalisation de matrices, valeurs propres, vecteurs propres

La diagonalisation de matrice possède de nombreuses application, en machine learning la principale application se nomme analyse en composantes principales (ou PCA en anglais). Regardons ici comment transformer en R une matrice en matrice diagonale en trouvant une nouvelle base (la base de vecteurs propres) associée aux valeurs propres de la matrice.

```
# valeurs propres et vecteurs propres d'une matrice
V_eigen<- eigen(V) # la fonction vecteurs propres
print(V_eigen) # résultat à décomposer
```

L'application de la procedure **eigen()** nous a produit un objet composite qu'il va falloir découper :

```
$values
[1] 680.9266583 17.6584861 1.9730603 0.4123561 0.0294392

$vectors
[,1]      [,2]      [,3]      [,4]      [,5]
[1,] -0.002133113 -0.08865214 0.66256386 -0.85603382 .../...
[2,] -0.026814798 -0.33857978 0.65963083 0.27225055
.../...
```

Extrayons tout d'abord la matrice diagonale triée par valeur propre :

```
V_valp <- diag(V_eigen$values) # matrice diagonale avec les
                                # valeurs propres classées

[,1]      [,2]      [,3]      [,4]      [,5]
[1,] 680.9267 0.000000 0.000000 0.0000000 0.0000000
[2,] 0.0000 17.65849 0.000000 0.0000000 0.0000000
.../...
```

Puis une matrice où chaque colonne est le vecteur propre correspondant à la valeur propre :

```
V_vectp <- V_eigen$vectors      # matrice des vecteurs propres
print(V_vectp)

[,1]      [,2]      [,3]      [,4]      [,5]
[1,] -0.002133113 -0.08865214 0.66256386 -0.85603382 .../...
[2,] -0.026814798 -0.33857978 0.65963083 0.27225055
```

Maintenant que nous disposons des outils mathématiques et statistiques de base, nous pouvons aborder notre problème d'estimation avec une certaine rigueur.

Estimations

Positionnement du problème d'estimation

Formulation générale du problème

Nous avons évoqué l'apprentissage supervisé au travers de l'expression suivante :

$$y = f(X) + \varepsilon.$$

Dans le cas courant, un grand nombre des éléments du vecteur y est inconnu, l'application $f()$ est à déterminer et le vecteur d'erreur est supposé avoir une moyenne à 0 et une norme petite par rapport à la norme de y .

Dans ces conditions on cherche à produire une estimation de f que nous noterons \hat{f} et une estimation de y que nous noterons \hat{y} tels que :

$$\hat{y} = \hat{f}(X)$$

Notre objectif est de produire une mécanique pour trouver $\hat{f}()$ telle qu'une "forme" d'écart entre y et \hat{y} soit minimum.

Pour exprimer l'écart global à minimiser, on se dote d'une application pour mesurer l'écart ou "perte" (*loss* en anglais). L'écart est calculé entre une valeur connue et une valeur estimée pour une observation donnée.

Nous noterons $\ell(\dots)$ cette application qui, dans le cas où les y_i sont des valeurs de \mathbb{R} , est une application de \mathbb{R}^2 vers \mathbb{R}^+ .

$\ell(y_i, \hat{y}_i)$ pourrait naturellement prendre la forme d'une distance induite par une norme ou d'une fonction monotone sur \mathbb{R}^+ d'une distance induite par une norme.

On peut donc par exemple chercher à trouver \hat{f} telle que la moyenne des $\ell(y_i, \hat{y}_i) = \text{distance}(y_i, \hat{y}_i)$ soit minimum, ou que la moyenne de leurs carrés le soit (notez que nous n'avons pas défini ici la distance choisie).

On va plutôt rechercher une fonction telle que l'espérance mathématique de $\ell(\cdot, \cdot)$ soit minimum.

Cette recherche se fera sur les observations de l'ensemble d'entraînement (training set : T).

On a coutume de nommer "risque" (R) l'espérance mathématique de $\ell(\cdot, \cdot)$.

Pratiquement, on n'accède facilement qu'au calcul **empirique** du risque dont l'espérance mathématique est la moyenne sur l'effectif de l'ensemble d'entraînement (au lieu d'être une intégrale), ce qui donne, en remplaçant \hat{y}_i par sa valeur et en notant la moyenne sur les observations du training set par \hat{y} :

$$R(X_T, y_T, \hat{f}) = \langle \ell(y_i, \hat{y}_i) \rangle_T$$

Ce qui peut se lire : "le risque R empirique est une fonction des observations (X_T, y_T) du "Training" set T et de l'estimation de f . R est égal à la moyenne sur T de $\ell(\cdot, \cdot)$."

La méthode à mettre en œuvre est de **trouver la fonction \hat{f} qui minimise R** sur les observations du training set T . La loi des grands nombres, sous certaines conditions qui sortent du cadre de cet ouvrage, garanti que cette estimation du risque R converge vers l'espérance mathématique de la fonction de perte qui représente le "vrai risque".

On résume un tel problème en exhibant la fonction **argmin()**, qui signifie tout simplement "trouver la valeur qui minimise cette expression". Comme R dépend en fait de l'ensemble de training T , et que l'on va faire varier f jusqu'à ce que l'**on considère avoir trouvé un minimum pour R , f sera alors la \hat{f} cherchée**.

On note cela de la façon synthétique suivante :

$$\hat{f} = \text{argmin}_f (R_T (f))$$

En synthèse : "Pour pouvoir être à même de comparer deux écarts, il convient souvent d'en prendre la moyenne, l'espérance mathématique. C'est ainsi que l'on a défini la fonction "risque". Cela est d'autant plus utile, que l'on est amené à entraîner le modèle sur un jeu d'essai avec un certain effectif pour trouver un modèle optimal \hat{f} , puis à le tester ou à l'utiliser de façon opérationnelle sur des effectifs différents".

Suivant les contextes, vous trouverez différents termes ayant des contours très similaires à nos fonctions de risque et de perte : *loss*, *cost*, *reward*, *objective function*... ne vous laissez pas impressionner !

Remarque importante : Souvent nous choisirons un aspect particulier pour \hat{f} , ce qui correspond au choix d'un "modèle". Nous pourrons par exemple essayer un modèle multilinéaire, polynomial... Dans ce cas, comme le modèle est fixé (provisoirement), notre problème se ramène à trouver **une estimation des paramètres** de \hat{f} . En effet, quand nous disposerons de ces paramètres, nous disposerons de \hat{f} .

Nous avons exprimé notre problème au travers **d'une application f, qui une fois appliquée à une matrice "complète" nous donne un vecteur "complet"**. C'est une vision très générale et vous remarquerez **qu'elle diffère de la vision qui à partir d'une ligne unique de la matrice nous donnerait une composante unique du vecteur**. En effet on peut concevoir des modèles qui utilisent plusieurs lignes de X (ou même toutes) pour produire une estimation d'une composante donnée du vecteur y.

Dans de nombreux cas, vous trouverez la mention d'une fonction "ligne à ligne" sous le nom de **fonction hypothèse**. Le fait que cette fonction s'applique ligne à ligne permet alors de nombreux calculs de dérivation partielle par rapport aux paramètres d'un modèle donné.

Application et reformulation du problème d'estimation

La lecture attentive de cette section **peut vous permettre de survivre** au flou introduit par les différences de formulation dans les ouvrages et les articles que vous allez parcourir dans votre vie de data scientist. En effet, le problème que vous rencontrerez ne se limite pas au fait de traduire la notation d'un auteur à un autre, ces notations vous donnent des indications sur les hypothèses et sur la nature des objets manipulés par chaque auteur et qui ne sont pas toujours celles que vous allez identifier de prime abord.

Introduction syntaxique de la fonction hypothèse

La traduction ligne à ligne de $\hat{y} = \hat{f}(X)$ nous oblige à introduire une fonction \hat{f}_i pour chacune des lignes (enlevez mentalement l'indice i de f pour comprendre que cet indice est important car sinon chaque ligne aurait la même expression !) :

$$\hat{y}_i = \hat{f}_i(X)$$

Dans le cas simple où \hat{y}_i ne dépend que de la ligne correspondante de X on peut écrire :

$$\hat{y}_i = h(x_{i,:})$$

où $x_{i,:}$ signifie i ème ligne de X , qui est souvent noté plus simplement x_i si l'on sait que l'on est en train de parler des lignes de X et pas des colonnes de X .

Dans ce cas

$$\hat{f}_1 = \hat{f}_2 = \hat{f}_3 = \dots = h$$

avec h étant la fonction hypothèse.

Le tableau de nos observations se trouve être la matrice $Z = (X, y)$, chaque ligne z_i est le $(p+1).uplet(x_i, y_i)$. On peut donc redéfinir la fonction de perte comme dépendant de h et s'appliquant à z_i :

$$z_i : \ell_h(z_i) = \ell(y_i, h(x_i))$$

Dans le même ordre d'idée, notons $R(h)$ la fonction "vraie" de risque de h , et $\widehat{R}(h)$ la fonction empirique sur l'ensemble d'entraînement.

Théorème "No free lunch"

Les problèmes d'optimisation qui consistent à trouver un argmin, ou un argmax au travers d'un algorithme (c'est-à-dire pas de façon purement analytique) se déclinent en deux types : les optimisations de fonction (ex : trouver le minimum d'une fonction convexe dans un espace à n dimensions par la méthode de descente de gradient) et les optimisations stochastiques qui consistent à rechercher un optimum de l'espérance mathématique d'un phénomène (ex : recherche du minimum de notre vraie fonction de risque).

Dans un cas comme dans l'autre il est impossible d'affirmer une réelle corrélation entre l'efficience de l'optimisation produite (effectuée sur un ensemble d'entraînement dans le cas stochastique : training set) et la généralisation que l'on pourrait faire de son efficacité (faible erreur) dans le cas général. Si bien que tout modèle qui fonctionne bien et qui optimise une fonction hypothèse sur un ensemble d'entraînement est en fait relié à une série d'hypothèses implicites effectuées sur la nature du phénomène étudié.

C'est une mauvaise nouvelle pour les technocrates ou les "pousse-bouton" qui imaginent que l'on peut traiter tous les problèmes avec la même artillerie, c'est une bonne nouvelle pour les passionnés que nous sommes : chaque problème stochastique nécessite du travail et de la réflexion sur la fonction de risque utilisée et sur le modèle à mettre en œuvre. Cela s'oppose à une conception commune du machine learning qui consisterait à ne travailler que sur la recherche et le conditionnement de features et à confier la prédiction ou la classification à un ensemble d'algorithmes figés et déterminés par d'autres au fur et à mesure de l'évolution de la technologie.

No free lunch = on n'a rien sans rien.

Régression linéaire

Le cas d'un modèle de régression linéaire simple

Prenons le cas d'un modèle linéaire simple. La matrice X se réduit à une colonne. La forme de $h()$ est la suivante : $h(x) = ax + b$.

On est dans le cas où l'on a qu'une "feature" en entrée et que l'on imagine trouver une droite de régression simple qui approxime le nuage de nos points sur le plan.

Les paramètres de $h()$ sont donc a et b et l'effort d'estimation portera donc sur ces paramètres a et b .

La fonction \hat{f} quant à elle a **exactement le même aspect visuel que h** (et d'ailleurs on leur donne souvent le même nom par abus de langage), à savoir que :

$$\hat{f}(\vec{X}) = \vec{\hat{y}} = \vec{a} \vec{X} + \vec{b}$$

\vec{b} est un vecteur colonne dont toutes les composantes valent b .

Quand ils pensent qu'il n'y pas d'ambiguïté, les auteurs ne vous signalent pas que \vec{X} , $\vec{\hat{y}}$ et \vec{b} sont des vecteurs et omettent les flèches. D'autres, de culture anglo-saxonne, utilisent différentes polices de caractère, ou mettent en gras les vecteurs...

