MASTER EN SCIENCES INFORMATIQUES À FINALITÉ SPÉCIALISÉE EN SOFTWARE ENGINEERING

Etude et simulations numériques de méthodes de classification supervisée pour les données fonctionnelles

Gruslin, Alice

Award date:
2024

Awarding institution:
Universite de Namur

UNIVERSITE DE NAMUR

Faculté des sciences

Etude et simulations numériques de méthodes de classification supervisée pour les données fonctionnelles

Mémoire présenté pour l'obtention du grade académique
de master à finalité spécialisée en Project Engineering

Alice GRUSLIN

Janvier 2024

Remerciements

Je tiens à exprimer ma profonde gratitude envers toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce mémoire. Pour commencer, je tiens à remercier sincèrement Monsieur Germain Van Bever, mon promoteur, pour son guidage éclairé, sa patience et son soutien constant tout au long de ce projet. Merci pour vos conseils avisés et pour avoir pris le temps de relire ce mémoire avec attention. Je remercie aussi le jury qui lira ce mémoire.

Je souhaite également exprimer ma reconnaissance envers Madame Sophie Jonlet, professeure de mathématiques à l'Institut Sainte-Anne de Florenville. Vous m'avez fait découvrir et surtout apprécier les mathématiques lors de mes trois dernières années de secondaire. Je n'aurais jamais pris ce chemin sans vous.

Je n'oublie pas de mentionner le soutien indéfectible de ma famille, en particulier mes parents. Leur encouragement et leur compréhension ont été une source inestimable de motivation tout au long de ce parcours académique. J'adresse aussi un merci particulier à ma marraine.

Je remercie également mes amis pour leurs encouragements lors de la réalisation de ce mémoire, pour avoir égayé nos nombreuses sessions d'examens et pour avoir embelli toutes ces années d'études. Pour tout ça, je dirais même gloire au Cercle Math !

Enfin, je tiens à remercier profondément mon compagnon, Olivier Didier, pour son soutien inconditionnel. Merci de m'avoir épaulée lors de mes baisses de moral, de m'avoir donné la motivation d'aller jusqu'au bout et d'avoir toujours cru en moi.

Merci à tous ceux qui ont joué un rôle dans la réalisation de ce mémoire. Votre contribution a été précieuse et a grandement enrichi cette expérience.

Alice

Résumé

Ce mémoire explore l'univers complexe de la classification de données fonctionnelles, allant des principes fondamentaux de la classification aux méthodes appliquées aux données fonctionnelles. Nous commençons par établir les bases de la classification pour des données multivariées, puis nous nous plongeons dans l'analyse des données fonctionnelles par le lissage des données à l'aide d'un paramètre de rugosité. Nous explorons ensuite cinq classificateurs fonctionnels, chacun adapté à des caractéristiques spécifiques : la méthode des k plus proches voisins, la classification par arbre de décision, celle basée sur la profondeur, la classification à l'aide des fonctions noyaux et enfin une méthode de classification binaire par projection à deux dimensions. Finalement, nous évaluons les performances des classificateurs, avec une validation croisée *leave-one-out*, en ajustant différents paramètres appropriés pour une perspective plus complète sur leur efficacité. Ces évaluations numériques sont faites pour trois jeux de données différents afin d'offrir des conclusions les plus générales possibles sur les performances des classificateurs étudiés.

Mots-clés : classification, données fonctionnelles, lissage, rugosité, classificateurs fonctionnels, k plus proches voisins, arbre de décision, profondeur, fonctions noyaux, validation croisée, évaluation, performances.

Abstract

This thesis explores the complex concept of functional data classification, ranging from the fundamental aspects of classification to the methods applied to functional data. We begin by the bases of classification for multivariate data, then we do the analysis of functional data by smoothing the data using a roughness parameter. We then explore five functional classifiers, each adapted to specific characteristics : the k nearest neighbours method, decision tree classification, depth-based classification, classification using kernel functions and a binary classification method using two-dimensional projection. Finally, we evaluate the performance of the classifiers, with a cross-validation *leave-one-out*, by adjusting various appropriate parameters for complete perspective on their effectiveness. These numerical evaluations are done for three different datasets in order to offer the most general conclusions about the performance of the classifiers studied.

Key-words : classification, functional data, smoothing, roughness, functional classifiers, k nearest neighbours, decision tree, depth, kernel functions, cross-validation, evaluation, performances.

Table des matières

Table des figures

Liste des tableaux

Introduction

La classification, en tant que concept fondamental en statistique, revêt une importance cruciale dans une multitude d'applications du monde réel. Le principe sous-jacent à la classification consiste à attribuer des catégories ou des étiquettes Y à des observations X en fonction de leurs caractéristiques distinctives. Etduier des jeu de données pour déterminer ces caractéritiques permet de pouvoir associer une étiquette à de nouvelles observations dont cette dernière est justement inconnue.

Le premier chapitre de ce mémoire se consacre à la classification appliquée aux données multivariées, caractérisées par plusieurs variables explicatives. Lorsque $X = (X_1, ..., X_p)$, la complexité de la tâche de classification s'accroît, nécessitant des approches plus avancées pour extraire des informations pertinentes. Ce chapitre est une exploration des bases de la classification.

Ensuite, le deuxième chapitre élargit l'horizon vers l'analyse de données fonctionnelles, offrant une approche puissante pour les ensembles de données représentés par des fonctions continues. Les courbes, omniprésentes dans des domaines tels que la biologie, la physique, la médecine, et la finance, deviennent le sujet central de cette exploration. Les bases théoriques nécessaires à la compréhension des méthodes de classification appliquées à ces données sont établies.

Le troisième chapitre nous plonge dans les défis et les opportunités de la classification de données fonctionnelles. Il s'agit de l'exploration de différents classificateurs fonctionnels. La variété de ces méthodes, chacune adaptée à des caractéristiques spécifiques des données, est présentée avec un éclairage sur leurs avantages et inconvénients.

Enfin, le quatrième et dernier chapitre se penche sur l'aspect pratique de l'évaluation des performances des classificateurs étudiés dans le troisième chapitre. Nous réalisons des simulations numériques comparatives. Les méthodes de classification sont soumises à une série de tests pour évaluer leur robustesse et leur efficacité. Ces évaluations sont réalisées sur différents jeux de données, offrant une perspective complète sur la performance des classificateurs et facilitant la généralisation des résultats.

Chacun de ces chapitres contribue à une compréhension approfondie des concepts, des méthodes, et de leur application dans le domaine fascinant de la classification de données fonctionnelles. Ce mémoire vise à offrir de nouvelles perspectives dans la compréhension et l'exploitation des données fonctionnelles.

Chapitre 1

Classification supervisée multivariée

La classification est un concept essentiel en statistique, elle détient une importance capitale dans de nombreuses applications du monde réel. Le principe de classification consiste à attribuer des catégories ou des étiquettes Y à des observations, des données X, en fonction de leurs caractéristiques ou de leurs attributs.

Le présent chapitre se consacre à la classification pour des données multivariées, c'est-à-dire composées de plusieurs caractéristiques. Nous disposons alors de plusieurs variables explicatives pour prédire une variable cible. On note $X = (X_1, ..., X_p)$. Dans ce cas, la tâche de classification devient plus complexe et exige des approches plus avancées pour extraire des informations pertinentes. Nous pouvons distinguer deux principales approches en classification : la classification supervisée, où les modèles sont formés sur des données étiquetées, et la classification non supervisée, où les données ne sont pas étiquetées. Ce mémoire se penche sur la classification supervisée.

Nous débutons par une exploration des bases de la classification, en commençant par comprendre les règles de classification pour l'analyse discriminante et le principe de procédure optimale. Nous explorons en détail le cas gaussien et illustrons nos propos à l'aide d'exemples de données réelles. Il s'agit des sections 1.1 à 1.3 inspirées de [Van Bever, 2021]. Par la même source, nous définissons également un concept fondamental en classification à la section 1.4 : la cross-validation.

Finalement, nous présentons une seule application pour simplement introduire les méthodes de classification. Il s'agit de la méthode des k plus proches voisins (k-NN) qui émerge comme une approche simple et puissante reposant sur le principe de voisinage pour résoudre un large éventail de problèmes. Cette méthode est détaillée à la section 1.5. Elle a été rédigée à l'aide des sources [Sharma, 2021] et [Van Bever, 2021]. Nous n'explorons pas d'autres méthodes car la classification multivariée supervisée nous sert uniquement de base pour comprendre le réel intérêt de ce mémoire : les techniques de classification pour les données fonctionnelles.

1.1 Règles de classification

La classification multivariée supervisée se caractérise par l'analyse de plusieurs variables X appartenant chacune à un groupe connu π_i ($i = 1, ..., m$). L'objectif est de déterminer à quel groupe on pourrait assigner une nouvelle observation X^*. Pour ce faire, il faut construire des discriminants R_i qui caractérisent les différents groupes π_i. Les discriminants sont des fonctions mathématiques ou des seuils qui permettent de séparer les données en différentes classes dans un problème de classification. Il existe plusieurs méthodes pour définir les discriminants en fonction de l'algorithme de classification utilisé. Par exemple, en classification linéaire, on peut illustrer les discriminants par des fonctions du premier degré, des droites qui séparent les classes. Alors qu'en classification quadratique, les frontières de décision sont non linéaires, on peut les illustrer par des fonctions de degré supérieur à 1, c'est-à-dire des courbes.

Prenons l'exemple d'un célèbre jeu de données prénommé `Iris`. Ce jeu de données reprend les caractéristiques de 150 fleurs Iris. Ces caractéristiques sont la longueur et la largeur des pétales, la longueur et la largeur des sépales ainsi que l'espèce d'Iris. Il y a au total 3 espèces : Setosa, Versicolor et Virginica. Le jeu de données comprend 50 fleurs de chaque espèce. La question à se poser est la suivante :

Comment prédire l'espèce d'une nouvelle fleur d'Iris sur base de la longueur et la largeur de ses pétales ainsi que la longueur et la largeur de ses sépales ?

Pour répondre à cette question, nous devons établir des règles de classification, c'est-à-dire, effectuer une analyse discrimante. Pour ce faire, deux définitions sont nécessaires : celle d'un vecteur aléatoire et celle d'une partition d'un ensemble.

Définition 1.1.1. *Un vecteur aléatoire est une application X de Ω dans $\mathbb{R}^n$. $\forall w \in \Omega$, $X(w)$ est un élément de $\mathbb{R}^n$, c'est-à-dire $X(w) = (X_1(w), ..., X_n(w))'$, où X_i est une application de Ω dans $\mathbb{R}$.*

Définition 1.1.2. *Une partition P d'un ensemble X est un ensemble de parties non vides, disjointes et dont l'union est P. Soit $P = P_1, ..., P_n$ une partition d'un ensemble X. On a :*

- $\forall P_i \in P$, $P_i \neq \emptyset$ $(i = 1, ..., n)$;
- $\bigcup_{1=1}^{n} P_i = P$;
- $\forall P_i, P_j \in P$, $i \neq j$, $P_i \bigcap P_j = \emptyset$, $(i, j = 1, ..., n)$.

Nous pouvons à présent exprimer ce qu'est une règle de classification.

Définition 1.1.3. *Soient $X = (X_1, ..., X_p)'$ un vecteur aléatoire et $x = (x_1, ..., x_p)'$ les valeurs prises par X, considérons π_i comme une population du jeu de données $(\forall i = 1, ..., m)$. Appliquer une règle de classification revient à déterminer une partition $R = R_1, ..., R_m$ de l'ensemble des valeurs possibles pour X, telle que*

$$x \text{ est classifié en } \pi_i \iff x \in R_i$$

Pour l'exemple du jeu de données `Iris`, π_i correspond à population d'Iris d'espèce i et R_i aux attributs de l'espèce i. Dans ce cas, m vaut 3 car il y a trois espèces d'Iris dans le jeu de données.

1.2 Procédure de classification optimale

Pour rendre la procédure de classification optimale, nous devons éviter la misclassification, c'est-à-dire le fait que l'observation x ne soit pas classifiée dans la bonne population. Pour minimiser les erreurs de classification, il faut en calculer le coût. Commençons par définir quelques termes.

Soient $c_{i,j}$ le coût qu'un x provenant de la population π_j soit classifié en π_i et $p_{i,j}$ la probabilité qu'un x provenant de la population π_j soit classifié en π_i $(\forall i, j = 1, ..., m)$. Cette dernière est notée $P[x \in R_i | x \in \pi_j]$.

