Module 6
Algorithmes de base en apprentissage machine

Sommaire

Dernière mise à jour le 6 novembre 2022

Introduction

L'apprentissage machine (aussi appelé apprentissage artificiel ou automatique, en anglais *machine learning*) est ...

> *le processus par lequel un ordinateur acquiert de nouvelles connaissances et améliore son mode de fonctionnement en tenant compte des résultats obtenus lors de traitements antérieurs.(Office québécois de la langue française)*

Il existe deux approches principales en apprentissage machine. La première est issue de l'intelligence artificielle *syntaxique* ou *symbolique*. Elle est fondée sur la modélisation du raisonnement logique et sur la représentation et la manipulation de la connaissance par des symboles formels. La deuxième est issue l'intelligence artificielle *statistique*, aussi parfois appelée numérique parce que, souvent, la représentation des données est sous une forme numérique. Le cours SCI 1016 s'intéresse à l'apprentissage machine statistique.

6.1 Principe de l'apprentissage machine

La définition présentée en introduction permet d'enchainer avec les éléments suivants sur le principe de l'apprentissage machine : il s'agit d'un ensemble de méthodes qui permettent de construire un modèle de la réalité à partir de données, soit en améliorant un modèle existant moins général, soit en créant un nouveau modèle représentatif de nouvelles données. Le choix du modèle dépend essentiellement des données à analyser. Les paramètres du modèle sont déterminés durant la phase d'apprentissage en utilisant l'algorithme qui lui est spécifique. Les modèles servent souvent à prendre des décisions. La décision à prendre dépend essentiellement de la problématique de l'apprentissage machine à résoudre. Par exemple, s'il s'agit d'un problème de classification de forme, la décision consiste souvent à reconnaître la classe de la forme. Par contre s'il s'agit d'un problème de regroupement, la décision consistera à identifier les différents regroupements.

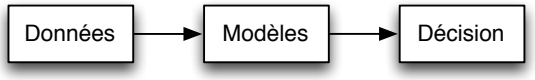

Figure 6.1: Neurone formel.

6.1.1 Entraînement, test et validation du modèle

L'ensemble des données considérées pour une analyse par apprentissage machine peut être réparti en trois sous-ensembles :

- Un ensemble d'entraînement qui est utilisé pour l'apprentissage (entraînement) des paramètres du modèle.
- Un ensemble de validation. Il s'agit d'un ensemble de données qui permet d'évaluer le modèle pendant la phase d'entraînement. Cette étape, appelée validation du modèle peut être omise en passant à la phase de test directement.
- Un ensemble de test. Une fois le modèle construit à partir de l'ensemble d'entraînement, le modèle est évalué en utilisant un ensemble de test : un ensemble d'échantillons n'ayant pas servi pour l'apprentissage.

▶ **Exemple 5.1** Nous considérons ici la base de données IRIS qui contient 150 échantillons provenant de trois classes décrites par quatre variables. Nous désirons faire une répartition des données en 80% pour l'ensemble d'entrainement `training_dataset` et le reste pour l'ensemble de test `testing_dataset` .

```
# partitions reproductibles
set.seed(123)
# Créer une partir de 80%
random_sample <- createDataPartition(iris$Species,p = 0.8, list = FALSE)
# Train Partition
print('Train Partition :')
summary(iris$Species[random_sample]);
# Test Partition
print('Test Partition :')
summary(iris$Species[-random_sample]);
# Générer les données d'entrainement
training_dataset  <- iris[random_sample, ]
# Générer les données de test
# à partir de celles qui ne sont pas inclus dans l'entrainement
testing_dataset <- iris[-random_sample, ]
```


La fonction `createDataPartition` peut être utilisée pour créer des divisions équilibrées des données. Si l'argument de cette fonction est un facteur, l'échantillonnage aléatoire se produit au sein de chaque classe et doit préserver la distribution globale des classes des données.
Dans notre cas, l'eéchantillonnage est fait selon l'argument `iris$Species`. Nous obtenons, ainsi, deux ensembles `training_dataset` et `testing_dataset` contenant respectivement 120 et 30 échantillons (soit un total de 150 échantillons). Les différentes divisions sont équilibrées puisque nous avons 40 échantillons par classe d'espèces dans `training_dataset` et 10 échantillons par classe dans `testing_dataset`.

```
> print('Train Partition :')
[1] "Train Partition :"
> summary(iris$Species[random_sample]);
setosa versicolor  virginica
40         40         40
>
> # Test Partition
> print('Test Partition :')
[1] "Test Partition :"
> summary(iris$Species[-random_sample]);
setosa versicolor  virginica
10         10         10
```

Il est à noter que si nous désirons créer un sous-ensemble de validation, nous utiliserons de la même manière la fonction `createDataPartition` sur le sous-ensemble `training_dataset` pour en sortir un sous-ensemble de données de validation.