Ceux qui aiment le calcul matriciel et dont l'auteur de cet ouvrage fait partie introduisent souvent un vecteur unité (un 1 en gras) : **1**. Ce vecteur étant une colonne de "1". Ce qui donne quelque chose comme cela :

$$\hat{f}(X) = \hat{y} = \hat{a} \cdot X + \hat{b} \cdot \mathbf{1}$$

ou :

$$\hat{f}(X) = \hat{y} = \hat{a}X + \hat{b}\mathbf{1}$$

Parfois, pour éviter toute ambiguïté on vous rappelle que X et $\mathbf{1}$ ont les formes suivantes, suivant les notations utilisées plus haut dans cet ouvrage ; les crochets étant parfois des parenthèses :

$$X = [x_i]_{1 \leq i \leq n}$$

$$\hat{y} = [\hat{y}_i]_{1 \leq i \leq n}$$

$$\mathbf{1} = [1]_{1 \leq i \leq n}$$

Mais vous trouverez d'autres notations, possédant leurs avantages et leurs inconvénients... Le karma du data scientist est ainsi fait qu'il ne se laisse pas troubler par ces détails visuels mais qu'il vérifie toujours soigneusement le sens des notations qui lui sont proposées, comme il est rappelé à plusieurs reprises dans cet ouvrage. Remarquez par ailleurs que suivant le problème, on utilise parfois des notations différentes pour des raisons historiques ou par le fait d'une habitude répandue qui aide chacun à se repérer dans le contexte précis du sujet abordé.

Application au cas d'un modèle de régression linéaire multiple

Dans le modèle de régression linéaire multiple, il n'y a pas qu'une seule colonne de features mais plusieurs : X n'est donc plus un vecteur mais une matrice.

$$\hat{f}(X) = \hat{y} = X \cdot \hat{a} + \hat{b}\mathbf{1}$$

$$X = [x_{i,j}]_{\substack{1 \leq i \leq n \\ 1 \leq j \leq p}}$$

$$\hat{y} = [\hat{y}_i]_{1 \leq i \leq n}$$

$$\mathbf{1} = [1]_{1 \leq i \leq n}$$

$$\hat{a} = [\hat{a}_i]_{1 \leq i \leq p}$$

Attention, **remarquez que la dimension du vecteur \hat{a}** est le nombre de features, soit p .

Au niveau de chaque ligne on a la fonction hypothèse : $h(x_i) = \sum_{j=1}^p x_{i,j} \hat{a}_j + \hat{b}$

Si l'on renomme \hat{b} en \hat{a}_{p+1} , cela donne :

$$h(x_i) = \sum_{j=1}^p x_{i,j} \hat{a}_j + \hat{a}_{p+1}$$

Ce qui peut s'écrire

$$h(x_i) = \sum_{j=1}^{p+1} x_{i,j} \hat{a}_j$$

en définissant \hat{X} comme une matrice construite d'une colonne de 1 ajoutée à X :

$$\hat{X} = [x_1, x_2, \dots, x_j, \dots, x_p, \mathbf{1}]$$

avec :

$$x_j = \begin{pmatrix} x_{1,j} \\ \vdots \\ x_{n,j} \end{pmatrix}$$

Comme pour toute matrice, notre nouvelle matrice peut s'exprimer de façon générique en fonction de ses membres :

$$\hat{X} = [\hat{x}_{i,j}]_{\substack{1 \leq i \leq n \\ 1 \leq j \leq p+1}}$$

Avec cette nouvelle matrice augmentée d'une colonne on obtient maintenant une formulation plus compacte, mais qui peut introduire une confusion si vous confondez X et \hat{X} .

$$\hat{f}(X) = \hat{y} = \hat{X} \cdot \hat{a}$$

avec :

$$\hat{a} = [\hat{a}_i]_{1 \leq i \leq p+1}$$

D'ailleurs dans la pratique les auteurs notent souvent $y = X.a + \varepsilon$ sans mettre en évidence que X n'est pas la matrice des variables explicatives.

Régression linéaire multiple et méthode des moindres carrés

Nous allons maintenant étudier rapidement la résolution de notre modèle de régression linéaire multiple. La méthode de résolution employée ici se nomme "méthode des moindres carrés" (remarquez que la régression linéaire simple par la méthode des moindres carrés s'en déduit de façon triviale).

À retenir absolument : en règle générale, trouver l'estimation de nos fonctions f conformes à un modèle donné, c'est tout simplement rechercher les estimations des paramètres de cette fonction tels que définis par le modèle.

On cherche : $\hat{f} = \operatorname{argmin}_f (R_T(f))$

Mais comme on a choisi le modèle de f , **cela revient à chercher à estimer ses paramètres**, donc notre problème se ramène à chercher :

$$\hat{a} = \operatorname{argmin}_a (R_T(a))$$

avec :

$$f(X) = y = X \cdot a + \varepsilon$$

et :

$$\hat{f}(X) = \hat{y} = X \cdot \hat{a}$$

Où on a reporté l'erreur d'estimation de a comme estimation de l'erreur sur y .

Sachant que la méthode des moindres carrés consiste d'abord à poser :

$$\ell(y_i, \hat{y}_i) = |y_i - \hat{y}_i|^2 = (y_i - \hat{y}_i)^2$$

On obtient l'expression à minimiser suivante :

$$\frac{1}{n} \|y - \hat{y}\|_2^2$$

Une petite démonstration hors du propos de notre ouvrage donne une solution analytique, sachant que l'on peut également chercher cette solution par un algorithme dédié à la recherche d'optimum, comme la méthode de descente de gradient :

$$\hat{a} = (\hat{X}^T \hat{X})^{-1} \hat{X}^T y$$

Cette solution est élaborée sur l'ensemble d'entraînement, ce qui explique que nous connaissons y !

Dans le cas où les données sont centrées (c'est-à-dire où on leur a soustrait leur moyenne) ou réduites (données centrées divisées par leur écart-type), le produit de la transposée de la matrice des observations par elle-même est respectivement la matrice de covariance ou la matrice de corrélation entre les features.

En règle générale, les algorithmes opérationnels n'implémentent pas directement les solutions analytiques quand celles-ci comportent des opérations aussi coûteuses ou dangereuses qu'une inversion de matrice, mais cela ne vous empêche pas d'utiliser directement une solution analytique dans le cas d'un prototypage pour mieux "sentir" et comprendre les conditions et les impacts de vos choix de solution. En fait, avant de manipuler une matrice pour ce type de calcul, il est souvent utile de faire différentes vérifications de ce que l'on appelle le bon **conditionnement de la matrice** car certains calculs peuvent prendre des valeurs extrêmes qui déforment la logique des calculs suivants.

Pour savoir si votre **problème est bien conditionné**, vous pouvez injecter de petites **perturbations aléatoires** négatives et positives, de valeur supérieure à la précision de votre ordinateur, sur vos données en entrée et vérifier que vos résultats ne changent pas de façon significative.

À retenir : l'expression à minimiser ne possédait qu'**un seul minimum** quand on a fait varier toutes les composantes de \vec{a} , c'était une **fonction convexe** de \vec{a} . Avec une autre fonction de risque, basée sur une autre distance on n'a pas toujours cette "chance" et la recherche du minimum via une technique d'optimisation numérique s'impose alors.

Par ailleurs cette méthode suppose certaines hypothèses, trois d'entre elles vous sembleront peut-être évidentes :

- Il faut un nombre d'observations (nettement) plus grand que le nombre de variables.
- Deux variables ne doivent pas être colinéaires.
- L'écart-type de l'erreur obtenue ne doit pas dépendre des valeurs des observations : hypothèse d'**Homoscédasticité** que vous pourrez tester en utilisant le **test de Breusch-Pagan**.

Un théorème vient à notre secours pour limiter la difficulté de vérifier ces hypothèses. Son application nous rappelle d'ailleurs qu'il **ne faut pas chercher à se rendre conforme à des hypothèses inutiles** : le théorème Gauss–Markov qui s'applique aux modèles linéaires.

Ce théorème nous apprend que si nos erreurs ont une espérance mathématique nulle (moyenne = 0), si l'hypothèse d'**Homoscédasticité est respectée** et si les erreurs sont non corrélées, la meilleure fonction de perte est bien celle qui correspond aux moindres carrés. Sous ces deux conditions, **il n'est pas utile de supposer et/ou de vérifier que les erreurs se répartissent sous une loi normale ou qu'elles soient indépendantes.**

Par ailleurs, sachez que cette méthode est très résistante au fait que certaines hypothèses ne soient pas tout à fait tenues.

En anglais on parle de "best linear unbiased estimator (BLUE)", ce qui exprime que le meilleur estimateur non biaisé pour un modèle linéaire est obtenu via la méthode des moindres carrés.

Matrice Hat

Nous avions l'expression :

$$\hat{y} = \hat{X} \hat{\alpha}$$

Dans le cas de la régression vue plus haut on a :

$$\hat{\alpha} = (\hat{X}^T \hat{X})^{-1} \hat{X}^T y$$

En remplaçant cette valeur dans notre première expression on obtient :

$$\hat{y} = \hat{X} (\hat{X}^T \hat{X})^{-1} \hat{X}^T y$$

Si l'on pose maintenant :

$$H = \hat{X} (\hat{X}^T \hat{X})^{-1} \hat{X}^T$$

On obtient :

$$\hat{y} = H y$$

C'est cette matrice H que l'on nomme la matrice Hat, car *hat* signifie chapeau en anglais, et que c'est cette matrice qui ajoute le chapeau sur y (amusant, n'est-ce pas ?).

La matrice Hat est une matrice de projection qui projette y sur l'hyperplan du modèle linéaire. Une telle matrice Hat est aussi calculée dans le cas de la généralisation des modèles linéaires dite "GLM" (mais elle a une expression plus complexe). Vous la retrouverez donc dans d'autres contextes. Si vous disposez de la matrice Hat, vous pouvez exprimer le vecteur des erreurs de prédiction de façon triviale sans passer par la fonction \hat{f} , ce qui fait gagner un temps de calcul parfois non négligeable :

$$e = y - Hy = (I-H)y$$

Certains auteurs expriment la magie de cette matrice Hat, par exemple :

- $(I-H)$ est symétrique par rapport à sa diagonale (elle est égale à sa transposée).
- $(I-H)$ est idempotente (elle est égale à son carré).

À retenir : c'est surtout le point suivant qui nous semble le plus intéressant : quand un des éléments de la diagonale est nettement supérieur à la moyenne des éléments de sa diagonale (on parle de deux fois supérieur) cela indique que la ligne d'observation correspondante est un **point de levier**, autrement dit que cette observation a un fort impact sur le modèle. Il faut donc identifier ces points et vérifier soigneusement la pertinence de ces échantillons. On peut également tenter de comparer le modèle avec ou sans ces points, le pire cas étant d'avoir créé un modèle surdéterminé (overfitté) à partir d'un de ces points de levier et que ce point relève de plus d'une erreur de collecte d'information !

Distance de Cook

Cette dernière remarque est généralisable à tous les types de modèles, au sens où vous devez toujours avoir la préoccupation **d'identifier les observations qui ont un effet de levier sur votre modèle et qui risquent de le surdéterminer.**

La question est de regarder comment évolue notre vecteur de prédictions \hat{y} quand on enlève une observation de l'ensemble d'entraînement. Supposons que l'observation enlevée soit l'observation k, nous noterons ici le vecteur $\hat{y}_{(-k)}$ résultant :

$$\hat{y}_{(-k)}$$

L'écart entre ces deux vecteurs sera le vecteur :

$$\delta_{(-k)} = \hat{y} - \hat{y}_{(-k)}$$

Et comme d'habitude on peut évaluer sa taille en utilisant une norme, ici la norme euclidienne au carré que l'on obtient toujours en multipliant la transposé d'un vecteur par lui-même :

$$\delta_{(-k)}^T \cdot \delta_{(-k)}$$

Pour pouvoir comparer cette quantité pour différents cas de figure, il faut comme toujours trouver une pondération qui normalise la quantité que l'on veut utiliser comme évaluateur. Cook propose la pondération suivante, qui est exprimée sous la forme d'un ratio utilisant la taille de la matrice des valeurs prédictives, notre matrice X. Les diviseurs de ce ratio sont le nombre de features (colonnes de la matrice) et la variance de l'erreur estimée. On normalise donc bien cette quantité en fonction du niveau global d'erreur de notre estimation.

$$\frac{\mathbf{X}^T \mathbf{X}}{p \sigma_e^2}$$

La pondération proposée est donc : $\frac{\mathbf{X}^T \mathbf{X}}{p \sigma_e^2}$.

