MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE ABDELHAMID IBN BADIS - MOSTAGANEM

Faculté des Sciences Exactes et d'Informatique

Département de Mathématiques et informatique

Filière : Informatique

POLYCOPIE DE COURS ET DE TP

Fouille de Données

Ce polycopié est destiné aux étudiants de la 1ère année Master ISI

Filière Informatique

PREPARE PAR :

Moumene Mohammed Elamine

Courriel : elamine.moumene@univ-mosta.dz

Année Universitaire 2020-2021

Avant-propos

Appelée aussi Data Mining, la fouille de données regroupe une panoplie de méthodes permettant d'extraire des connaissances à partir du volume extraordinaire de données présent un peu partout dans le monde. Les plus grands gouvernements et entreprises mondiales investissent massivement dans ce domaine à cause de son intérêt grandissant. La fouille de données est devenu primordiale pour l'éducation et à l'enseignement. Elle permet de prédire les performances scolaires des élèves en se basant sur les données d'anciens élèves, de les orienter en définissant leurs penchants scientifiques, d'augmenter le rendement des enseignant et d'améliorer le système en générale. La fouille de données est cruciale aussi en médecine. Elle permet par exemple de connaitre les effets des médicaments à l'aide de données rassemblés sur les patients au fil des années. Elle permet aussi de desceller les maladies propagées dans certaines zones afin d'aider les décideurs à prendre les mesures adéquates. La fouille de données oriente les décisions politiques et les stratégies importantes mises en place par les gouvernements pour le développement. Elle aide notamment l'humain à faire face aux changements climatiques que connait le monde en ce moment.

Ce polycopié de cours et de TP est destiné aux étudiants en Master 1 ISI de la filière informatique. Son objectif est de présenter à l'étudiant quelques outils essentiels de fouille de données et leurs applications sur de vrais problèmes de la vie quotidienne. Des problèmes réalistes sont traités dans ce document et les solutions sont détaillées et implémentées à l'aide du langage R. Nous allons voir par exemple comment la régression peut être utilisée pour conseiller un investisseur ou bien prédire les prix de maisons en se basant sur des critères tels que leurs surfaces, le nombre de pièces, l'Age de la maison…etc. Nous abordons également quelques méthodes de classification comme les arbres de décision, Naive bayes et nous démontrons à l'aide du langage R comment les réseaux sociaux utilisent la régression logistique pour cibler les utilisateurs avec de la publicité. Aussi, nous montrerons à l'étudiant comment à partir d'une base de transactions de supermarché (listes d'achats) on peut extraire des règles utiles en marketing.

Table des matières

Chapitre 1 :

Introduction Générale

Introduction générale :

Nous vivons au 21èm siècle ou une grandissante quantité de données nous entoure. Allant des données textuelles que nous échangeons tous les jours et que nous lisons, les images capturées par nos smartphones jusqu'aux positions GPS enregistrées en tout temps par les réseaux. Nous produisons donc constamment des données que ce soit d'une manière consciente ou pas. Ces données ne cessent de croitre d'une manière exceptionnelle et de s'accumuler tout le temps. Il a été estimé que l'humanité aie cumulé 130hexaoctets jusqu'à l'an 2005. Que le volume des données produit jusqu'en 2010 est de 1200 Exabyte. En 2020 le nombre a été estimé à 40900 Exabyte.

Figure 1.1 : Les données grandissantes dans le monde

A cet effet, le domaine de la fouille de donnée, connu aussi sous l'expression data mining, devient une priorité. Ce domaine a pour objectif l'extraction de connaissances à partir de ces grandes quantités de données, par le biais de méthodes automatiques ou semi-automatiques. Une panoplie d'algorithmes issus des statistiques, de l'intelligence artificielle ou bien de l'informatique tentent de desceller des structures intéressantes dans les données.
Nous retrouvons la fouille de données chez Facebook. L'application sait déjà qui sont nos amis et sait les identifier automatiquement. Amazon, Netflix utilisent de la fouille de données pour leurs systèmes de recommandation. La fouille de données est également utilisée en médecine. Les scientifiques utilisent la fouille de données pour sauver des vies tous les jours.

La fouille de données passe par des étapes. Au début les données sont présentes dans différents endroits (des supports de stockage) tels que dans des serveurs appartenant à de grandes entreprises dédiées aux réseaux sociaux par exemple. Ces données sont à l'état brut. Il faut d'abord cibler les données utiles au projet d'extraction de connaissance visé. Par la suite des algorithmes adéquats sont appliqués afin de procéder à la fouille de données.

Il existe différents algorithmes de fouilles de données et il y a deux principaux types : Algorithmes supervisés et ceux non supervisés. Pour le premier type, nous intervenons au moment de l'apprentissage en montrant à la machine comment elle apprend à partir de données. Pour les algorithmes non supervisés, la machine apprend par elle-même.

Apprentissage supervisé :

Supposons que nous voulons prédire les prix de maisons. Nous avons comme montré dans la Figure 1.2 un ensemble de données récolté sur des maisons déjà vendues. Les points rouges sont les observations qui représentent le prix de vente de chaque maison et sa superficie. Connaissant ces données, nous voulons prédire le prix de la maison d'un ami ayant une maison avec une superficie de 275M². Comment un algorithme de fouille de donnée peut-il nous aider à le faire. L'un des algorithmes de fouille de données permet par exemple d'ajuster une ligne droite sur les données tel que montré dans la figure 1.2 avec une ligne bleu. Cette ligne permet de prédire le prix de n'importe quelle maison à partir de sa surface. Une meilleure idée serait d'ajuster non pas une ligne droite mais une fonction quadratique afin que la prédiction du prix soit plus précise (voir la courbe en vert). Ceci est un exemple d'apprentissage supervisé. Dans ce cas il est fourni à la méthode de fouille de données un ensemble de données contenant les bonnes réponses (les prix exactes de maisons en fonction de leurs surfaces). La tâche de la méthode de fouille de donnée serait de produire plus de bonnes réponses. Lorsqu'on veut prédire une variable continue tel que montré par l'exemple précédant, il s'agit d'un problème de régression. Nous détaillerons plus tard dans ce document la régression et ses multiples formes.

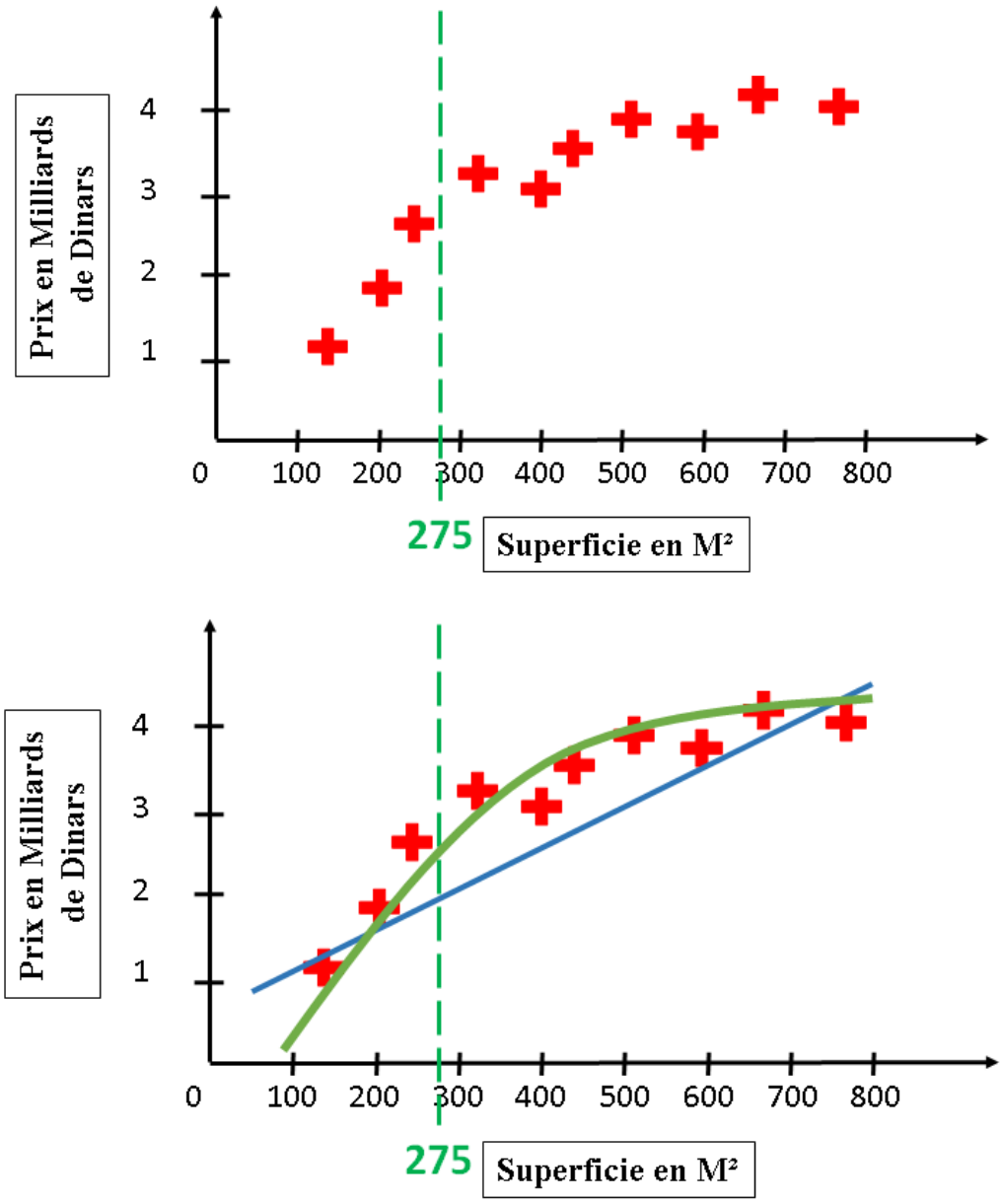


Figure 1.2 : Apprentissage supervisé : La régression pour prédire les prix de maison

Un autre exemple d'apprentissage supervisé serait de pouvoir détecter la présence d'un cancer chez un patient en se basant sur la taille de la tumeur présentée. La figure 1.3 montre un ensemble de données ou chaque observation est représenté par la taille de la tumeur chez le patient et le fait que le patient soit diagnostiqué comme cancéreux ou pas. L'idée est d'utiliser cet ensemble de données afin de prédire par la suite la présence de cancer

chez un nouveau patient en se basant sur la taille de la tumeur. L'apprentissage sera en mesure de fournir une probabilité de la maladie. Cet exemple reflète ce qu'on appelle en fouille de donnée un problème de classification. Lorsqu'on veut prédire des valeurs discrètes (comme ici 0 ou 1) il s'agit d'un problème de classification.

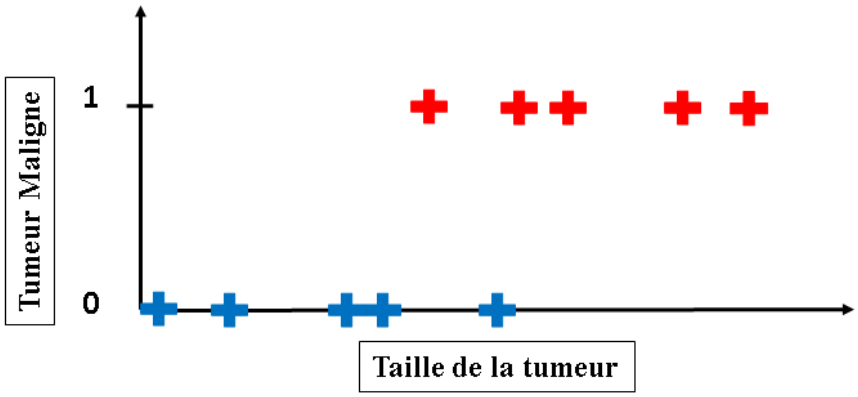


Figure 1.3 : Apprentissage supervisé : la classification

Supposons que nous connaissons maintenant l'âge du patient en plus de la taille de la tumeur. Dans ce cas l'ensemble de données ressemble plutôt à ce qu'on voit dans la Figure 1.4. Nous avons un ensemble de patients (en rouge) dont la tumeur est maline et un autre ensemble (en bleu) de patients dont la tumeur est bénigne. Dans ce cas l'algorithme d'apprentissage va faire en sorte d'ajuster une ligne qui va séparer les deux ensembles de données. Dans cet exemple nous avons deux variables indépendantes qui sont l'âge et la taille de la tumeur. Dans d'autres problèmes d'apprentissage on peut avoir bien plus de variables indépendantes. Dans ce cours, nous allons aborder des techniques de fouilles de données qui permettent de gérer plusieurs variables.

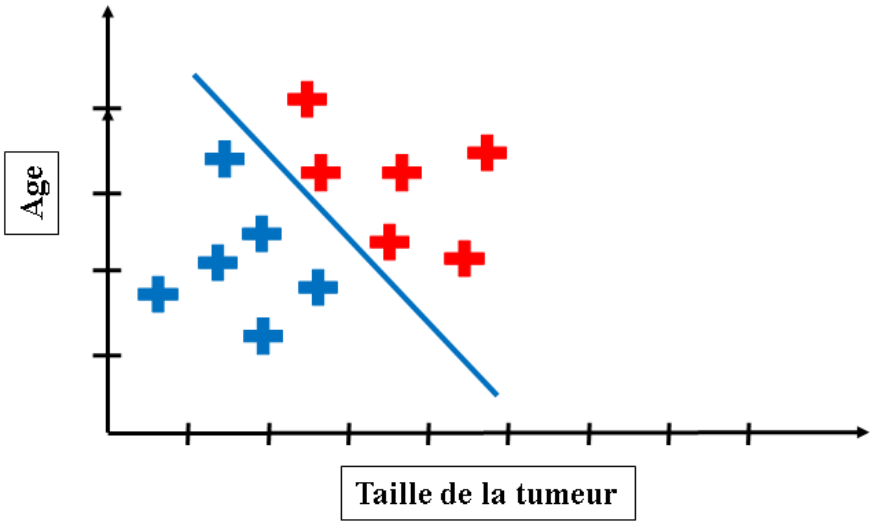


Figure 1.4 : Apprentissage avec plusieurs variables indépendantes

Apprentissage non supervisé

Dans la Figure 1.4 nous avons montré un exemple d'ensemble de données ou les observations étaient classifiées en deux ensembles (tumeurs malignes et bénignes). Nous avions donc la bonne réponse si le patient avait un cancer ou pas. Cependant, dans un problème d'apprentissage non supervisé la réponse n'est pas présente dans l'ensemble de données. Les observations ressembleraient à ce qui est montré dans la Figure 1.5. Les données sont fournies sans label et c'est à l'algorithme de fouille de données de trouver une sorte de structure à ces données. L'algorithme qui permet de diviser ce genre de données en plusieurs ensemble est appelé un algorithme de clustering. Une des applications de ce genre d'apprentissage non supervisé est l'analyse de réseaux sociaux. Dépendamment des personnes à qui envoyer plus de mail, suivant les amis sur Facebook…etc, il est facile de déterminer quels sont les groupes de personnes qui se connaissent.

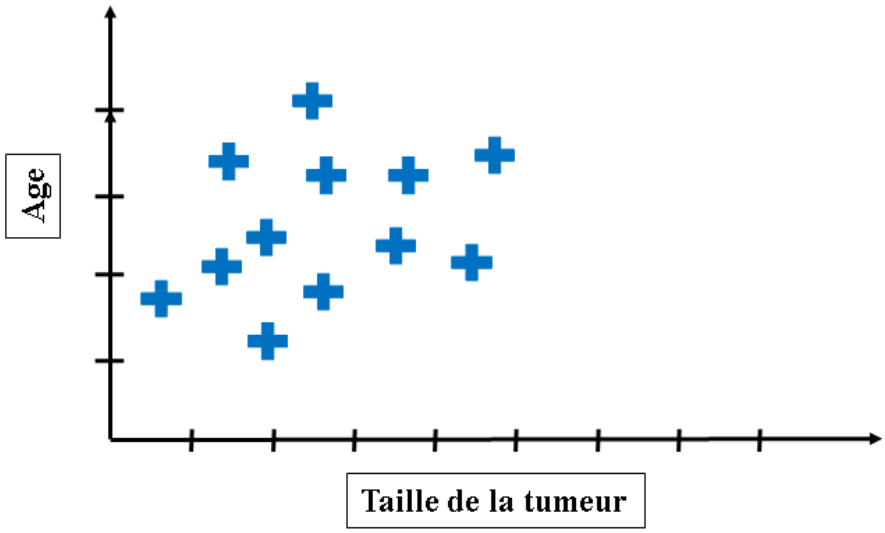


Figure 1.5 : Apprentissage non supervisé : Clustering

Langage R et l'environnement de développement R Studio

Certains langages de programmation conviennent plus à un domaine qu'un autre. Le Langage R a été conçu spécialement pour les calculs mathématiques et statistiques ainsi que pour la manipulation de données. De plus une panoplie de méthodes de fouille de données est implémentée dans les différentes librairies R et il est très facile et intuitif de les utiliser. De ce fait, Le langage R sera utilisé à l'aide de l'environnement RStudio dans ce document pour résoudre tous les problèmes de fouille de données abordés.

Afin d'installer les outils nécessaires, la première étape consiste à ouvrir un navigateur web et de taper le site web (https://cran.r-project.org). Ensuite, télécharger R en fonction du système qu'on a puis l'installer. Une fois R installé, vous on a besoin de télécharger l'environnement de développement RStudio. On peut le télécharger sur le site www.rstudio.com puis l'installer normalement. R Studio détectera le langage de programmation R automatiquement. Pour un premier essai on peut taper l'instruction : print(«hello world ») dans la console puis taper entrer.

Chapitre 2 :

Prétraitements

Prétraitements des données

Le prétraitement des données est une phase très importante dans le processus de fouille de données. Dans ce chapitre nous allons aborder à l'aide du langage R les différents prétraitements dont nous avons souvent besoin d'effectuer avant chaque application d'algorithme de fouille de données.