6.1.2 Validation par K-fold

Il existe différents méthodes pour réaliser le partage des données en un ensemble de données d'apprentissage et un ensemble de données de test. Parmi ces méthodes on trouve la validation par K-plis (plus connu sous le termes K-fold).

Le principe de cette méthode de validation consiste à diviser l'échantillon original en K échantillons de même taille et à prendre un échantillon pour procéder à la validation. Le processus est répété jusqu'à parcours des K échantillons. En d'autres termes, (1) nous divisons l'échantillon original en K échantillons, (2) nous sélectionnons un des K échantillons comme ensemble de tests et (3) nous gardons les $(K-1)$ autres échantillons pour réaliser l'apprentissage et la conception du système de classification. À la fin, (4) nous calculons la moyenne des résultats de validations pour avoir un seul résultat.

La répartition la plus communément utilisée entre ces échantillons est une proportion de 2/3 pour l'apprentissage et 1/3 pour le test, c'est-à-dire avec un $K = 3$ fold. Dans le cas de grande taille de base de données, la validation 10-fold est plus appropriée.

▶ **Exemple 5.2** Nous considérons ici la base de données IRIS et nous désirons faire une validation croisée 10-fold (9 fold pour l'entrainement et 1 pour le test). L'exemple suivant utilise la validation croisée 10-fold avec 3 répétitions pour estimer un arbre de décision sur le jeu de données de l'iris.

```
# load the library
library(caret)
# load the iris dataset
data(iris)
# The following example uses 10-fold cross validation with 3 repeats to estimate a decision on the iris dataset
#define training control
train_control <- trainControl(method="repeatedcv", number=10, repeats=3)
# train the model (sera traité dans la prochaine section)
model <- train(Species~., data=iris, trControl=train_control, method="ctree")
```

Nous avons ainsi 135 échantillons d'entrainement (9/10 de 150 échantillons au total) à chaque `fold` et 15 échantillons de test (1/10 de 150 échantillons au total).

```
Conditional Inference Tree

150 samples
4 predictor
3 classes: 'setosa', 'versicolor', 'virginica'

No pre-processing
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 135, 135, 135, 135, 135, 135, ...
Resampling results across tuning parameters:

mincriterion  Accuracy   Kappa
0.01          0.9422222  0.9133333
0.50          0.9422222  0.9133333
0.99          0.9422222  0.9133333

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mincriterion = 0.99.
```

6.1.3 Validation croisée Leave One Out (LOOCV)

La validation croisée Leave One Out (LOOCV) est un cas particulier de la validation k-fold pour laquelle un seul échantillon de données est laissée de côté pour le test. Le modèle est construit en utilisant tout les autres échantillons restant. Cette opération est répétée pour tous les échantillons de la base de données. Une moyenne est calculée par la suite.

▶ **Exemple 5.3** Nous considérons la base de données IRIS et nous désirons faire une validation croisée LOOCV.

Les lignes de codes suivants permettent de réaliser ce type de validation. Nous avons ainsi 149 échantillons d'entrainement et 1 échantillon de test.

```r
# load the library
library(caret)
# load the iris dataset
data(iris)
# define training control
train_control <- trainControl(method="LOOCV")
# train the model
model <- train(Species~., data=iris, trControl=train_control, method="nb")
# summarize results
print(model)
```

```
Conditional Inference Tree

150 samples
4 predictor
3 classes: 'setosa', 'versicolor', 'virginica'

No pre-processing
Resampling: Leave-One-Out Cross-Validation
Summary of sample sizes: 149, 149, 149, 149, 149, 149, ...
Resampling results across tuning parameters:

mincriterion  Accuracy   Kappa
0.01          0.9333333  0.9
0.50          0.9333333  0.9
0.99          0.9333333  0.9

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mincriterion = 0.99.
```


6.2 Techniques d'apprentissage machine

Les algorithmes d'apprentissage sont catégorisés selon les techniques d'apprentissage qu'ils emploient. Nous en citons l'apprentissage supervisé, l'apprentissage non supervisé, l'apprentissage par renforcement et l'apprentissage semi-supervisé.