La quantité que nous allons obtenir et qui nous permettra d'évaluer et de comparer ce qui se passe quand on enlève telle ou telle observation de l'ensemble de données d'entraînement se nomme "distance de Cook" et a pour expression :

$$D_{(-k)} = \frac{\delta_{(-k)}^T \mathbf{X}^T \mathbf{X} \delta_{(-k)}}{p \sigma_e^2}$$

À ce stade la formulation de **cette distance n'est pas spécifiquement liée aux modèles linéaires**, vous pouvez donc toujours calculer une telle distance (mais à vos risques et périls en termes de temps de calcul...).

Dans le cas linéaire, cette distance s'exprime très simplement en fonction de la matrice Hat et les calculs sont alors très rapides.

Globalement, retenez que plus la distance de Cook est élevée (typiquement aux alentours de 1) plus le point pourrait être un point de levier ou un point aberrant, ou les deux.

La liste des indices des observations qui ont entraîné votre modèle linéaire nommé **f_1m**, tels que la distance de Cook soit supérieure à 0.9, s'obtient de la façon suivante :

```
which(cooks.distance(f_1m) > 0.9)
```

La recherche d'une bonne estimation de f s'effectue en cherchant à trouver une configuration qui minimise une fonction de risque, souvent exprimée comme un indicateur d'écart.

Cette recherche est faite sur un ensemble d'entraînement ou au travers de méthodes que nous verrons plus loin qui scindent l'ensemble d'entraînement et "croisent" ou "compilent" leurs résultats. En final, lors du test du modèle, d'autres indicateurs d'écart sont souvent mis en œuvre, qui ne sont pas ceux qui ont servi à mettre au point le modèle. D'autant plus que souvent on les utilise pour comparer l'efficacité de plusieurs modèles !

Nous allons donc maintenant explorer les indicateurs d'écart les plus simples couramment mis à notre disposition.

Les indicateurs d'écart utilisés en machine learning

MSE, RMSE, SSE, SST

RMSE, l'indicateur "roi"

Un premier usage courant, qui tient compte de nos dernières remarques est d'utiliser l'erreur quadratique moyenne, MSE (*Mean Squared Error*) en anglais. On peut constater que cela correspond au choix de la moyenne de norme euclidienne au carré comme fonction de risque :

$$\text{MSE}(y, \hat{y}) = \mathbb{E}(\ell(\cdot, \cdot)) = \frac{1}{n} \|y - \hat{y}\|_2^2$$

avec $\ell(y_i, \hat{y}_i) = |y_i - \hat{y}_i|^2 = (y_i - \hat{y}_i)^2$

En exprimant la MSE au travers d'une norme vectorielle, on s'ouvre à l'idée d'utiliser d'autres normes pour concevoir des fonctions de risque qui ne seraient d'ailleurs donc pas forcément définies comme l'espérance mathématique d'une fonction de perte appliquée à chaque écart.

On appelle souvent SSE (*Sum of Square Errors*) la quantité :

$$\|y - \hat{y}\|_2^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Avec cette notation on a donc :

$$\text{MSE}(y, \hat{y}) = \text{SSE}(y, \hat{y}) / n$$

Il ne faut absolument pas la confondre avec la SST (*Total Sum of Squares*) qui ne fait pas intervenir l'estimation de y , à savoir \hat{y} mais sa moyenne \bar{y} . Cette quantité ne dépend que de y :

$$\text{SST}(y) = \|y - \bar{y}\|_2^2 = \sum_{i=1}^n (y_i - \bar{y})^2$$

La MSE n'est pas homogène à y (c'est-à-dire que par exemple si y est exprimé en mètres, une MSE correspondante serait exprimée en m^2). On utilise donc souvent sa racine pour mieux l'interpréter, on nomme cette quantité **root-mean-square deviation (RMSD)** ou **root-mean-square error (RMSE)**. La première dénomination rappelle que l'on a de fait affaire à un écart-type, la deuxième rappelant la façon dont on a construit cet indicateur d'écart. On a donc :

$$\text{RMSE}(y, \hat{y}) = \sqrt{\text{MSE}(y, \hat{y})}$$

Globalement on peut penser que plus les MSE ou RMSE sont faibles, meilleur est le modèle (c'est plus ou moins le cas, mais cela nécessiterait une discussion bien plus approfondie).

À quoi servent les indicateurs d'écart ?

Les usages typiques des MSE, RMSE et de la plupart des indicateurs que nous verrons sont similaires dans leurs objectifs :

- Premier usage possible : l'algorithme utilisé pour construire le modèle est conçu pour améliorer l'indicateur en question, soit par un calcul direct soit par itérations successives. On parle souvent de méthode de descente de gradient/gradiant dans le cas où on itère en faisant varier les paramètres à optimiser dans une direction qui avait donné une amélioration dans le cycle précédent. Par exemple la méthode des "moindres carrés" consiste justement à calculer une solution qui minimise la MSE !
- Deuxième usage possible : on compare les valeurs obtenues par un indicateur (ou plusieurs) pour déterminer le meilleur modèle pour un jeu de données d'entraînement, ou le cas échéant pour un contexte donné si l'on est en recherche d'une généralisation ou de l'élaboration d'une méthode.
- Troisième usage possible, proche du précédent et plus rare : on cherche quel modèle appliquer sur différentes parties du jeu d'essai avant de composer une prédiction qui sera la composition de ces modèles (ce type de méthode est une des méthodes génériquement classées sous le vocable méthodes "Ensemble").
- Quatrième usage : au regard des indicateurs on déclare que notre estimation est bonne, ou utilisable, ou assez aboutie (ou pas !) pour conclure sur la qualité ou l'employabilité de notre modèle dans un contexte donné. Cet objectif, très opérationnel, consistant à pouvoir affirmer que l'on a obtenu un "bon modèle" est en grande partie un leurre pour tout problème issu du monde réel. Mais c'est ce par quoi les data scientists opérationnels sont jugés. En fait, vos indicateurs vous aideront à façonner une conviction que vous pourrez partager au travers d'explications très didactiques, mais restez extrêmement vigilants "le plus beau jeu de données ne peut donner que ce qu'il a"...

MAE, ME

Un autre usage courant, très utilisé dans le cas des séries temporelles, est d'utiliser la moyenne de la valeur absolue des erreurs, MAE (*Mean Absolute Error*) en anglais :

$$\text{MAE}(y, \hat{y}) = \frac{1}{n} \| y - \hat{y} \|_1$$

Pour faire la moyenne on divise par l'effectif n , dans certain cas on choisit en fait de diviser par $n-1$ ou $n-2$. Nous aborderons succinctement cela dans le chapitre Feature Engineering quand nous décrirons la PCA. Notez que comme nous nous plaçons dans le cas du big

data, les effectifs sont très grands et que diviser par n ou $n-1$ ou $n-2$ ne change pas le résultat d'une façon notable. La fonction `sd()` de R utilise $n-1$ pour vous calculer l'écart-type (standard deviation).

Point d'attention : vous entendrez parfois parler de biais, ou bias (B), ou Mean Error (ME) en anglais, qui représente une moyenne de l'écart mais en conservant les signes, si bien qu'un écart négatif peut être compensé par un écart positif et vice versa. Cette notion de biais est intéressante, elle permet de percevoir un décalage systématique du modèle dans une direction ou une autre. Mais il est dangereux de vouloir optimiser un modèle sous la contrainte de diminution de ce biais ! En effet imaginez que vous ayez des éléments estimés avec de grosses erreurs positives (ou négatives), vous pourriez générer lors de votre estimation de grosses erreurs négatives (ou positives) pour diminuer le biais !

À l'inverse, travailler à rechercher un modèle dont l'optimisation serait effectuée sous la contrainte de diminuer la MAE (ou un autre indicateur d'écart moyennant des valeurs toutes de même signe) n'a pas cet inconvénient, la MAE étant en fait une moyenne de valeurs absolues :

$$MAE(y, \hat{y}) = \langle |y - \hat{y}| \rangle$$

$$ME(y, \hat{y}) = \langle y - \hat{y} \rangle$$

avec une des notations proposées plus haut sur l'espérance mathématique.

Paradoxalement, ces indicateurs qui semblent très simples posent un petit problème mathématique : il n'est pas agréable de les dériver car ils contiennent une "valeur absolue".

Dériver un indicateur permet d'étudier comment il évolue quand une variable ou un paramètre du modèle évolue. C'est un élément très utile à l'amélioration de la compréhension de ce que l'on est en train de manipuler. Imaginez par exemple qu'un indicateur d'erreur en pourcentage évolue de façon manifeste en fonction d'une mesure quelconque effectuée sur une ou plusieurs de vos variables, vous pourriez peut-être en déduire une nouvelle stratégie quant au(x) modèle(s) que vous allez utiliser.

Dans le même esprit, on aime que la dérivée d'une régression par rapport aux valeurs prédictes d'une mesure d'erreur soit proche de zéro en tout point (nous verrons cela dans la partie "pratique" de ce chapitre), c'est-à-dire qu'elle se rapproche d'**une droite horizontale**.

Pour appréhender les caractéristiques un peu plus intrinsèques d'une quantité, sans apprécier sa dépendance à une variable en utilisant la notion de dérivée, on peut utiliser une mécanique de ratio et/ou de coefficient de variation par rapport à une variable ou un paramètre : c'est ce type d'indicateur appliqué à la RMSE, que nous allons aborder maintenant.

NRMSE/NRMSD, CV_RMSE

Quand on veut comparer le calcul d'indicateurs d'écart sur des prédictions correspondant à des natures différentes (des \mathbf{y} de natures différentes, poids, âge...) afin d'évaluer par exemple l'efficacité d'un algorithme dans différents cas, ou tout simplement pour interpréter rapidement la valeur d'une RMSE ou d'une MAE, on tente de supprimer l'effet d'échelle en normalisant la valeur. La question est alors de choisir une référence : ce peut être l'écart maximum entre deux valeurs de la variable y , son écart-type, sa moyenne, que l'on exprimera le cas échéant en pourcentage... Dans cet esprit on a :

$$\text{NRMSE}(y, \hat{y}) = \text{NRMSD}(y, \hat{y}) = \frac{\text{RMSE}(y, \hat{y})}{y_{\max} - y_{\min}}$$

Le "N" de NRMSE signifie "Normalized".

On a vu plus haut la notion de coefficient de variation que nous avons exprimée sous la forme suivante :

$$c_v = \frac{\sigma}{\mu}$$

Dans cette formule le σ représentait l'écart-type de notre variable ; ici la variable dont on va mesurer l'écart-type est l'erreur $e = y - \hat{y}$. Dans cette expression μ représentait la moyenne de cette variable soit ici la moyenne ou l'espérance mathématique de e . Or, nous souhaitions que cette espérance mathématique soit nulle. Ce coefficient de variation là serait donc incohérent. Dans le cas de la RMSE il est défini un autre coefficient de variation qui n'est plus normé sur la moyenne de ce sur quoi on a calculé l'écart-type (à savoir e), mais sur la moyenne de la variable y elle-même, que nous noterons ici \bar{y} .

On a donc :

$$\text{CV_RMSE}(y, \hat{y}) = \frac{\text{RMSE}(y, \hat{y})}{\bar{y}}$$

Le "CV" signifie "coefficient de variation", mais a un sens propre à cette expression (c'est-à-dire non systématiquement généralisable à d'autres indicateurs, mais généralisable à d'autres indicateurs d'écart d'erreur).

On a vu précédemment (dans la partie initiation aux statistiques et aux probabilités) que l'on pouvait étudier différents "moments" d'une distribution. Quand on a une idée de la tendance centrale (moyenne, médiane...) on est amené à étudier la dispersion. C'est ce que nous allons évoquer avec l'indicateur suivant.

SDR

On nomme parfois les écarts (erreurs) sous le vocable **résidus**. Pour **percevoir leur dispersion, on est amené à calculer l'écart type de ces résidus** et on le nomme traditionnellement "Standard Deviation of Residuals" soit SDR en anglais.

$$\sigma_e = \text{SDR}(y, \hat{y})$$

On démontre facilement l'assertion suivante : $MSE = RMSE^2 = ME^2 + SDR^2$

En clair : **variance de l'erreur = biais au carré + variance des résidus.**

On peut agir sur le biais, qui est réductible, mais on n'agit pas formellement sur la variance des résidus que l'on peut ou doit considérer comme irréductible si on ne veut pas capturer le bruit inhérent à nos données au travers du modèle que l'on cherche à concevoir.