Description des données :

L'ensemble de données que nous allons utiliser, celui montré dans la Figure 2.1, contient quatre colonnes (Pays, Age, Salaire et Achat) et 10 observations. Ces données représentent les informations de clients d'une compagnie sur les trois premières colonnes ainsi que le fait qu'ils aient acheté ou non le produit de cette compagnie. Il est important de distinguer ici la différence entre les variables indépendante ainsi que celle dépendante. Les variables indépendantes sont celles représentées par les trois premières colonnes (Pays, Age, Salaire) et la variable dépendante est celle représentée par la quatrième colonne (Achat). L'idée est d'utiliser les variables indépendantes pour prédire la variable dépendante. C'est-à-dire qu'a partir du pays, de l'age et du salaire du client nous voulons être en mesure de prédire si ce dernier va acheter ou pas le produit. La phase de prétraitement que nous allons effectuer durant cette section est importante est doit être effectuée pour toutes les techniques de fouilles de données que nous allons voir par la suite.

	Pays	Age	Salaire	Achat
1	France	44	72000	Non
2	Espagne	27	48000	Oui
3	Allemagne	30	54000	Non
4	Espagne	38	61000	Non
5	Allemagne	40	NA	Oui
6	France	35	58000	Oui
7	Espagne	NA	52000	Non
8	France	48	79000	Oui
9	Allemagne	50	83000	Non
10	France	37	67000	Oui

Figure 2.1. Base de données Entreprise

Importer les données :

Afin d'importer les données sur l'environnement de développement R nous devons commencer par fixer le répertoire de travail. Pour cela il faut aller à Outils (Tools) puis sélectionner Options globales (Global options). Dans l'onglet General, modifier le Répertoire de travail par défaut (Default working directory) en sélectionnant le dossier contenant le fichier de données.csv.

Une fois le répertoire de travail fixé, créer un nouveau fichier (R script) et le sauvegarder sous le nom de Prétraitement.R.

Nous allons commencer par importer les données qui nous intéressent. Pour importer les données nous avons besoin de taper une seule ligne de code en langage R :

```
dataset = read.csv('Données1.csv')
```

Le fichier Données.csv est un fichier contenant les observations sous forme de texte ou les valeurs sont séparées par des virgules. La première observation serait donc représentée dans le fichier par :

France,44,72000,no

Une fois ce script exécuté à l'aide du raccourcis clavier Ctrl+Entrer, les données sont chargées. Vous pouvez visualiser les données dans l'onglet Environnement qui se trouve en haut à droite de l'interface graphique de R Studio. Double cliquer sur la variable dataset permet de l'afficher.

Traiter les données manquantes :

Maintenant nous allons commencer à faire des prétraitements aux données afin d'assurer que la méthode de fouille de données appliquée fonctionne correctement. Le premier problème à quoi on peut faire face est celui de données manquantes. Ce problème est récurrent dans la majorité des données collectées du monde réel. Comme on peut le voir sur la figure 2.1, il y a deux données manquantes (le salaire dans l'observation 5 et l'âge dans l'observation 7). Une première idée serait de supprimer les lignes ayant des données manquantes mais ça serait une perte d'informations qui sont parfois cruciales. Une méthode plus courante utilisée pour faire face au manque de données est de remplacer la donnée manquante par une moyenne (ou bien une médiane). Pour le faire avec le langage R il suffit d'utiliser les lignes de code suivantes :

```
# Prétraitement des données
# Importer la base de données
dataset = read.csv('Données1.csv')
# Traiter les données manquantes
dataset$Age = ifelse(is.na(dataset$Age),
                     ave(dataset$Age, FUN = function(x) mean(x, na.rm = TRUE)),
                     dataset$Age)
dataset$Salaire = ifelse(is.na(dataset$Salaire),
                       ave(dataset$Salaire, FUN = function(x) mean(x, na.rm = TRUE)),
                       dataset$Salaire)
```

dataset$Age: Permet de sélectionner la colonne Age de la base de données.

Ifelse() contient trois paramètres. Le premier est la condition, le second est le paramètre retourné si la condition est vraie. Le troisième paramètre est le paramètre retourné si la condition est fausse.

Is.na est une instruction qui permet de tester si la valeur entrée en paramètre est manquante ou pas.

Pour utiliser la médiane au lieu de la moyenne il suffit de remplacer mean par median.

Une fois le traitement effectué la base de donnée ressemble à celle montrée dans la Figure 2.2.

	Pays	Age	Salaire	Achat
1	France	44.00000	72000.00	Non
2	Espagne	27.00000	48000.00	Oui
3	Allemagne	30.00000	54000.00	Non
4	Espagne	38.00000	61000.00	Non
5	Allemagne	40.00000	63777.78	Oui
6	France	35.00000	58000.00	Oui
7	Espagne	38.77778	52000.00	Non
8	France	48.00000	79000.00	Oui
9	Allemagne	50.00000	83000.00	Non
10	France	37.00000	67000.00	Oui

Figure 2.2. Traitement des données manquantes

Données catégorielles :

Nous pouvons voir dans la base de données présentée en haut qu'il y a deux variables catégorielles (Pays et Achat). Ces variables contiennent des catégories. Les méthodes de fouilles de données sont basées sur des formules mathématiques. Il est facile donc à deviner que les données catégorielles posent problème si leurs valeurs textuelles sont gardées. Ceci nous pousse à trouver le moyen d'encoder le texte des variables catégoriques en nombres.

Pour le faire nous utilisons la fonction Factor du langage R comme suit :

```
# Encoder les données catégorielles
dataset$Pays = factor(dataset$Pays,
                      levels = c('France', 'Esp       agne', 'Allemagne'),
                      labels = c(1, 2, 3))
dataset$Achat = factor(dataset$Achat,
                       levels = c('Non', 'Oui'),
                       labels = c(0, 1))
```

Une fois ce code exécuté en plus du code de traitement des données manquantes, on obtient la base de données montrée dans la Figure 2.3. Les données catégorielles ont été échangées

par des nombres. France a été substituée par le nombre 1. Acheter (oui) a été modifié par le nombre 1.

	Pays	Age	Salaire	Achat
1	1	44.00000	72000.00	0
2	2	27.00000	48000.00	1
3	3	30.00000	54000.00	0
4	2	38.00000	61000.00	0
5	3	40.00000	63777.78	1
6	1	35.00000	58000.00	1
7	2	38.77778	52000.00	0
8	1	48.00000	79000.00	1
9	3	50.00000	83000.00	0
10	1	37.00000	67000.00	1

Figure 2.3 : Encodage données catégorielles

Diviser les données pour l'apprentissage et pour le Test

L'ensemble de la base de données que nous avons utilisé jusqu'à présent contient 10 observations. Ce qu'il devrait être fait pour chaque méthode de fouille de données est de diviser la base de données en deux. Un ensemble d'observations pour l'apprentissage et un autre ensemble pour le test. Le modèle choisi pour la fouille de données va utiliser les données d'apprentissage, trouver les corrélations qui permettent par la suite faire des prédictions. Les données de test seront légèrement différentes de celle utilisées durant l'apprentissage, ce qui va garantir un test adéquat.

Pour diviser la base de données en ensemble d'apprentissage et ensemble de Test, nous devons d'abord importer la librairie « caTools ». Nous pouvons visualiser les librairies déjà installées dans l'oglet « Package ». Si la librairie caTools n'est pas installée il suffit de taper la ligne de code :

```
install.packages('caTools')
```

Une fois installée, nous invoquons la librairie caTool à l'aide de la ligne de code :

library(caTools)

Maintenant nous pouvons diviser la base de données en deux partie en utilisant le code suivant :

```
set.seed(123)
split=sample.split(dataset$Achat, SplitRatio=0.8)
training_Set=subset(dataset,split==TRUE)
test_Set=subset(dataset,split==FALSE)
```

Où « set.seed » permet la génération des mêmes ensembles à chaque exécution.

La fonction « Sample.split » prend en paramètre la variable dépendante ainsi que le ratio des données d'apprentissage. Elle retourne Vrai ou Faux pour chaque observation selon le cas où celle-ci est sélectionnée pour l'apprentissage (Vrai) ou bien pour le Test (Faux). Une fois exécutée, on obtient une nouvelle variable split contenant les valeurs (vrai, faux). Les deux dernières instructions permettent de diviser la base de données chargée dans la variable « dataset » en deux ensembles (Apprentissage et Test) en fonction de la variable « split ».

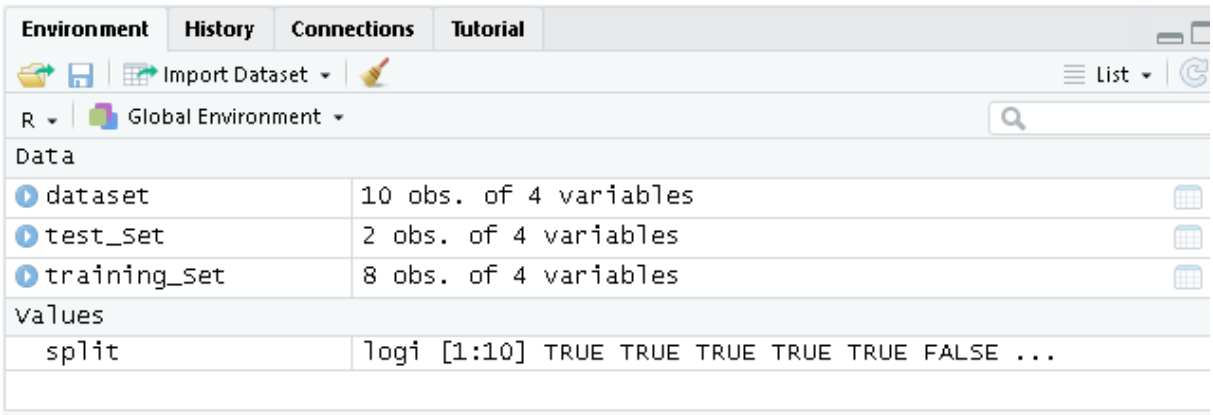


Figure 2.4 : Division de la base de données en deux ensembles (Apprentissage et Test)

Mise à l'échelle :

Si on regarde la base de données utilisée jusque-là, les variables âge et salaire ne sont pas dans la même échelle. Age varie de 27 à 50. Salaire de 48000 jusqu'à 83000. Cette différence d'échelle cause problème l'ors de l'application de nombreux algorithmes de fouille de données. Par exemple, de nombreux algorithmes utilisent la distance Euclidienne (voir Figure ci-dessous). La distance euclidienne si calculée en utilisant l'âge et le salaire sera erronée. La distance Euclidienne sera dominée par le salaire car les valeurs de ce dernier sont grands comparées à celle de l'âge.

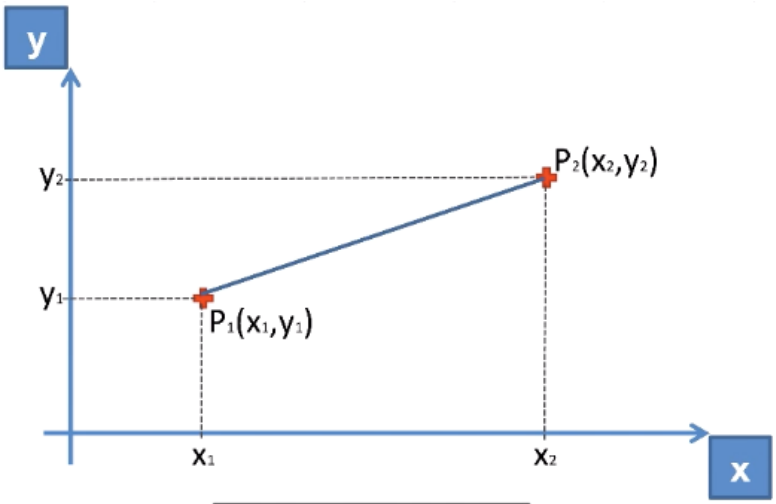


Figure 2.5 : Calcul de la distance Euclidienne

Deux façons communes de mettre à l'échelle les données sont la standarisation et la normalisation telles que décrites par les formules ci-dessous.

Standardisation	Normalisation
$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation } (x)}$	$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$

Afin de mettre à l'échelle les données en langage R, il suffit d'utiliser le code ci-dessous :

```
# Mise à l'echelle
training_Set[,2:3] = scale(training_Set[,2:3])
test_Set[,2:3] = scale(test_Set[,2:3])
```

Filter

	Pays	Age	Salaire	Achat
1	1	0.91654447	0.97382739	0
2	2	-1.57645649	-1.30635381	1
3	3	-1.13651514	-0.73630851	0
4	2	0.03666178	-0.07125566	0
5	3	0.32995601	-0.07125566	1
7	2	0.03666178	-0.92632361	0
8	1	1.50313293	1.63888024	1
10	1	-0.10998534	0.49878964	1

Figure 2.6: Mise à l'échelle de l'ensemble d'entrainement (Training_Set)

Remarque : Une erreur est retournée lorsqu'on essaye de mettre à l'échelle les variables Pays et Achat car ce ne sont pas vraiment des nombres mais des facteurs (à la base variable catégorielle).

Chapitre 3 : La Régression Linéaire

3.1. La Régression Linéaire Simple

Si on revenait à notre exemple de prédiction des prix de maisons en se basant sur leurs superficie, invoqué un peu plus haut dans ce document et montré dans la Figure suivante. On comprend que la régression linéaire permet d'ajuster une ligne droite sur les données que nous avons. Cette ligne droite permettra de faire une prédiction sur la valeur de la variable dépendante d'une observation n'appartenant pas à l'ensemble de données de départ. Grâce à la ligne droite, nous pourrons définir le prix d'une maison en se basant uniquement sur sa surface. Lorsqu'on veut prédire une variable dépendante continue tel que dans l'exemple de vente de maison, on devrait utiliser la régression. Dans cette section nous allons détailler la régression linéaire simple.

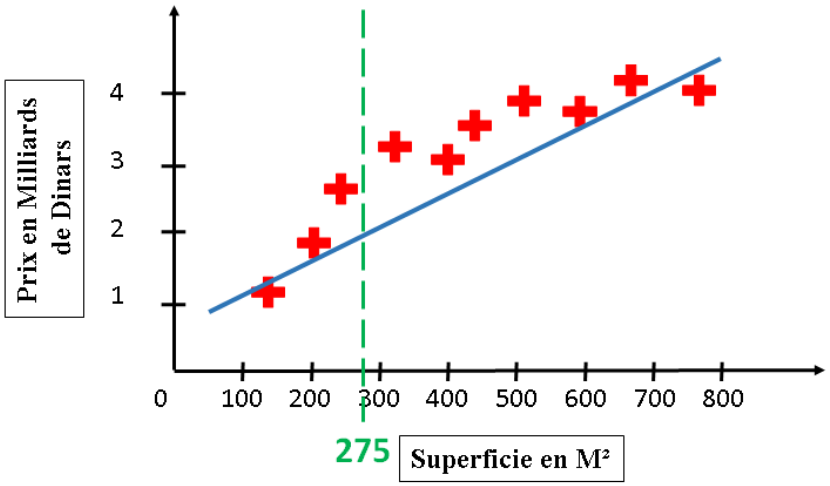


Figure 3.1 : Régression linéaire simple sur l'ensemble de données « vente de maison »

Dans un apprentissage supervisé nous avons un ensemble de données appelé données d'apprentissage. Le travail d'une méthode de fouille de données serait d'apprendre à faire des prédictions en fonctions des données d'apprentissage.

Données d'apprentissage sur les prix de maisons

Superficie en M² (x)	Prix en Milliard de Centimes (y)
110	1,1
170	1,5
220	2,05
270	2,5
350	3,1
…	…

Notation :

m : Nombre d'observations (exemples dans l'ensemble d'apprentissage).

x : variables indépendante (variables d'entrée).

y : variable dépendante (variable cible/de sortie).

(x,y) : un exemple de l'ensemble d'apprentissage (une observation).

(x^i, y^i) : C'est l'ième élément de l'ensemble d'apprentissage.

Nous venons de définir ce qu'est l'ensemble d'apprentissage. Nous allons dans ce qui suit expliquer comment l'apprentissage supervisé fonctionne. Nous commençons par un ensemble d'apprentissage tel que celui des prix de maison et nous appliquons un algorithme d'apprentissage qui va retourner une fonction *h* (hypothèse) dont le rôle est de prendre en entrée la superficie d'une maison et de retourner un prix estimé. *h* est donc une fonction qui lie entre x et y. L'un des problèmes important des algorithmes d'apprentissage est celui de la modélisation de la fonction hypothèse h. Pour cet exemple nous allons supposer que cette fonction est une ligne droite. Ce qui veut dire que nous supposons que y détient une relation linéaire avec la variable x. Dans ce cas la fonction h sera représentée par :

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

Les $\theta_i$sont les paramètres du modèle choisi. Maintenant comment choisir ces paramètres de telle sorte que la ligne résultante représente au mieux l'ensemble de données. La figure suivante montre le comportement de la ligne droite en fonction des paramètres θ_i.

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

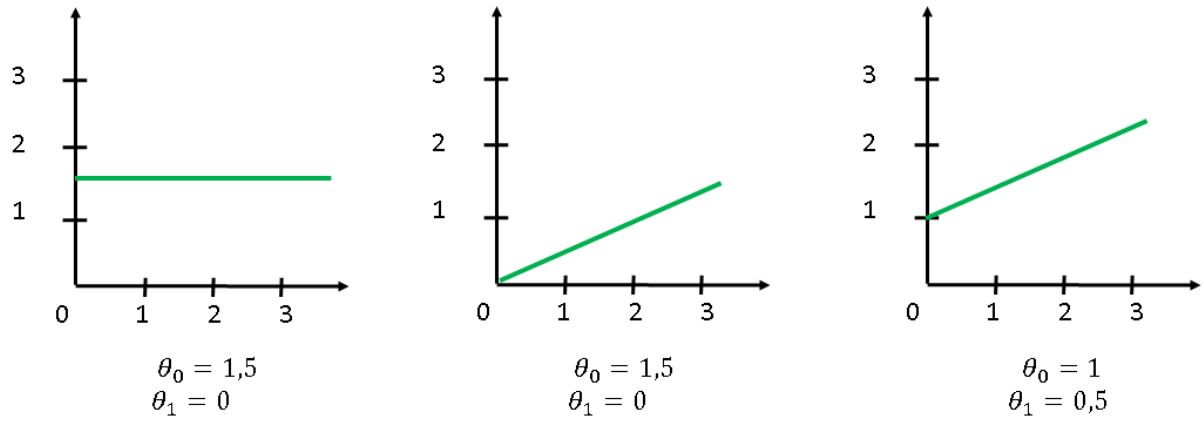


Figure 3.2 : paramètres de la ligne droite

Pour la régression linéaire nous avons un ensemble de données d'apprentissage et nous voulons trouver les paramètres de la ligne droite qui s'ajuste au mieux à ces données. L'idée est de choisir les paramètres θ_i de telle sorte que $h_\theta(x)$ soit proche de y pour les observations de l'apprentissage. En quelque sorte nous voulons minimiser la différence entre $h_\theta(x)$ et y. Ceci peut être formalisé par ce qui suit :

$$\min_{\theta_0,\theta_1} \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$

Nous voulons minimiser la somme des différences entre la prédiction de l'hypothèse pour tout les éléments i ($h_\theta(x^i)$, le prix de la maison estimé) et le prix de la maison effectif dans l'ensemble des observations y^i. Ceci est la fonction objective à minimiser.