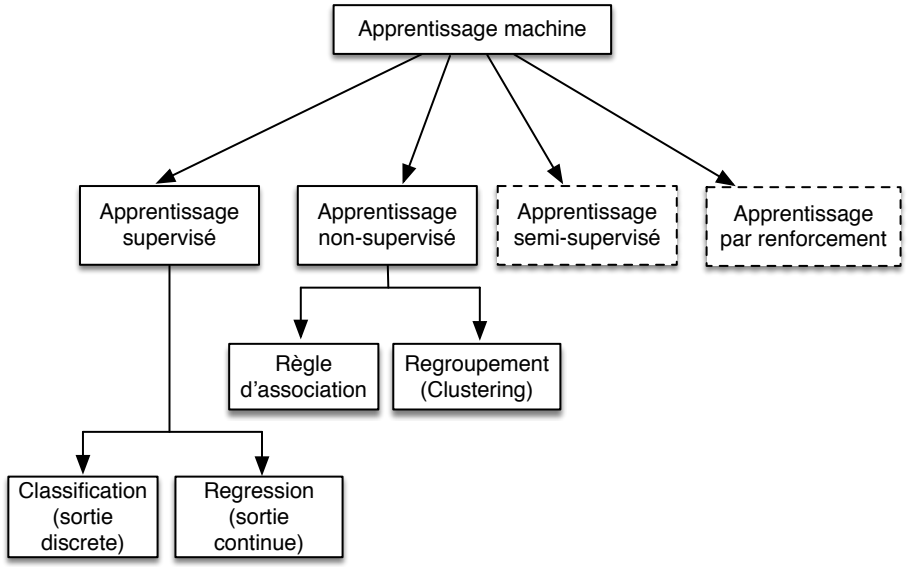

Figure 6.2: Principe de l'apprentissage machine.

Apprentissage supervisé

Dans le cas d'un apprentissage supervisé (en anglais *supervised learning*), le système observe des couples de types entrée-sortie et apprend une fonction (un modèle) qui permet d'aboutir à la sortie à partir de l'entrée. Cette phase est appelée *phase d'apprentissage* ou *d'entraînement*. C'est en ce sens que l'apprentissage est appelé supervisé, métaphore

qui signifie qu'un professeur apprend au système la sortie à fournir pour chaque entrée. Les données d'entrée et les données de sortie correspondantes, aussi appelées classes, sont connues (aussi dites labellisées ou étiquetées). Telles que décrits précédemment, elles sont regroupées dans un ensemble de données appelées *données d'apprentissage* ou *d'entrainement* qui se présentent sous la forme de couples $(\mathbf{x}_i, y_i)_{1 \leq i \leq N}$ avec $\mathbf{x}_i \in \mathbb{R}$ un vecteur de caractéristique et y_i la classe (ou étiquette ou attribut) correspondante. N étant le nombre d'échantillons.

Apprentissage non supervisé

Contrairement à l'apprentissage supervisé, dans le cas non supervisé les données de sortie ne sont pas connues. Le système apprend alors de lui-même à organiser les données ou à déterminer des structures dans les données. La tâche d'apprentissage la plus courante est le regroupement (*clustering* en anglais) qui consiste à regrouper les données d'entrées selon leurs caractéristiques communes. Ce type d'apprentissage est utilisé dans le but de visualiser ou explorer des données.

Apprentissage semi supervisé

L'apprentissage semi-supervisé se base sur un mélange de données étiquetées et non-étiquetées. Ceci permet, d'une part, d'améliorer la qualité de l'apprentissage, et d'autre part, de réduire le temps de préparation des données pour leur étiquetage.

Apprentissage par renforcement

L'apprentissage par renforcement se base sur des données d'entrée similaires à celles utilisées en apprentissage supervisé. Cependant dans ce cas, l'apprentissage est guidé par l'environnement sous la forme de récompenses (positive ou négative) calculées en fonction de l'erreur commise lors de l'apprentissage. En robotique, l'apprentissage par renforcement a permis de mettre au point des robots plus autonomes et adaptatifs à leurs environnements.

Le cours SCI1016 présente en détails les deux techniques les plus connues en apprentissage à savoir l'apprentissage supervisé et l'apprentissage non supervisé.