Parmi les quatre usages que nous avons évoqués précédemment, le dernier usage, à savoir déterminer la qualité de notre modèle, s'avérait le plus polémique. Les indicateurs suivants, qui peuvent évidemment être utilisés avec succès à d'autres usages, sont couramment utilisés par tout à chacun pour établir une perception, une conviction eu égard à la qualité de leur modèle.

Accuracy, R2

Les indicateurs suivants sont également très courants et donnent une perception rapide du niveau de la "qualité" de l'estimation.

Accuracy

Dans le cas où l'on cherche à prédire l'appartenance à une classe, typiquement quand **y** peut valoir **0** ou **1**, on utilise tout naturellement le pourcentage de prédictions réussies par rapport au nombre total de prédictions. Cet indicateur se nomme *Accuracy* en anglais. Cet indicateur est facilement utilisable si vous cherchez à prédire l'appartenance à une classe. Si vous cherchez à prédire une valeur réelle et que vous êtes à même de définir une notion de seuil (threshold) sur les valeurs à prédire, vous pouvez étendre cette notion pour déterminer des indicateurs similaires.

La recherche d'une meilleure *accuracy* peut être nuancée dans certains cas. En effet on préfère parfois optimiser son modèle sur la recherche d'autres bons taux de prédiction. Par exemple, un médecin préfère prendre le risque d'une mécanique de dépistage statistique qui lui trouve parfois des faux positifs, à savoir des personnes identifiées comme malades alors qu'elles s'avéreront ne pas l'être après un examen plus approfondi, alors qu'un gestionnaire de la sécurité sociale préférera peut-être prendre le risque d'utiliser un modèle fabriquant des faux négatifs au risque d'avoir des faux positifs car il désire limiter le nombre d'examens coûteux et inutiles... Vous trouverez une abondante littérature sur ces sujets et des sujets très connexes sous les termes "matrice de confusion", ROC et AUC que nous avons esquissés dans le chapitre Introduction.

R2, coefficient de détermination

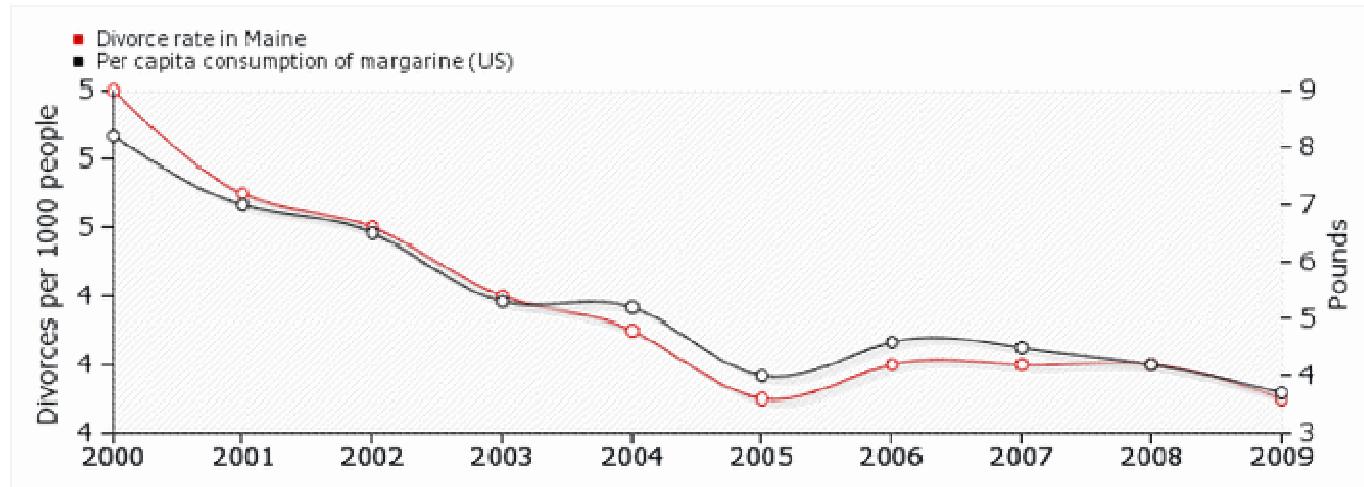
Pour évaluer si le modèle représente bien la variation des données on peut utiliser :

$$R^2 = 1 - \frac{SSE}{SST}$$

R2 se nomme coefficient de détermination, on dit "R carré", *R squared*, car dans le cas des régressions linéaires il est égal au carré du coefficient de corrélation. Normalement ses valeurs sont comprises entre 0 et 1, mais comme son expression comporte une soustraction, qu'il est employé dans différents contextes et qu'il existe plusieurs définitions (formules) pour R2 vous aurez parfois affaire à des R2 qui ne seront pas dans cet intervalle, ce au gré de vos lectures ou lors de l'utilisation de différents packages.

Pratiquement, en ce qui concerne notre problème d'estimation, plus la valeur de R2 est élevée plus on peut considérer que la correspondance entre l'application du modèle sur les données de test versus et les données réelles est forte.

L'utilisation de R2 pour affirmer ceci ou cela sur la véracité d'une relation entre les **x** et **y** est une pratique courante en politique et en consulting, souvent pernicieuse, ou tout au moins maladroite. Au moment où nous écrivons ces lignes, un site illustre avec humour ce type de conclusions aberrantes. Je vous invite à le parcourir pour faire vaciller vos certitudes : <http://tylervigen.com/spurious-correlations>. Vous y verrez par exemple la corrélation suivante (R2 de l'ordre de 0.98), qui pourrait laisser penser que le taux de divorces dans le Maine peut se laisser prédire en connaissant le taux de consommation de margarine aux US et/ou vice versa :



Étrange similitude

Remarquons quand même que ces séries sont des séries temporelles, qu'elles comportent peu de points, et que l'on ne parle que d'une seule variable explicative, ce qui favorise les "hasards" générant de forts R2.

Il faut avoir conscience que le **fait d'ajouter des variables a tendance à augmenter les valeurs de R2** pour estimer un **y** donné (ce qui peut sembler contradictoire avec la remarque précédente, mais à tort car on y évoquait la probabilité de trouver deux courbes similaires en termes de probabilité d'avoir un grand R2 sans avoir fixé la cible **y** à prédire).

Pour limiter ce phénomène on utilise souvent un R2 dit "ajusté" : "adjusted R2". La formule de ce R2 tient compte des degrés de liberté via l'application d'un ratio $(n-p-1)/(n-1)$. Dans cette expression p est le nombre de variables et n la taille de l'échantillon. Comme stipulé à d'autres endroits de cet ouvrage, quand les échantillons sont importants, ce ratio tend vers 1 et ce type de correction perd beaucoup d'intérêt.

Bien que cela ne transparaisse pas de prime abord dans sa formulation R2 est très sensible à la variance des X. Personne ne peut affirmer qu'une valeur de R2 est effectivement "bonne" avec certitude (ni même mauvaise en fait), sauf dans le cas du test de la multi colinéarité de différentes variables.

Conclusion intermédiaire

Au sujet des indicateurs, utilisés dans un contexte opérationnel, l'idée est plutôt la suivante : **mes efforts n'améliorant plus notablement mes indicateurs, j'ai maintenant le choix entre considérer mon travail achevé ou changer drastiquement les paradigmes de mon étude** (sans doute au travers d'une remise en question importante et/ou d'un coût ou d'un délai dont il faut évaluer la pertinence). Avec comme corollaire : **la difficulté liée à l'amélioration de mes indicateurs peut signifier que j'ai extrait toute l'information disponible dans mes données et que d'autres étapes m'amèneraient en fait à surdéterminer mon modèle et à y intégrer du bruit (danger d'overfitting)**.

Nous allons maintenant effectuer deux prédictions sur les mêmes données en utilisant des modèles différents. Puis nous comparerons leurs R² respectifs.

Mise en pratique : apprentissage supervisé

Nous allons effectuer un petit cycle de construction d'une prédiction afin de mettre en pratique les notions vues précédemment. Tout d'abord nous allons nous doter d'un ensemble d'observations. Cet ensemble comportera 10000 observations, deux features en entrée et une à prédire. On cherchera le modèle f , la feature cible étant y , l'entrée étant $X(x_1, x_2)$. L'ensemble des observations $Z=(X, y)$ est scindé en un ensemble d'entraînement Z_e de 6000 lignes et un ensemble de validation Z_v de 4000 lignes.

Nous allons entraîner le modèle sur l'ensemble d'entraînement Z_e , en cherchant à estimer le modèle f , ce qui nous donnera une fonction "estimation de f " telle que :

$$\hat{y}_e = \hat{f}(X_e)$$

Puis nous appliquerons \hat{f} sur l'ensemble de validation X_v et nous obtiendrons un vecteur de prédictions \hat{y}_v que nous comparerons avec le vecteur y_v des valeurs réelles issues de l'ensemble de validation.

Pour ce faire nous allons créer un ensemble Z "factice" où nous avons imposé, puis perturbé aléatoirement le fait que y soit plus ou moins situé sur le plan :

$$2 \cdot x_1 - 7 \cdot x_2 + 1.$$

Notre espoir est de constater que les algorithmes utilisés retrouvent l'équation de ce plan avec une erreur de faible amplitude.

Préparation

Le code de création de l'ensemble factice d'observation est le suivant (sans intérêt spécifique) :

```
# un dataset factice, création des matrices #  
rm(list=ls())  
  
g <- function(x) {s <- round(abs(sin(x))*1000 )}  
set.seed(s) # randomize  
# mais assure que g(constante) est une constante  
x*(1- 0.5*rnorm(1)*sin(x)) } # pour perturber  
  
set.seed(2); x1 <- matrix(rnorm(10000))  
set.seed(3); x2 <- matrix(rnorm(10000))  
y <- matrix(2*x1-7*x2+1)
```

```

y      <- apply(y, 1, g)
Z      <- data.frame(x1, x2, y)

```

Nous allons maintenant découper l'ensemble des observations en deux ensembles, l'un d'entraînement, l'autre de validation.

Tout d'abord vérifions le nom des colonnes, cela évite bien des déboires.

```

names(Z)                      # X = [x1, x2] ; y = [y]
[1] "x1" "x2" "y"

```

La fonction **createDataPartition** de la librairie **caret** va nous permettre d'extraire une partition de Z. Elle nous permet de constituer aléatoirement un index d'une taille donnée correspondant à une proportion des numéros de lignes de Z. Le vecteur **index** résultant nous permettra d'extraire les lignes de notre jeu d'entraînement, et en prenant son opposé **-index** nous pourrons extraire les lignes de notre jeu de validation/test.

```

require(caret)                 # nous apporte un partitioning élégant
set.seed(1)                     # initialise générateur de nombres aléatoires
index <- createDataPartition(y=Z$y,           # extraction index
                            p= 60/100,    # qui est une suite de
                            list=FALSE)   # 6000 nombres aléatoires

str(index)                      # vérification c'est ok

Ze  <- Z[ index, ]            # extraction de 6000 lignes
Zv  <- Z[-index, ]            # extractions des 4000 lignes complémentaires

int [1:6000, 1] 1 3 5 7 9 10 11 12 13 14 ...

head(index)                    # vérifications - 5 premiers index
head(Ze)                        # 5 premières lignes sur ces index
dim (Ze)                         # on a bien 6000 lignes
head(Zv)                        # 5 premières lignes sur d'autres index
dim(Zv)                          # on a bien 4000 lignes

```

Les six premières lignes du jeu d'entraînement Ze :

	x1	x2	y
1	-0.89691455	-0.96193342	7.3642765
3	1.58784533	0.25878822	2.2410705
5	-0.08025176	0.19578283	-0.5627873
7	0.70795473	0.08541773	1.3363726
9	1.98447394	-1.21885742	15.9955753
10	-0.13878701	1.26736872	-2.3581976

Les quatre premières lignes du jeu de validation/test Zv :

	x1	x2	y
2	0.18484918	-0.29252572	3.915574
4	-1.13037567	-1.15213189	6.875471
6	0.13242028	0.03012394	1.143770
8	-0.23969802	1.11661021	-9.811021

On constate que sur les 10 premières lignes on a bien 60 % de lignes dans le jeu d'entraînement et 40 % dans le jeu de validation/test, avec un aspect aléatoire sur le choix de la répartition des numéros de lignes entre les deux **data.frames**.