$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$ est la fonction de cout. Elle est aussi appelée l'erreur quadratique moyenne. C'est la fonction de cout la plus utilisée pour des problèmes de régression.

Résumons ce que nous avons comme problème :

Nous avons l'hypothèse : $h_\theta(x) = \theta_0 + \theta_1 x$
Les paramètres du modèle : $\theta_0 + \theta_1$ à retrouver
La fonction de cout $J(\theta_0, \theta_1)$
Le but : Minimiser $J(\theta_0, \theta_1)$ en fonction de $\theta_0 + \theta_1$

Supposons que nous avons l'ensemble de données représenté dans la figure suivante par les points rouges. On suppose aussi que le paramètre θ_0 de l'hypothèse à retrouver soit égal à 0. Si nous fixons le paramètre θ_1, $h_\theta(x)$ devient une fonction de x uniquement. Quand θ_1 est égal à 1, la fonction hypothèse, on obtient la ligne droite verte sur la figure à gauche. Ce que nous voulons savoir est la valeur de la fonction de cout $J(\theta_1)$:

$$J(\theta_0) = \frac{1}{2m}\sum_{i=1}^{m}(h_{\theta 1}(x^i) - y^i)^2$$

$$J(\theta_0) = \frac{1}{2m}\sum_{i=1}^{m}(0^2 + 0^2 + 0^2)^2 = 0^2$$

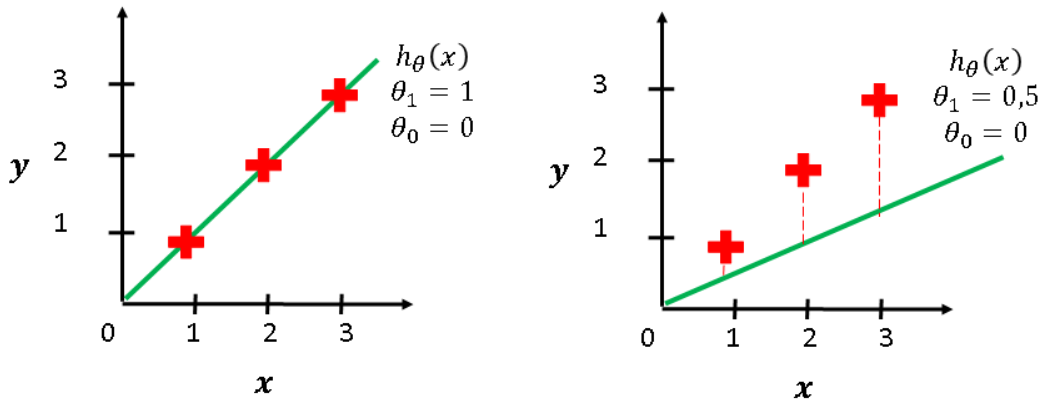


Figure 3.3 : Intuition sur l'Erreur quadratique

Ce qui veut dire que pour le paramètre θ_1, $h_\theta(x^i)$ équivaut exactement à y^i. La fonction de cout pour $\theta_1 = 1$ est égale à 0. Pour $\theta_1 = 0.5$ la fonction de cout sera égale à 0.58. Pour $\theta_1 = 0$ la fonction de cout sera égale à 2.3…etc. Si on affiche la fonction de cout on obtient la figure suivante. L'objectif de la méthode de regression est de choisir les paramètres de telle sorte que la foction de cout soit minimale.

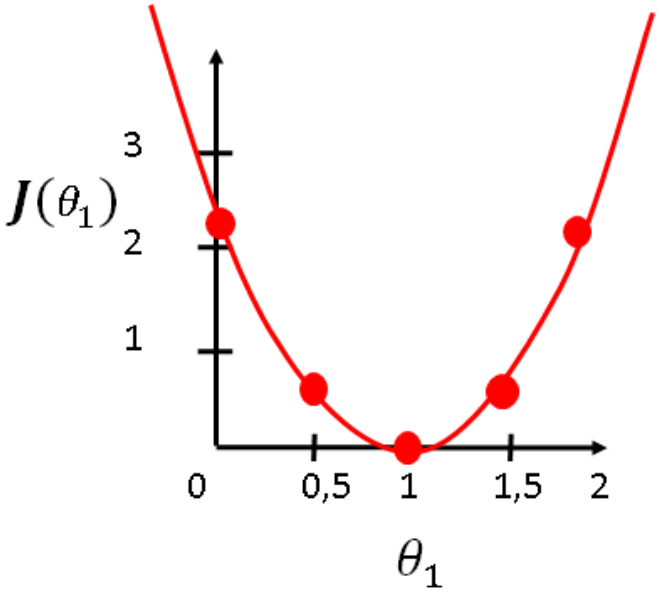


Figure 3.4 : Fonction de cout

Maintenant supposons que nous avons l'ensemble des observations sur la vente de maison. Si on choisi $\theta_1 = 0.06$ et $\theta_0 = 50$ nous allons obtenir la fonction hypothèse $h_\theta(x) = 50 + 0.06x$ qui est dessinée el ligne droite sur le nuage de points. Si on veut dessiner la fonction de cout en fonction des deux paramètres nous allons obtenir l'affichage sur la Figure 3.5. Tant que nous varions les paramètres θ_0 et $\theta_1$nous obtenons différentes valeurs de la fonction de cout *J*. La hauteur de la surface indique la valeur de la fonction de cout. Nous pouvons facilement après avoir affiché cette fonction déduire le point minimun et donc les paramètres qui reflètent la meilleure ligne qui représente les données. Le défi maintenant est celui de pouvoir retrouver le minimum de cette fonction de cout sans à avoir à dessiner la surface de cette fonction.

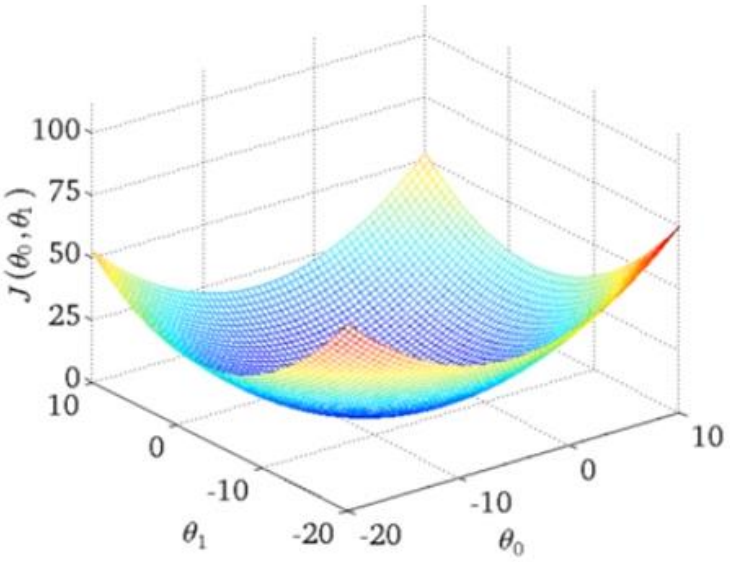


Figure 3.5 : Fonction de cout à deux paramètres

L'algorithme de Gradient Descent est utilisé pour résoudre ce problème de minimisation. Voici le fonctionnement de cet algorithme :

- Commencer avec des paramètres initiaux $\theta_0 \; et \; \theta_1$
- Continuer à changer ces paramètres pour réduire la fonction de cout $J(\theta_0, \theta_1)$ et espérer finir sur un minimum.

Nous supposons que nous somme sur un point d'une montagne et nous voulons descendre le plus vite possible au point le plus bas. Pour le faire nous allons voir autour de nous sur 360° et nous choisissons de faire le premier pas. Une fois dans la deuxième position on refait un autre pas suivant le même résonnement. L'opération continue jusqu'à converger au point le plus bas. C'est ce que fait l'algorithme de gradient descent à travers la formule suivante :

Répéter

{

$$\theta_i := \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1) \; Pour \; J = 0 \; et \; J = 1$$

} Jusqu'à convergence

α étant le taux d'apprentissage. La longueur du pas, Ce taux influence la vitesse par laquelle l'algorithme convergera vers le minimum. Les paramètres doivent être mis à jours simultanément.

Si on avait un seul paramètre à mettre à jour θ_0 le terme de dérivée représente la pente (le gradient) de la fonction J dans le point ayant le paramètre θ_0.

La formule $\theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1)$ permet de rajouter une valeur au paramètre qui est le négatif du gradient. Ceci va permettre une mise à jour du paramètre qui le conduira à réduire la fonction de cout. Si le gradient est positif, ça veut dire que la fonction de cout est ascendante. La logique serait donc de réduire la valeur du paramètre. L'inverse est aussi vrai.

L'algorithme ci-dessous représente la descente de gradient :

Répéter

{

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)\, x^i$$

} Jusqu'à convergence

3.2. Implémentation avec R

Dans ce qui suit nous allons implémenter la régression linéaire simple avec R. La première étape consiste à mettre à jour le répertoire de travail contenant la base de données à traiter. Ensuite nous faisons appel au module de prétraitement de données comme suit :

```
# Régresion linéaire Simple
#Vider l'environnement
remove(list = ls())
# Importer la base de données
dataset = read.csv('Salaires.csv')
# Diviser la base de données (apprentissage/test)
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Salaire, SplitRatio = 2/3)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

	AnnéesExperience	Salaire
1	1.1	39343
2	1.3	46205
3	1.5	37731
4	2.0	43525
5	2.2	39891
6	2.9	56642
7	3.0	60150
8	3.2	54445
9	3.2	64445
10	3.7	57189
11	3.9	63218

La base de données contient des observations sur le nombre d'années d'expérience de 30 employés ainsi que leurs salaires. Nous allons essayer de voir s'il y a une corrélation entre le nombre d'années d'expérience et le salaire. Nous allons voir s'il y a une dépendance linéaire entre la variable dépendante « Salaire » et la variable indépendante « AnnéesExperience ». Une fois la base de données chargée nous divisons celle-ci en un ensemble d'apprentissage et un autre ensemble de test. Nous allons maintenant entrainer notre modèle linéaire sur l'ensemble d'apprentissage. Celui-ci va apprendre la corrélation entre le salaire d'un employé et le nombre d'années d'expérience. Par la suite nous vérifierons la performance de ce modèle sur l'ensemble de Test. Nous aurions pu faire aussi la mise à l'échelle des données mais la librairie que nous allons utiliser pour faire la régression linéaire prend en charge cette étape automatiquement.

Nous allons appeler une nouvelle variable que nous nommerons « regressor » avec la fonction « lm » (fitting linear model) qui est la fonction de la régression linéaire simple(voir le code ci-dessous). Le premier argument est la formule qui doit montrer le rapport variable indépendante avec la variable dépendante. Dans le deuxième argument on met les données.

```
# Ajuster la régression linéaire simple aux données
regressor = lm(formula = Salaire ~ AnnéesExperience, data = training_set)
```

Une fois ce code exécuté le modèle de régression est calculé et la variable regressor devient disponible dans l'environnement globale de RStrudio. Si on veut avoir plus d'information sur le regresseur, la meilleure méthode est de taper dans la console l'instruction

summary(regressor)

ceci affichera des informations sur la régression comme le montre la figure suivante.

```
Call:
lm(formula = Salaire ~ AnnéesExperience, data = training_set)

Residuals:
    Min      1Q  Median      3Q     Max
-7325.1 -3814.4   427.7  3559.7  8884.6

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)         25592       2646   9.672 1.49e-08 ***
AnnéesExperience     9365        421  22.245 1.52e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5391 on 18 degrees of freedom
Multiple R-squared:  0.9649,    Adjusted R-squared:  0.963
F-statistic: 494.8 on 1 and 18 DF,  p-value: 1.524e-14
```

Figure 3.6 : Information sur la régression linéaire simple à l'aide de l'instruction summary(regressor)

Nous avons commencé par charger la base de données ensuite nous avons effectué une phase de prétraitements. Une fois le modèle de régression linéaire estimé, nous pouvons maintenant l'utiliser pour la prédiction en utilisant l'ensemble de Test. Pour cela nous allons appeler le vecteur des prédictions « y_pred ». Celui-ci va contenir les valeurs prédites par notre régression. Ensuite, la fonction « predict » est utilisée comme suit :

```
# Prediction de l'ensemble de Test
y_pred = predict(regressor, newdata = test_set)
```

Le premier argument est notre regression, le deuxième argument représente les données de Test. La fonction « predict » permet de calculer le salaire de ces dix observations à l'aide du modèle entrainé. Pour afficher le vecteur y_pred il suffit d'écrire le nom de la variable dans la console. La vraie valeur du salaire de l'observation 2 est de 46205. Le modèle prédit une valeur de 37766.77. Pour l'observation 4 le vrai salaire est de 435225 et celui prédit est de 44322.33.

```
2          4          5          8       ….
37766.77  44322.33  46195.35  55560.43  …
```

	AnnéesExperience	Salaire
2	1.3	46205
4	2.0	43525
5	2.2	39891
8	3.2	54445
11	3.9	63218
16	4.9	67938
20	6.0	93940
21	6.8	91738
24	8.2	113812
26	9.0	105582

Pour mieux visualiser le résultat nous pouvons afficher des graphes. Pour le faire nous devons commencer par importer la librairie ggplot2 à l'aide du code suivant :

```
# VIisualisation des résultats
Install.packages('ggplot2')
library(ggplot2)
```

Commençons par afficher les observations à l'aide du code ci-dessous. Avant chaque affichage il faut écrire l'instruction : ggplot()+. Le signe + c'est pour séparer les affichages. Ensuite « geom_point() » est l'instruction qui permet d'afficher point par point les observations de la base de données. Il faut préciser les axes x, y à cette instruction. Pour afficher la ligne de régression on utilise linstruction « geom_line() ». Cette fois ci pour l'axe de y il faut mettre les prédictions du modèle de régression sur les données d'apprentissage.

```
# Visualisation des résultats
library(ggplot2)
ggplot() +
geom_point(aes(x = training_set$AnnéesExperience, y = training_set$Salaire),colour = 'red')
+
  geom_line(aes(x = training_set$AnnéesExperience, y = predict(regressor, newdata =
training_set)), colour = 'blue') +
  ggtitle('Salaire  vs Experience (Ensemble d\'Apprentissage)') +
  xlab('Années d\'experience') +
  ylab('Salaire')
```

Une fois le code ci-dessus exécuté on obtient le graphe sur la Figure 3.7. C'est l'ensemble d'observations d'apprentissage avec les valeurs réelles en rouge. La ligne bleue représente la prédiction par la régression linéaire simple.

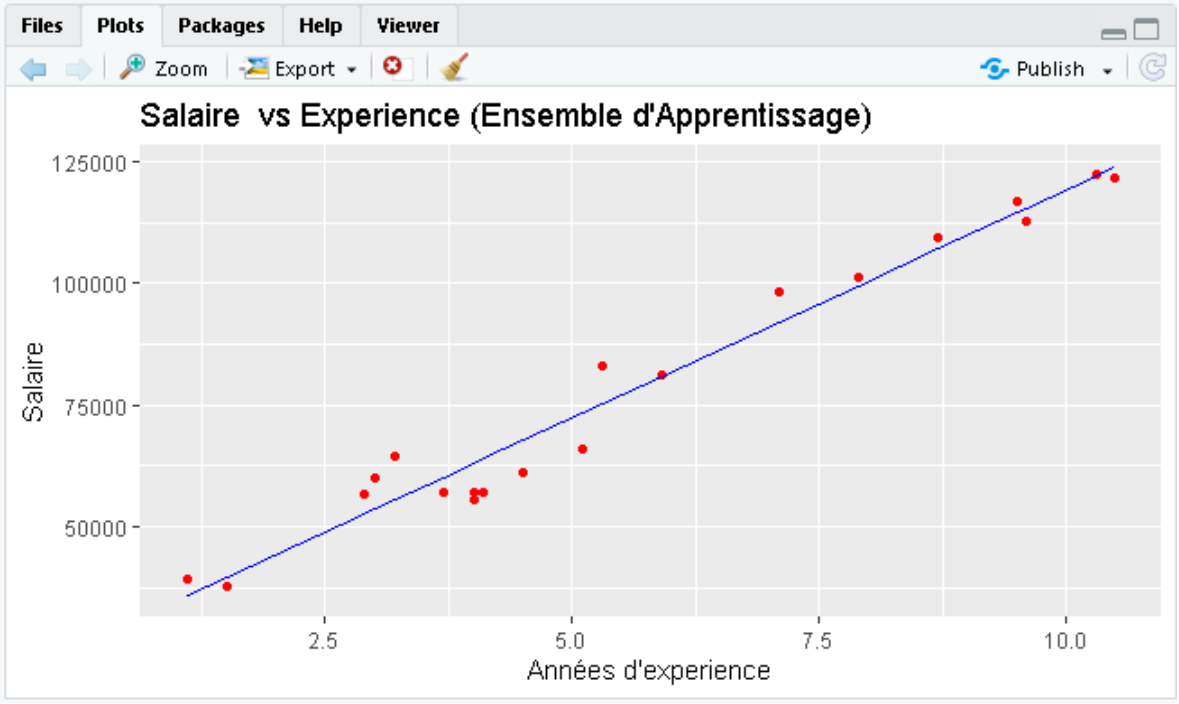


Figure 3.7 : Régression linéaire estimée + observations de l'apprentissage

Nous pouvons voir maintenant comment ce modèle prédit les salaires des employés appartenant à l'ensemble de Test. Le code ci-dessous permet de le faire. Cette fois ci l'ensemble de Test est utilisé en entré pour la fonction geom_point(). Par contre ça ne sert à rien de remplacer l'ensemble d'apprentissage par celui de Test pour la fonction geom_line() car la ligne de régression restera la même en fonction des données en entrée. Nous pouvons voir dans la figure 3.8 que la ligne droite proposée par la méthode de régression représente bien les données de Test. On peut voir sur quelques points que la prédiction est très précise, un peu moins pour d'autres. Ceci est du au fait que le salaire n'est pas corrélé à 100% avec la variable années d'expérience.