6.3 Classification

La reconnaissance ou classification de formes (en anglais, *pattern recognition*) est « l'ensemble des techniques permettant à l'ordinateur de détecter la présence de formes, en comparant leurs caractéristiques avec celles de motifs de référence »(Office québécois de la langue française, 2010). La classification de formes est l'une des branches de l'intelligence artificielle qui fait largement appel aux techniques d'apprentissage machine ; plus particulièrement aux réseaux de neurones.

6.3.1 Réseau de neurones

Un *réseau de neurones* aussi appelé *réseau de neurones artificiels* est un modèle mathématique très simple dérivé du neurone biologique dont les composantes élémentaires sont des unités de calcul qui reçoivent des entrées pour produire des sorties. Ce modèle reprend les principes du fonctionnement du neurone biologique, en particulier par la sommation des entrées et la pondération de la somme de ses entrées par des *poids synaptiques* (aussi appelés coefficients synaptiques) tels que illustrés à la Figure 6.3.
Un réseau de neurones est défini par sa structure (c'est-à-dire le nombre de couches, le nombre de noeuds par couche et le nombre de sorties) et par la règle d'apprentissage qui permet de calculer les poids synaptiques. La règle d'apprentissage dépend du type d'apprentissage comme nous l'avons décrit précédemment.

Il existe plusieurs types de réseau de neurones tels que le perceptron simple, le perceptron multi-couche, le réseau de Kohonen. Ces réseau se différencient aux niveaux de leur architecture et de la technique d'apprentissage sur laquelle ils se basent.

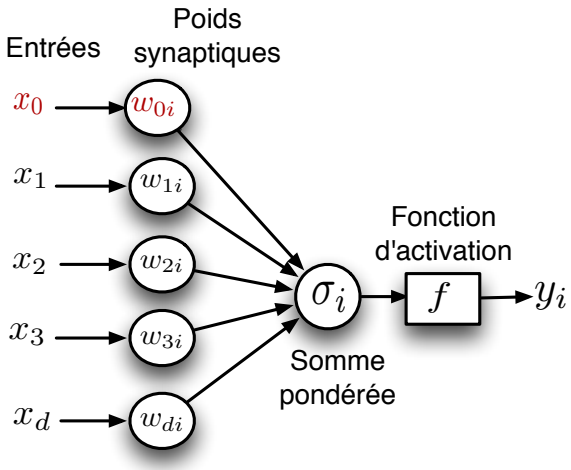

Figure 6.4: Neurone formel.

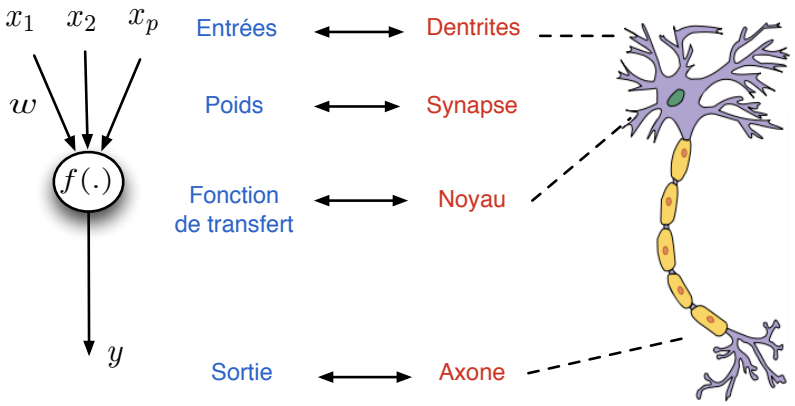

Figure 6.3: Analogie entre le neurone biologique et le neurone artificiel.

6.3.2 Évaluation d'un système de classification

Lorsque l'apprentissage machine a pour objectif de développer un système de classification de formes, la validation se fait en calculant le taux de bonne classification et à travers la matrice de confusion.

Taux de classification

Soit S un ensemble d'échantillons d'apprentissage, et T un ensemble d'échantillons de test. L'estimation du taux de bonne classification est mesurée sur l'ensemble de test selon :

$$\tau = \frac{\text{nbr bien classifié(T)}}{\text{nbr(T)}}$$

C'est-à-dire le rapport entre le nombre d'échantillons de l'ensemble de test qui sont bien classifiés par rapport au nombre total d'échantillons de la base de test.
Le taux de bonne classification est généralement donné en pourcentage. Il lui correspond une valeur complémentaire à 100 correspondant au taux d'erreur ($\epsilon(\%) = 100 - \tau(\%)$).