Pour les calculs qui suivent, nous avons besoin d'extraire la colonne **y** et ses partitions :

```
y <- matrix(Z$y)           # les y
ye <- matrix(Ze$y)          # les y d'entraînement
yv <- matrix(Zv$y)          # les y de validation

dim(y)                      # vecteur 10000 coord
dim(ye)                     # vecteur 6000 coord
dim(yv)                     # vecteur 4000 coord
```

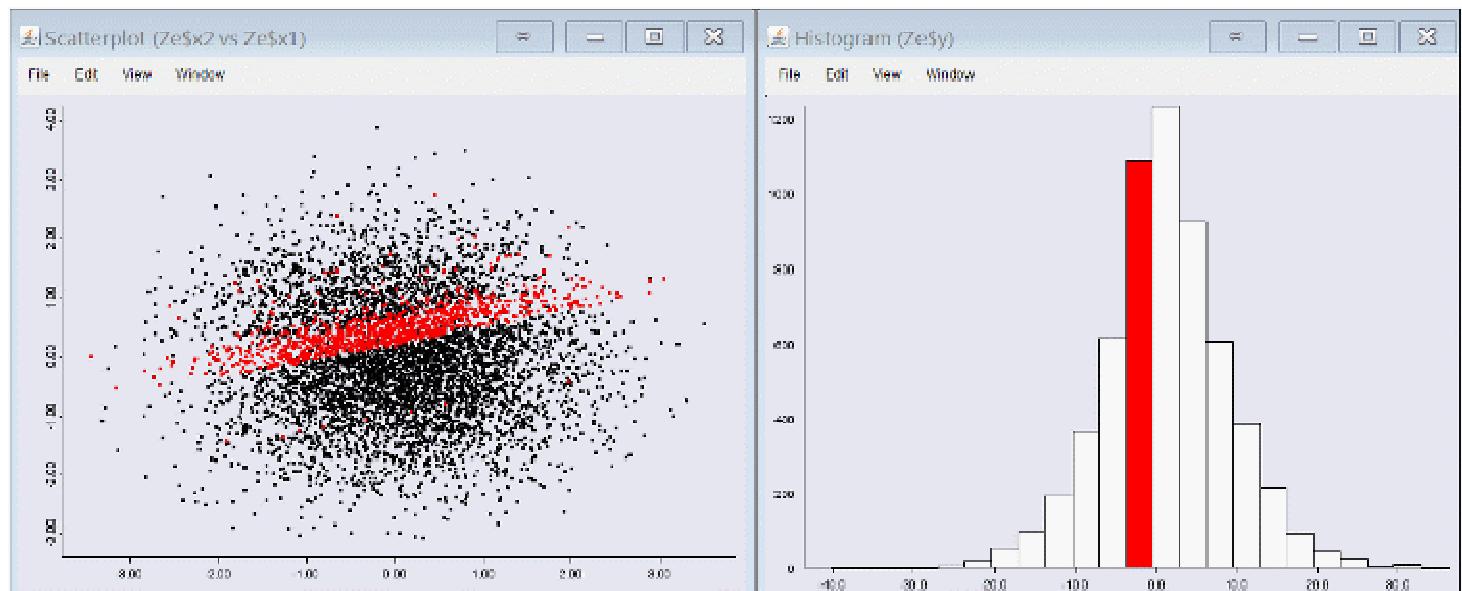
Tester des hypothèses, p_value

À ce niveau, il faudrait étudier visuellement et statistiquement nos données comme nous l'évoquons dans d'autres chapitres.

Analyse graphique interactive avec iplots

En termes graphiques, nous allons nous contenter ici d'exploiter un package très agréable de R, le package **iplots**, qui permet une visualisation interactive et synchronisée sur plusieurs fenêtres. Ce package permet de sélectionner à la souris certaines données sur un des graphiques et de visualiser les points concernés sur les autres graphiques. Ici on a sélectionné une bande dans la distribution des **y** et on voit les points concernés dans le nuage de points **x1** par **x2**.

```
library(iplots)  
  
iplot(Ze$x1, Ze$x2)  
ihist(Ze$y)
```



Données synchronisées interactivement sur deux graphiques

La bande sélectionnée sur l'histogramme de droite correspond à une série de points alignés sur le "scatterplot" de gauche, si l'on fait cela sur d'autres bandes on constate que l'on obtient à gauche une autre série de points alignés et parallèle à la première, cela vous confirme visuellement une certaine linéarité des données **x1** et **x2** par rapport à **y**.

On peut (et doit) utiliser d'autres représentations des données si on veut les percevoir avec acuité. Par ailleurs les tests statistiques vont nous aider à élaborer des stratégies d'exploration de nos data et à formaliser nos premières intuitions.

Test de Breush-Pagan et zoom sur p_value

En termes de test statistique d'hypothèses, pour cette petite application pratique, nous allons nous contenter d'effectuer le test de **Breush-Pagan** pour obtenir de bonnes présomptions d'**homoscédasticité des résidus** de l'application d'un modèle de régression linéaire. En fait le test de Breush-Pagan est un test d'hétéroscléasticité ! (pour s'en convaincre, il faut lire l'aide du test dans R : "*Performs the Breusch-Pagan test against heteroskedasticity*").

Pour comprendre le résultat du test il faut connaître la notion de **p-value**, qui est très utilisée en statistiques (parfois à tort !) et qui permet de statuer avec plus ou moins de conviction sur le résultat d'un test.

L'idée est de tester une hypothèse nommée "**hypothèse nulle**" que l'on écrit souvent H_0 . Dans la théorie sous-jacente à l'usage de la p-value, on considère que cette **hypothèse nulle ne peut jamais être validée**. Attention, ce point est très important : on ne peut jamais conclure avec cet outil que H_0 est vraie.

Au contraire, on sait qu'à partir d'un certain seuil on pourra s'orienter vers le fait d'invalider l'hypothèse nulle.

Les seuils souvent utilisés sont les suivants :

- Si la p_value est **inférieure à 0.01** on se propose d'**invalider** l'hypothèse nulle.
- Si la p_value est **supérieure à 0.1** on ne peut **pas invalider** l'hypothèse nulle.

Le code correspondant est le suivant :

```
library(lmtest)          # package comportant des tests pour lm
bptest(y ~ x1 + x2, data = Ze) # le test
? bptest                  # aide du test

studentized Breusch-Pagan test

data: y ~ x1 + x2
BP = 50.6275, df = 2, p-value = 1.015e-11
```

La p_value est très petite, on invalide l'hypothèse nulle qui était l'hétéroscléasticité de nos résidus. C'est une bonne nouvelle, nous disposons de données dont les résidus face à une régression linéaire ont une forte présomption d'homoscédasticité.

Comme vous l'aurez compris, il est vital de bien comprendre quelle est l'hypothèse nulle des tests que vous utilisez. Ce qui n'est pas toujours simple, méfiez-vous des nombreuses erreurs de documentation et **testez vos tests sur des données** que vous maîtrisez et dont vous savez anticiper les résultats avant de généraliser l'usage d'un test que vous appréhendez pour la première fois.

Création d'un modèle (régression linéaire multiple)

Maintenant nous allons façonner notre estimation de la fonction de prédiction f , en utilisant un algorithme des plus simples et des plus rapides, qui sera appliqué sur l'ensemble d'entraînement : le "fitting linear model". C'est cet entraînement, ou apprentissage (machine learning), qui donnera le modèle \hat{f} .

```
# création modèle linéaire sur set d'entraînement          #
# application sur set de validation                         #

f <- lm(y ~ x1+x2,data=Ze)      # création du modèle y=f(X)
```

Notez la syntaxe : **lm()** nous retourne un modèle f , élaboré par un apprentissage sur le **data.frame Ze**. La variable à prédire est la colonne **y** et cette prédiction doit être effectuée au regard de la conjonction des effets des attributs représentés par les colonnes **x1** et **x2**. La syntaxe des "formulas" du type **y ~ x1+x2** est caractéristique du langage R, elle est très puissante et permet de décrire des cas très variés de relations potentielles entre les variables.

Établissement d'une prédiction

Nous pouvons maintenant effectuer une prédiction sur notre ensemble de validation. La syntaxe de **predict** est utilisable sur une grande part des modèles "fittés" par R.

```
yv_ <- predict(f,newdata = Zv) # f(Xv)
yv_ <- matrix(yv_)
```

Nous pouvons stocker nos prédictions mais aussi notre modèle (c'est vraiment très utile) :

```
# stockons nos résultats
# pour un usage futur
prediction_lm <- data.frame(Zv[,-3], # on enlève y
                                prediction_y = yv_,
                                row.names = NULL)
head(prediction_lm)
tail(prediction_lm)

f_lm <- f                         # mémorisation du modèle pour une
                                    # future comparaison
```

Quelques lignes issues de notre prédiction :

```

          x1           x2 prediction_y
2      0.18484918 -0.29252572     3.421428
4     -1.13037567 -1.15213189     6.713933
6      0.13242028  0.03012394     1.070163
8     -0.23969802  1.11661021    -7.246921
18     0.03580672 -0.64824281     5.591171
19     1.01282869  1.22431362    -5.437667

          x1           x2 prediction_y
9984   0.88058277  0.4480796   -0.3087302
9987   -0.95327970 -0.4346129    2.0849951
9992   -0.42200874 -0.1059597    0.8842236
9995   -0.97046292  0.7907912   -6.4733440
9999   -0.06147666  0.8628701   -5.1180128
10000  -0.40880672 -2.5715141   18.0602384

```

La prédiction commence à la ligne 2 et finit à la ligne 10000, c'est aléatoire. Souvenez-vous que vous avez ici 4000 lignes sur les 10000 d'origine !

Oui, mais quelle est la qualité de notre prédiction ? Voyons cela...

Étude des résultats et représentations graphiques

Nous avons obtenu les prédictions sur notre ensemble de validation/test **yv_** et nous aimerais bien les comparer avec les valeurs réelles **yv**. Une première façon de procéder consiste tout simplement à tracer les points de l'un versus les points de l'autre. Plus les points sont proches de la diagonale, moins il y a d'erreur. Mais attention, il faut avoir à l'esprit qu'une part de l'erreur est irréductible parce que les données possèdent un aléa 'intrinsèque', un bruit. Par maladresse on intègre parfois ce bruit dans le modèle et celui-ci a alors moins de capacité prédictive sur de nouveaux jeux de données : c'est la hantise du data scientist, bien connue sous le terme overfitting.