Nous pouvons désormais calculer le coût C_j de misclassifier un objet provenant de la population π_j, celui-ci est donnée par

$$C_j = \sum_{i=1}^{m} p_{i,j} c_{i,j}.$$

Il est alors possible de calculer ce coût pour chacune des populations de l'ensemble de données. Ceci nous permet de calculer le coût global C de toutes les misclassifications

$$C = \sum_{j=1}^{m} C_j p_j,$$
$$C = \sum_{j=1}^{m} \sum_{i=1}^{m} p_{i,j} c_{i,j} p_j,$$

avec p_j, la probabilité a priori que x provienne de la population π_j. On la note $P[x \in \pi_j]$.

Exemple 1.2.1. *Prenons le cas où l'ensemble des données se distinguent en deux populations, autrement dit $m = 2$. Le coût des misclassifications est donné par*

$$\begin{aligned} C =& \sum_{j=1}^{2} \sum_{i=1}^{2} p_{i,j} c_{i,j} p_j, \\ C =& \sum_{j=1}^{2} \left(p_{1,j} c_{1,j} p_j + p_{2,j} c_{2,j} p_j \right), \\ C =& p_{1,1} c_{1,1} p_1 + p_{2,1} c_{2,1} p_1 + p_{1,2} c_{1,2} p_2 + p_{2,2} c_{2,2} p_2, \\ C =& p_{2,1} c_{2,1} p_1 + p_{1,2} c_{1,2} p_2, \end{aligned}$$

car $c_{1,1} = c_{2,2} = 0$.

Après analyse de l'exemple 1.2.1, on remarque que le cas où $i = j$ est inutile. Il est donc plus juste de déterminer le coût global de toutes les misclassifications par

$$C = \sum_{1 \leq i \neq j \leq m} p_{i,j} c_{i,j} p_j. \tag{1.1}$$

Pour rendre la procédure de classification optimale, il faut minimiser le coût C déterminé à l'équation (1.1). Pour ce faire, nous utilisons les fonctions de densité de probabilité.

Considérons f_j la fonction de densité de probabilité associée à la population π_j $(j = 1, ..., m)$. La probabilité de classifier en π_i un objet provenant de π_j peut être exprimée grâce à cette densité, on note

$$p_{i,j} = \int_{R_i} f_j(x) dx. \tag{1.2}$$

En utilisant les équations (1.1) et (1.2), nous pouvons dire que pour classifier x en π_i, il faut que le coût suivant soit minimal

$$C = \sum_{\substack{j=1 \\ j \neq i}}^{n} f_j(x) c_{i,j} p_j.$$

Supposons que les $c_{i,j}$ sont des coûts pour lesquels nous n'avons pas d'information. Pour la suite de cette section, prenons $c_{i,j} = 1$, $\forall i \neq j$. Nous allons alors essayer de minimiser

$$C = \sum_{\substack{j=1 \\ j \neq i}}^{n} f_j(x) p_j.$$

C'est-à-dire minimiser

$$C = \sum_{j=1}^{n} f_j(x) p_j - f_i(x) p_i. \tag{1.3}$$

La règle qui minimise l'équation (1.3), et donc qui rend la procédure de classification optimale, se traduit par

$$f_i(x)p_i = \max_j \{f_j(x)p_j\}. \tag{1.4}$$

Cette règle dépend des fonctions de densité et donc des lois de probabilité. Par exemple, supposons que la probabilité a priori p_j que x provienne de la classe π_j vale $\frac{1}{m}$ pour tout $j = 1, ..., m$ avec m le nombre de classes. Cela signifie que chaque observation à la même chance d'appartenir à chacune des classes. L'équation 1.4 devient

$$f_i(x)\frac{1}{m} = \max_j \left\{ f_j(x)\frac{1}{m} \right\}.$$

Comme $\dfrac{1}{m}$ est indépendant de j, nous pouvons écrire

$$f_i(x)\frac{1}{m} = \frac{1}{m}\max_j \{f_j(x)\}.$$

Ce qui revient à l'équation

$$f_i(x) = \max_j \{f_j(x)\}.$$

Dans ce cas particulier où tous les $p_j = \frac{1}{m}$, la règle qui rend la procédure de classification optimale ne dépend plus que de la fonction de densité.

Analysons à présent un des cas les plus rencontrés, le cas gaussien. Il s'agit du cas où la loi de probabilité considérée est une loi normale.

1.3 Le cas gaussien

Commençons par définir la densité de probabilité d'une loi Gausienne, ou loi normale, mutivariée.

Définition 1.3.1. *Soit X une variable aléatoire suivant une loi normale multivariée de moyenne μ et de variance Σ, notée $N_p(\mu, \Sigma)$. La densité de probabilité de X est donnée par*

$$f(x) = \frac{1}{(2\pi)^{\frac{p}{2}}\,|\Sigma|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu)\right).$$

Comme il est optimal de classifier x en π_i si $f_i(x)p_i = \max_j \{f_j(x)p_j\}$, il est toujours optimal de classifier x en π_i si on applique le logarithme néperien à l'équation (1.4), autrement dit, si

$$ln(f_i(x)p_i) = \max_j \{ln(f_j(x)p_j)\}.$$

Nous allons à présent considérer le cas non homoscédastique et le cas homoscédastique. Définissons alors ce qu'est l'homoscédasticité.

Définition 1.3.2. *Une séquence de variables aléatoires est dite homoscédastique lorsque toutes ses variables aléatoires ont la même variance.*

1.3.1 Le cas non-homoscédastique

Le cas non homoscédastique est donc celui où la variance de l'erreur des variables est différente pour chaque population. C'est ce qui caractérise l'analyse disriminante quadratique (QDA). Comme nous sommes dans le cas gaussien, considérons $\pi_j = N_p(\mu_j, \Sigma_j)$. Par la définition 1.3.1, on a

$$\begin{aligned} ln(p_j f_j(x)) &= ln(p_j) + ln(f_j(x)) \\ &= ln(p_j) + ln\left(\frac{1}{(2\pi)^{\frac{p}{2}}\Sigma^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu)\right)\right) \\ &= ln(p_j) + ln\left(\frac{1}{(2\pi)^{\frac{p}{2}}\Sigma^{\frac{1}{2}}}\right) - \frac{1}{2}(x-\mu)'\Sigma^{-1}(x-\mu) \\ &= ln(p_j) - \frac{p}{2}ln(2\pi) - \frac{1}{2}ln(\Sigma_j) - \frac{1}{2}(x-\mu_j)'\Sigma_j^{-1}(x-\mu_j). \end{aligned}$$

Comme nous cherchons le j qui engendre le maximum de $ln(p_j, f_j(x))$, nous n'avons pas besoin du terme $-\frac{p}{2}ln(2\pi)$ étant donné qu'il est indépendant de j. Nous pouvons finalement déterminer la procédure optimale de classification pour le cas gaussien non homoscédastique.

Proposition 1.3.1. *Soit $\pi_i = N_p(\mu_i, \Sigma_i)$. Notons $d_j(x) = ln(p_j) - \frac{1}{2}ln(\Sigma_j) - \frac{1}{2}(x-\mu_j)'\Sigma_j^{-1}(x-\mu_j)$. Dans ce cas,*

$$\textit{Il est optimal de classifier } x \textit{ en } \pi_i \iff d_i(x) = \max_j \{d_j(x)\}.$$

En d'autres termes, prendre le maximum des d_j revient à prendre le minimum des distances de Mahalanobis. Quand $m = 2$, nous souhaitons classifier x en π_1 ou en π_2 selon la proposition 1.3.1. En l'occurence, nous souhaitons trouver le minimum de deux distances pour $m = 2$. Dans le cas de Σ_j différents, la surface de séparation des deux classes est alors quadratique.

1.3.2 Le cas homoscédastique

Cela correspond à l'analyse discriminante linéaire (LDA). Il s'agit du cas particulier où $\Sigma_i = \Sigma$, $\forall i = 1, ..., m$. Ainsi, on peut définir π_i comme $N_p(\mu_i, \Sigma)$. Dans ce cas, on a

$$\begin{aligned} ln(p_j f_j(x)) &= ln(p_j) + ln(f_j(x)) \\ &= ln(p_j) - \frac{p}{2}ln(2\pi) - \frac{1}{2}ln(\Sigma) - \frac{1}{2}(x-\mu_j)'\Sigma^{-1}(x-\mu_j) \\ &= ln(p_j) - \frac{p}{2}ln(2\pi) - \frac{1}{2}ln(\Sigma) - \frac{1}{2}x'\Sigma^{-1}x + \frac{1}{2}\mu_j'\Sigma^{-1}x + \frac{1}{2}x'\Sigma^{-1}\mu_j - \frac{1}{2}\mu_j'\Sigma^{-1}\mu_j \\ &= ln(p_j) - \frac{p}{2}ln(2\pi) - \frac{1}{2}ln(\Sigma) - \frac{1}{2}x'\Sigma^{-1}x + \mu_j'\Sigma^{-1}x - \frac{1}{2}\mu_j'\Sigma^{-1}\mu_j. \end{aligned}$$

En supprimant à nouveau tous les termes indépendants de j qui ne nous intéressent pas, nous déterminons un $d_j(x)$ différent de celui pour le cas non homoscédastique. Dans ce cas, nous obtenons une nouvelle proposition.

Proposition 1.3.2. *Soit $\pi_i = N_p(\mu_i, \Sigma)$. Notons $d_j(x) = ln(p_j) + \mu_j'\Sigma^{-1}x - \frac{1}{2}\mu_j'\Sigma^{-1}\mu_j$. Dans ce cas,*

$$\textit{Il est optimal de classifier } x \textit{ en } \pi_i \iff d_i(x) = \max_j \{d_j(x)\}.$$

Comme pour le cas non-homoscédastique, nous cherchons le minimum des distances de Mahalanobis. Lorsque $m = 2$ et que les Σ_j sont identiques $\forall j$, la surface de séparation des deux classes π_1 et π_2 est linéaire.

1.3.3 Exemple de données réelles

Pour illustrer la théorie de la classification multivariée supervisée, prenons à nouveau l'exemple du jeu de données `Iris`, présenté à la section 1.1. Pour plus de facilité, nous utilisons uniquement les 100 dernières observations, c'est-à-dire uniquement les observations concernant les Iris d'espèce Versicolor et d'espèce Virginica. Deux classifications sont réalisées. La première pour une analyse discriminante linéaire dont les résultats sont à la FIGURE 1.1. Il s'agit du cas homoscédastique à deux dimensions. La seconde classification est pour une analyse discriminante quadratique. C'est le cas non homoscédastique à deux dimensions. Le résultat est présenté à la FIGURE 1.2. Ces classifications sont réalisées à l'aide du logiciel RStudio.

Sur les FIGURES 1.1 et 1.2, les étoiles roses correspondent aux observations des Iris considérées comme étant d'espèce Versicolor et les étoiles vertes aux observations des Iris considérées comme étant d'espèce Virginica. Les observations entourées en rouge sont les observations misclassifiées, c'est-à-dire celles pour lesquelles la classe prédite est mauvaise.

A la FIGURE 1.1, quatre observations de la population Versicolor sont misclassifiées. Il s'agit des observations 19, 21, 23 et 34. L'observation 84 de la population Virginica est également misclassifiée. Nos pouvons résumer cette classification à l'aide de sa table de contingence.

Définition 1.3.3. *Une table de contingence est un tableau statistique qui permet de présenter simultanément 2 séries statistiques afin d'en croiser les informations.*

La table de contingence de la classification pour l'analyse discriminante linéaire est réprésentée à la TABLE 1.1.

En ce qui concerne la FIGURE 1.2, seulement 3 observations sont misclassifiées. Ces observations sont communes au cas linéaire. En effet, les observations 21 et 34 ne sont pas correctement classifiées pour l'espèce Versicolor et l'observation 84 n'est pas correctement classifiée pour l'espèce Virginica. Nous pouvons également dresser sa table de contingence. Elle est regprésentée à la TABLE 1.2.