Matrice de confusion

La matrice de confusion est aussi connue sous les termes matrice d'erreur, tableau de contingence ou matrice d'erreur de classification. C'est une matrice affichant les statistiques de la précision de classification et plus particulièrement les taux de classification par classes. Généralement, L'information des lignes (données horizontales) correspond aux classes réelles des formes. Quant aux colonnes (données verticales), elles contiennent l'information prédite résultant de la classification.
Les valeurs de la diagonale de la matrice représentent le nombre de formes correctement classifié. La somme des valeurs par ligne correspond au nombre d'échantillons de test par classe. Le taux de classification par classes est donné par la valeur à la diagonale divisée par la somme des valeurs par ligne. Des exemples d'utilisation de la matrice de confusion pour l'évaluation de systèmes de classification seront donnés dans ce qui suit.

▶ **Exemple 5.4** Nous nous intéressons aux données `IRIS`. L'objectif de cet exemple est de développer un réseau de neurones qui permet la classification de données selon les différentes espèces en se basant les 4 variables descriptives : `Petal.Length`, `Petal.width`, `Setal.Length` et `Setal.width`. La commande `sample` (ligne 5) permet de générer des échantillons aléatoires. La ligne 7 du code suivant permet de générer une matrice nulle dans laquelle seront stockés les valeurs prédites : (1 0 0) pour la classe `setosa`, (0 1 0) pour `versicolor` et (0 0 1) pour `virginica` (lignes 8 à 11). La diagonale de la matrice de confusion donne le taux de classification par classe (`table(ciris2[,5])`)

```
1   # Réseaux de neurones
2   library(nnet);
3   data(iris)
4   # générer des valeurs aléatoires
5   samp <- c(sample(1:50,25), sample(51:100,25), sample(101:150,25))
6   x <- iris[1:4]
7   y <- matrix(0,nr=length(iris[,1]),nc=3)
8   for(i in 1:length(iris[,1])) {
9   if(iris[i,5]=='setosa') {y[i,] <- c(1,0,0)}
10  else if(iris[i,5]=='versicolor') {y[i,] <- c(0,1,0)}
```

```
    else {y[i,] <- c(0,0,1)}
    }
    # Initialisation des paramètres
    valInitWgt=0.005;
    linearOutput=FALSE;
    nbNeurons=10;
    maxLearnIter=3000;
    # Apprentissage
    rna01<-nnet(x[samp,],y[samp,], size=nbNeurons, linout=linearOutput,
    rang=valInitWgt, decay=0, maxit=maxLearnIter, Hess=TRUE);
    #Prediction sur les données de tests
    y_rna2<-predict(rna01,x[-samp,]);
    ciris2=iris[-samp,]
    clust2=matrix(0,nr=length(y_rna2[,1]),nc=1)
    for(i in 1:length(y_rna2[,1])) {
    index=which.max(y_rna2[i,])
    if(index==1) {clust2[i,1]='setosa'}
    else if(index==2) {clust2[i,1]='versicolor'}
    else {clust2[i,1]='virginica'}
    }
    table(ciris2[,5])
```

```
> table(ciris2[,5])

setosa versicolor  virginica
25         25         25
```

6.4 Regroupement

Le *regroupement*, aussi appelé *agrégation* (*clustering*), est une méthode d'apprentissage machine non supervisée, qui a pour objectif de former des groupes ou agrégats (*clusters*) d'objets similaires à partir d'un ensemble hétérogène d'objets (aussi appelées individus

ou points).
Formellement, soit un ensemble de données formé de N objets décrits par P variables. Le regroupement consiste à trouver une répartition de ces données en K groupes de sorte que les objets à l'intérieur du même groupe soient similaires. Le nombre de groupes K peut être déterminé a priori.

6.4.1 Algorithme $K-$moyennes

L'algorithme $K-$moyennes ($K-means$) utilise une technique d'affinement itératif qui consiste à améliorer progressivement la qualité des groupes selon un indice de similarité entre les différents individus. Le paramètre d'entrée de l'algorithme $K-$moyennes est le nombre de groupes désiré K. La distance euclidienne est souvent utilisée pour mesurer le rapprochement des groupes.

Algorithme $K-$moyennes : On cherche à déterminer K regroupements à partir d'un ensemble de N individus.

1. Choisir aléatirement K centres initiaux G_i avec $i = 1, 2, ...K$.
2. **Répéter**
3. Répartir chaque individu dans le groupe G_i le plus proche.
4. Recalculer les nouveaux centres G_i.
5. **Jusqu'à ce que** les centres deviennent stables.