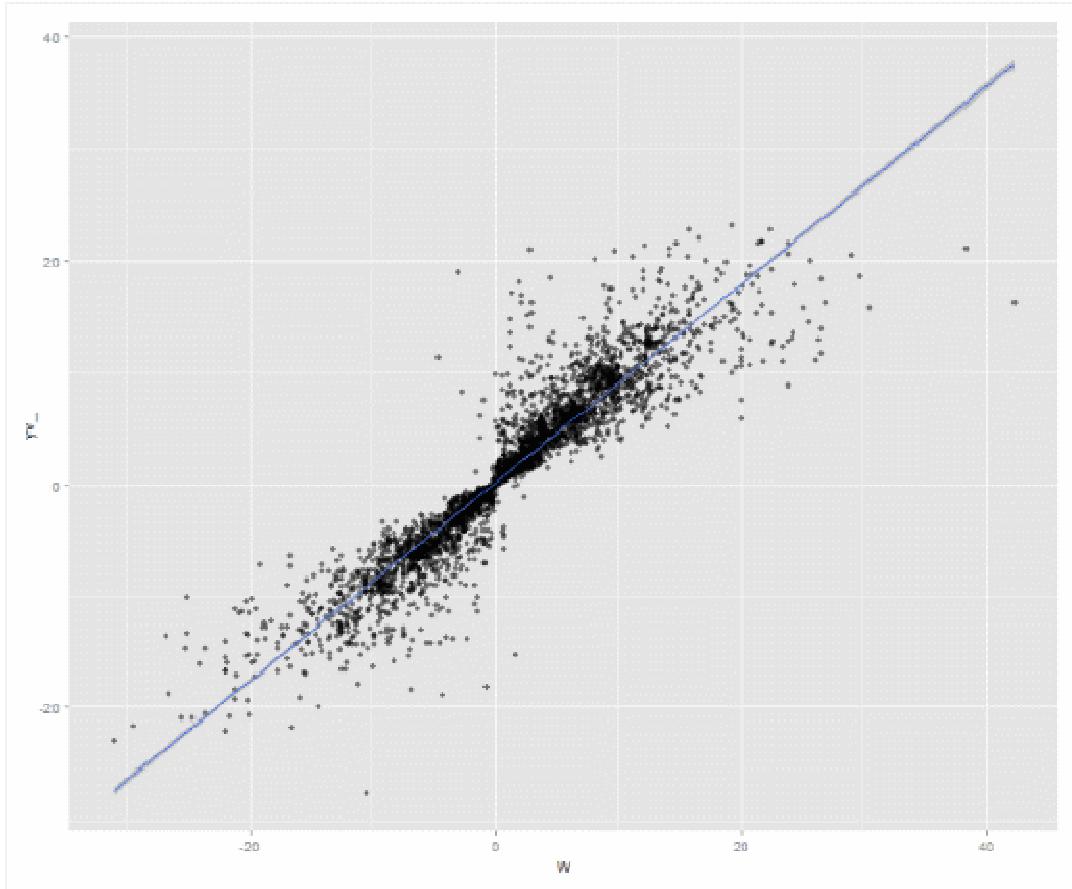
```

require(ggplot2)                      # pour disposer de qplot()
                                         # visualisation du résultat
qplot(yv,yv_,                         # comparaison y et y predit
      geom = c("point", "smooth"),
      method ="lm",
      alpha = I(1/2)
)                                     # aspect satisfaisant

```

Notez les paramètres très utiles de la fonction **qplot()**, qui permettent d'introduire une régression, ici linéaire, et une transparence (**alpha**) des points ceci afin de mieux déterminer les zones constituées de nombreux points ou pas (essayez de faire varier **alpha**).

On obtient des points dont la régression linéaire (**lm**) est effectivement proche d'une droite à 45°, c'est une bonne nouvelle quant à la pertinence de notre modèle.



Prédiction VS valeurs réelles

Une fausse bonne idée commune et dangereuse dont il faut absolument vous méfier :

Nous avions deux variables, x_1 et x_2 ; on aurait pu imaginer estimer y étant une fonction de x_1 . On aurait obtenu $y = f_1(x_1) + e_1$. Puis vouloir d'estimer e_1 en fonction de x_1 , $e_1 = f_2(x_2) + e_2$... et, au final, on aurait obtenu quelque chose comme :

$$y = f_1(x_1) + f_2(x_2) + e_2.$$

Eh bien cela n'est pas correct (sauf cas particuliers, et souvent triviaux), car on transporte un biais d'une estimation sur l'autre (et de fait on néglige l'impact de l'interaction entre x_1 et x_2 sur y).

Dans la littérature anglaise, quand vous trouvez les termes "stepwise blabla" ou ROR (regression on residual)... soyez très attentif, vous êtes peut-être face à une erreur méthodologique commune qu'il ne serait pas judicieux de perpétuer !

L'étude des résidus est une pratique simple qui nous permet de mieux qualifier le résultat apporté par un modèle. Nous allons constituer un vecteur de résidu ϵ et le tracer en fonction des valeurs estimées :

```
e <- yv - yv_
# vecteur d'erreurs (= residus)

qplot(yv_, e,
      # dispersion des erreurs fonct de yv
      geom = c("point", "smooth"),
      method = "lm",
```

```

alpha = I(1/2)
)

```

Ce qui nous donne :

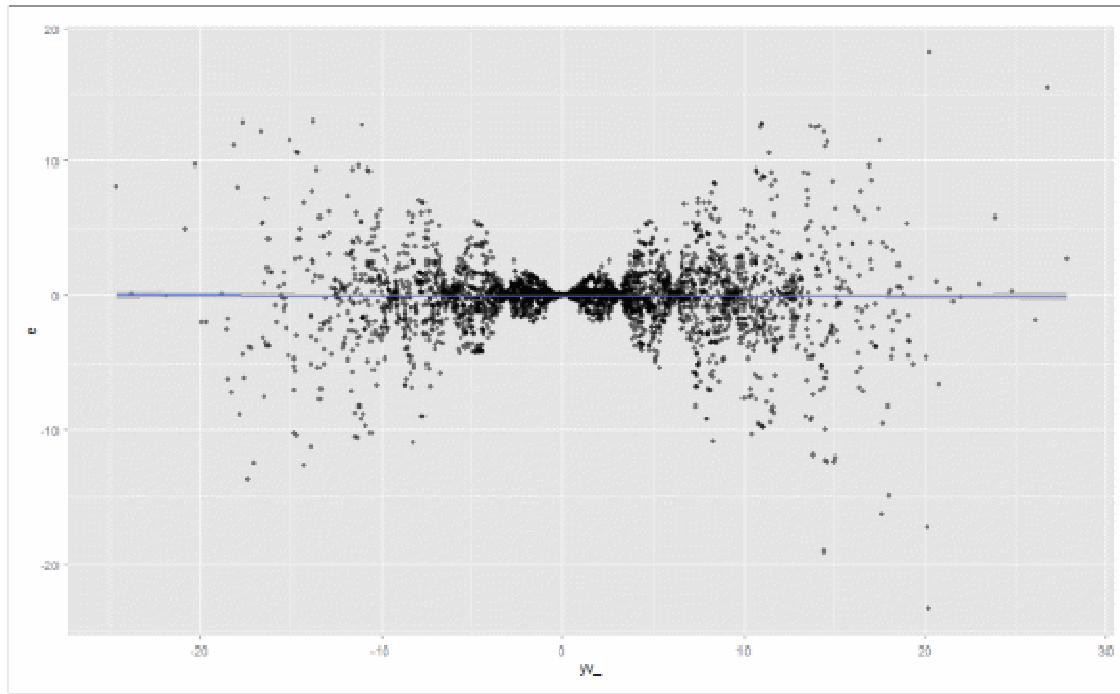


Diagramme des résidus en fonction des prédictions

On constate que la régression linéaire sur les résidus montre que "globalement" les résidus ne dépendent pas des valeurs prédites. En effet la droite est horizontale (si on avait utilisé une approximation, un autre lissage, on aurait regardé si sa dérivée était presque toujours nulle) : c'est une autre bonne nouvelle.

Notez que l'on aurait préféré obtenir une bande homogène, horizontale et répartie symétriquement autour de l'axe horizontal : notre modèle a sans doute un petit défaut, ou l'échantillonnage est un peu biaisé !

On aurait pu tracer le même diagramme en proportion d'erreur :

```

# en pourcentage
d <- max(yv_) - min(yv_)           # écart max dans prédiction
p <- e/d *100                      # proportion d'erreur

qplot(yv_, p,                         # étude proportion erreur
      geom = c("point", "smooth"),
      method = "lm",
      ylab = "pourcentage d'erreur",
      alpha = I(1/2)
)

```

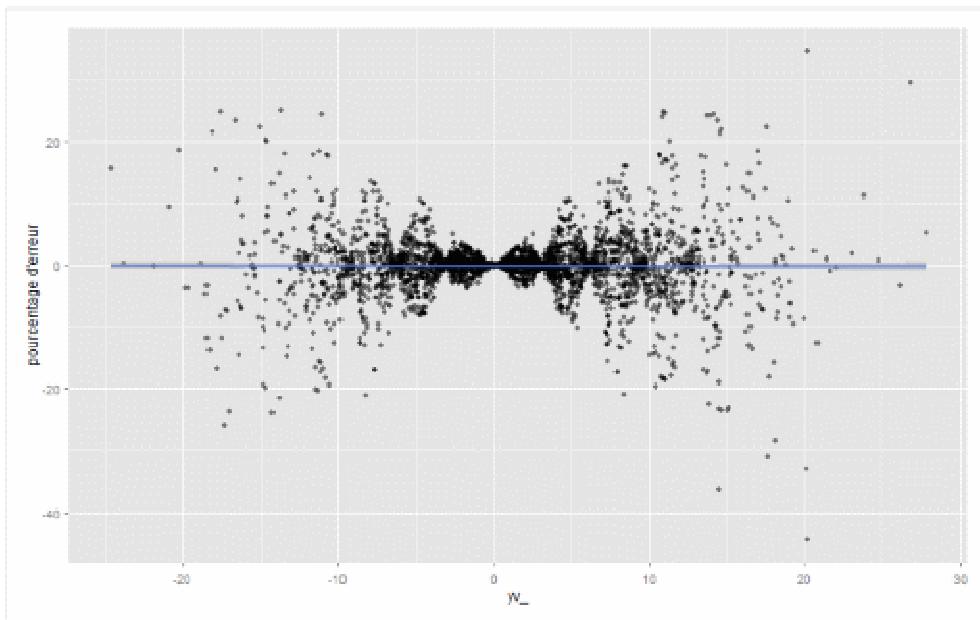


Diagramme des résidus (en %) en fonction des prédictions

Il est ainsi plus facilement interprétable, on visualise mieux la proportion de bruit très présent.

Pour mieux appréhender la qualité du résultat, nous pouvons également faire appel aux indicateurs évoqués plus haut.

Il faut donc les calculer ; c'est ce que nous allons effectuer maintenant.

Indicateurs courants - calculs

Pour ce faire, le code est assez simple, mais il ne faut pas confondre les données d'entraînement et les données de test. On effectue ici ces **calculs sur les données de test/validation** et sur le vecteur d'erreur **e**.

Vous remarquerez également qu'ici le calcul est fait en partant de la norme calculée à l'aide de la fonction **entrywise.norm()** du package **matrixcalc** (parfois l'auteur appelle les packages librairies, c'est un abus de langage pratique mais pas du tout compris par les anglais dans les forums R...). Le fait de se référer à la norme nous ouvre le chemin pour fabriquer nos propres indicateurs à partir d'autres normes qui seraient mieux adaptées à notre problème, en soi c'est presque un champ de recherche, mais c'est parfois très intéressant. Évidemment si l'on sort du monde de la MSE, il faut se donner d'autres noms et la formule **(ME**2 + SRD**2) – MSE** ne donne plus zéro.

```
# calcul manuel des indicateurs courants #  
  
library(matrixcalc)      # pour faire des calculs matriciels  
  
n  <- length(yv)          # nombre d'éléments de yv  
e  <- yv - yv_             # vecteur d'erreurs (= résidus)  
n  <- length(yv)          # nombre d'éléments de yv  
  
SSE  <- entrywise.norm( e, 2 )**2      # SSE  
MSE  <- SSE / n            # MSE  
RMSE <- sqrt(MSE)          # RMSE  
MAE   <- entrywise.norm( e, 1 ) / n      # MAE  
ME    <- mean(e)            # ME  
NRMSE <- RMSE/(max(yv) - min(yv))       # NRMSE  
CV_RMSE <- RMSE/mean(yv)           # CV_RMSE  
SRD   <- sqrt(mean(e**2) - mean(e)**2)    # SRD  
  
(ME**2 + SRD**2) – MSE           # vérification  
  
MEAN   <- mean(yv)            # moyenne yv  
CENTER <- yv-MEAN             # centrage yv  
SST    <- entrywise.norm( CENTER, 2 )**2  # SST  
R2    <- 1- SSE/SST            # R2  
  
id1 <- data.frame(n, SSE, MSE, RMSE, MAE, ME, NRMSE, CV_RMSE, SRD, SST, R2)  
id1
```

On vérifie bien que **(ME**2 + SRD**2) = MSE**, puisque leur différence est nulle :

```
[1] 0
```

Il est toujours intéressant de vérifier la cohérence de ses calculs.