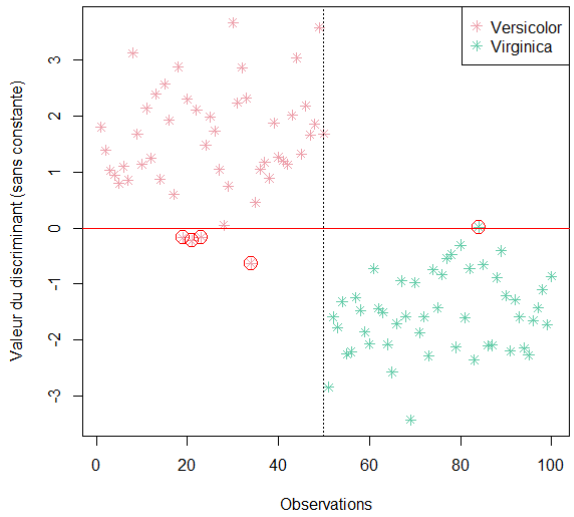

FIGURE 1.1 – Classification d'une partie du jeu de données `Iris` pour une analyse discriminante linéaire.

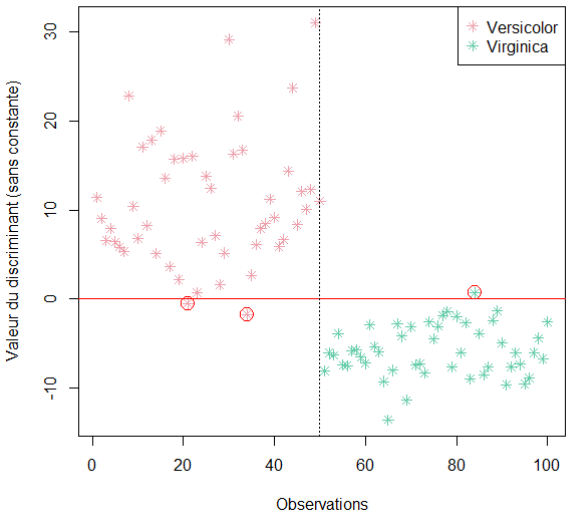

FIGURE 1.2 – Classification d'une partie du jeu de données `Iris` pour une analyse discriminante quadratique.

	Versicolor	Virginica
Versicolor	46	4
Virginica	1	49

TABLE 1.1 – Table de contingence de la classification des espèces d'Iris pour l'analyse discriminante linéaire.

	Versicolor	Virginica
Versicolor	48	2
Virginica	1	49

TABLE 1.2 – Table de contingence de la classification des espèces d'Iris pour l'analyse discriminante quadratique.

Les tables de contingence liées aux graphes de misclassifications nous permettent de déterminer le pourcentage d'erreur de classification du jeu de données. Nous estimons le taux d'erreur de classification par

$$\hat{e} = \frac{\sum_{i=1}^{m} n_{i,e}}{\sum_{i=1}^{m} n_i},$$

avec

- m, le nombre de populations du jeu de données ;
- n_i, le nombre d'observations de la population i avec $\sum_i n_i = n$ (le nombre total d'observations) ;
- $n_{i,e}$, le nombre d'observations de la population i qui ne sont pas correctement classifiées.

Pour l'exemple présenté, le taux d'erreur est donc de 5% pour le cas de l'analyse discriminante linéaire et de 3% pour le cas de l'analyse discriminante quadratrique.

1.4 Cross validation leave-one-out

Dans le cas du calcul du taux d'erreur à la section précédente, l'estimation $\hat{e}$ a tendance à sous-estimer le taux d'erreur. En effet, ce sont les mêmes observations qui déterminent la règle de classification et qui l'évaluent. Cela peut donner l'impression d'avoir une performance élevée, car les caractéristiques spécifiques de cet ensemble de données ont été prises en compte. Cependant, cette performance peut ne pas se généraliser à de nouvelles données. Une solution pour éviter ce genre de sous-estimation est la cross validation. Le principe est de déterminer une règle de classification à l'aide de certaines observations et d'évaluer cette règle sur base d'autres observations. La FIGURE 1.3 illustre ce concept.

Le principe de la méthode est d'utiliser une partie des observations pour définir un échantillon appelé échantillon d'entraînement. Celui-ci permet de définir la règle de classification. Ensuite, un second échantillon est défini. Il s'agit de l'échantillon de test. Comme nous traitons la cross validation leave-one-out, cet échantillon ne contient qu'une observation.

Pour commencer, c'est l'observation 1 qui sert de test. On regarde alors si la règle déterminée par l'échantillon d'entraînement, soit par les $n-1$ observations restantes, engendre une missclassification de l'observation 1. On répète ensuite ce procédé pour chaque observation de chaque population i. Le nombre de misclassifications ainsi engendrées pour les observations de la population i est noté $n_{i,e}^{l}$.

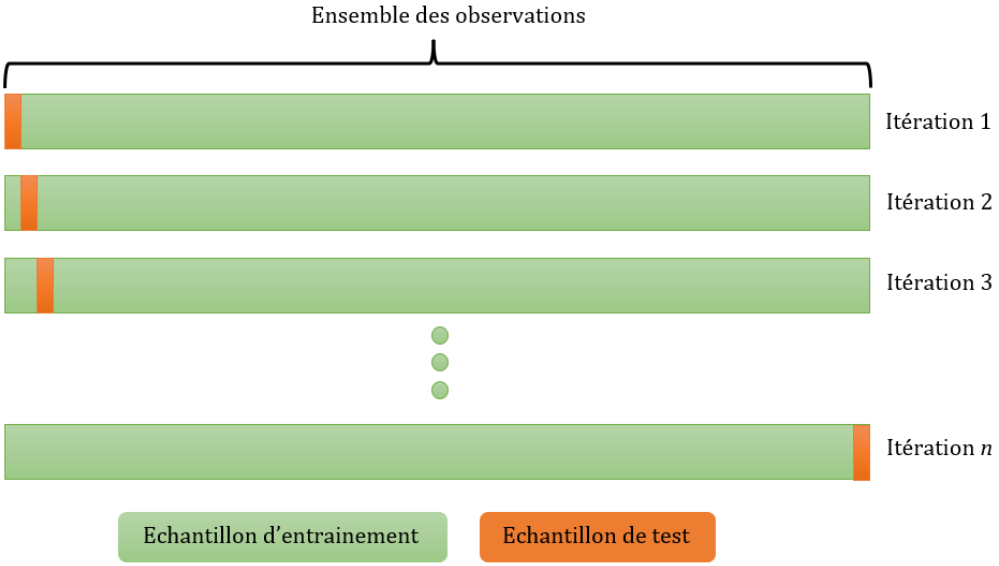

FIGURE 1.3 – Principe de cross-validation.

Dans ce cas, on estime le taux d'erreur de classification par cross validation leave-one-out comme

$$\hat{e}^l = \frac{\sum_{i=1}^{p} n_{i,e}^l}{\sum_{i=1}^{p} n_i}.$$

L'avantage de cette cross validation leave-one-out est l'utilisation totale des données pour définir la règle de classification. De plus, contrairement à $\hat{e}$, l'estimateur leave-one-out $\hat{e}^l$ ne sous-estime pas le taux d'erreur de clasification. En revanche, dans le cas où on étudie des grands ensembles de données, le temps de calcul est coûteux étant donné qu'on répète n fois la procédure, en considérant n comme étant la taille de l'échantillon. Cela est d'autant plus coûteux lorsque la quantité de données est importante. Dans ce cas, nous considérons la k-fold cross validation. Le principe est le même que la cross validation leave-one-out à la seule différence que nous considérons k plis (de plusieurs observations) plutôt qu'une seule observation à la fois. Le modèle est alors formé k fois, chaque fois en utilisant k-1 plis pour l'apprentissage et le pli restant pour la validation. Cela permet d'obtenir k estimations du taux d'erreur, dont la moyenne est souvent utilisée comme estimation finale. Cette approche réduit significativement le coût en terme de temps de calcul par rapport à la cross validation leave-one-out tout en utilisant encore une grande partie des données. Cependant, le choix de k peut être délicat. En effet, un k trop faible peut conduire à une estimation instable, c'est-à-dire que de légères modifications dans la composition des plis peuvent avoir un impact significatif dans les performances du modèle. A l'inverse, un k trop élevé implique qu'une partie importante des données peut ne pas être pleinement exploitée.

1.4.1 Exemple de données réelles

Nous allons procéder à la classification pour l'analyse discriminante linéaire et pour l'analyse discriminante quadratique dans le cas de la cross validation leave-one-out pour le jeu de données `Iris`. Ces classifications sont représentées respectivement aux figures 1.4 et 1.5. Nous comparons ensuite les résultats à ceux de la section 1.3.3.

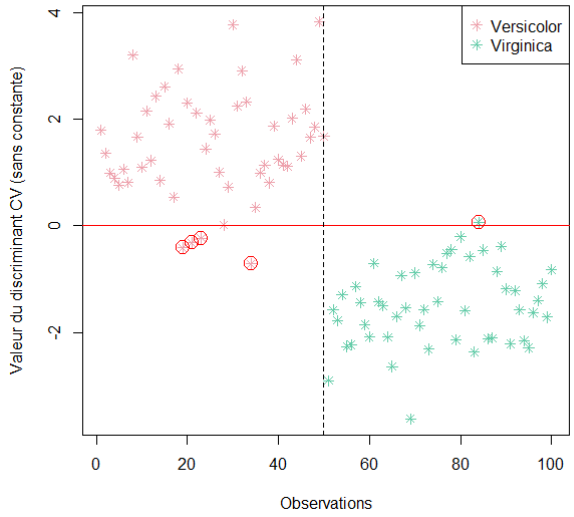

FIGURE 1.4 – Classification d'une partie du jeu de données `Iris` pour une analyse discriminante linéaire avec cross validation.

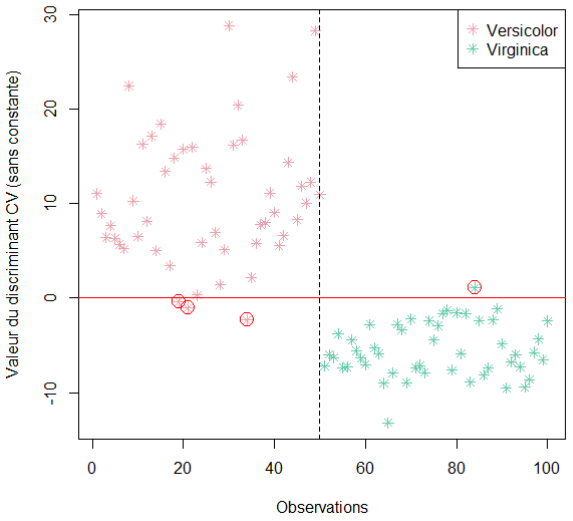

FIGURE 1.5 – Classification d'une partie du jeu de données `Iris` pour une analyse discriminante quadratique avec cross validation.

Pour la FIGURE 1.4, les observations misclassifiées sont exactement les mêmes que pour la FIGURE 1.1. Il s'agit des observations 19, 21, 23, 34 et 84. Il y a donc toujours 5% d'erreur de classification. En revanche, si on observe correctement la FIGURE 1.4, on remarque que les observations misclassifiées ne se trouvent pas exactement aux mêmes endroits que celles de la FIGURE 1.1. Il est inutile de faire la table de contingence pour cette classification car elle est identique à la TABLE 1.1.

En ce qui concerne le cas de l'analyse discriminante quadratique, la FIGURE 1.5 ne présente pas les mêmes résulats que la FIGURE 1.2. En effet, lorsqu'on utilise la cross validation, les observations 21, 34 et 84 sont toujours misclasifiées mais ce n'est pas tout. L'observation 19, faisant partie de l'espèce Versicolor, est également considérée comme misclassifiée. Ce n'était pas le cas pour la FIGURE 1.2. La table de contingence de cette classification est la TABLE 1.3.

	Versicolor	Virginica
Versicolor	47	3
Virginica	1	49

TABLE 1.3 – Table de contingence de la classification pour l'analyse discriminante quadratique avec cross validation.

Dans ce cas, le taux d'erreur a bel et bien été sous-estimé à la section 1.3.3 pour la classification pour l'analyse discriminante quadratique. Avec la cross validation leave-one-out, ce taux d'erreur est de 4%.

1.5 Méthode k-NN

Une des méthodes de classification les plus connues est la méthode k-NN. Elle porte ce nom car il s'agit du diminutif de k-Nearest-Neighbours, c'est-à-dire la méthode des k plus proches voisins. C'est une méthode de classfication supervisée non paramétrique simple. Cela signifie que cette technique de classification s'appuie sur des données d'apprentissage pour prendre des décisions de classification et non sur des hypothèses strictes. Nous considérons à nouveau un ensemble d'observations p-dimensionnelles X_i appartenant à des classes connues π_j ($\forall i = 1, ..., n$ et $\forall j = 1, ..., m$).