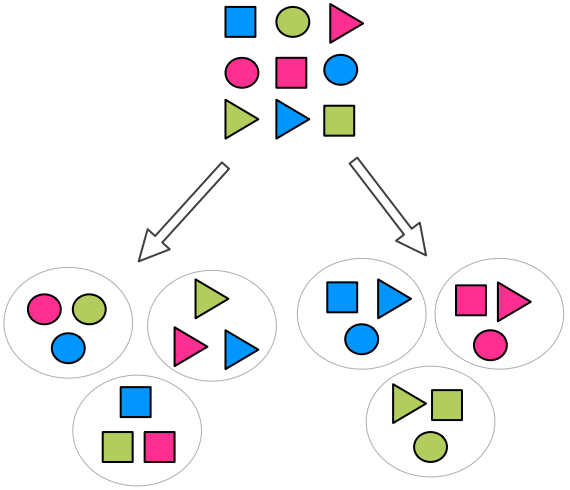

Figure 6.5: Regroupement des données en trois groupes selon la forme (à gauche) et selon la couleur (à droite).

6.4.2 Évaluation de la qualité du regroupement : Indice de silhouette

L'indice de silhouette est un indicateur efficace pour valider une méthode de regroupement. Pour tout individu i de l'ensemble des données, l'indice de silhouette est défini par

la formule suivante :

$$s(i) = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{6.1}$$

où a_i est la dissimilarité moyenne entre l'individu i et tous les autres individus du groupe G_j auquel il appartient et b_i est le minimum des dissimilarités moyennes entre l'individu i et tous les autres individus des groupes G_k $(k = 1, 2, ..., c$ avec $k \neq j)$.

▶ **Exemple 5.5** Revenons aux données `IRIS`. L'objectif de cet exemple est de faire le regroupement des différentes espèces en se basant sur la longueur (`Petal.Length`) et la largeur des pétales (`Petal.Width`). La figure 6.6 représente la variation de la largeur (`Petal.Width`) en fonction de la longueur des pétales (`Petal.Length`) pour les trois espèces. Nous pouvons déjà voir visuellement la présence d'au moins deux regroupements.

```
library(datasets)
library(ggplot2)
head(iris)
ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom_point()
set.seed(20)
irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
# Matrise de confusion
table(irisCluster$cluster, iris$Species)
# Representation des clusters
irisCluster$cluster <- as.factor(irisCluster$cluster)
ggplot(iris, aes(Petal.Length, Petal.Width, color = irisCluster$cluster)) + geom_point()
# Representation de la silhouette
library (cluster)
irisCluster <- kmeans(iris[, 3:4], 3, nstart = 20)
s <- silhouette(irisCluster$cluster,dist(iris[,3:4]))
plot(s)
```

La représentation graphique de l'indice de silhouette (Figure 6.7) montre une bonne séparation des données `IRIS` en trois groupes. Sur l'axe des ordonnées sont représentées

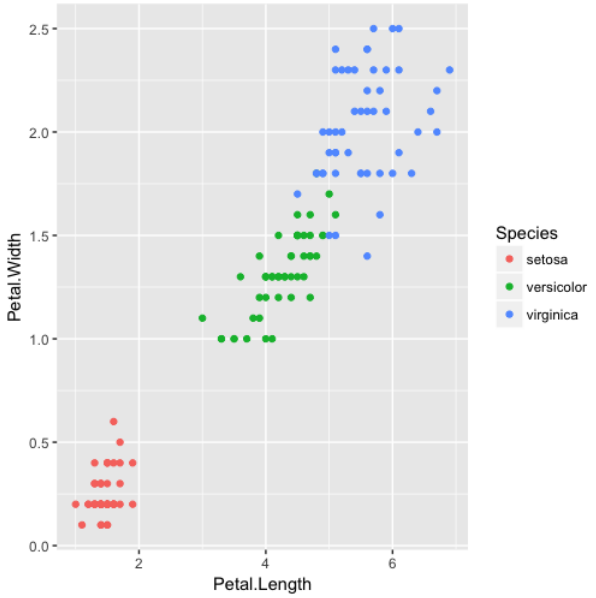

Figure 6.6: Variation de la largeur (`Petal.Width`) en fonction de la longueur des pétales (`Petal.Length`).