Et on obtient nos résultats de calculs (stockés dans un mini **data.frame** pour usage futur éventuel). Évidemment on aurait pu encapsuler tout cela dans une fonction R mais ce n'est pas le sujet ici.

Résultat de nos calculs :

n	SSE	MSE	RMSE	MAE	ME	NRMSE
4000	26482.29	6.620573	2.573047	1.4765	-0.06345662	0.03505314
CV_RMSE	SRD	SST		R2		
2.511445	2.572265	232643.2		0.8861678		

La valeur normalisée de la racine de l'erreur quadratique moyenne (NRMSE) a l'air petite (3.5 %) et le coefficient de détermination (R2) plutôt proche de 1. C'est rassurant.

Nous allons maintenant regarder plus avant le modèle créé.

Étude du modèle linéaire généré

Chaque package définit sa propre visualisation des éléments décrivant le modèle généré. Dans de nombreux cas, c'est la fonction **summary()** qui vous permet d'accéder à cette information.

```
summary(f) # détail du modèle
```

Ce qui nous donne les informations suivantes (que nous interpréterons en fonction de nos connaissances à ce stade de l'ouvrage).

```
Call:  
lm(formula = y ~ x1 + x2, data = Ze)
```

Residuals:

Min	1Q	Median	3Q	Max
-18.9865	-0.6949	0.0014	0.7709	20.9093

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	1.00921	0.03438	29.36	<2e-16 ***
x1	2.04257	0.03439	59.39	<2e-16 ***
x2	-6.95545	0.03398	-204.72	<2e-16 ***

Signif. codes:	0	'***'	0.001	'**'
	0.01	'*'	0.05	'.'
	0.1	' '	1	

Tout d'abord, le résumé nous rappelle la formule utilisée. C'est très utile lorsque vous avez stocké le modèle et que vous l'abordez plus tard.

La ligne concernant les résidus est très intéressante, mais notez bien que, comme pour les autres indicateurs, elle a été effectuée en utilisant les données d'entraînement et pas les données de test et que donc la fiabilité de cette information est partielle.

La valeur médiane des résidus est presque nulle : il y a autant de résidus inférieurs à zéro que supérieurs, comme nous pouvions l'imaginer en regardant nos précédents diagrammes. Cinquante pour cent des erreurs sont comprises entre -0.69 et 0.77 (du premier quartile au troisième quartile il y a toujours 50 % des données par définition).

Les valeurs du premier et du deuxième quartiles sont très proches de la médiane et très éloignées des valeurs extrêmes (par exemple 0.7709 est très inférieur à 20.9093).

La suite nous donne l'équation du modèle, en effet notre modèle est linéaire et dépend de deux variables, donc on connaît sa forme :

$$y = a.x1 + b.x2 + c$$

On peut lire :

Coefficients:

	Estimate
(Intercept)	1.00921
x1	2.04257
x2	-6.95545

Donc :

$$y = 2.04257 * x1 - 6.95545 * x2 + 1.00921$$

C'est l'équation d'un plan en trois dimensions, la valeur de c (**Intercept**) représente le décalage par rapport à un plan qui passerait par l'origine.

Les *** après les valeurs "a, b, c" signifient que l'on peut avoir grandement confiance en elles.

Nous allons essayer de manipuler ce modèle par nous-mêmes. Comme nous n'avons pas l'intention de recopier à la main ces coefficients, ce qui serait source d'erreurs, nous allons utiliser le fait que la fonction **summary()** renvoie en fait un objet dont nous pouvons lire le contenu. Cet objet c'est **f** lui-même.

```
a <- f$coefficients[2]      # extraction des coefficients du modèle
b <- f$coefficients[3]
c <- f$coefficients[1]

f_ <- function(x1,x2) {      # recréation de la fonction du modèle
  a*x1+b*x2+c
}
```

Nous avons créé une fonction de deux variables, nommée **f_()**, qui exprime l'équation linéaire correspondant à notre modèle.

Nous pouvons maintenant dessiner notre fonction dans l'espace : par construction c'est un plan de l'espace 3D.

Un hyper-plan est une combinaison linéaire d'une dimension inférieure d'une unité à la dimension de l'espace. Par exemple une droite ($y = ax+b$) est un hyper plan d'un espace en 2D, et un plan est hyperplan d'un espace en 3D.

```

x1_ <- seq(min(x1), max(x1), by=0.2) # maillage x1
x2_ <- seq(min(x2), max(x2), by=0.2) # maillage x2
y_ <- outer(x1_, x2_, f_)           # une valeur par couple

persp(x1_,
      x2_,
      y_,
      phi=40,
      theta=60,
      col= "green",
      ticktype = "detailed",          # axe détaillé
      shade=.0001,
      cex.lab=1.0 )                  # attention zoomer
                                    # pour le visualiser

```

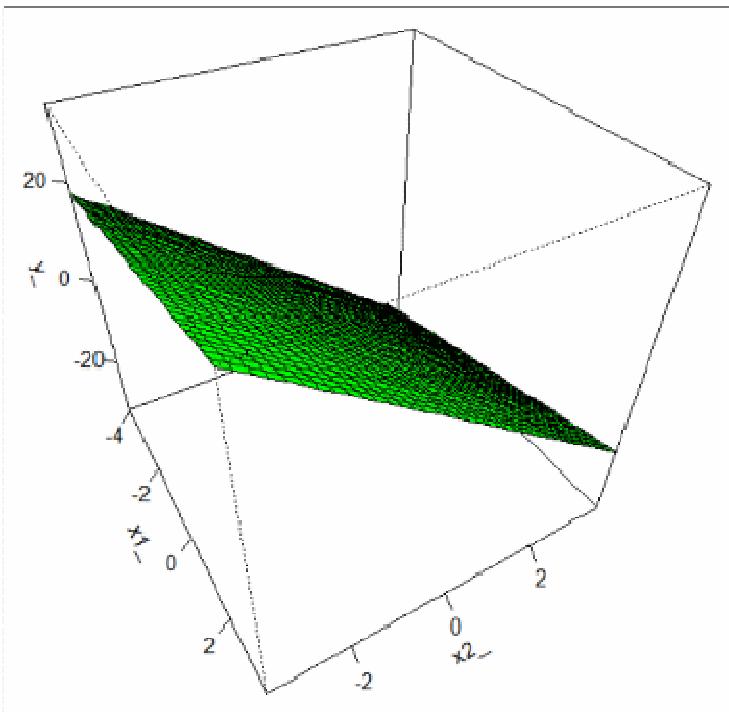
Ce code nous permet d'aborder l'utilisation de la fonction **persp()** de R qui fabrique des représentations en 3D.

Tout d'abord on y fabrique une série de valeurs factices de **x1** et **x2** couvrant l'espace maximum de toutes les valeurs effectives de **x1** et **x2** avec un "pas" de 0.2. En fait cela nous permettra de fabriquer différentes combinaisons de couples de valeurs (**x1_**, **x2_**) puis de leur appliquer la fonction **f_()**.

On obtient 39 valeurs pour **x1_** et 38 pour **x2_**.

Puis en appliquant **f_()** sur l'ensemble des 1482 couples (**x1_**, **x2_**) via la fonction **outer()** on obtient une matrice **y_** de dimension 39 par 38 qui contient les 1482 valeurs correspondantes **f_(x1_,x2_)**.

Ce travail préliminaire nous permet de renseigner les structures de données attendues par la fonction **persp()**, les autres paramètres étant des paramètres de présentation. Le résultat pourra paraître minuscule sur votre écran, il faudra dans ce cas "zoomer" pour obtenir la figure suivante :



Modèle de prédiction linéaire $y=f(X)$

Nous avons obtenu le modèle. Il est très intéressant en soi, mais cela ne nous montre pas comment les observations se répartissent **réellement** autour de ce modèle. Pour avoir une visualisation, bien imparfaite, des points du jeu d'entraînement, nous allons utiliser le code suivant.

Comme il y a 6000 points, nous ne tracerons qu'un petit pourcentage aléatoire de ceux-ci, sinon le diagramme sera totalement illisible. Afin de distinguer les points du haut et du bas nous allons jouer sur les couleurs et la forme des points, via quelques petites astuces.

Le code transformé :

```

palette <- colorRampPalette(c("red","yellow"))(100)
diag <- persp(x1_,
               x2_,
               Y_,
               phi=40,
               theta=60,
               col=palette[
                   round((y_+max(y_))/(max(y_)-min(y_))* 100)],
               ticktype = "detailed",
               shade=.0001,
               cex.lab=1.5 )           # attention zoomer
                           # pour le visualiser

index_ <- createDataPartition(y=Z$y,          # extraction index
                             p= 2/100,    # qui est une suite de
                             list=FALSE) # 6000 nombres aléatoires

u <- as.vector(matrix(Z[index_,1]))
v <- as.vector(matrix(Z[index_,2]))

```

```

w <- as.vector(matrix(Z[index_, 3]))

nuage <- trans3d(u,
                  v,
                  v,
                  pmat = diag)

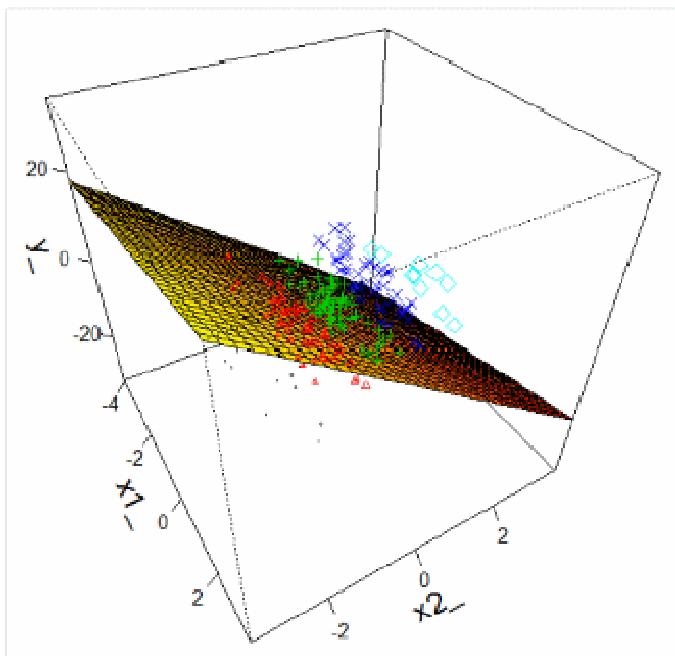
points(nuage,                                     # insertion dans le diagramme
       pch = v + max(v)+1,      # jeu sur forme des symboles
       cex = v/3.5 + max(v/3),  # jeu sur taille des symboles
       col = v + max(v)+1      # jeux sur couleur des symboles
)

```

Le code est un peu plus riche. Tout d'abord on se dote d'une palette de couleurs, qui nous fabrique 100 codes couleurs (comme les couleurs dans HTML). Puis on fabrique le diagramme que l'on stocke car on va l'amender après. Dans les paramètres, la couleur fait l'objet d'une petite formule qui a pour but de sélectionner une couleur différente de la palette pour chaque valeur de $y_{_}$.

L'idée de cette expression étant de ne pas avoir de valeur négative et de couvrir la centaine d'indices de couleurs de la palette, mais aussi d'avoir des valeurs d'indices entières (pas de virgule). Après on extrait 2 % des points de notre jeu d'essai et on déclare le nuage de points via l'application d'une fonction spécialisée. Il nous reste à tracer les points sur le diagramme stocké précédemment tout en faisant varier la taille des symboles en fonction de la valeur de $f(X)$ et en faisant varier les couleurs et les symboles de façon relativement aléatoire.

Nous obtenons le diagramme suivant.