```
# Visualiser la prédiction sur l'ensemble de Test
library(ggplot2)
ggplot() +
  geom_point(aes(x = test_set$AnnéesExperience, y = test_set$Salaire),  colour = 'red') +
  geom_line(aes(x = training_set$AnnéesExperience, y = predict(regressor, newdata = training_set)), colour = 'blue') +
  ggtitle('Salaire vs Experience (Ensemble de Test)') +
  xlab('Années d\'experience') +
  ylab('Salaire')
```

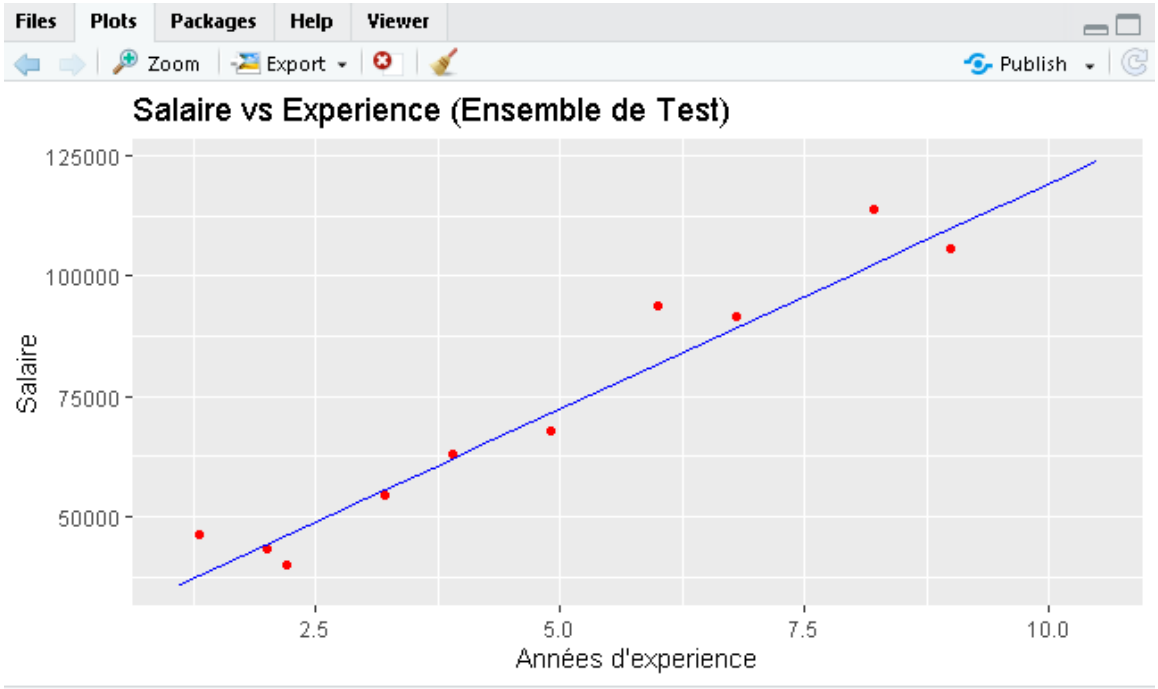


Figure 3.8 : Régression linéaire estimée + Test Set

3.3. La Régression Linéaire Multiple

Dans cette section nous allons parler d'une autre version de la régression linéaire. Celle-ci prend en charge plusieurs variables indépendantes au lieu d'une seule. Dans la première version de la régression simple nous avions comme variable dépendante la surface de la maison et nous voulions l'utiliser pour prédire son prix. La forme de l'hypothèse était une fonction d'une ligne droite. Pour la régression linaire multiple nous avons non seulement la surface de la maison comme variable dépendante mais aussi le nombre de chambres, nombre d'étages, l'âge de la maison qui sont aussi des variables qui vont aider à prédire le prix exacte d'une maison.

Superficie en M² (x)	Nombre Pièces	Nombre étages	Age de la Maison	Prix en Milliard de Centimes (y)
110	2	1	36	1,1
170	3	2	30	1,5
220	3	2	30	2,05
270	5	2	40	2,5
350	6	1	45	3,1
…	…	…	…	…

Figure 3.9 : Variables indépendantes multiples

Nous allons utiliser dans ce qui suit les notations x_1, x_2, x_3, x_4 pour les variables indépendantes et y pour la variable dépendante à prédire. Aussi :
n : est le nombre de variables (dans ce cas-ci il est égal à 4)

x^i : Représente un vecteur de 4 dimensions, c'est le ième élement de la base de données contenant les valeurs pour les variables indépendantes.

x_j^i : Représente la valeur de la variable j dans le ième élément de la base de données.

Maintenant que nous avons plusieurs variables indépendantes, la fonction hypothèse de la régression devrait ressembler à :

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

Si on définit x_0=1, nous pouvons écrire :

$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

D'une autre manière :

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \cdot\cdot \\ \theta_{n+1} \end{bmatrix}, x = \begin{bmatrix} x_0 \\ x_1 \\ \cdot\cdot \\ x_{n+1} \end{bmatrix} \text{ et } h_\theta(x) = \theta^T x$$

Dans ce qui suit nous allons discuter comment estimer les paramètres de cette fonction hypothèse. En d'autres termes, comment appliquer la descente de gradient pour ajuster le modèle de régression multiple aux données.

Pour résumer notre notation:

Hypothèse : $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Paramètres : $\theta_0, \theta_1, \theta_2, \ldots, \theta_n$

Fonction de cout :

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$

Descente de gradient :

Répéter {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_i} J(\theta_j)$$

} Jusqu'à convergence

Après les dérivations on obtient l'algorithme de descente de gradient suivant :

Répéter {

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

} Jusqu'à convergence Avec j=0,…n.

3.4. Implémentation avec R

Dans cette section nous allons voir comment implémenter la régression linéaire multiple à l'aide du langage R. Pour commencer, il faut mettre à jour le répertoire de travail de RStudio sur le dossier contenant la base de données. Ensuite, commencer par la phase une qui est celle de la préparation des données (prétraitement) pour la régression linéaire multiple. Pour cela on se base sur les démarches de prétraitement vu jusque-là. Nous commençons par charger la base de donnée avec :

```
# Régression linéaire Multiple
# Importer la base de données
dataset = read.csv('50_Startups.csv')
```

Cette base de données contient des informations sur 50 startups. Les valeurs des variables sont des montants d'argent dépensés dans la recherche et le développement, l'administration, le marketing et la ville ou la startup opère. Finalement nous avons la variable dépendante « Profit » qui est la variable que nous voulons prédire grâce au modèle linéaire multiple en se basant sur les variables indépendantes. Cette prédiction permettra à des futurs investisseurs de savoir sur quelle startup investir.

	R.D	Administration	Marketing	Ville	Profit
1	165349.20	136897.80	471784.10	Alger	192261.83
2	162597.70	151377.59	443898.53	Oran	191792.06
3	153441.51	101145.55	407934.54	Annaba	191050.39
4	144372.41	118671.85	383199.62	Alger	182901.99
5	142107.34	91391.77	366168.42	Annaba	166187.94
6	131876.90	99814.71	362861.36	Alger	156991.12
7	134615.46	147198.87	127716.82	Oran	156122.51
8	130298.13	145530.06	323876.68	Annaba	155752.60
9	120542.52	148718.95	311613.29	Alger	152211.77
10	123334.88	108679.17	304981.62	Oran	149759.96

Figure 3.10 : Base de données Startups

Dans cette base de données il faut faire attention à la variable « Ville » qui est une variable catégorielle. Celle-ci peut causer un problème à l'algorithme d'apprentissage. Ce que nous devons faire est donc d'encoder cette variable tel que vu dans la section des prétraitements en utilisant le code ci-dessous, qui une fois exécuté retourne l'ensemble de données qu'on peut voir dans la Figure 3.11. Les dénominations des villes deviennent maintenant sous forme de nombre (1,2,3 pour Alger, Oran, Annaba respectivement).

```
# Encoder les données catégorielles
dataset$Ville = factor(dataset$Ville,
                       levels = c('Alger', 'Oran', 'Annaba'),
                       labels = c(1, 2, 3))
```

Filter

	R.D	Administration	Marketing	Ville	Profit
1	165349.20	136897.80	471784.10	1	192261.83
2	162597.70	151377.59	443898.53	2	191792.06
3	153441.51	101145.55	407934.54	3	191050.39
4	144372.41	118671.85	383199.62	1	182901.99
5	142107.34	91391.77	366168.42	3	166187.94
6	131876.90	99814.71	362861.36	1	156991.12
7	134615.46	147198.87	127716.82	2	156122.51
8	130298.13	145530.06	323876.68	3	155752.60
9	120542.52	148718.95	311613.29	1	152211.77
10	123334.88	108679.17	304981.62	2	149759.96

Figure 3.11 : Encodage de la variable catégorielle Ville

Une dernière chose à faire en termes de prétraitement est de diviser l'ensemble de données en deux. L'ensemble d'apprentissage et l'ensemble de test.

```
# Diviser les données en données d'Apprentissage/Test
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Profit, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

Nous n'avons pas besoin de mettre à l'échelle manuellement les données car ça sera fait à l'aide de l'implémentation de la régression multiple sous R.

```
# Mise à l'echelle
# training_set = scale(training_set)
# test_set = scale(test_set)
```

Nous sommes prêts maintenant à ajuster le modèle linéaire multiple sur les données. Pour la régression linéaire simple la variable dépendante était liée uniquement à une seule variable dépendante. Dans le cas de la régression multiple la variable dépendante « profit » sera proportionnelle à quatre variables indépendantes. La formule de régression à mettre comme paramètre à la fonction lm serait donc «~R.D + Administration + Marketing + Ville ». Une autre méthode pour écrire cette formule quand la variable à prédire dépend de toutes les variables indépendante ressemble à ce qui suit :

```
# Ajuster la régression linéaire Multiples aux données d'apprentissage
regressor = lm(formula = Profit ~ ., data = training_set)
```

Une tâche importante maintenant est celle d'analyser les informations de la régression obtenue. Pour le faire il suffit de taper sur la consôle 'summary(regressor)' et on obtient les informations affichées dans la Figure. Dans la partie coefficients, nous pouvons voir des informations du modèle concernant les variables indépendantes. Vous pouvez remarquer que la variable « Ville » est remplacée par les deux variables muettes Ville 2, Ville 3. Pour chaque variable nous avons le coefficient, l'erreur standard, valeur t et valeur p et le niveau de signification. Les informations les plus importantes sont les deux dernières colonnes, la valeur P et le niveau de signification. Ces deux informations nous donne une idée sur la dépendance de la variable à prédire de la variable indépendante. Une petite valeur P témoigne d'une plus grande influence de la variable indépendante sur la variable à prédire. Généralement, un seuil de 5% est utilisé pour définir si la variable est statistiquement signifiante ou pas. La dernière colonne est une méthode rapide d'interprétation des coefficients. L'interprétation de ces données est comme suit : La variable Recherche et Développement est la seule variable qui a une influence sur le profit de la startup. Les autres variables n'ont aucune influence sur la variable à prédire. Cela veut dire que nous pouvons transformer le problème de régression linéaire multiple en un problème de régression linéaire simple avec une seule variable indépendante qui est la recherche et le développement.

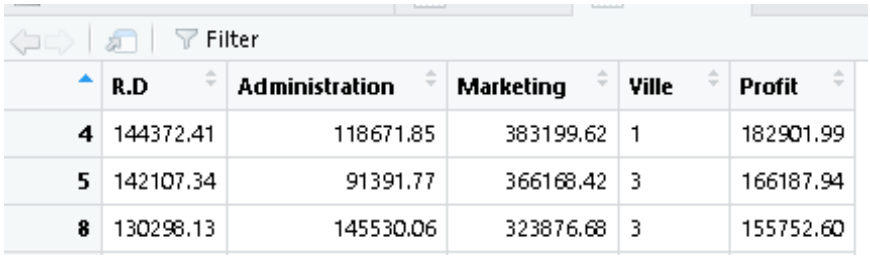
```
Call:
lm(formula = Profit ~ ., data = training_set)

Residuals:
   Min     1Q Median     3Q    Max
-33128  -4865      5   6098  18065

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)     4.965e+04  7.637e+03   6.501 1.94e-07 ***
R.D             7.986e-01  5.604e-02  14.251 6.70e-16 ***
Administration -2.942e-02  5.828e-02  -0.505    0.617
Marketing       3.268e-02  2.127e-02   1.537    0.134
Ville2          1.213e+02  3.751e+03   0.032    0.974
Ville3          2.376e+02  4.127e+03   0.058    0.954
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 9908 on 34 degrees of freedom
Multiple R-squared:  0.9499,    Adjusted R-squared:  0.9425
F-statistic:   129 on 5 and 34 DF,  p-value: < 2.2e-16
```

Figure 3.12: Les informations du modèle de régression retourné par le programme R

Pour prédire le profit des startups de Test nous utilisons le code :

```
# Prédiction de l'ensemble de Test
y_pred = predict(regressor, newdata = test_set)
```

Pour afficher les valeurs prédites par le modèle de régression estimé il suffit de taper sur la console ‘y_pred’. Les valeurs de tests et ceux de la prédiction sont affichés ci-dessous.
Trois valeurs de Test :



Filter

	R.D	Administration	Marketing	Ville	Profit
4	144372.41	118671.85	383199.62	1	182901.99
5	142107.34	91391.77	366168.42	3	166187.94
8	130298.13	145530.06	323876.68	3	155752.60

Les trois valeurs prédites pour les mêmes observations :

4	5	8
173981.09	172655.64	160250.02

Elimination rétroactive (Backward Elimination) :

Nous avons déjà mis en œuvre un modèle de régression multiple que nous avons ajusté aux données d’apprentissage. Mais est-ce que c’est le modèle optimal que nous pouvons créer à l’aide de ces données ? Lorsque nous avons construit ce modèle, nous avons utilisé toutes les variables indépendantes. Parmi ces variables indépendantes, certaines sont hautement significatives et ont un impact important sur la variable dépendante. D’autres ne le sont pas du tout. Cela signifie que si nous retirions les variables non significatives du modèle, nous aurons des prédictions encore plus justes.

L'objectif de l’élimination rétroactive est de trouver l’ensemble optimale de variables indépendantes, de sorte que chaque variable indépendante de cet ensemble ait un impact important sur la variable dépendante. Pour le faire, nous allons simplement nous baser sur le modèle que nous avons déjà créé. La première étape consiste à prendre le modèle de régression multiple et le réécrire sous la forme suivante:

regresseur = lm(formula = Profit ~ R.D + Administration + Marketing + Ville, data = training_set)

Comme ceci, il représente toutes les variables indépendantes (l'espace doit être remplacé par un point), séparées par le signe plus. Le principe de l'élimination rétroactive et d'enlever de cette équation les variables non significative une par une. L'ajustement du modèle peut se faire sur l'ensemble de données totale, non pas sur l'ensemble d'apprentissage uniquement, afin d'avoir une meilleure évaluation des variables indépendantes.

Maintenant que c'est prêt, nous sommes prêts à entamer les premières étapes de notre élimination rétroactive. On va utiliser la fonction summary(regresseur). Mais avant cela rappelons les cinq étapes de l'algorithme de l'élimination rétroactive, qu'on peut voir sur la Figure suivante.

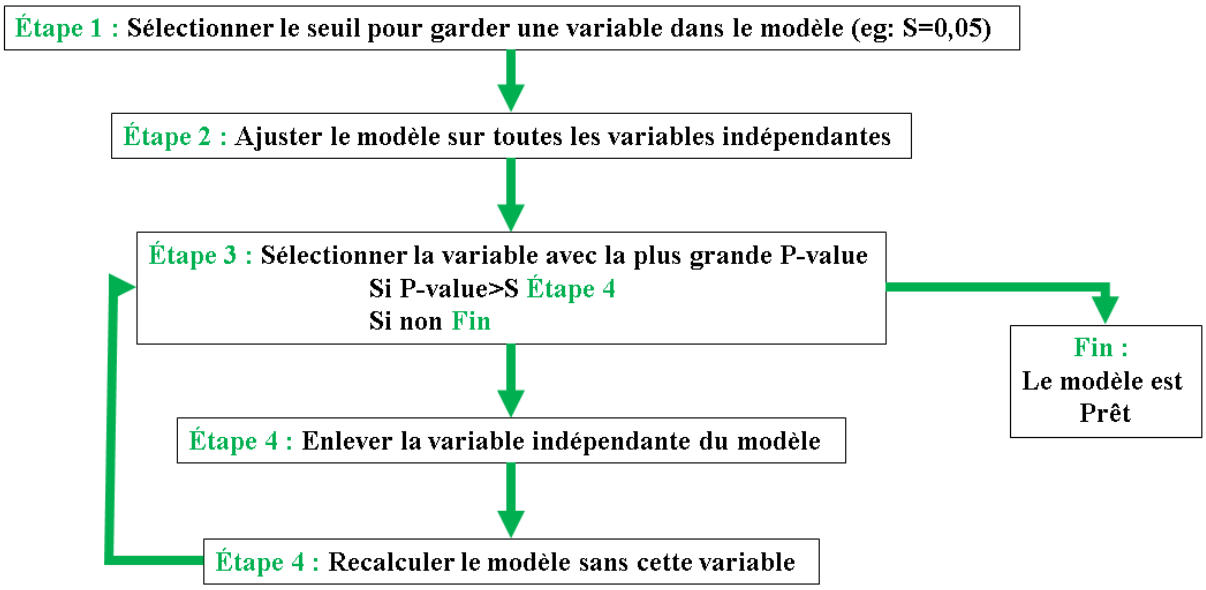


Figure 3.13 : Elimination rétroactive (Backward Elimination)

La première étape consiste à sélectionner le niveau de signification qui constitue un seuil pour notre valeur P, de sorte que si la valeur p d'une variable indépendante est inférieure au niveau de signification, cette variable indépendante reste dans le modèle. Si la valeur p de la variable indépendante est supérieure au niveau de signification, elle ne restera pas dans le

modèle. Le seuil est souvent définit avec la valeur 5% (0. 05). La deuxième étape consiste à calculer le modèle en prenant en considération à tous les prédicteurs possibles (variables indépendantes). La troisième étape consiste à examiner à l'aide de la fonction summary(regresseur) la variable qui a la valeur P la plus élevée. Si la valeur p est supérieure au niveau de signification définit, on passe à l'étape 4. Si ce n'est pas le cas, notre modèle est prêt. Supposons que nous avons trouvé la valeur la plus élevée supérieure au seuil de signification de 5%. Il faut donc supprimer la viable indépendant puis passer à la cinquième étape, qui consiste à ajuster le modèle sans cette viable.