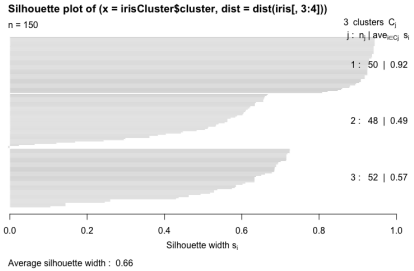

Figure 6.7: Indice de silhouette d'un ensemble de données en 3 groupes.

les données des différents groupes selon le regroupement assigné. Sur l'axe des abscisses la valeur de l'indice de silhouette calculé selon l'équation 6.1. L'indice de silhouette est positif pour tout les échantillons. Ce qui démontre l'homogénéité des données au sein des différents groupes.

6.5 Régression

Dans le cas d'un modèle de régression, le principe de l'apprentissage machine consiste à déterminer des modèles mathématiques qui permettent de représenter une variable à expliquer (ou variable dépendante ou variable endogène ou réponse) Y, en fonction d'une ou plusieurs variables explicatives (ou indépendantes ou variables exogènes) $X = (x_1, x_2, ...x_p)$. Le modèle de régression est la relation entre la variable à expliquer Y et les variables explicatives en entrée x_i. L'algorithme de régression permet de trouver le modèle en se basant sur les données d'entrainement. Le modèle calculé permet de donner une estimation sur les données de tests.

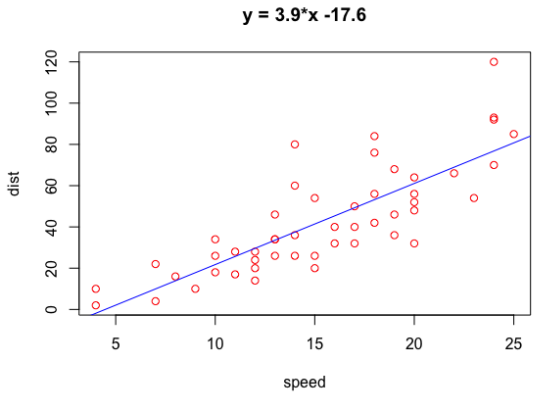

Figure 6.8: Modèle de regression linéaire.

6.5.1 Modèle de régression univarié

Le modèle de régression est dit univarié ou simple s'il n'inclut qu'une seule variable explicative. Il consiste à trouver la meilleure droite qui s'approche le plus des données d'apprentissage. Le modèle obtenu est une droite de la forme :

$$Y = f(X) = \alpha + \beta x_1 \tag{6.2}$$

avec α et β les coefficients de la droite.