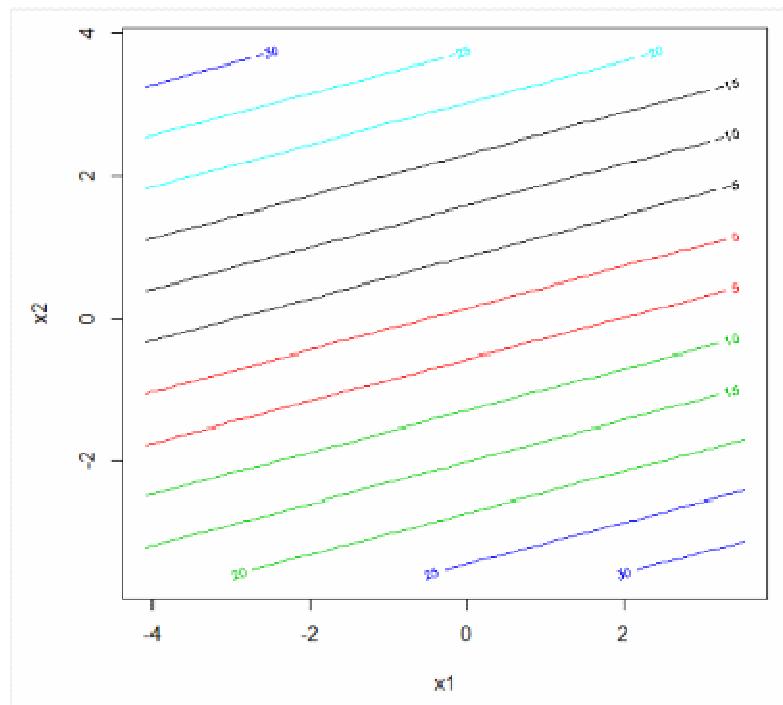
Modèle de prédiction versus 2 % des observations

En termes d'interprétation géométrique et "mathématique", vous avez visualisé un plan affine, à partir du sous-espace vectoriel $\text{Vect}(\vec{X}_1, \vec{X}_2)$ et les points (vecteurs) de l'ensemble d'entraînement. Le plan accueille la projection orthogonale des points correspondant à vos observations, telle que la distance moyenne entre le plan et les points de l'ensemble d'entraînement, soit minimale.

Pour visualiser ce modèle 3D d'une façon un peu plus prosaïque et surtout telle que l'on puisse l'utiliser pour comprendre sa topologie ou même l'utiliser comme un abaque permettant d'avoir une idée de la valeur de y en fonction de x_1 et x_2 , nous disposons d'une autre fonction dans R. Elle est moins spectaculaire mais s'avère très utile car elle nous permet de tracer des contours, c'est-à-dire des lignes de niveau obtenues si on regarde le diagramme par dessus.

```
# lignes de niveau
contour(y_,
        method = "edge",
        xlab="x1",
        ylab="x2",
        col=round(abs(y_) %%5+1) # jeu sur les couleurs #
```

Ce qui nous donne :



Abaque des valeurs de y en fonction de X - Courbes de niveau

Grâce à cette représentation, on peut vérifier que l'on a affaire à un plan (les lignes de niveau sont parallèles et équidistantes). Par ailleurs on peut approximer visuellement la valeur estimée de y en fonction de x_1 et x_2 . Par exemple sur le diagramme on peut lire que pour une valeur de $x_1 = -2$ et $x_2 = 2$ on a y compris entre -20 et -15. On peut aussi visualiser la pente en chaque endroit, qui n'est autre que la perpendiculaire aux

lignes de niveau (si ce n'était pas un plan, on pourrait même déduire une augmentation de la pente là où les lignes se resserrent).

C'est en regardant cette représentation que l'on comprend l'usage du mot gradient, utilisé dans les techniques d'optimisation. Descendre un gradient, ce serait ici se déplacer perpendiculairement aux lignes de niveau, d'ailleurs représentées sous forme d'un gradient de couleurs.

Conclusion sur le modèle linéaire

Nous avons fabriqué un ensemble de données factices à partir d'une relation linéaire que nous avions perturbée par un bruit important revêtant un aléa certain (preuve en est : le volume des nuages de points). Pourtant l'algorithme de régression linéaire multiple a bien extrait l'information, au point de retrouver les coefficients de régression avec une grande efficacité.

L'équation utilisée pour préparer le jeu de données factices puis notamment perturbées servant à élaborer ce jeu d'observations était :

$$y = f(X) = 2.x_1 - 7.x_2 + 1$$

Le modèle issu de nos travaux est :

$$y = f(X) = 2.04257 * x_1 - 6.95545 * x_2 + 1.00921$$

Quand il y a de la linéarité dans les données, le très simple et extrêmement performant algorithme de régression linéaire est d'une efficacité redoutable pour la mettre en évidence. En fait, on retrouve l'utilisation des concepts sous-jacents de linéarité dans de nombreux algorithmes de machine learning.

Du fait de ce que nous savons de la construction du jeu de données, nous pouvons présumer une efficacité plus faible des autres algorithmes sur ce jeu de données, ou le cas échéant un risque d'overfitting intégrant une part du bruit aléatoire que nous avions introduit.

Explorons maintenant ce que donnerait sur ces données un autre algorithme, plus agnostique quant à la linéarité. Pour cet essai nous allons utiliser Random Forest (un algorithme utilisant "ensemble", c'est le terme consacré, plusieurs instances d'un algorithme d'arbre de décision). Cet algorithme possède une grande universalité quant aux hypothèses de son utilisation. Voyons dans quelle mesure sa performance sera plus faible ou égale (qu'elle soit supérieure serait étonnant car nous avions auparavant fabriqué un jeu de données basé sur la linéarité !).

Utilisation d'un modèle "Random Forest"

Pour utiliser cet algorithme, nous allons charger le package approprié et utiliser un code très semblable au précédent.

```
# essai avec l'algorithme randomForest          #
# application sur set de validation            #

require(randomForest)           # forêt d'arbres de décision
f <- randomForest(y ~ x1+x2,
                   data=Ze,
                   ntree = 30)      # création du modèle y=f(X)
yv_ <- predict(f,newdata = Zv)    # f(X)
yv_ <- matrix(yv_)
```

C'est toute la beauté de R : en dehors des paramètres spécifiques à notre nouveau modèle nous utilisons les mêmes syntaxes.

Pour obtenir les diagrammes de résidu, le code est strictement le même que précédemment.

Toujours en effectuant les mêmes calculs, mais en les stockant dans un **data.frame** nommé **id2** et en faisant :

```
# comparaison des indicateurs
id <- rbind(id1,id2)
id
```

On obtient une comparaison de nos valeurs calculées. Jetons un œil sur NRMSE et R2 :

```
NRMSE
0.03505314
0.03775296

R2
0.8861678
0.8679577
```

Les valeurs se sont légèrement dégradées, mais cette différence ne semble pas particulièrement significative.

Voyons cela via une représentation dans l'espace en utilisant un code similaire à celui utilisé précédemment, mais avec une façon de définir la fonction à représenter qui sera plus générale (donc que vous pourrez utiliser sur de nombreux autres modèles).

Nouveau code pour la fonction à représenter :

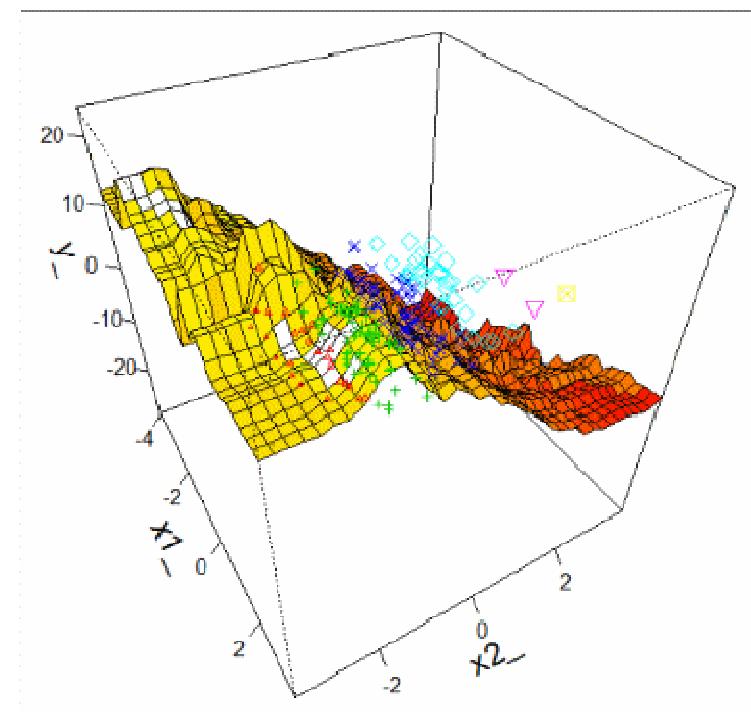
```

f_ <- function(x1,x2){      # création fonction prédiction
  x1 <- matrix(c(x1))
  x2 <- matrix(c(x2))
  y  <- matrix(c(1))
  ligne_X <- data.frame(x1,x2,y)
  predict(f,newdata = ligne_X )
}

```

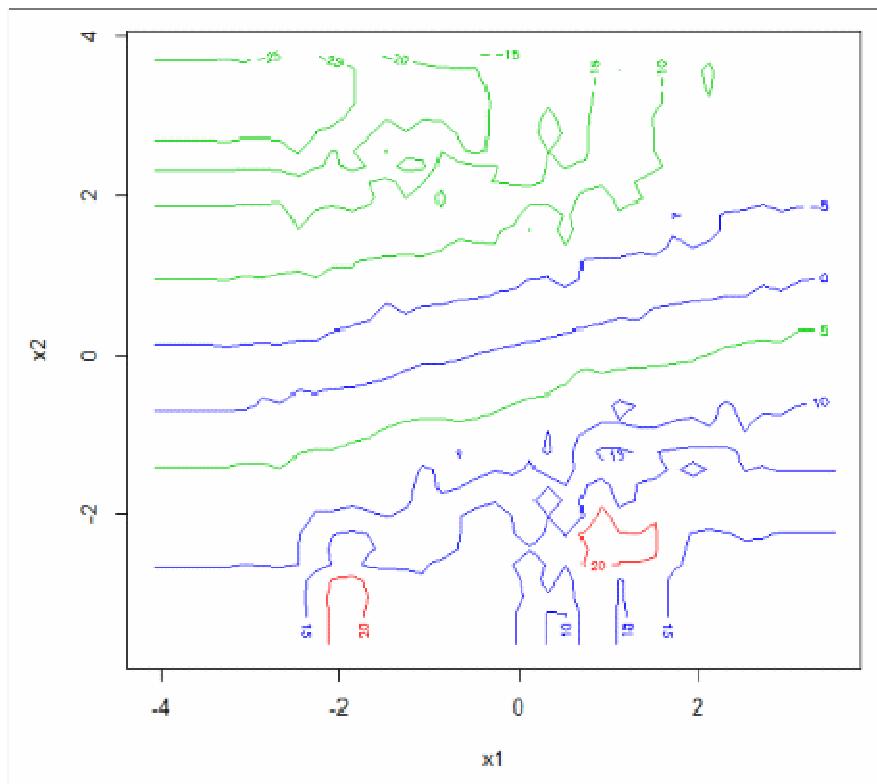
Dans ce code on n'utilise aucune connaissance sur le type de modèle choisi, on se contente de l'invoquer.

Le code qui suit est toujours le même (aux paramètres de couleurs près) et on obtient alors :



Prédiction avec Random Forest et points réels

Du côté des courbes de niveau, on obtient :



Courbes de niveau, Random Forest

La plage de valeurs de y est moins large que celle identifiable dans le modèle linéaire, de -25 à 25 au lieu de -30 à 30.

La topologie de ce modèle est manifestement moins linéaire en dehors des plages de y allant de -5 à 10, mais pourtant on a vu plus haut que ses performances n'étaient pas forcément plus mauvaises au vu de la valeur normalisée de la racine de l'erreur quadratique moyenne (NRMSE) et du R2.

Le niveau de pertinence de nos outils d'évaluation de modèles à ce chapitre de notre ouvrage ne nous donne pas la capacité de choisir entre les deux modèles. Dans un tel cas et sans outils et investigations supplémentaires, n'hésitez pas et prenez le modèle qui semble le plus simple, à savoir celui qui semble le moins chercher à utiliser les moindres détails de variation de X (en quelque sorte le plus fluide). Ce modèle est sans doute le plus généralisable à d'autres données, le moins "overfitté".

Voilà, nous disposons maintenant du bagage qui va nous permettre d'aborder avec confiance la compréhension puis l'utilisation de techniques très diverses de machine learning ainsi que le décryptage de nombreux articles de recherche.