On peut voir ci-dessous le résultat de l'ajustement du modèle de régression multiple sur la base de données « 50_Startups ».

```
Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)     5.008e+04  6.953e+03   7.204 5.76e-09 ***
R.D             8.060e-01  4.641e-02  17.369  < 2e-16 ***
Administration -2.700e-02  5.223e-02  -0.517    0.608
Marketing       2.698e-02  1.714e-02   1.574    0.123
Ville2          4.189e+01  3.256e+03   0.013    0.990
Ville3          2.407e+02  3.339e+03   0.072    0.943
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Nous pouvons voir que les dépenses en R&D sont très importantes sur le plan statistique, mais le reste ne semble pas être important. L'élimination rétroactive doit être effectuée étape par étape, car en supprimant certaines variables indépendantes, la signification d'autres variables peut changer considérablement tel que pour la variable Marketing :

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 4.698e+04  2.690e+03  17.464   <2e-16 ***
R.D         7.966e-01  4.135e-02  19.266   <2e-16 ***
Marketing   2.991e-02  1.552e-02   1.927     0.06 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Chapitre 4 : La Classification

4.1. La Régression Logistique

Dans cette section nous développons la régression logistique qui est une méthode largement utilisée pour la classification. Comme décrit dans l'introduction, la classification permet par exemple de dire si une tumeur est maline ou bénigne. Cela veut dire que nous voulons prédire la variable y pouvant être représentée par deux valeurs : $y = \{0,1\}$. Où 0 est la première classe (tumeur bégnine), 1 est la deuxième classe (Tumeur Maline). Ci-dessous un exemple de base d'apprentissage sur laquelle nous voulons nous baser pour faire de la classification si une tumeur est maligne ou bégnine.

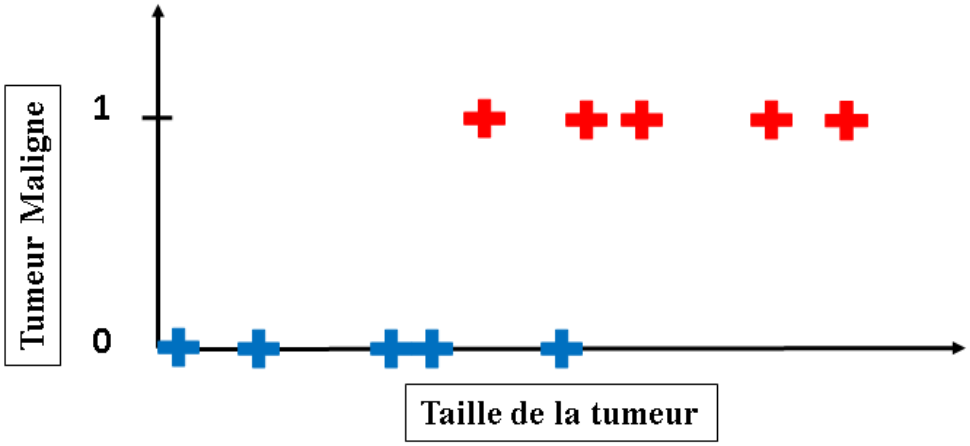


Figure 4.1 : Exemple base d'apprentissage de classification

Précédemment nous avons vu que le modèle de régression linéaire simple était sous la forme $h_\theta(x) = \theta^T x$. Ce modèle n'est plus adapté à la classification. La régression logistique a besoin d'un modèle $h_\theta(x)$ qui retourne une estimation entre 0 et 1 (On veut que $0 \leq h_\theta(x) \leq 1$). Pour cela la fonction Sigmoide (logistique) est combiné au modèle précèdent comme suit :

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

Comme le montre la Figure suivante, la fonction sigmoide traduit la valeur de l'ancien modèle de régression linéaire en valeurs comprises en 0 et 1. L'idée de la régression logistique et donc d'utiliser ce nouveau modèle afin de faire la classification. Il s'agit de faire l'estimation des paramètres θ à l'aide de l'ensemble d'apprentissage. Nous discuterons dans ce qui suit l'algorithme permettant de retrouver ces paramètres. Mais avant cela interprétons l'hypothèse de la régression logistique. La valeur retournée par l'hypothèse de la régression logistique est considérée comme probabilité que y=1 pour une donnée x. Si la valeur retournée est de 0.8 par exemple, on pourra l'interpréter comme étant le patient à 80% de chance d'avoir une tumeur maligne. Ceci peut être écrit de la forme :

$P(y = 1|x; \theta)$

Où : $P(y = 1|x; \theta) + P(y = 0|x; \theta) = 1$

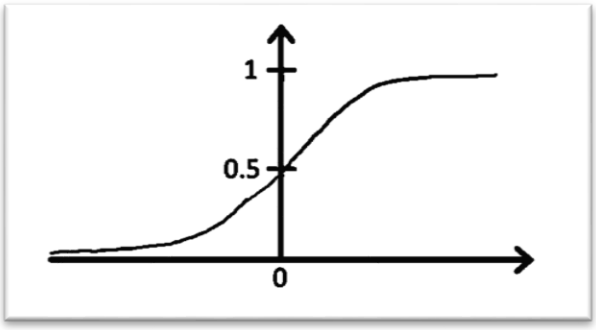


Figure 4.2 : La fonction Sigmoide

La prédiction doit avoir la valeur de 0 ou bien 1. On prédit que :

- $y = 1 \; si \; h_\theta(x) \geq 0{,}5$
- $y = 0 \; si \; h_\theta(x) < 0{,}5$

Nous avons :

- $h_\theta(x) \geq 0{,}5 \; si \; \theta^T x \geq 0$
- $h_\theta(x) < 0{,}5 \; si \; \theta^T x < 0$

Supposons que nous avons $h_\theta(x) = \frac{1}{1+e^{\theta_0+\theta_1 x_1+\theta_2 x_2}}$

Supposons qu'une fois les paramètres estimés nous obtenons : $\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$. Nous avons donc :

- $y = 1\ si\ \theta^T x \geq 0$ $\qquad\qquad$ $y = 1\ si\ -3 + x_1 + x_2 \geq 0$
- $y = 0\ si\ \theta^T x < 0$ $\qquad\qquad$ $y = 0\ si\ -3 + x_1 + x_2 < 0$

Toute donnée ayant les valeurs (x_1, x_2) vérifiant $-3 + x_1 + x_2 \geq 0$ sera donc classée dans la classe 1.

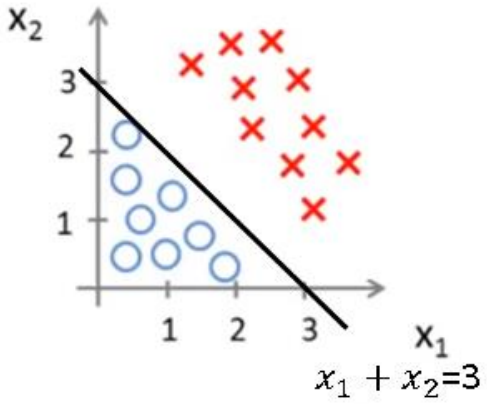


Figure 4.3 : Limite de décision

Il est possible que la limite de décision soit non linéaire tel que montré dans l'exemple ci-dessous :

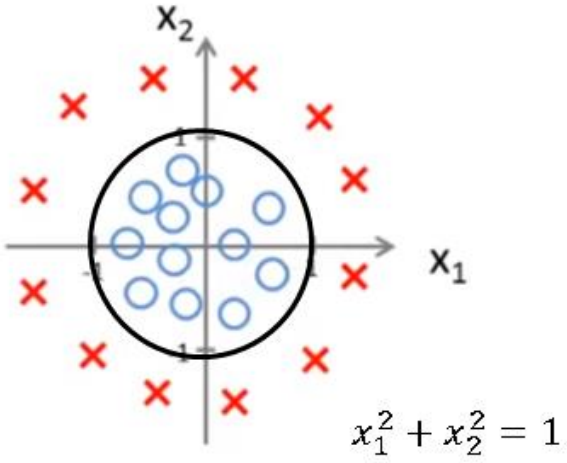


Figure 4.4 : Limite de décision non linéaire

Dans ce cas l'hypothèse ressemble à ce qui suit :

$$h_\theta(x) = \frac{1}{1 + e^{\theta_0+\theta_1 x_1+\theta_2 x_2+\theta_1 x_1^2+\theta_2 x_2^2}}$$

Les paramètres retrouvé après l'ajustement du modèle sur les données sont :

$$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

La classification sera donc faite à l'aide des règles suivantes :

- $y = 1\ si\ -1 + x_1^2 + x_2^2 \geq 0$
- $y = 0\ si - 1 + x_1^2 + x_2^2 < 0$

L'estimation du modèle de régression linéaire peut être effectué à l'aide de l'algorithme de descente de gradient où la fonction de cout est définie comme suit :

$$Cost(h_\theta(x), y) = \begin{cases} -\log\big(h_\theta(x)\big)\ if\ y = 1 \\ -\log\big(1 - h_\theta(x)\big)\ if\ y = 0 \end{cases}$$

La fonction à minimiser devient :

$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\big(h_\theta\big(x^i\big), y^i\big)$

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} y^i \log\Big(h_\theta\big(x^i\big)\Big) + (1 - y^i)\log\Big(1 - h_\theta\big(x^i\big)\Big)$$

4.2. La Régression Logistique avec R

Dans cette section nous allons implémenter la régression logistique sur une base de données contenant des informations sur des utilisateurs d'un réseau social. La méthode de fouille de donnée va permettre au réseau social de connaitre les utilisateurs qui seront ciblés par la publicité.

Nous commençons d'abord par importer la base de données à l'aide de l'instruction :

```
# Importer la base de données
dataset = read.csv('RéseauxSociaux.csv')
```

Cette base de données contient des informations sur des utilisateurs du réseau social. Ces informations sont l'identificateur de l'utilisateur, le genre, l'Age et le salaire estimé. La dernière variable signifie si l'utilisateur a acheté un produit qui a été précédemment publié dans le réseau sociale ou pas.

Le travail qui va suivre consiste à implémenter la régression logistique afin de comprendre la corrélation entre la variable dépendante Achat et les autres variables indépendantes.

	ID.Utilisateur	Genre	Age	Salaire.Estimé	Achat
1	15624510	Homme	19	19000	0
2	15810944	Homme	35	20000	0
3	15668575	Femme	26	43000	0
4	15603246	Femme	27	57000	0
5	15804002	Homme	19	76000	0
6	15728773	Homme	27	58000	0
7	15598044	Femme	27	84000	0
8	15694829	Femme	32	150000	1
9	15600575	Homme	25	33000	0
10	15727311	Femme	35	65000	0
11	15570769	Femme	26	80000	0

Figure 4.5 : Base de données Réseau Social

Nous pouvons sélectionner les variables indépendantes avec lesquelles nous voulons faire l'apprentissage à l'aide de l'instruction : dataset = dataset[3:5]. Dans ce cas, uniquement l'age et le salaire seront utilisé pour faire la prédiction.

La prochaine étape consiste à diviser la base de données en base d'apprentissage et base de test. Pour le faire on utilise le code suivant :

```
# Diviser la base de données en Apprentissage/Test
# install.packages('caTools')
library(caTools)
set.seed(123)
split = sample.split(dataset$Achat, SplitRatio = 0.75)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)
```

Par la suite nous allons mettre en echelle les données à l'aide du code:

```
# Mise à l'echelle
training_set[1:2] = scale(training_set[1:2])
test_set[1:2] = scale(test_set[1:2])
```

Une fois la mise à l'échelle éffectuée on obtient une base d'apprentissage qui ressemble au tableau dans la figure suivante.

	Age	Salaire.Estimé	Achat
1	-1.76554750	-1.47334137	0
3	-1.09629664	-0.78837605	0
6	-1.00068938	-0.36027273	0
7	-1.00068938	0.38177303	0
8	-0.52265305	2.26542765	1
10	-0.23583125	-0.16049118	0
11	-1.09629664	0.26761214	0
13	-1.66994024	0.43885347	0
14	-0.52265305	-1.50188159	0
15	-1.86115477	0.32469259	0
16	-0.80947485	0.26761214	0

Figure 4.6 : Base d'apprentissage après sélection de variable et mise à l'échelle

A présent nous avons effectué la phase de prétraitement, nous pouvons passer maintenant à l'estimation du modèle de régression logistique :

```
# Ajuster la régression logistique aux données d'apprentissage
classifieur = glm(formula = Achat ~ .,
                  family = binomial,
                  data = training_set)
```

Le premier argument de la fonction glm est la formule. On mentionne la variable dépendante Achat et comme on l'a fait pour la régression multi variable, on utilise le point pour prendre en compte les variables durant l'apprentissage. Pour la régression logistique il faut choisir la valeur binomial dans le paramètre family. En fin, mettre la base d'apprentissage. Ce code permet de construire le modèle de régression logistique.
Une fois le modèle estimé, nous pouvons l'utiliser pour la prédiction comme suit :

```
# Prédiciton de l'ensemble de Test
prob_pred = predict(classifieur, type = 'response', newdata = test_set[-3])
```

Si on tape prob_pred dans la console on obtient les probabilités de chaque élément de l'ensemble de Test.

```
           2            4            5            9           12           18
0.0162395375 0.0117148379 0.0037846461 0.0024527456 0.0073339436 0.2061576580
```

Le modèle a prédit pour l'élément 2 une probabilité de l'achat du produit égale à 0.016. L'utilisateur a donc de très mince chance d'effectuer l'achat. Pour avoir les résultats affichés sous forme de 0 et de 1 on peut faire la conversion suivante :

```
y_pred = ifelse(prob_pred > 0.5, 1, 0)
```

```
 2  4  5  9 12 18 19 20 22 29 32 34 35 38 45 46 48 52 66 69
 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0
```

Afin d'évaluer la qualité de la prédiction nous pouvons calculer la matrice de confusion comme suit:

```
# Construire la Matrice de Confusion
cm = table(test_set[, 3], y_pred > 0.5)
```

La matrice obtenue est :

	FALSE	TRUE
0	57	7
1	10	26

Les valeurs 57 et 26 sont les prédictions correctes. Les valeurs 7 et 10 sont les prédictions incorrectes. Ce qui est intéressant à savoir est que le modèle estimé a donné des résultats acceptables.

Dans ce qui suit nous allons visualiser les résultats grâce à des graphes. Afin d'afficher les prédictions du modèle de régression logistique estimé jusqu'à présent sur la base d'apprentissage nous pouvons utiliser le code suivant :

```
# Visualisation des résultats sur la base d'apprentissage
library(ElemStatLearn)
set = training_set
X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
grid_set = expand.grid(X1, X2)
colnames(grid_set) = c('Age', 'Salaire')
prob_set = predict(classifieur, type = 'response', newdata = grid_set)
y_grid = ifelse(prob_set > 0.5, 1, 0)
plot(set[, -3],
```

```
    main = 'Regression Logistique (Training set)',
    xlab = 'Age', ylab = 'Salaire Estimé',
    xlim = range(X1), ylim = range(X2))
contour(X1, X2, matrix(as.numeric(y_grid), length(X1), length(X2)), add = TRUE)
points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'springgreen3', 'tomato'))
points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

Figure 4.7: Régression logistique estimée

Les points dans le graphes représentent les utilisateurs du réseau social présents dans l'ensemble d'apprentissage. Ces utilisateurs sont caractérisés par l'age dans l'axe des X et par le salaire dans l'axe des Y. Nous avons des points rouges et des points verts. Les points rouges représentent les utilisateurs n'ayant pas acheté de produit contrairement aux points verts. L'objectif de la régression logistique est de représenté les utilisateurs dans leurs bonnes catégories (classes). Le classifieur est représenté dans le graphe par des régions de prédiction, les deux régions rouge et verte. Pour chaque utilisateur présent dans la zone rouge, la prédiction serait la classe 0, le non achat du produit. Pour les utilisateurs de la zone verte, la

prédiction serait la classe 1, l'achat du produit. Cette connaissance que l'on vient d'extraire à partir des données peut être utilisée par le réseau social afin de cibler le publique avec la publicité. Dans ce cas, la publicité serait affichée uniquement aux utilisateurs ayant les caractéristiques de la région verte.

4.3. Arbres de Décision

Une autre méthode de classification très commune est celle des arbres de décision. Un arbre de décision est une structure de type organigramme dans laquelle chaque nœud interne représente un Test sur un attribut (voir la figure ci-dessous). Chaque branche représente le résultat du test, et chaque nœud feuille représente une étiquette de classe (décision prise après le calcul de tous les attributs). Les chemins de la racine à la feuille représentent les règles de classification.

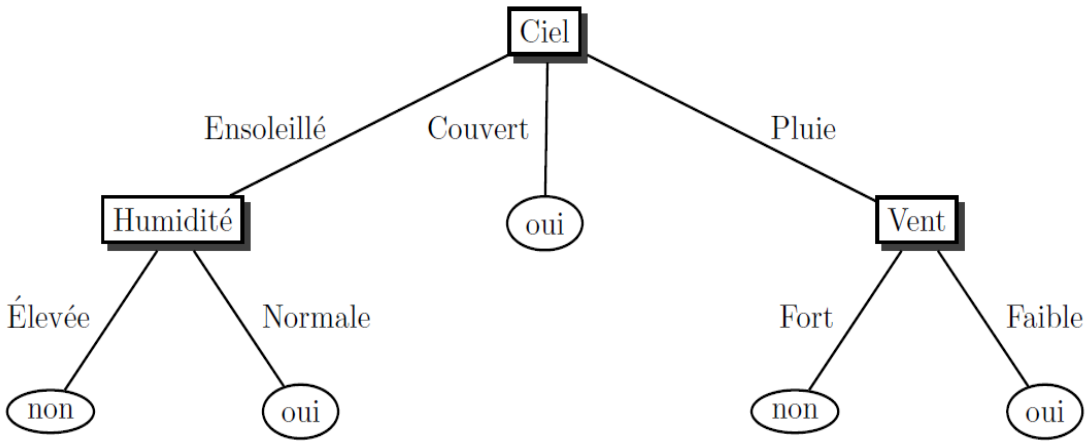


Figure 4.8 : Arbre de décision

Il existe plusieurs algorithmes permettant de construire un arbre de décision à partir d'un ensemble de données. Les plus communs sont les algorithmes ID3 ou bien C4.5. Dans cette section nous allons détailler l'algorithme ID3 pour la conception d'arbre de décision. Le tableau ci-dessous représente un ensemble de données sur le jeu de tennis. Il s'agit de 14 observations ayant quatre attributs et une variable dépendante. Nous voulons prédire si la personne va jouer au tennis en partant des conditions météorologiques (Ciel, Température, Humidité, Vent).