▶ **Exemple 5.6** Nous traitons dans cet exemple la base de données disponible dans R dénommée `cars` (décrite dans l'Exemple 4.6). `cars` contient 50 voitures caractérisées chacune par deux variables : `dist` et `speed`. Nous désirons développer un modèle de régression qui permet de relier la variable prédictive `dist` et la variable à prédire `speed`.

```r
require(stats)
reg <- lm(cars$dist ~ cars$speed)
coeff=coefficients(reg)
# Equation de la droite de regression :
eq = paste0("y = ", round(coeff[2],1), "*x ", round(coeff[1],1))
# Graphes
plot(cars, main=eq, col="red")
abline(reg, col="blue")
```

```
> summary(reg)

Call:
lm(formula = cars$dist ~ cars$speed)

Residuals:
Min      1Q  Median      3Q     Max
-29.069  -9.525  -2.272   9.215  43.201

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -17.5791     6.7584  -2.601   0.0123 *
cars$speed    3.9324     0.4155   9.464 1.49e-12 ***
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1  1
Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-squared:  0.6511,Adjusted R-squared:  0.6438
F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

Le résultat du modèle de régression contient le modèle `dist` = α + β `speed` avec $\alpha = -17.5791$ et $\beta = 3.9324$. Selon cette équation, la distance dépend linéairement de la vitesse : la distance augmente de 3.9324 pour chaque kilomètre/h de vitesse additionnelle.

La valeur $p = 1.49e^{-12}$ étant inférieure à 0.05, nous pouvons conclure qu'Il y a donc une relation significative entre les variables du modèle de régression linéaire de l'ensemble

de données. Le valeur du coefficient de détermination R^2 permet d'évaluer la qualité de prédiction du modèle. La valeur du coefficient de détermination ajusté R^2_{adj} prend en considération le nombre de variable. Dans le cas d'une regression simple, ce coefficient correspond au rapport entre la variance expliquée et la variance totale.
La valeur de `R-squared de 0.651` est assez élevé ; ce qui permet de conclure que le modèle de régression est de bonne qualité.

Nous ouvrons une parenthèse sur la relation entre le test de significativité et le modèle de régression qui permet de tester l'apport de chaque variable dans le modèle. Le modèle à estimer a pour équation $Y = f(X) = \alpha + \beta * X$. Le test de significativité pour le coefficient β est le suivant :

$$\begin{cases} \mathrm{H}_0 : \beta = 0 \\ \mathrm{H}_1 : m \neq 0 \end{cases} \tag{6.3}$$

La règle de décision est la suivante : si la valeur absolue de la statistique observée est supérieure à la valeur théorique de la Student on rejette au seuil de α l'hypothèse nulle en faveur de l'hypothèse alternative. La variable est alors supprimé du modèle de régression.

6.5.2 Modèle de régression multivariée

Le modèle de régression est dit multivarié ou multiple s'il inclut plusieurs variables explicatives. Dans le cas de deux variables explicatives, la relation obtenue est de la forme :

$$Y = f(X) = \alpha + \beta_1 x_1 + \beta_2 x_2 \tag{6.4}$$

α est une constante et β_1 et β_2 sont les coefficients des variables explicatives.

6.6 Règles d'association

La recherche de règles d'association est une approche importante en apprentissage machine non-supervisé qui consiste à extraire des informations à partir de la coïncidence dans les données. Ce qui permet d'identifier des relations implicites qui sont cachés dans d'importante quantité de données. L'application la plus connue des règles d'association est le "panier de la ménagère" qui consiste à analyser des tickets de caisse dans les supermarchés afin de découvrir des corrélations d'évènements transactionnels. Ces corrélations sont des relations d'implication conditionnelles entres les items d'un ensemble de données qui se présentent sous la forme :

"Si l'item `X` existe, Alors il est possible que l'item `B` existe aussi". Le tableau suivant illustre un exemple d'évènements transactionnels.

Client	Liste des items achetés
1	{ pain, beurre, lait }
2	{ pain, viande }
.	.....
.	.....
n	{ Jus de fruit, pain, beurre, fraise }

6.6.1 Support et confiance

Il existe deux mesures statistiques importantes associés aux règles d'sssociation *le support* et la *confiance*.
Le support d'un ensemble d'items est la fréquence d'apparition simultanée des items figurant dans l'ensemble des données.

La confiance dans une règle $R : X \rightarrow Y$ est définie par :

$$\begin{aligned} Conf(R) &= p(Y \subseteq T | X \subseteq T) \\ &= \frac{support(X \cup Y)}{support(X)} \end{aligned}$$

ou X et Y sont des items disjoints.

6.6.2 Algorithme Apriori

L'algorithme Apriori est un algorithme itératif de recherche des itemsets les plus fréquents par niveaux. Cet algorithme fonctionne en deux phases : La première consiste à déterminer les ensembles d'items fréquents (EIF) ; Ensuite, on utilise ces EIF pour déterminer les règles d'association dont la confiance est supérieure au seuil fixé.

▶ **Exemple 5.7** Nous traitons dans cet exemple la base de données disponible dans R dénommée `Groceries` qui contient des informations sur les ventes d'une épicerie.

La base de données comprend 9835 transactions et 169 articles (`items`). L'article `whole milk` étant le plus populaire avec une vente de 2513 unités. L'utilisation de l'algorithme Apriori (`arules`) permet de ressortir 5668 règles d'association (`rules`). Les trois règles retenus sont obtenus par la commande `inspect` en fixant le niveau `n` (Ligne 7 du code).

```
1 library(arules)
2 library("arulesViz")
3 data("Groceries")
4 summary(Groceries)
5 rules <- apriori(Groceries, parameter=list(support=0.001, confidence↩
     =0.5))
6 rules
7 inspect(head(rules, n = 3, by ="lift"))
```

```
> summary(Groceries)
transactions as itemMatrix in sparse format with
9835 rows (elements/itemsets/transactions) and
169 columns (items) and a density of 0.02609146

most frequent items:
whole milk other vegetables       rolls/buns      soda      yogurt    (Other)
2513             1903             1809             1715     1372      34055

> rules
set of 5668 rules

> inspect(head(rules, n = 3, by ="lift"))
lhs                              rhs              support     confidence lift     count
[1] {Instant food products,soda} => {hamburger meat} 0.001220132 0.6315789  18.99565 12
[2] {soda,popcorn}               => {salty snack}    0.001220132 0.6315789  16.69779 12
[3] {flour,baking powder}        => {sugar}          0.001016777 0.5555556  16.40807 10
```