L'idée est alors de classer une nouvelle observation x à l'aide de celles qu'on connait déjà, c'est-à-dire à l'aide de ses observations voisines. Pour ce faire, nous commençons par regarder les k plus proches données de cette nouvelle observation, ainsi que les classes auxquelles elles appartiennent. On note $X_{(i)}(x) \in \{X_1, ..., X_n\}$ le $i^{\text{ème}}$ plus proche voisin de x parmi les X_i $\forall i = 1, ..., n$. On note

$$||x - X_{(1)}(x)|| \leq ||x - X_{(2)}(x)|| \leq ... \leq ||x - X_{(n)}(x)||.$$

On associe alors à chaque $X_{(i)}(x)$ un $Y_{(i)}(x)$ représentant sa classe. Si la majorité des voisins appartient à la population j, alors le nouveau point de données sera classifié comme étant également de la population j. Afin de déterminer ces k plus proches voisins, nous utilisons donc une notion de distance.

Définition 1.5.1. *Une distance est une fonction $d : E \times E \longrightarrow \mathbb{R}_+$, avec E un ensemble non vide. Cette fonction respecte les propriétés de séparation, de symétrie et de l'inégalité trianglaire.*

- $\forall x_1, x_2 \in E$, $d(x_1, x_2) = 0 \Longleftrightarrow x_1 = x_2$,
- $\forall x_1, x_2 \in E$, $d(x_1, x_2) = d(x_2, x_1)$,
- $\forall x_1, x_2, x_3 \in E$, $d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3)$.

La distance la plus utilisée est la distance euclidienne, exprimée à la définition 1.5.2. C'est celle que nous utilisons. Il s'agit d'une distance qui se réfère aux vecteurs à valeurs réelles. Elle mesure la distance entre deux points en ligne droite. Nous aurions pu utiliser d'autres distances métriques pour définir les k plus proches voisins d'une observation, comme par exemple la distance de Minkowski qui est une forme généralisée de la distance euclidienne.

Définition 1.5.2. *La distance euclidienne est une fonction $d : E \times E \longrightarrow \mathbb{R}_+$ telle que $\forall x, y \in E$, avec E un espace de dimension n,*

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

En pratique, il n'existe pas de k optimal. En effet, prendre $k = 1$ signifie qu'un nouveau point de données sera automatiquement de la même classe que son plus proche voisin. Or, si ce plus proche voisin est misclassifié, le résultat ne sera pas cohérent. D'autre part, si k est trop grand, les plus proches voisins peuvent être trop éloignés de la nouvelle observation. Dans ce cas, celle-ci risque à nouveau d'être mal classifiée.

Par exemple, la FIGURE 1.6, tirée directement de l'article [Sharma, 2021], montre de façon très simplifiée l'importance de bien choisir k. En prenant $k = 4$, il est difficile de classer la pièce de puzzle bleue car il y a autant de voisins *soleils* que de voisins *visages*. L'observation est alors assignée de façon aléatoire dans une des deux classes. En revanche, en choisissant $k = 11$, 7 voisins sont des *visages* alors que seulement 4 sont des *soleils*. Dans ce cas, la différence est flagrante et la pièce de puzzle sera classifiée dans la population des *visages*. Tout ça pour dire qu'il faut choisir k en fonction du jeu de données traité. Nous disons alors que le k optimal est celui qui minimise l'erreur de classification estimée par cross validation.

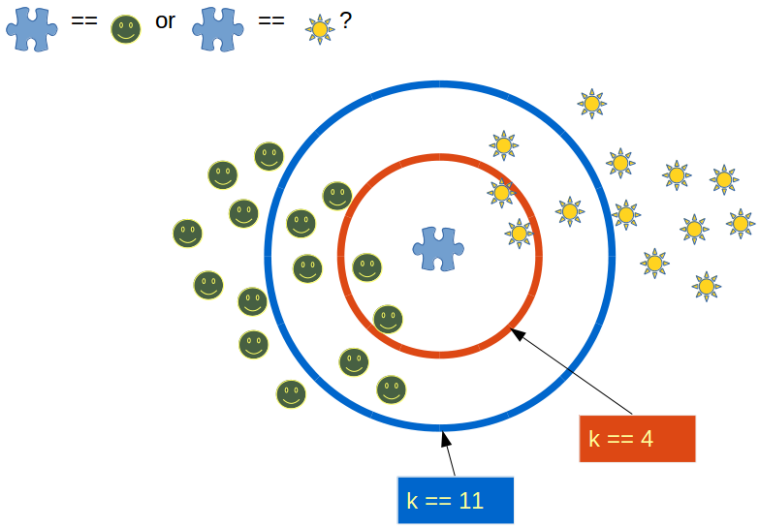

FIGURE 1.6 – Classification de données fictives par la méthode des k plus proches voisins. [Sharma, 2021]

Cette classification à l'aide des k plus proches voisins a plusieurs avantages. Premièrement, l'algorithme est simple et facile à mettre en oeuvre. C'est un des classificateurs de base. Ensuite, peu d'hyperparamètres sont nécessaires. Contrairement aux paramètres du modèle, les hyperparamètres sont les paramètres appris à partir des données d'entraînement. Ils sont choisis en fonction de la performance souhaitée du modèle. Pour la méthode k-NN, on a seulement besoin du paramètre k et d'une notion de distance. De plus, chaque nouvelle donnée est stockée en mémoire. L'algorithme s'adapte donc facilement pour les prochaines données en tenant compte des nouvelles informations en mémoire. En revanche, le fait de stocker toutes ces données implique qu'on ait besoin d'une grande mémoire. Cela est coûteux en terme de temps, l'algorithme va s'exécuter plus lentement. Un autre incovénient de la méthode est le fait qu'elle soit sensible aux caractéristiques non pertinentes. Quand on a un nombre de caractéristiques optimal, en ajouter entraine une augmentation des erreurs de classification. Il faut également rester vigilant car la méthode k-NN est de plus en plus inéfficace quand on considère un grand ensemble de données. Cette méthode est utilisée dans plusieurs domaines, notamment le prétraitement de données, les moteurs de recommandations, la finance, la santé ou encore la reconnaissance de formes.

1.5.1 Consistance de Bayes

Un résultat important de la classification k-NN est la convergence vers l'estimateur de Bayes. Cette section est inspirée de la source [Younso and Azhari, 2022]. Tout d'abord, considérons $(x_i,y_i) \in \mathbb{R}^p \times \mathcal{Y}$ la réalisation de variables indépendantes et identiquement distribuées (X_i, Y_i) où $\mathcal{Y} = \{1, ..., m\}$ représente les m classes des observations x_i. L'objectif d'une classification est de construire des classificateurs $g : \mathbb{R}^p \longrightarrow \mathcal{Y}$ qui minimise la probabilité d'erreur $P_{(X,Y)}[g(X) \neq Y]$. On peut traduire cette probabilité par le risque attendu $R(g) = \mathbb{E}_{(X,Y)}[\mathbb{1}_{g(X)\neq Y}]$. Un classificateur aléatoire pour deux classes $\mathcal{Y} = \{1, 2\}$ peut se traduire par une probabilité d'erreur (un risque attendu) de $\frac{1}{2}$. En se concentrant sur des classificateurs non triviaux, on défini le classificateur de Bayes par

$$g^*(x) = \begin{cases} 2 & \text{si } \eta(x) \geq \dfrac{1}{2}, \\ 1 & \text{sinon.} \end{cases}$$

avec η une fonction connue. On peut alors définir le risque de Bayes par le lemme 1.5.1. La démonstation de se lemme est disponible dans [Gaillard, 2018].

Lemme 1.5.1. *Le risque du classificateur de Bayes est*

$$R^* \triangleq R(g^*) = \mathbb{E}[\min(\eta(x), (1 - \eta(x))],$$

De plus, pour tout classifcateur g, on a

$$R(g) - R* = \mathbb{E}[|2\eta(x) - 1|\mathbb{1}_{g(X)\neq g^*(X)}] > 0.$$

On dit alors que le classificateur de Bayes R^* est optimal car

$$R^* = \min_{g:\mathbb{R}^p \longrightarrow \{1,2\}} R(g).$$

C'est ce qu'on appelle un estimateur consistant.

Pour la classification k-NN, on peut définir $\hat{\eta}_n^k(x)$ un estimateur de η.

$$\hat{\eta}_n^k(x) = \frac{1}{k}\sum_{i=1}^{k} Y_i(x)$$

On peut alors construire $\hat{g}_n^k$ un estimateur du classificateur k-NN. Le risque asymptotique du classificateur k-NN est donné par

$$R_{k-\text{NN}} \triangleq \lim_{n\longrightarrow\infty} \mathbb{E}_{(X_i,Y_i)}[R(\hat{g}_n^k].$$

Finalement, par le théorème de Stone (1.5.1), on peut dire que le classificateur k-NN converge vers l'estimateur optimal de Bayes. La preuve de ce théorème se trouve dans [Gaillard, 2018].

Théorème 1.5.1. ***Stone 1964.*** *Si $k \longrightarrow \infty$ et $\frac{k}{n} \longrightarrow 0$, alors le classificateur k-NN est universellement consistant. On a*

$$R_{k-NN} \triangleq \lim_{n\longrightarrow\infty} \mathbb{E}_{(X_i,Y_i)}[R(\hat{g}_n^k] = R^*.$$

La consistance universelle de k-NN assure que, sous certaines conditions, l'estimateur k-NN devient de plus en plus précis à mesure que la taille de l'échantillon augmente, fournissant ainsi une estimation convergente vers la vraie structure sous-jacente des données.

1.5.2 Exemple de données réelles

Nous allons à présent illustrer cette méthode des k-plus proches voisins avec le jeu de données `Iris` que nous utilisons depuis le début de ce chapitre. A nouveau, par facilité, nous ne considérons que les Iris d'espèces Versicolor et Virginica. Pour commencer, k a été determiné avec le logiciel RStudio. En effet, nous avons regardé parmi $k = 1, ..., 30$ lequel donnait le moins d'erreurs de classification. Il s'agit du cas où $k = 6$. En classifiant les observations à l'aide de leur 6 plus proches voisins, seulement 5 observations sont misclassifiées. Il s'agit des observations entourées en rouge sur la FIGURE 1.7. Ce graphique a été réalisé à l'aide du logiciel RStudio et le code correspondant a été inspiré des codes du chapitre 5 de la source [Van Bever, 2021].

Les données misclassifiées sur la FIGURE 1.7 sont les observations 21 et 28 pour la population Versicolor et les observations 70, 84 et 85 pour la population Virginica. Il est intéressant de comparer les observations misclassifiées de la méthodes k-NN avec celles de la méthode d'analyse discriminante. Les observations 21 et 84 sont communes alors que les autres ne le sont pas. Par

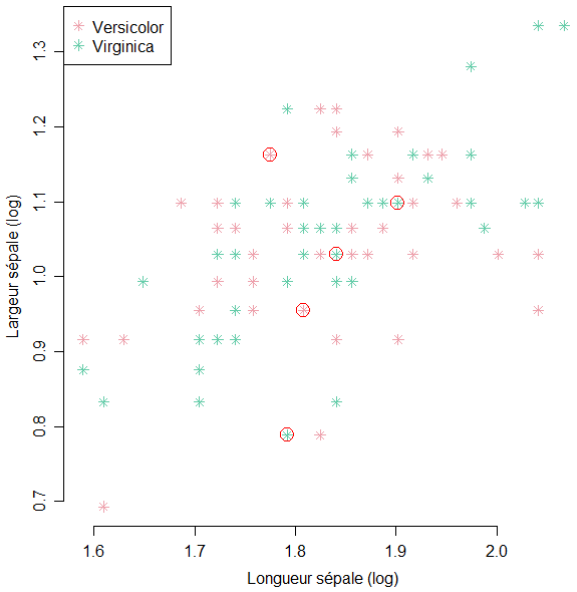

FIGURE 1.7 – Classification d'une partie du jeu de données Iris à l'aide de la méthode des k plus proches voisins.

exemple, il y a plus d'observations misclassifiées dans la population Virginica avec la méthode k-NN. Nous pouvons conclure que les classifications sont différentes en fonction des méthodes. Nous pouvons établir la table de contingence des misclassifications à l'aide de la méthode des k plus proches voisins. Il s'agit de la table TABLE 1.4.