	Ciel	Température	Humidité	Vent	Jouer au Tennis
1	Ensoleillé	Chaude	Elevée	Faible	Non
2	Ensoleillé	Chaude	Elevée	Fort	Non
3	Couvert	Chaude	Elevée	Faible	Oui
4	Pluie	Tiède	Elevée	Faible	Oui
5	Pluie	Fraiche	Normale	Faible	Oui
6	Pluie	Fraiche	Normale	Fort	Non
7	Couvert	Fraiche	Normale	Faible	Oui
8	Ensoleillé	Tiède	Elevée	Faible	Non
9	Ensoleillé	Fraiche	Normale	Faible	Oui
10	Pluie	Tiède	Normale	Fort	Oui
11	Ensoleillé	Tiède	Normale	Fort	Oui
12	Couvert	Tiède	Elevée	Fort	Oui
13	Couvert	Chaude	Normale	Faible	Oui
14	Pluie	Tiède	Elevée	Fort	Non

Tableau 4.1: Observations jeu de Tennis

La construction de l'arbres de décision repose sur deux formules : L'entropie et le Gain.

$$Entropie: H(X) = -\sum_{i=1}^{n} p_i log_2(p_i)$$

Où pi est la probabilité de l'évènement i et X étant l'ensemble de données.

$$Gain: Gain(X, a_i) = H(X) - \sum \frac{|X_{a_i=v}|}{|X|} H(X_{a_i=v})$$

Parmi les observations de notre ensemble de données nous avons 9 individus qui ont joué au tennis et 5 qui n'ont pas joué. L'entropie de cette population est donc :

$$H(X) = -\left(\frac{9}{14} log_2\left(\frac{9}{14}\right) + \frac{5}{14} log_2\left(\frac{5}{14}\right)\right) = 0.940$$

Ensuite nous pouvons calculer le Gain pour tous les variables indépendantes comme suit :

$$Gain(X, Ciel) = H(X) - \sum \frac{|X_{Ciel}|}{|X|} H(X_{Ciel})$$

D'abord il faut calculer l'entropie de la population X_{Ciel} pour chaque valeur (Ensoleillé, Couvert, Pluie). Pour la valeur ensoleillée nous avons les données suivantes :

	Ciel	Température	Humidité	Vent	Jouer au Tennis
1	Ensoleillé	Chaude	Elevée	Faible	Non
2	Ensoleillé	Chaude	Elevée	Fort	Non
8	Ensoleillé	Tiède	Elevée	Faible	Non
9	Ensoleillé	Fraiche	Normale	Faible	Oui
11	Ensoleillé	Tiède	Normale	Fort	Oui

$$H(X_{Ciel=Ensoleillé}) = -\left(\frac{2}{5} log_2\left(\frac{2}{5}\right) + \frac{3}{5} log_2\left(\frac{3}{5}\right)\right) = 0.971$$

$$H(X_{Ciel=Couvert}) = -\left(\frac{4}{4} log_2\left(\frac{4}{4}\right) + \frac{0}{4} log_2\left(\frac{0}{4}\right)\right) = 0.971$$

$$H(X_{Ciel=Pluie}) = -\left(\frac{4}{4} log_2\left(\frac{4}{4}\right) + \frac{0}{4} log_2\left(\frac{0}{4}\right)\right) = 0.971$$

Nous pouvons maintenant calculer le Gain de l'attribut Ciel :

$$Gain(X, Ciel) = H(X) - (\frac{5}{14} H(X_{Ciel=Ensoleillé}) + \frac{4}{14} H(X_{Ciel=couvert}) + \frac{5}{14} H(X_{Ciel=pluie}))$$

$$Gain(X, Ciel) = 0.246$$

De la même manière on peut calculer le gain de l'attribut Température :

$$Gain(X, Température) = H(X) - \sum \frac{|X_{Température}|}{|X|} H(X_{Température})$$

$$H(X_{Température=Chaude}) = -\left(\frac{2}{4} log_2\left(\frac{2}{4}\right) + \frac{3}{4} log_2\left(\frac{3}{4}\right)\right) = 1$$

$$H(X_{Température=Tiède}) = -\left(\frac{4}{6} log_2\left(\frac{4}{6}\right) + \frac{2}{6} log_2\left(\frac{2}{6}\right)\right) = 0.918$$

$$H(X_{Température=Fraiche}) = -\left(\frac{3}{4} log_2\left(\frac{3}{4}\right) + \frac{1}{4} log_2\left(\frac{1}{4}\right)\right) = 0.811$$

$$Gain(X, Température)$$
$$= H(X) - (\frac{4}{14} H(X_{Température=Chaude}) + \frac{6}{14} H(X_{Température=Tiède})$$
$$+ \frac{4}{14} H(X_{Température=Fraice}))$$

$$Gain(X, Température) = 0.030$$

Ensuite nous calculons le Gain des attributs restants :

$Gain(X, Humidité) = 0.152$

$Gain(X, Vent) = 0.048$

Le choix de la racine de l'arbre de décision se base sur le Gain calculé sur les attributs. La variable indépendante ayant le plus grand gain représentera la racine de l'arbre. Ce qui veut dire pour l'ensemble de donnée du jeu de tennis que l'attribut Ciel est la variable indépendante qui devrait représenter la racine de notre arbre :

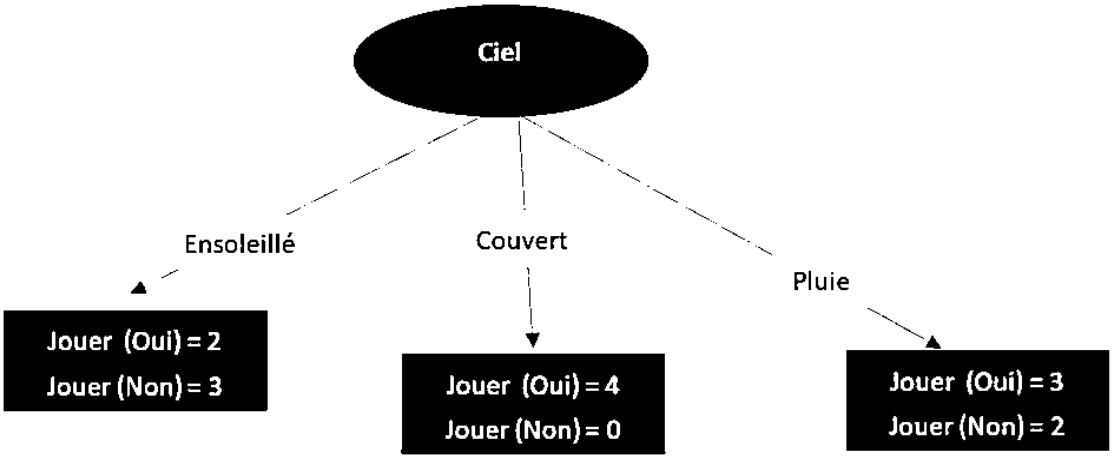


Figure 4.9 : Choix de la racine de l'arbre à l'aide du Gain

Jusqu'ici nous avons obtenu trois nœuds et nous pouvons remarquer que celui du milieu est pur. Le nœud généré par la valeur couvert de l'attribut Ciel ne contient que des éléments appartenant à une seule classe (la classe Jouer=oui). De ce fait, ce nœud restera sous forme de feuille. Les autres nœuds sont des branches.

Sous branche 1 :

	Ciel	Température	Humidité	Vent	Jouer au Tennis
1	Ensoleillé	Chaude	Elevée	Faible	Non
2	Ensoleillé	Chaude	Elevée	Fort	Non
8	Ensoleillé	Tiède	Elevée	Faible	Non
9	Ensoleillé	Fraiche	Normale	Faible	Oui
11	Ensoleillé	Tiède	Normale	Fort	Oui

Parmi les observations de cette sous branche, nous avons 2 individus qui ont joué au tennis et 3 qui n'ont pas joué. L'entropie de cette population est donc :

$$H(X) = -\left(\frac{2}{5}log_2\left(\frac{2}{5}\right) + \frac{3}{5}log_2\left(\frac{3}{5}\right)\right) = 0.971$$

Calculons maintenant les Gains des attributs restants (Autre que le Ciel) :

Température :

$$Gain(X, Température) = H(X) - \sum \frac{|X_{Température}|}{|X|} H(X_{Température})$$

$$H(X_{Température=Chaude}) = -\left(\frac{0}{2} log_2\left(\frac{0}{2}\right) + \frac{2}{2} log_2\left(\frac{2}{2}\right)\right) = 0$$

$$H(X_{Température=Tiède}) = -\left(\frac{1}{2} log_2\left(\frac{1}{2}\right) + \frac{1}{2} log_2\left(\frac{1}{2}\right)\right) = 1$$

$$H(X_{Température=Fraiche}) = -\left(\frac{1}{1} log_2\left(\frac{1}{1}\right) + \frac{0}{1} log_2\left(\frac{0}{1}\right)\right) = 0$$

$$Gain(X, Température)$$

$$= H(X) - (\frac{2}{5} H(X_{Température=Chaude}) + \frac{2}{5} H(X_{Température=Tiède})$$

$$+ \frac{1}{5} H(X_{Température=Fraice}))$$

$$Gain(X, Température) = 0.571$$

Humidité :

$$Gain(X, Humidité) = 0.971$$

Vent:

$$Gain(X, Vent) = 0.020$$

Le plus grand gain est celui de l'attribut Humidité. Ce qui veut dire que la branche représentée par la valeur ensoleillé va être raccordée au nœud Humidité. Une fois fait, on obient des nœuds purs (feuilles). Le même raisonnement est effectué sur les branches restantes ayant généré des nœuds non purs.

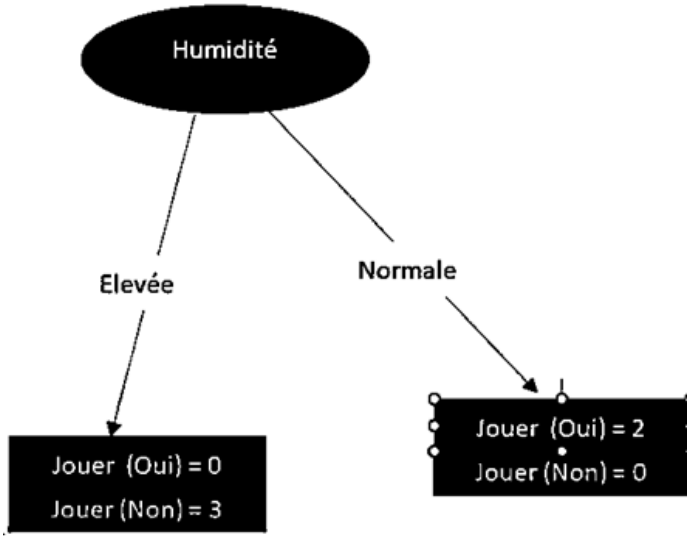


Figure 4.10 : Feuilles pures obtenus après le choix de la variable Humidité.

4.4. Implémentation avec R

L'implémentation des arbres de décision en langage R ressemble grandement à celle de la régression logistique vue dans la section 4.2 mise à part quelques changements que nous allons voir dans ce qui suit. Nous allons travailler sur la même base de données concernant les utilisateurs d'un réseau social. D'abord on doit importer la librairie « rpart » tel que montré dans le code ci-dessous. Ensuite on appelle la fonction rpart qui prend en entrée deux paramètres. La formule, écrite comme vue précédemment dans ce document ainsi que l'ensemble d'apprentissage.

```
# Ajuster l'arbre de décision à l'ensemble d'apprentissage
# install.packages('rpart')
library(rpart)
classifieur = rpart(formula = Achat ~ .,
                    data = training_set)
```

En utilisant le code suivant on obtient les prédictions sur l'ensemble de Test ainsi que la matrice de confusion :

```
# Prédiction sur l'ensemble de Test
y_pred = predict(classifieur, newdata = test_set[-3])
y_grid = ifelse(y_pred > 0.5, 1, 0)
# Calculer la matrice de confusion
cm = table(test_set[, 3], y_grid)
```

	0	1
0	53	11
1	8	28

La matrice de confusion obtenue en haut veut dire qu'il y a eu 19 fausses prédiction obtenues avec ce modèle de classification. Le modèle estimé est utilisé avec l'ensemble d'apprentissage pour obtenir le graphe ci-dessous. Nous remarquons que nous avons obtenu un classifieur différent de celui vu avec la régression logistique. De nouvelles limites de décision se sont dessinées.

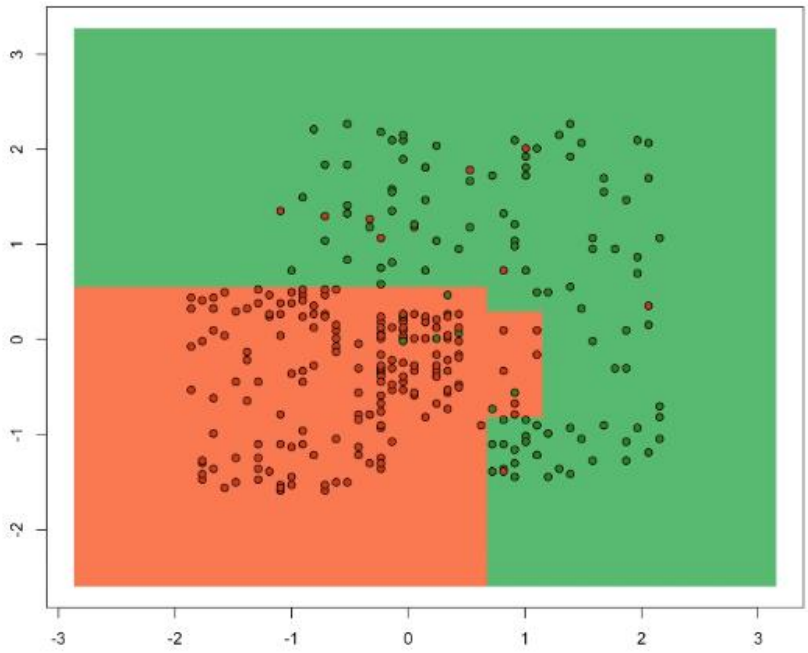


Figure 4.11 : Modèle Arbres de décision estimé

4.5. Naive Bayes

La classification par Naive Bayes est une classification probabiliste. Il s'agit d'estimer la probabilité de chaque classe sachant l'observation de la donnée x et sachant la base d'apprentissage X.

Règle de Bayes :

$$\Pr[y|x,X] = \frac{\Pr[x|y,X]\ \Pr[y|X]}{\Pr[x,X]}$$

$\Pr[y|x,X]$ Appelée probabilité à posteriori.

Pr[y|X] est la probabilité à priori d'observer la classe y étant donné l'ensemble X.

Pr[x|y,X] est la vraisemblance de l'évènement x s'il est de classe y disposant des exemples X.

D'après l'hypothèse de Naive Bayes :

$$\Pr[\mathrm{x}|\mathrm{y},\mathrm{X}] = \prod_{i=1}^{i=P} \Pr[a_i = v_i|y,X]$$

Où x est décrit par P attributs notés a_i dont les valeurs sont notés v_i .