	Versicolor	Virginica
Versicolor	48	2
Virginica	3	47

TABLE 1.4 – Table de contingence de la classification par la méthode des k plus proches voisins.

Il est également important de souligner le fait que la FIGURE 1.7 est visuellement moins parlante que la FIGURE 1.6 car la procédure se fait dans $\mathbb{R}^4$. En effet, nous analysons quatre caractéristiques des Iris. Cette représentation en deux dimensions ne présente que la longueur et la largeur des sépales alors que nous considérons également la longueur et la largeur des pétales.

Chapitre 2

Concepts théoriques

L'analyse de données fonctionnelles offre une approche puissante pour traiter des ensembles de données où les observations sont représentées sous la forme de fonctions continues. Dans ce second chapitre, nous établissons les bases théoriques nécessaires à la compréhension des méthodes de classification appliquées à ce type de données.

Nous débutons par une introduction aux données fonctionnelles, explorant leur modélisation sous forme de fonctions continues. Un aspect crucial de la classification des données fonctionnelles réside dans la mesure de la similarité entre différentes fonctions. Nous introduisons des outils permettant de capturer les relations entre les fonctions tout en tenant compte de leur nature continue. Nous abordons les semi-métriques pour qualifier les distances entre les fonctions, la notion de profondeur pour évaluer la position relative d'une fonction par rapport à un ensemble de données ou encore les fonctions noyaux permettant de traiter des relations non linéaires entre les données.

Nous définissons également un concept fondammental dans la théorie des données fonctionnelles : l'Analyse en Composantes Principales Fonctionnelles (ACPF). C'est une extension de l'Analyse en Composantes Principales (ACP) adaptée à l'analyse de données fonctionnelles. Contrairement à l'ACP traditionnelle qui traite des variables scalaires, l'ACPF s'applique lorsque les observations sont des fonctions continues dépendant d'une variable (par exemple, des courbes temporelles, des trajectoires spatiales, des spectres).

Ce chapitre établit ainsi les bases théoriques nécessaires à une exploration approfondie des méthodes de classification dans le domaine des données fonctionnelles, jetant les fondations indispensables à la compréhension du chapitre suivant, où nous appliquons ces notions pour définir des méthodes de classification.

2.1 Introduction aux données fonctionnelles

La première section de ce chapitre consiste à expliquer ce que sont les données fonctionnelles. Elle est inspirée des sources [Ramsay et al., 2005] et [Ferraty and Vieu, 2006]. Commençons par déterminer ce qu'est une variable fonctionnelle.

Définition 2.1.1. *Une variable aléatoire fonctionnelle est une variable aléatoire X qui prend ses valeurs dans un espace de dimension infinie,*

$$X : \Omega \longrightarrow \mathcal{F},$$

avec Ω un ensemble et $\mathcal{F}$ un espace fonctionnel.

Pour ce mémoire, nous considérons $\mathcal{F}$ comme l'espace de L^2 de dimension finie. Quelques définitions son nécessaire pour définir ce sous-espace L^2. Commençons par définir une suite de Cauchy afin d'exprimer ce qu'est un espace complet pour arriver à la définition d'un espace de Hibert.

Définition 2.1.2. *Une suite (x_n) d'éléments sur un espace vectoriel V muni de la norme $||.||$ est dite de Cauchy si $\forall \epsilon > 0, \exists n_\epsilon \in \mathbb{N}$ tel que $\forall n, m \geq n_\epsilon, ||x_n - x_m|| < \epsilon$.*

Définition 2.1.3. *Un espace vectoriel V est complet (de Banach) si toute suite de Cauchy sur $(V, ||.||)$ est convergente.*

Définition 2.1.4. *Un espace de Hilbert est un espace verctoriel V équipé d'un produit scalaire $< ., . >$ et complet pour la norme $< x, x >^{\frac{1}{2}}$.*

Une fois que l'espace de Hilbert est défini, nous pouvons donner la définition d'un espace L^2.

Définition 2.1.5. *L'espace $L^2(\tau)$ est l'ensemble des fonctions x mesurables sur τ telles que $\int_\tau x^2(t)dt < \infty$. C'est un espace de Hilbert séparable muni du produit scalaire $< x, y >_2 = \int_\tau x(t)y(t)dt$ et dont la norme est $||x||_2 = \left(\int_\tau x^2(t)dt\right)^{\frac{1}{2}}$.*

L'espace fonctionnel choisi et défini, nous pouvons donner la définition d'une donnée fonctionnelle.

Définition 2.1.6. *Une donnée fonctionelle est une réalisation x d'une variable fonctionnelle X. Un ensemble de données fonctionnelles $x_1, ..., x_n$ est donc une réalisation de n variables fonctionnelles $X_1, ..., X_n$.*

Premièrement, une variable aléatoire peut être observée à plusieurs moments dans un intervalle $[t_{min}, t_{max}]$. Une observation peut donc être exprimée en fonction du temps, c'est-à-dire par une famille aléatoire $X = \{X(t_j)\}_{j=1,...,J}$. Comme il y a énormément de possibilités dans cet intervalle de temps, une observation peut être exprimée par la famille continue $X = \{X(t) : t \in [t_{min}, t_{max}]\}$. Pour obtenir alors des données fonctionnellles et non discrètes, nous procédons à un lissage des données, expliqué à la sous-section (2.1.1).

Deuxièmement, il est possible que la caractéristique fonctionnelle provienne directement des observations, c'est-à-dire que X désigne une courbe aléatoire. On note $X = \{X(t) : t \in T\}$. Lorsque nous analysons des courbes, $T \in \mathbb{R}$ est un ensemble unidimensionnel. Dans le cas où on analyse des surfaces, $T \in \mathbb{R}^2$ est un ensemble bidimensionnel. Il est également possible de traiter des ensembles de dimension infinie. L'observation d'une telle variable aléatoire est désignée par $x = \{x(t) : t \in T\}$. Pour ce mémoire, nous traitons uniquement du cas où $T \in \mathbb{R}$, c'est-à-dire des courbes.

L'objectif de l'utilisation de données fonctionnelles est tout d'abord de rendre l'analyse de ces données la plus simple possible. Ce format peut permettre de déterminer les sources de certaines variations des données ou encore de mettre en évidence certaines caractéristiques pas forcément flagrantes au premier abord. De plus, l'utilisation des données fonctionnelles est alors intéressante pour comparer des observations dont la discrétisation initiale est différente.

2.1.1 Lissage des données

Afin de rendre fonctionnel un jeu de données discrétisées, nous allons procéder à un lissage des données. Pour détailler ce concept, nous nous sommes renseignés sur la documentation RStudio et nous avons également consulté la source [Bayart, 2011].

Pour commencer, les objets de données fonctionnelles sont construits à partir de combinaisons linéaires de fonctions de base. En effet, l'objectif est de condtruire une fonction $x(t)$ sur des mesures exactes $x(t_j)$ $\forall j$ pour chaque observation. La première idée est de relier les points. C'est ce qu'on appelle l'interpolation linéaire. Ce n'est évidemment pas une bonne solution, d'autant plus s'il y a des erreurs de mesure. On souhaite alors interpoler les données avec des polynômes. Pour cela, nous avons besoin de l'espace fonctionnel $\mathcal{F}$, ici L^2 et pour s'assurer de la proximité de l'interpolation, nous avons besoin d'une distance et d'une norme, en l'occurence la norme $||.||_2$.

Soit $\{\phi_k\}$ une base de L^2. Les éléments de L^2 sont linéairement indépendants tels qu'il existe des constantes $\{c_{k,K}\}$ $\forall x \in L^2$ telles que

$$\left\| x - \sum_{k=1}^{K} c_{k,K}\phi_k \right\|_2 \longrightarrow 0, \qquad \text{pour } k \longrightarrow \infty.$$

K est fixé et correspond au nombre de fonctions de base. Prendre $K = N$ n'a pas de sens car cela revient à un ajustement parfait et donc aucun lissage. L'objectif est de trouver un petit nombre K de fonctions de base. Les seules éléments à stocker sont les coefficients c_k. On souhaite les choisir tels que l'approximation de $x(t_i)$ soit la meilleure, c'est-à-dire que

$$\left(\sum_{i=1}^{N} \left[x(t_i) - \sum_{k=1}^{K} c_k\phi_k(t_i) \right]^2 \right)^{\frac{1}{2}}$$

soit minimal. Notons que l'on peut négliger l'exposant $\frac{1}{2}$.

Le choix de la base est primordial. Un première exemple de base est la base simple, c'est-à-dire la base des monômes. Le choix de cette base est assez mauvais. Les fonctions de base sont données par

$$\phi_k(t) = t^k, k > 0.$$

Ensuite, une base populaire est la base de Fourier. Elle représente un outil pour les fonctions périodiques et les fonctions de base sont les suivantes

$$\psi_{2k}(t) = cos(2k\pi t) \text{ ou } \psi_{2k+1}(t) = sin(2k\pi t), \qquad \text{pour } k = 0, 1, 2, ...$$

Pour ce mémoire, nous nous intéressons aux bases b-splines. Il s'agit d'une méthode stable et efficace numériquement permettant d'ajuster une courbe lisse. Elle est bien meilleure que les bases classiques comme la base des monômes ou celle de Fourier.

Commençons par considérer une fonction $x(t)$ univariée. Le but est de construire une courbe paramétrique dans $\mathbb{R}^2$. La moyenne pondérée entre deux éléments c_0 et c_1 du plan est donnée par $c = (1 - \lambda)c_0 + \lambda c_1$ avec $\lambda \in [0, 1]$ et est considérée comme stable car le résultat sera toujours entre c_0 et c_1. La ligne entre ces deux points est appelée enveloppe convexe. Entre n points, on aura

$$c = \sum_{i=1}^{n} \lambda_i c_i \quad \text{ avec } \sum_{i=1}^{n} \lambda_i = 1 \text{ et } \lambda_i > 0.$$

Si la courbe qui ajuste les points $c_0, ..., c_n$ se trouve dans son enveloppe convexe, on considère qu'elle est stable.

Le segment entre c_0 et c_1 peut également être donné par la courbe paramétrique

$$q(t|c_0, c_1, t_0, t_1) = \frac{t_1 - t}{t_1 - t_0}c_0 + \frac{t - t_0}{t_1 - t_0}c_1, \quad \forall t \in [t_0, t_1].$$

Notons $q(t|c_0, c_1, t_0, t_1)$ par $q_{0,1}(t)$ avec 0 en référence au point de départ c_0 et 1 pour le degré de la courbe. On appelle $q_{0,1}(t)$ une combinaison convexe.

Il est possible de généraliser cela à trois points c_0, c_1 et c_2. On note $q(t|c_0, c_1, c_2, t_0, t_1, t_2)$ par $q_{0,2}(t)$ et on obtient

$$q_{0,2}(t) = \frac{t_2 - t}{t_2 - t_0} q_{0,1}(t) + \frac{t - t_0}{t_2 - t_0} q_{1,1}(t), \quad \text{avec } t_0 < t_1 < t_2.$$

Cela correspond à une courbe quadratique à travers c_0, c_1 et c_2. C'est bien une interpolation.

On peut généraliser le procédé pour $d + 1$ points. Pour ce faire, nous devons interpoler $(c_i)_{i=0}^{d-1}$ et $(c_i)_{i=1}^{d}$ puis procéder à la combinaison convexe suivante

$$q_{0,d}(t) = \frac{t_d - t}{t_d - t_0} q_{0,d-1}(t) + \frac{t - t_0}{t_d - t_0} q_{1,d-1}(t).$$

Notons que cette courbe passe par les points de données puisque c'est une interpolation et donc $q_{0,d}(t_k) = c_k$.

Dans le cas où $q_{0,1}(t)$ ou $q_{1,1}(t)$ n'est pas une combinaion convexe, on choisit le même intervalle $[t_0, t_1]$ pour les deux courbes. Si par exemple $[t_0, t_1] = [0, 1]$, on a

$$\begin{aligned} p_{0,1} &= p(t|c_0, c_1) = (1 - t)c_0 + tc_1, \\ p_{1,1} &= p(t|c_1, c_2) = (1 - t)c_1 + tc_2, \\ p_{0,2} &= p(t|c_0, c_1, c_2) = (1 - t)p_{0,1}(t) + tp_{1,1}(t). \end{aligned}$$

C'est ce qu'on appelle une courbe de Bézier. Contrairement aux combinaisons convexes, les courbes de Bézier ne passent pas par les c_i, il ne s'agit pas d'une interpolation. On peut généaliser cela à $(c_i)_{i=0}^{d-1}$ et $(c_i)_{i=1}^{d}$ avec les courbes de Bézier $p_{0,d-1}(t)$ et $p_{1,d-1}(t)$ respectivement. On a

$$p_{0,d} = (1 - t)p_{0,d-1}(t) + tp_{1,d-1}(t).$$

Quand d est grand, c'est très complexe à calculer. Dans ce cas, on va considérer des courbes de Bézier composées. L'idée est de joindre plusieurs courbes de Béziers de faible degré (et donc plus facile à calculer) les unes à la suite des autres. La fonction composée obtenue porte également le nom de courbe de Bézier et est donnée par

$$f(t) = \sum_{i=1}^{M} p_{0,d}^{(i)} \frac{t - t_i}{t_{i+1} - t_i} \mathbb{1}\{t \in [t_i, t_{i+1}[]\},$$

pour $n = Md + 1$ points.

Nous devons alors choisir l'ordre des b-splines qui composent cette courbe de Bézier. Le plus souvent, l'ordre choisi est 4, ce qui correspond à des splines cubiques, dont la définition est la 2.1.7. Notons que les bases b-spline sont utilisées pour des fonctions non périodiques.

Définition 2.1.7. *Soient x_i, y_i des nombres réels pour tout $i = 1, ..., n$ ($x_i < x_{i1}$). Une spline cubique associée à la famille (x_i, y_i) est une fonction S de classe C^2, polynomiale de degré au plus trois sur chaque intervalle $[x_i, x_{i+1}[$ telle que $S(x_i) = y_i \forall i = 1, ..., n$ avec les dérivées premières et seconde de S continues.*

Cependant, cela créé des points de rupture pour lesquels nous devons procéder à un lissage. Pour avoir un polynôme final lisse, les valeurs de polynômes adjacents doivent correspondre, ainsi que leurs dérivées jusqu'à l'odre $m - 1$ avec m le degré des polynômes par morceaux.

Ensuite, nous devons ajuster les données à partir d'un ensemble de courbes lisses déterminées à partir des fonctions de base. Le critère d'ajustement est le critère des moindres carrés pondérés. Il permet de sélectionner la fonction qui reproduit le mieux les données expérimentales. Considérons $y = x + e$ avec x la mesure exacte et e l'erreur telle que $var(e) = \sigma^2 I$. Soient $\Phi \in \mathbb{R}^{N\times K}$ et $c \in \mathbb{R}^K$ pour $x = \Phi c$. On a $\hat{c}$ l'estimateur de $\hat{x} = \Phi\hat{c}$ par les moindres carrés. $\hat{x}$ est construit tel que la somme des carrés de résidus

$$SSR = \sum_{i=1}^{n}(y(t_i) - \hat{x}(t_i))^2$$

est minimale afin d'obtenir le meilleur ajustement possible.

Cependant, ce qui nous intéresse aussi est de minimiser la somme des carrés des erreurs

$$SSE = \sum_{i=1}^{n}(x(t_i) - \hat{x}(t_i))^2.$$

Pour savoir s'il est possible de minimiser à la fois le SSR et le SSE, nous nous intéressons au théorème suivant et à son lemme (non démontrés dans le cardre de ce mémoire).

Théorème 2.1.1. *Soit $y \in \mathbb{R}^N$. Notons S le sous espace engendré par $\phi_1, ..., \phi_K$, des vecteurs indépendants de $\mathbb{R}^N$. L'élément $\hat{y} \in S$ le plus proche de y en terme de distance est caractérisé par le fait que $(y - \hat{y})$ soit orthogonal à S, c'est à dire que $< y - \hat{y}, \phi_i >= 0$, $\forall i = 1, ..., K$.*

Lemme 2.1.1. ***Lemme du théorème 2.1.1****. L'estimateur des moindres carrés $\hat{x}$ est caractérisé par $< y - \hat{x}, \phi_i >= 0$ $\forall i$ et satisfait*

$$SSE = ||x - y||^2 - SSR.$$

Le lemme 2.1.1 nous informe qu'en diminuant le SSR pour obtenir un ajustement correct, on augmente le SSE et donc les erreurs. On souhaite alors un compromis entre les deux.

Nous définissons alors le lissage avec la pénalité de rugosité. Cette pénalité est une mesure quantitative de la rugosité d'une courbe qui s'adapte aux données. Une mesure de pénalité populaire est la courbure intégrée. Soit D^k l'opérateur différentiel d'ordre k pour $k = 1, 2, ...$ La mesure de pénalité est définie par

$$Pen_k^\lambda(x) = \lambda \int_0^1 [(D^k x)(t)]^2 dt,$$

avec λ le paramètre de rugosité ou de lissage. Ce paramètre permet de contrôler l'importance accordée à l'ajustement des données par rapport au lissage. Plus le paramètre de rugosité est grand, plus la courbe sera lisse mais moins les données seront ajustées et inversement.

Finalement, on veut minimiser

$$||y - \Phi c||^2 + Pen_k^\lambda\left(\sum_{j=0}^{K} c_j\phi_j\right).$$

Dans le cadre lu lissage des données, nous devons être vigilent à propos de l'overfitting, ou surajustement en français. Il se produit lorsqu'un modèle d'apprentissage s'adapte trop précisément aux données d'entraînement. En conséquence, le modèle donne de très bons résultats sur l'ensemble de données d'entraînement, car il a mémorisé les détails spécifiques de cet ensemble. Cependant, lorsque le modèle est confronté à de nouvelles données, sa performance est mauvaise, car il a du mal à généraliser au-delà des exemples spécifiques d'entraînement. Pour faire le lien avec la rugosité,

un modèle lissé est moins sujet à l'overfitting car il tente de capturer les tendances générales plutôt que les détails spécifiques.

Pour illustrer le lissage des données, nous prenons l'exemple du jeu de données `EuStockMarkets`, celui-ci est représenté à la FIGURE 2.1. Ce jeu de données contient les cours de clôture quotidiens des principaux indices boursiers européens : DAX (allemand), SMI (suisse), CAC (français) et FTSE (britannique) de 1991 à 1998. Les données sont échantillonnées en heures ouvrables, c'est-à-dire que les week-ends et les jours fériés ne comptent pas.

Pour obtenir des données fonctionnelles, nous commençons par analyser un ensemble de n observations discrétisées. Il est alors intéressant de regarder si ces observations reflètent une variation lisse de ce qu'on observe et si l'allure est la même d'une courbe à l'autre. Pour commencer, nous considérons des splines cubiques, ainsi qu'un paramètre de lissage nul. En ce qui concerne le nombre de fonctions de base, plus il est élevé, plus le résultat sera proche des données discrétisées. Pour le jeu de données sur les indices boursiers, nous montrons un exemple de lissage des données avec 20 fonctions de base à la FIGURE 2.2 et un exemple avec 10 fonctions de base à la FIGURE 2.3.

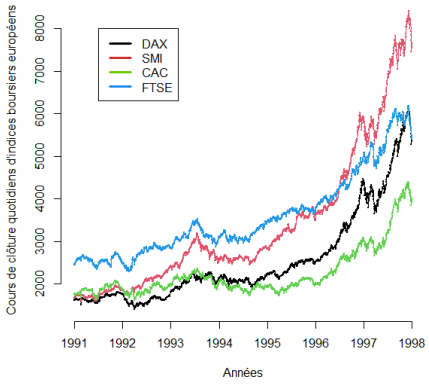

FIGURE 2.1 – Cours de clôture quotidiens des principaux indices boursiers européens de 1991 à 1998 : données discrétisées.

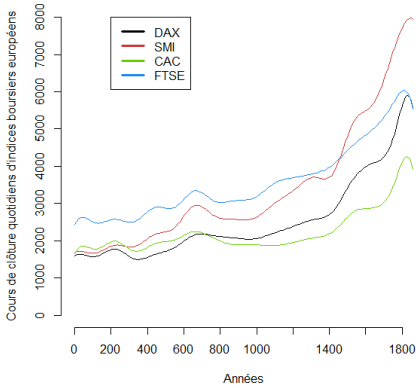

FIGURE 2.2 – Cours de clôture quotidiens des principaux indices boursiers européens de 1991 à 1998 : données lissées (20 fonctions de base) sans paramètre de rugosité.

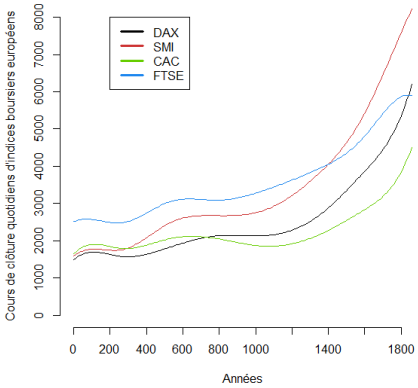

FIGURE 2.3 – Cours de clôture quotidiens des principaux indices boursiers européens de 1991 à 1998 : données lissées (10 fonctions de base) sans paramètre de rugosité.

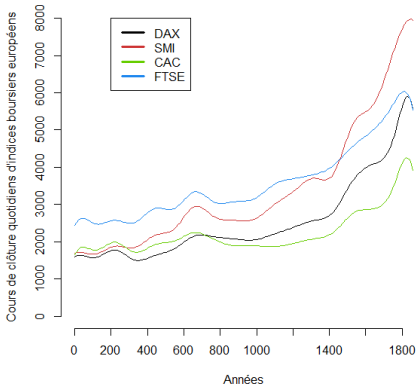

FIGURE 2.4 – Cours de clôture quotidiens des principaux indices boursiers européens de 1991 à 1998 : données lissées (20 fonctions de base) sans paramètre de rugosité.

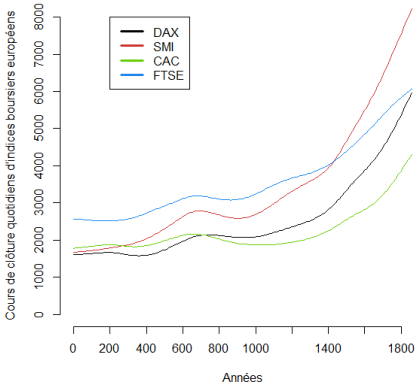

FIGURE 2.5 – Cours de clôture quotidiens des principaux indices boursiers européens de 1991 à 1998 : données lissées (20 fonctions de base) avec paramètre de rugosité λ =20 000 000.

Ensuite, nous pouvons faire un constat sur le paramètre de lissage. En effet, à la FIGURE 2.4 nous observons les indices boursiers pour des données lissés avec 20 fonctions de base sans pénalité de rugosité et à la FIGURE2.5, le nombre de fonctions de base est identique mais nous avons ajouté un paramètre de lissage élevé soit λ =20 000 000. On en conclut que plus le paramètre de rugosité est élevé, plus les données sont lisses et plus ce dernier est faible, plus l'ajustement est proche des données discrétisées, d'où l'intéret de trouver un compromis entre le nombre de fonctions de base et le paramètre λ.

2.2 Semi-métriques

Dans le contexte de la classification de données fonctionnelles, la notion de semi-métrique peut être utilisée pour modéliser les similarités ou les distances entre différentes courbes. Les données fonctionnelles peuvent être complexes, avec des variations sur l'ensemble des courbes. Une semi-métrique permet une certaine flexibilité en relâchant certaines propriétés strictes des métriques, comme l'inégalité triangulaire. Cela peut être important car la notion de distance peut être plus subjective dans le contexte de données fonctionnelles.

Définition 2.2.1. *Une semi-métrique ou semi-distance est une fonction $d : E \times E \longrightarrow \mathbb{R}_+$, avec E un ensemble non vide. Cette fonction respecte les propriétés de séparation et de symétrie.*

- $\forall x_1, x_2 \in E,\ d(x_1, x_2) = 0 \Longleftrightarrow x_1 = x_2,$
- $\forall x_1, x_2 \in E,\ d(x_1, x_2) = d(x_2, x_1).$

Contrairement à la notion de distance (définition 1.5.1), l'inégalité triangulaire n'est pas obligatoire pour une semi-métrique. La propriété de symétrie d'une semi-métrique implique que deux courbes sont similaires uniquement selon leurs caractéristiques, peu importe si l'une est au-dessus de l'autre ou inversément.