$\Pr[x,X]$ est la probabilité d'observer la données x ayant l'ensemble de données X.

$$\sum_{y_i \in y} \Pr[y_i|x,X] = 1$$

$$\Pr[y|x,X] = \frac{\Pr[y|x,X]}{1} = \frac{\Pr[y|x,X]}{\sum_{y_i \in y} \Pr[y_i|x,X]} = \frac{\frac{\Pr[x|y,X]\ \Pr[y|X]}{\Pr[x,X]}}{\sum_{y_i \in y} \frac{\Pr[x|y_i,X]\ \Pr[y_i|X]}{\Pr[x,X]}}$$

$$\Pr[y|x,X] = \frac{\Pr[x|y,X]\ \Pr[y|X]}{\sum_{y_i \in y} \Pr[x|y_i,X]\ \Pr[y_i|X]}$$

On peut prédire maintenant la classe de la données x (Maximal a posteriori) :

$$y_{MAP} = \arg\max_{y \in Y} \Pr[y|x, X]$$

$$y_{MAP} = \arg\max_{y \in Y} \Pr[x|y, X]\Pr[y|X]$$

Maximal Likelihood :

$$y_{ML} = \arg\max_{y \in Y} \Pr[x|y, X]$$

Exemple : Soit l'ensemble de données représenté par le tableau ci-dessous. Il s'agit d'un nombre d'observations sur un joueur de tennis. Ces observations représentent le fait de jouer ou pas en fonction de quatre variables indépendantes qui sont le Ciel, la Température, L'humidité et le vent. On souhaite à travers cet ensemble de données pouvoir prédire la classe de la donnée x=(ciel=Soleil, Température=Fraiche, Humidité=élevée, Vent=Fort)

Jour	Ciel	Température	Humidité	Vent	Jouer
J1	Soleil	Chaude	élevée	Faible	Non
J2	Soleil	Chaude	élevée	Fort	Non
J3	Couvert	Chaude	élevée	Faible	Oui
J4	Pluie	Tiède	élevée	Faible	Oui
J5	Pluie	Fraiche	Normale	Faible	Oui
J6	Pluie	Fraiche	Normale	Fort	Non
J7	Couvert	Fraiche	Normale	Fort	Oui
J8	Soleil	Tiède	élevée	Faible	Non
J9	Soleil	Fraiche	Normale	Faible	Oui
J10	Pluie	Tiède	Normale	Faible	Oui
J11	Soleil	Tiède	Normale	Fort	Oui
J12	Couvert	Tiède	élevée	Fort	Oui
J13	Couvert	Chaude	Normale	Faible	Oui
J14	Pluie	Tiède	élevée	Fort	Non

Tableau 4.2 : Ensemble de données sur le jeu de Tennis

$$\Pr[jouer = oui|x, X] = \frac{\Pr[x|jouer = oui, X]\Pr[jouer = oui|X]}{\sum_{y\in\{oui,non\}} \Pr[x|jouer = y, X]Pr\,[jouer = y|X]}$$

Attribut	Valeur	Jouer=oui	Jouer=Non
Ciel	Ensoleillé	2	3
	Couvert	4	0
	Pluie	3	2
Température	Chaude	2	2
	Tiède	4	2
	Fraiche	3	1
Humidité	Elevée	3	4
	Normale	6	1
Vent	Fort	3	3
	Faible	6	2
Jouer		9	5

$$Pr[jouer = oui|X] = \frac{9}{14}$$

$$Pr[jouer = non|X] = \frac{5}{14}$$

$$Pr[x|jouer = oui, X] = \Pr[(ciel = Ensoleillé, Température = Fraiche, Humidité = elevée, Vent = Fort)|jouer = oui, X]$$

= Pr [(ciel=Ensoleillé)|jouer=oui, X] ×

= Pr [(Température=Fraiche)|jouer=oui, X] ×

= Pr [(Humidité=Elevée)|jouer=oui, X] ×

= Pr [(Vent=Fort)|jouer=oui, X]

$$=\frac{2}{9}\times\frac{3}{9}\times\frac{3}{9}\times\frac{3}{9} = \frac{2}{243}$$

$$Pr[x|jouer = non, X] = \Pr[(ciel = Ensoleillé, Température = Fraiche, Humidité = elevée, Vent = Fort)|jouer = non, X]$$

$$= \Pr[(\text{ciel=Ensoleillé})|\text{jouer=non, X}] \times$$

$$= \Pr[(\text{Température=Fraiche})|\text{jouer=non, X}] \times$$

$$= \Pr[(\text{Humidité=Elevée})|\text{jouer=non, X}] \times$$

$$= \Pr[(\text{Vent=Fort})|\text{jouer=non, X}]$$

$$= \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} = \frac{36}{625}$$

$$Pr[jouer = oui|x, X] = \frac{\frac{2}{243} \times \frac{9}{14}}{\frac{2}{243} \times \frac{9}{14} \times \frac{36}{625} \times \frac{5}{14}} = 0.205$$

$$Pr[jouer = non|x, X] = \frac{\frac{36}{625} \times \frac{5}{14}}{\frac{2}{243} \times \frac{9}{14} \times \frac{36}{625} \times \frac{5}{14}} = 0.795$$

En cas d'Attributs numériques :

Exemple : On souhaite prédire la classe de la donnée x=(ciel=soleil, température=18, humidité=90, vent=fort) sachant le jeux de données montré sur le tableau ci-dessous.

On utiliser la distribution normale comme suit :

$$N(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\overline{x})^2}{2\sigma\sigma}}$$

Température		Humidité	
Oui	Non	Oui	Non
26.5	27.5	86	85
20	25	96	90
19	17.5	80	70
17	21	65	95
19.5	20.5	70	91
22.5		80	
22.5		70	
21		90	
25.5		75	
21.5	22.3	79.1	86.2
2.91	3.53	10.2	9.7

$\Pr[température = 18|jouer = oui, X] = \frac{1}{\sqrt{2\pi 2.91^2}} e^{-\frac{(18-21.5)^2}{2\times 2.91^2}}$=0.0665

$\Pr[température = 18|jouer = non, X] = \frac{1}{\sqrt{2\pi 3.53^2}} e^{-\frac{(18-22.3)^2}{2\times 3.53^2}}$=0.0538

$\Pr[Humidité = 90|jouer = oui, X] = \frac{1}{\sqrt{2\pi 10.2^2}} e^{-\frac{(90-79.1)^2}{2\times 10.2^2}}$=0.0221

$\Pr[Humidité = 90|jouer = non, X] = \frac{1}{\sqrt{2\pi 9.7^2}} e^{-\frac{(90-86.2)^2}{2\times 9.7^2}}$=0.0381

$$\Pr[x|jouer = oui, X] = \frac{2}{9} \times 0.0665 \times 0.0221 \times \frac{3}{9} = 1.08 \times 10^{-4}$$

$$\Pr[x|jouer = non, X] = \frac{3}{5} \times 0.0538 \times 0.0381 \times \frac{3}{5} = 7.37 \times 10^{-4}$$

$$\Pr[\text{jouer} = \text{oui}|\text{x}, \text{X}] = \frac{1.08 \times 10^{-4} \times \frac{9}{14}}{1.08 \times 10^{-4} \times \frac{9}{14} + 7.37 \times 10^{-4} \times \frac{5}{14}} = 0.208$$

$$\Pr[\text{jouer} = \text{non}|\text{x}, \text{X}] = \frac{7.37 \times 10^{-4} \times \frac{5}{14}}{1.08 \times 10^{-4} \times \frac{9}{14} + 7.37 \times 10^{-4} \times \frac{5}{14}} = 0.791$$

4.6. Implémentation avec R

Une fois de Plus, l'implémentation en R de ce classifieur ressemble à celles vues précédemment dans les deux sections 4.2 et 4.4. Le même programme el langage R est utilisé pour implémenter et visualiser le classifieur de Naive Bayes. La seule différence consiste en le code utilisé pour estimer le classifieur Naive Bayes à l'aide de l'ensemble d'apprentissage. D'abord il faut importer la librairie e1071 comme le montre le code ci-dessous. Par la suite la fonction naiveBayes permet de créer le classifieur Naive Bayes.

```
# Estimation du Modèle de Naive Bayes
# install.packages('e1071')
library(e1071)
classifieur = naiveBayes(x = training_set[-3],
                         y = training_set$Achat)
```

Une fois estimé sur l'ensemble d'apprentissage de la base de données réseau social, nous pouvons calculer la matrice de confusion ainsi que visualiser le modèle obtenu dans la figure suivante.

	0	1
0	57	7
1	87	29

D’après la matrice de confusion obtenue, on peut dire que le modèle de Naive Bayes permet de faire de meilleures prédictions par rapport aux modèles vus jusqu’à présent.

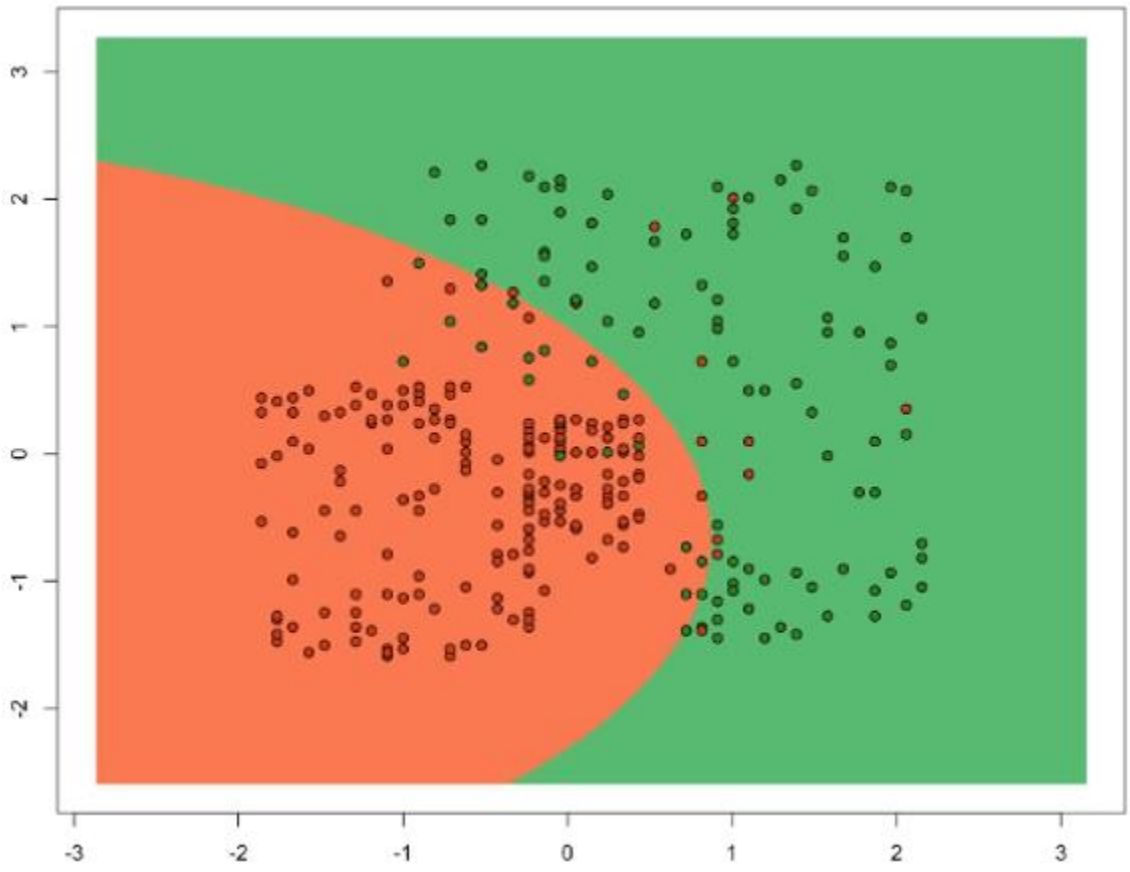


Figure 4.12 : Modèle Naive Bayes estimé pour la classification

Chapitre 5 : Les Règles D'Association

5.1. Algorithme Apriori

Lorsqu'on fait les courses, nous avons une liste de choses à acheter. Chaque individu a une liste unique qui dépend de ses préférences et de ses besoins. En visualisant les listes d'achats on remarque qu'il existe des associations entre les articles. Comprendre ces associations permet de faire de l'optimisation et d'augmenter les ventes. L'exemple ci-dessous montre par exemple que tous les clients ayant acheté le lait, ont aussi acheté le sucre. 60% entre eux ont acheté aussi le café. Ces informations qui permettent de mieux disposer ces éléments sur les rayons du supermarché sont appelés règles d'association.

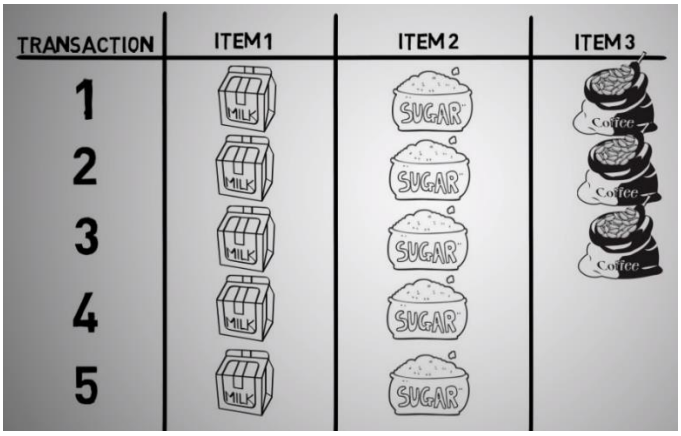


Figure 5.1 : Exemple de liste de transactions au supermarché

L'un des algorithmes les plus utilisés pour extraire ces règles d'association est l'algorithme Apriori. Dans cette section est déroulé l'algorithme d'Apriori sur l'ensemble de transactions du tableau ci-dessous. Ceux concepts sont à définir avant : le support et la confiance.

Le support signifie la popularité de l'élément. On peut le calculer grâce à la formule suivante :

$$Support(X) = \frac{Nombre\ de\ transactions\ où\ X\ apparait}{Nombre\ Totale\ de\ Transactions}$$

La confiance d'une règle A→B est définie comme suit :

$$\text{Confiance}(A \rightarrow B) = \frac{Support(AUB)}{Support(A)}$$

Transaction	**Liste d'Items**
1	A,B,E
2	B,D
3	B,C
4	A,B,D
5	A,C
6	B,C
7	A,C
8	A,B,C,E
9	A,B,C

Tableau : Liste de transactions

Nous allons appliquer l'algorithme Apriori sur le Tableau en prenant : SupportMinimum=2. Confiance 70%. La première étape à faire est celle de calculer le support de tous les éléments comme le montre le tableau ci-dessous. Dans la colonne L1 du tableau suivant, on garde les élements ayant un support supérieur ou égal au support minimum définit au préalable.

C1		L1	
Itemset	Support		
{A}	6	{A}	6
{B}	7	{B}	7
{C}	6	{C}	6
{D}	2	{D}	2
{E}	2	{E}	2

Par la suite, on génère des ensembles contenant toutes les combinaisons possibles deux-à-deux entre les éléments de la liste L1 comme le montre le tableau suivant. Ensuite une sélection des ensembles les plus significatifs est faite aussi à l'aide du support minimum.

C2		L2	
Itemset	Support		
{A,B}	4	{A,B}	4
{A,C}	4	{A,C}	4
{A,D}	1	{A,E}	2
{A,E}	2	{B,C}	4
{B,C}	4	{B,D}	2
{B,D}	2	{B,E}	2
{B,E}	2		
{C,D}	0		
{C,E}	1		
{D,E}	0		

L'opération est répétée sur les ensembles sélectionnées dans la liste L2 ensuite la liste L3 tel que montré par les tableaux ci-dessous. Une fois arrivé à la liste des ensemble L4 on obtient un seul ensemble ayant un support inférieur à 2 et donc L4 restera un ensemble vide.

C3		L3	
Itemset	Support	Itemset	Support
{A,B,C}	2	{A,B,C}	2
{A,B,E}	2	{A,B,E}	2
{A,B,D}	1		
{A,C,E}	0		
{B,C,D}	0		
{B,C,E}	1		
{B,D,E}	0		

C4 : {A,B,C,E}. L4 = Ø.

Une fois arrivé à ce stade, les règles d'association peuvent être générés à l'aide de la dernière liste d'ensembles significatifs, dans ce cas la liste L3. On procède à l'extraction des règles grâce au taux de confiance fixé au préalable comme suit:

1. {A,B,C}

{A,B} → C $\frac{2}{4}$ = 0.5 = 50 %

{A,C} → B $\frac{2}{4}$ = 0.5 = 50 %

{B,C} → A $\frac{2}{4}$ = 0.5 = 50 %

A → {B,C} $\frac{2}{6}$ = 33 %

B → {A,C} $\frac{2}{7}$ = 29 %

C → {A,B} $\frac{2}{6}$ = 33 %

1. {A,B,E}

{A,B} → E $\frac{2}{4}$ = 50 %

{A,E} → B $\frac{2}{2}$ = 100 %

{B,E} → A $\frac{2}{2}$ = 100 %

A → {B,E} $\frac{2}{6}$ = 33 %

B → {A,E} $\frac{2}{7}$ = 29 %

E → {A,B} $\frac{2}{2}$ = 100 %

Les règles d'association retenues sont donc : {A,E} → B , {B,E} → A, E→{A,B}

5.2. Implémentation avec R

Dans cette section nous implémentons l'algorithme d'Apriori afin de déceler des règles d'association à partir d'un ensemble de données constitué de 7501 observation (transaction) et de 20 variables. Nous commençons par importer les données à l'aide de l'instruction suivante :

dataset = read.csv('Supermarché.csv', header = FALSE)

La partie (header = FALSE) permet d'éviter de considérer la première ligne comme header. Une fois chargé nous pouvons visualiser les données en cliquant sur la variable dataset. On peut voir qu'il s'agit de transactions qui contiennent des listes de produits achetés.

	V1	V2	V3	V4	V5	V6
1	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour
2	burgers	meatballs	eggs			
3	chutney					
4	turkey	avocado				
5	mineral water	milk	energy bar	whole wheat rice	green tea	
6	low fat yogurt					
7	whole wheat pasta	french fries				
8	soup	light cream	shallot			
9	frozen vegetables	spaghetti	green tea			
10	french fries					

Figure 5.2 : Jeu de données Supermarché.CSV

Afin d'appliquer l'algorithme Apriori nous avons besoin de la librairie « arules ». On peut l'installer puis l'importer grâce aux deux instructions suivantes :

install.packages('arules')

library(arules)

Par la suite l'algorithme a besoin de recevoir les données sous forme d'une matrice. Nous appelons donc l'instruction :

dataset = read.transactions('Supermarché.csv', sep = ',', rm.duplicates = TRUE)

summary(dataset)

Pour avoir des détails sur les données nous pouvons utiliser la fonction summary. Les informations sont affichés dans la Figure suivante. La matrice contient les informations sur 7501 transactions. La fonction retourne les produits les plus fréquents. Nous pouvons voir que le produit le plus fréquent est l'eau. Ensuite nous avons la distribution des paniers des 7501 transactions. Par exemple, le premier élément veut dire que nous avons 1754 transaction contenant seulement un seul produit. La plus grande transaction contient 20 produits. En moyenne les clients mettent quatre produits dans leurs paniers…etc.

```
> summary(dataset)
transactions as itemMatrix in sparse format with
 7501 rows (elements/itemsets/transactions) and
 119 columns (items) and a density of 0.03288973

most frequent items:
mineral water          eggs     spaghetti  french fries     chocolate
         1788          1348          1306          1282          1229
      (other)
        22405

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
1754 1358 1044  816  667  493  391  324  259  139  102   67   40   22   17
  16   18   19   20
   4    1    2    1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   3.000   3.914   5.000  20.000
```

La fonction « itemFrequencyPlot(dataset, topN = 10) »permet d'afficher la fréquence avec laquelle les différents produits apparaissent dans la base de données. La fonction affiche les premiers 10 articles les plus fréquents (topN=10). Cette fonction est utile si on veut visualiser les supports afin d'en choisir un qui est adéquat.