Notons $x_i^{(a)}(t)$ la différentiation d'ordre a de la courbe $x_i(t)$. La semi-métrique des dérivées de deux covariables fonctionnelles $x_i^{(a)}(t)$ et $x_j^{(a)}(t)$ avec $i \neq j$ est donnée par $d(x_i^{(a)}(t), x_j^{(a)}(t))$. Des exemples de semi-métriques que l'on peut utiliser sont repris à la TABLE 2.1. Ce sont les plus utilisées et les plus connues. La TABLE 2.1 est directement traduite et issue de la source [Fuchs et al., 2017].

Semi-métriques	**La semi métrique est basée sur ...**
$d^{Eucl}(x_i^{(a)}(t), x_j^{(a)}(t)) = \sqrt{\int_T \left(x_i^{(a)}(t) - x_j^{(a)}(t)\right)^2 dt}$	la distance absolue entre les dérivées de deux courbes.
$d_\tau^{Scan}(x_i^{(a)}(t), x_j^{(a)}(t)) = \sqrt{\int_T \left(\phi_\tau(t)\left(x_i^{(a)}(t) - x_j^{(a)}(t)\right)\right)^2 dt}$	les distances absolues des profils pondérés des dérivées de courbes centrées autour de τ avec $\phi_\tau(t)$ une fonction de balayage appropriée (par exemple un noyau gaussien).
$d_{T_{small}}^{shortEucl}(x_i^{(a)}(t), x_j^{(a)}(t)) = \sqrt{\int_{T_{small}} \left(x_i^{(a)}(t) - x_j^{(a)}(t)\right)^2 dt}$	la distance absolue sur une partie limitée du domaine de définition $T_{small} \subset T$ des dérivées des deux courbes.
$d^{Mean}(x_i^{(a)}(t), x_j^{(a)}(t)) = \left\lvert \int_T x_i^{(a)}(t)dt - \int_T x_j^{(a)}(t)dt \right\rvert$	la similarité des valeurs moyennes des dérivées des courbes entières.
$d^{relAreas}(x_i^{(a)}(t), x_j^{(a)}(t)) = \left\lvert \left\lvert \frac{\int_{T_1} x_i^{(a)}(t)dt}{\int_{T_2} x_i^{(a)}(t)dt} \right\rvert - \left\lvert \frac{\int_{T_1} x_j^{(a)}(t)dt}{\int_{T_2} x_j^{(a)}(t)dt} \right\rvert \right\rvert$	la similarité de la relation des zones sur les parties du domaine de définition $T_1, T_2 \subset T$.
$d_{bo}^{Jump}(x_i(t), x_j(t)) = \lvert (x_i(t_b) - x_i(t_0)) - (x_j(t_b) - x_j(t_0)) \rvert$	la similarité des hauteurs de saut aux points $t_b, t_0 \in T$.
$d^{Max}(x_i^{(a)}(t), x_j^{(a)}(t)) = \left\lvert max\left(x_i^{(a)}(t)\right) - max\left(x_j^{(a)}(t)\right) \right\rvert$	la différence des maximas globaux des dérivées des courbes.
$d^{Min}(x_i^{(a)}(t), x_j^{((a)}(t)) = \left\lvert min\left(x_i^{(a)}(t)\right) - min\left(x_j^{(a)}(t)\right) \right\rvert$	la différence des minimas globaux des dérivées des courbes.
$d^{Points}(x_i^{(a)}(t), x_j^{(a)}(t)) = \frac{1}{E}\sum_{e=1} E \left\lvert x_i^{(a)}(t_e) - x_j^{(a)}(t_e) \right\rvert$	la différence à certains points d'observation t_e appelés points d'impact.

TABLE 2.1 – Différentes semi-métriques utilisées pour déterminer l'ensemble des k plus proches voisins. [Fuchs et al., 2017]

2.3 Profondeur

2.3.1 Idée de profondeur

La profondeur d'une observation est un outil statistique développé par John Tukey, un célèbre statiticien américain du XX^{me} siècle. La notion de profondeur statistique permet de situer une observation parmis l'entièreté de sa population. Ainsi, pour déterminer la profondeur d'une observation nous vérifions sa proximité avec la médiane statistique.

Définition 2.3.1. *La médiane statistique d'un jeu de données correspond au point central de celui-ci. C'est-à-dire que la moitié des observations se trouve au dessus de la médiane et l'autre moitié en dessous. Mathématiquement, considérons S une série statistique quantitative discrète d'effectif total $N \in \mathbb{N}$ définie par $S = \{x_i\}_{i=1}^{n}$.*

$$\textit{Médiane}(S) = \begin{cases} x_{p+1} & \textit{si } N \textit{ est impair avec } N = 2p+1 \quad \forall p \in \mathbb{N}, \\ \dfrac{x_p + x_{p+1}}{2} & \textit{si } N \textit{ est pair avec } N = 2p \quad \forall p \in \mathbb{N}. \end{cases}$$

De ce fait, une observation est dite de profondeur maximale lorsqu'elle correspond à la médiane. Il est alors possible d'avoir deux observations de profondeur maximale si le nombre total d'observations du jeu de données est pair. On peut également dire que la profondeur d'une observation correspond à son plus petit rang parmis les deux possibles (lorsque les observations sont triées par ordre croissant ou décroissant). Un définition de la profondeur dans $\mathbb{R}^d$ est donnée à la définition 2.3.2.

Définition 2.3.2. *Soit $C_n = x_1, ..., x_n avec y_i \in \mathbb{R}^d$. Une profondeur mesure la centralité d'un quelconque $\theta \in \mathbb{R}^d$ par rapport à C_n.*

$$\begin{array}{rccc} D(., C_n) & : & R^d & \to & [0,1] \\ & & \theta & \mapsto & D(\theta, C_n) \end{array}$$

Au plus $D(\theta, C_n)$ est grand, au plus θ est centré par rapport à C_n.

Par exemple, dans $\mathbb{R}^3$, la profondeur de Tukey représente la proportion de $x_i \in \mathbb{R}^3$ qui se trouvent à l'intérieur d'une sphère centrée sur θ par rapport au nombre total de x_i de l'ensemble C_n. Pour déterminer le rayon de la sphère, on peut calculer la distance de chaque point du jeu de données par rapport à θ puis calculer la médiane de ces distances. Ensuite, on multiplie la médiane des distances par une constante (le facteur de réduction ou de dilatation) pour obtenir le rayon de la sphère. Cette constante peut être ajustée en fonction de nos besoins.

2.3.2 Idée de profondeur de bande pour des données fonctionnelles

Dans la même idée que la profondeur expliqué à la section 2.3.1, la profondeur fonctionnelle permet de déterminer si une observation est représentative de sa classe, c'est-à-dire regarder si cette observation est plutôt au centre ou vers l'extérieur du groupe.

Pour définir la profondeur, il faut d'abord expliquer quelques concepts. Considérerons l'ensemble L^2 des fonctions réelles continues x_i $(i = 1, ..., n)$. Nous pouvons alors déterminer le graphe G d'une fonction x. Celui-ci est donné par

$$G(x) = \{(t, x(t)) : t \in T\}, \quad \text{avec } T \text{ un intervalle } [t_{min}, t_{max}].$$

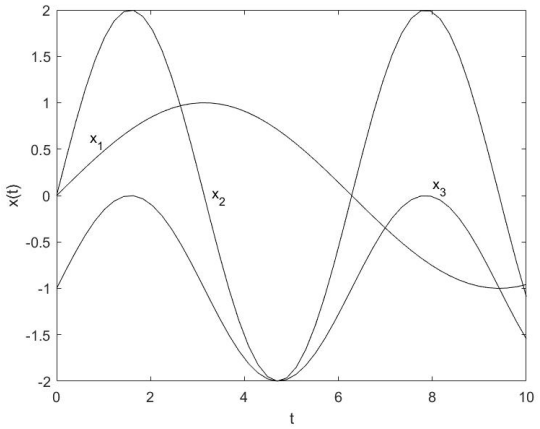

FIGURE 2.6 – Représentation de trois courbes x_1, x_2, x_3.

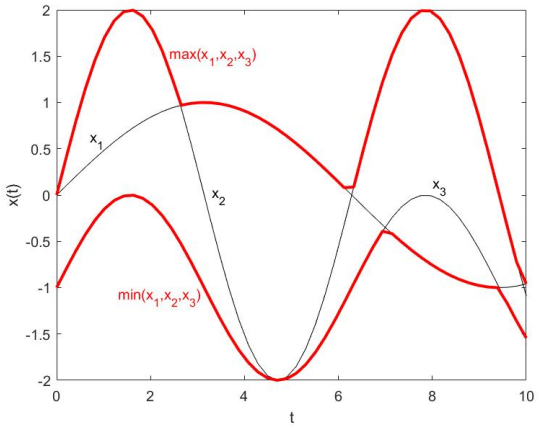

FIGURE 2.7 – Représentation de trois courbes x_1, x_2, x_3 ainsi que la bande déterminée par ces courbes.

Ensuite, il est facile de déterminer une bande entre deux courbes sur un graphe. Il est alors possible de déterminer une bande à l'aide de k courbes, avec $k \geq 2$. Il suffit de passer d'une courbe à l'autre quand celles-ci se croisent afin de former une bande qui contient entièrement les k courbes x_i considérées. Par exemple, sur la FIGURE 2.6, on ne distingue pas directement la bande déterminée par les trois courbes x_1, x_2 et x_3. Les courbes qui construisent la bande formée par x_1, x_2 et x_3 sont représentées en rouge à la FIGURE 2.7. Les FIGURES 2.6 et 2.7 ont été réalisées à l'aide du logiciel Matlab. Celles-ci correspondent aux trois fonctions suivantes ($\forall t \in [0, 10]$) :

$$\begin{aligned} x_1(t) &= sin(t/2); \\ x_2(t) &= 2 * sin(t); \\ x_3(t) &= sin(t) - 1. \end{aligned}$$

La bande ainsi déterminée par k courbes $x_{i_1}, ..., x_{i_k}$ s'exprime par

$$V(x_{i_1}, ..., x_{i_k}) = \left\{ (t, x(t)) \ : \ t \in T, \min_{r=1,...,k} x_{i_r}(t) \leq x(t) \leq \max_{r=1,...,k} x_{i_r}(t) \right\}.$$

Nous pouvons donc déterminer des bandes à l'aide de j courbes x_i et vérifier si ces bandes contiennent le graphe d'une nouvelle observation x ($j \geq 2$). La proportion de telles courbes est donnée par

$$S^j(x) = \binom{n}{j}^{-1} \sum_{1 \leq i_1 \leq ... \leq i_j \leq n} \mathbb{1}(G(x) \subset V(x_{i_1}, ..., x_{i_j})), \tag{2.1}$$

avec $\binom{n}{j}$ le nombre de combinaisons possibles pour choisir j courbes parmi les n existantes et $\mathbb{1}(G(x) \subset V(x_{i_1}, ..., x_{i_j}))$ une indicatrice qui vaut 1 si le graphe de x est inclus dans la bande formée par les j courbes considérées et 0 sinon.

Pour n'importe quelle courbe x, on définit la profondeur de la bande par

$$S^J(x) = \sum_{j=2}^{J} S^j(x). \tag{2.2}$$

Nous pouvons à présent déterminer la courbe $\hat{m}_{S^J}$ qui a la valeur de profondeur la plus élevée. Celle-ci est appelée fonction médiane d'échantillon et est notée

$$\hat{m}_{S^J} = \underset{x \in \{x_1,\ldots,x_n\}}{\operatorname{argmax}} S^J(x). \tag{2.3}$$

Les courbes profondes sont celles qui sont les plus représentatives de l'échantillon, celles qui ont les caractéristiques principales de l'ensemble. A l'inverse, les courbes les moins profondes sont celles qui s'éloignent le plus de l'échantillon et ne sont donc pas très représentatives de celui-ci.