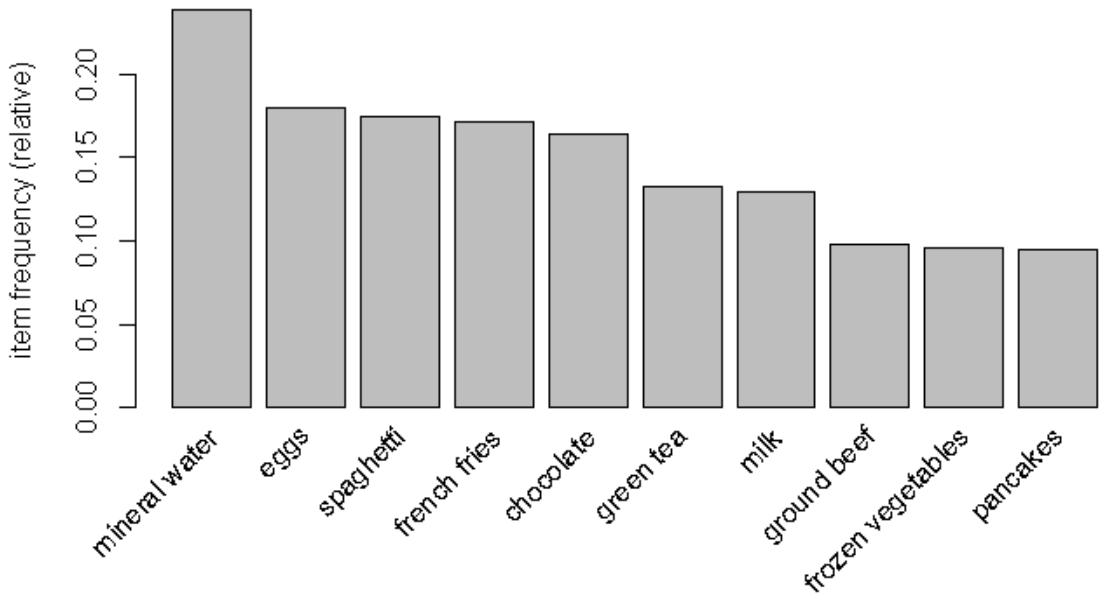


Figure 5.3 : Fréquence d'apparition des éléments

Maintenant nous allons voir comment apprendre les règles à partir de la base d'apprentissage. Pour le faire nous pouvons utiliser la fonction apriori comme suit:

```
# Apprentissage des règles à l'aide d'Apriori
Regles = apriori(data = dataset, parameter = list(support = 0.003, confidence = 0.2))
```

Le premier argument représente les données et le deuxième argument est le paramètre où on va introduire des valeurs pour le support et la confiance. Comme le montre la première étape de l'algorithme d'Apriori, il faut d'abord choisir le support minimum et la confiance. Le choix de ces deux paramètres ne repose pas sur une règle générale, il dépend de la base de données traitée et le but à atteindre. Commençons avec le support. En le définissant, on va choisir les éléments que vont contenir les règles. Ces éléments vont respecter un certain support minimal qui va nous générer à la fin des règles intéressantes. Si on affiche la fréquence d'apparition des éléments des 100 premiers éléments nous allons remarquer que beaucoup d'articles ne sont pas acheté souvent, et donc auront un très petit support. Ces éléments ne sont pas intéressants à l'étude des règles d'association. Supposons que nous

allons considérer uniquement les produits acheté au moins 3 fois par jour. Ces produits seront donc acheté 21 par semaine. Maintenant il faut diviser ce nombre par le nombre totale des transactions effectuées par semaine (7501) pour avoir le support minimal. Ce qui va revenir à choisir la valeur arrondie a 0.003. La deuxième étape consiste à choisir la confiance. L'idée est de choisir une confiance par défaut (0.8 pour Apriori) et de la réduire étape par étape jusqu'à aboutir à des règles intéressantes. Une fois l'instruction en haut exécutée, les règles sont générées. Nous pouvons visualiser dans les informations retournées dans la console le nombre de règles générées. Avec une confiance de 0.8, aucune règle n'est générée. Ce qui veut dire que toutes les règles générées ont une confiance inférieur à 0.8. Avec une confiance de 0.4, 281 règles sont générées.

Maintenant nous pouvons visualiser les règles et en choisir les plus intéressantes grâce à une métrique appelée Lift. Pour la visualisation nous pouvons utiliser l'instruction inspect comme suit :

```
# Visualisation des règles
inspect(sort(Regles, by = 'lift')[1:10])
```

Les résultats ci-dessous sont obtenus :

```
> inspect(sort(Regles, by = 'lift')[1:10])
     lhs                                rhs                     support confidence    coverage     lift count
[1]  {mineral water,
      whole wheat pasta}             => {olive oil}          0.003866151  0.4027778 0.009598720 6.115863    29
[2]  {spaghetti,
      tomato sauce}                  => {ground beef}        0.003066258  0.4893617 0.006265831 4.980600    23
[3]  {french fries,
      herb & pepper}                 => {ground beef}        0.003199573  0.4615385 0.006932409 4.697422    24
[4]  {cereals,
      spaghetti}                     => {ground beef}        0.003066258  0.4600000 0.006665778 4.681764    23
[5]  {frozen vegetables,
      mineral water,
      soup}                          => {milk}               0.003066258  0.6052632 0.005065991 4.670863    23
[6]  {chocolate,
      herb & pepper}                 => {ground beef}        0.003999467  0.4411765 0.009065458 4.490183    30
[7]  {chocolate,
      mineral water,
      shrimp}                        => {frozen vegetables} 0.003199573  0.4210526 0.007598987 4.417225    24
[8]  {frozen vegetables,
      mineral water,
      olive oil}                     => {milk}               0.003332889  0.5102041 0.006532462 3.937285    25
[9]  {cereals,
      ground beef}                   => {spaghetti}          0.003066258  0.6764706 0.004532729 3.885303    23
[10] {frozen vegetables,
      soup}                          => {milk}               0.003999467  0.5000000 0.007998933 3.858539    30
```

Nous obtenons donc les 10 premières règles les plus intéressantes. La première règle dit que si l'utilisateur achète de l'eau, du blé complet et des pâtes, ils vont aussi acheter de l'huile d'olive dans 40% des cas. Il est intéressant de modifier les paramètres de l'algorithme Apriori et de ré-interpréter les résultats. Si on applique l'algorithme Apriori une nouvelle fois à l'aide d'une confiance égale à 0.2 au lieu de 0.4, on obtient les résultats ci-dessous qui représentent des règles plus intuitifs.

```
> # Visualisation des règles
> inspect(sort(Regles, by = 'lift')[1:10])
     lhs                                    rhs             support     confidence coverage    lift
[1]  {mineral water,whole wheat pasta}      => {olive oil}     0.003866151 0.4027778  0.009598720 6.115863
[2]  {frozen vegetables,milk,mineral water} => {soup}          0.003066258 0.2771084  0.011065191 5.484407
[3]  {fromage blanc}                        => {honey}         0.003332889 0.2450980  0.013598187 5.164271
[4]  {spaghetti,tomato sauce}               => {ground beef}   0.003066258 0.4893617  0.006265831 4.980600
[5]  {light cream}                          => {chicken}       0.004532729 0.2905983  0.015597920 4.843951
[6]  {pasta}                                => {escalope}      0.005865885 0.3728814  0.015731236 4.700812
[7]  {french fries,herb & pepper}           => {ground beef}   0.003199573 0.4615385  0.006932409 4.697422
[8]  {cereals,spaghetti}                    => {ground beef}   0.003066258 0.4600000  0.006665778 4.681764
[9]  {frozen vegetables,mineral water,soup} => {milk}          0.003066258 0.6052632  0.005065991 4.670863
[10] {french fries,ground beef}             => {herb & pepper} 0.003199573 0.2307692  0.013864818 4.665768
```

Questions et Exercices

1. Quelle est la différence entre l'apprentissage supervisé et celui non supervisé ?

2. A quel moment peut-on opter pour l'une des approches : Régression ou Classification ?

3. Expliquer en quelques phrases le fonctionnement de l'algorithme de descente de gradient.

4. Peut-on prédire l'équipe qui gagne à un match de Foot à l'aide de la régression linéaire simple ?

5. Quelle est la fonction en langage R qui permet d'appliquer la régression linéaire simple ?

6. Cochez parmi les formules suivantes celles qui ne sont pas des formules de régression linéaire simple :

- Salaire = a*Expérience
- Salaire = a*Expérience + b
- Salaire = a*Expérience + b* Age

7. Quelle est la manière correcte d'écrire l'équation de régression linéaire simple dans le paramètre formule dans R

- Salaire=AnnéesExpérience
- Salaire~AnnéesExpérience
- Salaire==AnnéesExpérience
- Salaire=a*YearsExperience+b

8. Quelle est la formule qui n'est pas adéquate pour la régression linéaire multiple

- Salaire=a*Experience+b*Age+c
- Salaire=a*Experience+b*Age+c*Level
- Salaire=a*Experience+b*Age+c*Level+d
- Salaire=a*Experience+b*Age^2
- Salaire=a*Experience+b*Age

9. En langage R, le code qui fait la régression linéaire simple permet de faire la régression linéaire multiple :

- Vrai
- Faux

10. En langage R, quelle formule devrait-on introduire comme paramètre pour la régression linéaire multiple ?

- Salaire~*
- Salaire=*
- Salaire~.
- Salaire=experience+Age

11. On devrait utiliser la régression linéaire multiple pour prédire une variable qui est croissante d'une manière exponentielle :

- Vrai
- Faux

12. De quelle nature doit être la variable dépendante afin qu'on puisse appliquer la régression logistique ?

13. Comment peut-on estimer les paramètres du modèle de la régression logistique ?

Algorithme de descente de gradient

14. En langage R, quelle fonction permet de faire la régression logistique ?

- Lr
- Lm
- Glm
- Glr

15. La régression logistique retourne des probabilités ?

- Oui
- Non

16. En langage R, quelle est la valeur qu'on doit fournir au paramètre family ?

- Linear
- Logistic
- Binomial
- Response

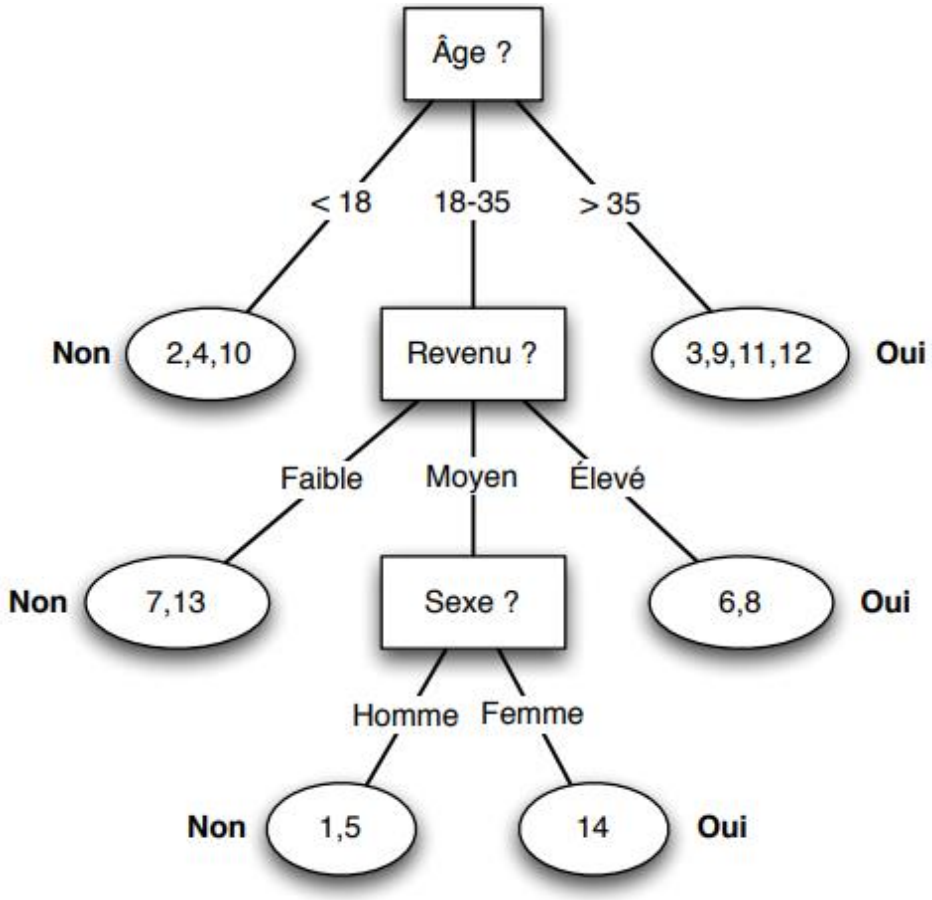
17. Décrire comment un nouvel exemple est classifié dans un arbre de décision entrainé.

18. On veut apprendre un modèle permettant de déterminer si un client est intéressé à acheter un certain produit (Oui ou Non), en fonction de son sexe (Homme ou Femme), son age (< 18, 18 − 35 ou > 35), son état civil (Célibataire ou Marie), et son revenu (Faible, Moyen ou Elevé). Soit l'échantillon suivant d'exemples d'entrainement, Construisez l'arbre de décision résultant de ces exemples en utilisant l'algorithme ID3. On suppose qu'on arrête la subdivision uniquement lorsque les nœuds sont purs (entropie de 0). Exprimez la classe des exemples positifs sous la forme d'un prédicat logique.

ID	Sex	Age	Etat Civil	Revenu	Achat
1	Homme	18-35	Marié	Moyen	Non
2	Homme	<18	Célibataire	Faible	Non
3	Homme	>35	Marié	Elevé	Oui
4	Femme	<18	Célibataire	Moyen	Non
5	Homme	18-35	Célibataire	Moyen	Non
6	Femme	18-35	Célibataire	Elevé	Oui
7	Femme	18-35	Marié	Faible	Non
8	Homme	18-35	Marié	Elevé	Oui
9	Homme	>35	Célibataire	Faible	Oui
10	Femme	<18	Célibataire	Moyen	Non
11	Femme	>35	Célibataire	Moyen	Oui
12	Femme	>35	Marié	Elevé	Oui
13	Homme	18-35	Célibataire	Faible	Non
14	Homme	18-35	Marié	Moyen	Oui

Corrigé :

Pour classifier un nouvel exemple, on traverse l'arbre depuis la racine jusqu'à une feuille. Pour chaque nœud interne rencontré, on emprunte la branche correspondant au résultat du test de ce nœud. Une fois dans la feuille, on assigne l'exemple à la classe la plus fréquente des exemples d'entrainement contenus dans cette feuille**.**



19. Ci-dessous un ensemble de données décrivant si des individus ont acheté un certain produit ou pas en fonction de leurs âges, revenus, score crédit, être étudiant ou pas. A travers ces données, construire, un arbre de décision permettant de prédire si un individu va acheter le produit ou pas.

ID	Age	Revenu	Score Crédit	Etudiant	Achat
1	Jeune	Elevé	Juste	Non	Non
2	Jeune	Elevé	Excellent	Non	Non
3	Moyen	Elevé	Juste	Non	Oui
4	Senior	Moyen	Juste	Moyen	Oui
5	Senior	Bas	Juste	Oui	Non
6	Senior	Bas	Excellent	Oui	Non
7	Moyen	Bas	Excellent	Oui	Non
8	Jeune	Moyen	Juste	Non	Non
9	Jeune	Bas	Juste	Oui	Oui
10	Senior	Moyen	Juste	Oui	Oui
11	Jeune	Moyen	Excellent	Oui	Oui
12	Moyen	Moyen	Excellent	Non	Oui
13	Moyen	Elevé	Juste	Oui	Oui
14	Senior	Moyen	Juste	Non	Non

20. En utilisant les données de la question précédente, appliquer l'algorithme de naïve bayes afin de prédire la classe de l'individu ayant les caractéristiques (Moyen, Bas, Juste, Oui).

21. Ci-dessous un ensemble de données sur huit individus se trouvant sur une plage. Nous voulons utiliser leurs informations pour prédire par la suite le risque d'attraper un coup de soleil pour d'autres individus se trouvant en plage.

N	Cheveux	Taille	Poids	Crème Solaire	Classe
1	Blond	Moyenne	Léger	Non	Coup de Soleil
2	Blond	Grande	Moyen	Oui	Bronzé
3	Brun	Petite	Moyen	Oui	Bronzé
4	Blond	Petite	Moyen	non	Coup de Soleil
5	Roux	Moyenne	Lourd	Non	Coup de Soleil
6	Brun	Grande	Lourd	Non	Bronzé
7	Brun	Moyenne	Lourd	Non	Bronzé
8	Blond	Peite	Léger	Oui	Bronzé

En se basant sur la classification par l'algorithme de naïve bayes, trouver les classes des exemples suivants :

N	**Cheveux**	**Taille**	**Poids**	**Crème Solaire**	Solution
1	Blond	Petite	Lourd	Oui	Bronzé
2	Brun	Grande	Moyen	Non	Bronzé
3	Blond	Moyenne	Léger	Non	Coup de Soleil

22. Appliquer l'algorithme Apriori pour extraire les règles d'association présentes dans la base de transactions ci-dessous (minimum support = 2).

Transaction 1 : 1 3 4

Transaction 2 : 2 3 5

Transaction 3 : 1 2 3 5

Transaction 4 : 2 5

23. Soit le support minimum=0.3 et la confiance minimale =60%. Trouver l'ensemble des règles d'association présentes dans le tableau suivant :

	X1	**X2**	**X3**	**X4**	**X5**
A1	0	1	0	0	1
A2	0	0	1	0	1
A3	0	0	1	0	0
A4	1	1	1	1	1
A5	1	1	1	1	0
A6	1	0	1	1	0
A7	1	0	0	0	1
A8	1	0	0	0	1

Corrigé :

Les règles d'association retenues	**Confiance**
X1=> X3	3/5 = 60%
X3=> X1	3/5 = 60%
X1=> X4	3/5 = 60%
X4 => X1	3/3 = 100%
X1=>X5	3/5 = 60%
X5=>X1	3/5 = 60%
X3=> X4	3/5 = 60%
X4=> X3	3/3 = 100%
X1=>X3X4	3/5 = 60%
X3=>X1X4	3/5 = 60%
X4=>X1X3	3/3 = 100%
X1X3=>X4	3/3 = 100%
X1X4=>X3	3/3 = 100%
X3X4=>X1	3/3 = 100%

Bibliographie

[1] Max Bramer, Principles of Data Mining, 2020.

[2] Jiawei Han, Micheline Kamber Jian Pei: Data Mining, Concepts and Techniques 3rd edition, 2012.

[3] Yanchang Zhao: R and Data Mining: Examples and Case Studies 1, 2014.

[4] Luis Torgo: Data Mining with R Learning with Case Studies, Second Edition, 2017.

[5] Graham Williams, Data Mining with Rattle and R, 2011.

[6] David L Olson, Dursun Delen : Advanced Data Mining Techniques, 2008.

[7] Wolfgang Ertel: Introduction to Artificial Intelligence, 2011.

[8] Miroslav Kubat: An introduction to Machine Learning, 2017.

[9] Massimiliano Bonamente: Statistics and Analysis of Scientific Data, 2017.