2.3.3 Idée de profondeur de bande généralisée

Pour généraliser cette notion de profondeur de bande, nous allons décortiquer les courbes. En effet, au lieu de vérifier si l'entièreté de la fonction x se trouve dans la bande déterminée par les k courbes considérées, nous allons regarder précisément quels points de x se trouvent dans cette bande. Pour ce faire, nous ne nous intéressons plus à $V(x_{i_1}, ..., x_{i_k})$ mais à un ensemble $A_k(x)$ défini de la façon suivante

$$A_k(x) = \left\{ t \in T \ : \min_{r=i_1,\ldots,i_k} x_r(t) \leq x(t) \leq \max_{r=i_1,\ldots,i_k} x_r(t) \right\}.$$

Nous pouvons alors adapter les équations (2.1), (2.2) et (2.3). En effet, en considérant $\lambda A_j(x)$ comme la proportion de temps où x est à l'intérieur de la bande formée par un ensemble de $j \geq 2$ courbes, on généralise l'équation (2.1) par

$$GS^j(x) = \binom{n}{j}^{-1} \sum_{1 \leq i_1 \leq \ldots \leq i_j \leq n} \lambda A_j(x).$$

L'équation (2.2) peut à son tour être généralisée.

$$GS^J(x) = \sum_{j=2}^{J} GS^j(x).$$

Finalement, nous pouvons adapter la fonction médiane (2.3) à l'idée de profondeur de bande généralisée. On a

$$\hat{m}_{GS^J} = \underset{x \in \{x_1,\ldots,x_n\}}{\operatorname{argmax}} GS^J(x).$$

Ainsi, nous pouvons déterminer les courbes les plus profondes de façon précise sur certains intervalles plutôt que sur l'entièreté du graphe.

2.4 Mesures statistiques de sélection d'attribut

Une mesure de sélection d'attribut est utilisée dans le domaine de l'exploration de données pour identifier les caractéristiques (attributs) qui sont les plus informatives ou discriminantes par rapport à la classification qui nous intéresse. Le gain d'information et l'indice de Gini sont des mesures statistiques de sélection d'attribut. C'est-à-dire que ces mesures permettent d'établir des règles dans le but de diviser les données afin de réduire la dimension d'un jeu de données en un sous-ensemble de variables pertinentes.

2.4.1 Entropie et gain d'information

Commençons par parler d'entropie. L'entropie est ce qui détermine le caractère aléatoire des données. Nous pouvons également définir mathématiquement l'entropie.

Définition 2.4.1. *Soient S un ensemble fini de données, m le nombre de classes dans S et p_i la probabilité qu'un élément de S appartienne à la classe C_i. L'entropie $H(S)$ de l'ensemble fini S est donnée par*

$$H(S) = -\sum_{i=1}^{m} p_i \, log_2 \, (p_i).$$

Ainsi, nous pouvons définir l'entropie relative à un attribut A, à l'origine de la division d'un ensemble de données S.

Définition 2.4.2. *Soient S un ensemble fini de données, V le nombre de valeurs possibles pour l'attribut A et S_j l'ensemble des éléments de la j^{me} partition relative à l'attribut A. L'entropie $H_A(S)$ de l'ensemble fini S relative à un attribut A est donnée par :*

$$H_A(S) = -\sum_{j=1}^{V} \frac{S_j}{S} H(S_j),$$

avec $H(S_j)$ l'entropie de l'ensemble fini S_j.

Nous pouvons désormais définir le gain d'information à l'aide des définitions 2.4.1 et 2.4.2 étant donné qu'il s'agit d'un critère de division qui mesure les changements d'entropie après la division d'un ensemble de données selon un attribut.

Définition 2.4.3. *Soient un ensemble S, $H(S)$ l'entropie de l'ensemble S et $H_A(S)$ l'entropie de l'ensemble S relative à un attribut A. Le gain d'information $IG_A(S)$ est donnée par*

$$IG_A(S) = H(S) - H_A(S).$$

Maximiser ce gain d'information revient à maximiser la discrimination entre les classes en identifiant les attributs qui apportent le plus d'informations pour distinguer les exemples de différentes classes.

2.4.2 Indice de Gini

L'indice de Gini est une mesure statistique qui permet de rendre compte de la répartition d'une variable au sein d'une population. Il s'agit d'une mesure de l'impureté d'un ensemble de données. Si toutes les données d'un ensemble sont identifiées comme étant de la même classe, on dit que cet ensemble est pur. A l'invere, un ensemble de données avec beaucoup de classes différentes est dit d'une grande impureté.

Définition 2.4.4. *Soient S un ensemble de données, m le nombre de classes et p_i la probabilité qu'un élément de S appartienne à la classe C_i. L'indice de Gini $G(S)$ de l'ensemble S est donné par*

$$G(S) = \sum_{i=1}^{m} p_i(1 - p_i).$$

Comme cet indice traite des divisions binaires, nous pouvons dire que l'attribut A divise l'ensemble S en deux ensembles S_1 et S_2. Ainsi, nous pouvons définir l'indice de Gini de l'ensemble S relatif à l'attribut A.

Définition 2.4.5. *Soient S_1 et S_2 les sous-ensembles de S déterminés à l'aide de l'attribut A. L'indice de Gini $G_A(S)$ d'un ensemble S relatif à un attribut A est déterminé par*

$$G_A(S) = \frac{|S_1|}{|S|}G(S_1) + \frac{|S_2|}{|S|}G(S_2),$$

avec G l'indice de Gini défini précédemment.

Grâce aux définitions 2.4.4 et 2.4.5, nous pouvons définir la variation de l'impureté, c'est-à-dire la variation de l'indice de Gini relatif à un attribut A.

Définition 2.4.6. *Soient un ensemble S, $G(S)$ l'indice de Gini de l'ensemble S et $G_A(S)$ l'indice de Gini de l'ensemble S relatif à un attribut A. La variation $\Delta G(A)$ de l'indice de Gini relatif à un attribut A est donnée par*

$$\Delta G(A) = G(S) - G_A(S).$$

En minimisant l'indice de Gini lors de la division des données, on vise à maximiser la pureté des ensembles résultants.

2.5 Fonction moyenne, fonction de covariance et opérateur de covariance

Les courbes issues de différentes populations π_k se distinguent par leurs fonctions moyennes μ_k et leurs fonctions de covariance K. La covariance entre deux variables aléatoires est un nombre permettant de quantifier leurs écarts conjoints par rapport à leurs espérances respectives. Ce sont des concepts fondamentaux pour comprendre la structure des données et les relations entre les variables.

Premièrement, la fonction moyenne nous donne une idée de la tendance centrale d'un ensemble de données. Elle permet de calculer la valeur moyenne d'une variable aléatoire sur l'ensemble de l'espace d'échantillonnage. Mathématiquement, il s'agit de la définition 2.5.1.

Définition 2.5.1. *La fonction moyenne de X est l'unique fonction $\mu \in L^2(\tau)$ telle que*

$$\mathbb{E}[< y, X >] =< y, \mu >, \quad \forall y \in L^2(\tau),$$

avec $\mu(t) = \mathbb{E}[X(t)]$ pour presque tout $t \in \tau$.

Ensuite, la fonction de covariance nous permet d'explorer la relation entre deux variables aléatoires. Elle mesure la façon dont ces variables varient ensemble. Une covariance positive indique que les variables augmentent ou diminuent ensemble, tandis qu'une covariance négative indique qu'elles varient en sens opposé. La covariance est donnée mathématiquement par la définition 2.5.2.

Définition 2.5.2. *Soit μ la fonction moyenne de X. La fonction de covariance K de X est donné par*

$$K(s,t) = \mathbb{E}[(X(t) - \mu(t))(X(s) - \mu(s))], \quad \forall t, s \in \tau.$$

Enfin, l'opérateur de covariance est un outil plus puissant qui généralise la covariance pour gérer les relations entre plusieurs variables aléatoires. Il nous permet de construire une matrice de covariance, qui contient des informations sur la manière dont chaque variable se comporte par rapport à toutes les autres. Mathématiquement, on note la covariance par la définition 2.5.3

Définition 2.5.3. *Soit μ la fonction moyenne de X. L'opérateur de covariance $\mathcal{K}$ de X est donné par*

$$\mathcal{K}(y) = \mathbb{E}[< (X-\mu), y > (X-\mu)], \quad \forall y \in L^2(\tau).$$

Le lien entre l'opérateur de covariance $\mathcal{K}$ de X et sa fonction de covaraince K est le suivant :

$$\mathcal{K}(y)(t) = \int_\tau K(t,s)y(s)ds.$$

2.6 Analyse en composantes principales fonctionnelles

L'analyse en composante principale à pour objectif de transformer des variables corrélées en nouvelles variables non corrélées les unes des autres. On les appelle composantes principales. Cette approche permet de résumer l'information tout en réduisant le nombre de variables. Cette section a été rédigée sur base de la source [Atto, 2021]. Commençons par détailler l'analyse en composantes principales pour les données discrètes avant de l'adapter aux données fonctionnelles.

2.6.1 Analyse en composantes principales pour les données discrètes

Soit X un ensemble de données pour n individus observés selon p variables numériques $\mathcal{X}_1, ..., \mathcal{X}_p$. Le jeu de données peut alors s'écrire comme la matrice X suivante

$$X = \underbrace{\left.\begin{pmatrix} x_{11} & \dots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{np} \end{pmatrix}\right\} n \text{ individus}}_{p \text{ variables}}$$

avec x_{ij} l'observation pour l'individu i et la variable j ($\forall i = 1, ..., n$ et $\forall j = 1, ..., p$).

Notons $x_1, ..., x_p$ les colonnes de X, soient les vecteurs de dimensions n représentant les p variables. L'objectif est alors de "comprimer" ces vecteurs en p' nouveaux vecteurs orthogonaux $x_1, ..., x_{p'}$ avec $p' < p$ san trop perdre d'informations. Ces p' nouveaux vecteurs sont des combinaisons linéaires des p vecteurs initiaux, c'est-à-dire les colonnes de X. Notons α le vecteur inconnu $(\alpha_1, ..., \alpha_p)^T$. Nous cherchons les combinaisons linéaires $X\alpha = \sum_{j=1}^{p} \alpha_j x_j$ telles que la variance soit maximale pour tenir compte de la diversité des données et telles que la contrainte $||\alpha||^2 = \alpha^T\alpha = 1$ soit respectée.

Pour cette analyse en composantes principales, la fonction à considérer est $Var(X\alpha)$ et la contrainte est que la norme de α doit valoir 1, soit $\alpha^T\alpha - 1 = 0$. On cherche alors à maximiser le Lagrangien L donné par

$$L(\alpha, \lambda) = Var(X\alpha) - \lambda(\alpha^T\alpha - 1),$$

avec λ le multiplicateur de Lagrange.

Maximiser le lagrangien revient à égaler ses différentielles à 0. Notons $\hat{\Sigma}$ la matrice de variance-covarience associées aux données. La variance est alors définie par $var(X\alpha) = \alpha^T \hat{\Sigma} \alpha$. On a

$$\begin{aligned} \frac{\partial L}{\partial \alpha} = 0 &\iff \hat{\Sigma}\alpha - \lambda\alpha = 0 \\ &\iff \hat{\Sigma}\alpha = \lambda\alpha. \end{aligned}$$

Cette équation signifie que α doit être un vecteur propre unitaire de $\hat{\Sigma}$, de valeur propre associée λ. C'est-à-dire que $\alpha^T \hat{\Sigma} \alpha = \alpha^T \lambda \alpha = \lambda$.

$\hat{\Sigma}$ est une matrice symétrique, elle est égale à sa propre transposée. Un des résultats importants de l'algèbre linéaire concernant les matrices symétriques est le théorème spectral.

Théorème 2.6.1. ***Théorème spectral.*** *Soit A une matrice $n \times n$. Alors A est symétrique si et seulement si elle peut se diagonaliser à l'aide d'une matrice de changement de base orthogonale.*

Cela implique qu'il existe une matrice orthogonale U dont les colonnes sont les vecteurs propres de $\hat{\Sigma}$. Cette matrice se note $U = [u_1, ..., u_p]$ telle que $U^T U = I$. On a

$$\hat{\Sigma} u_j = \lambda_j u_j,$$

tel que $U^T \hat{\Sigma} U = diag(\lambda_1, .., \lambda_p)$ avec $\lambda_1 > \lambda_2 > \cdots > \lambda_p$.

Nous pouvons désormais définir α :

— Etape 1 : comme λ_1 est le maximum des valeurs propres, on prend $\alpha = u_1$ (λ_1 est atteint quand $\alpha = u_1$).

— Etape 2 : comme λ_2 est le deuxième maximum des valeurs propres, on prend $\alpha = u_2$ pour la contrainte supplémentaire $\alpha^T u_1 = 0$.
⋮

— Etape m : comme λ_m est le m-ième maximum des valeurs propres, on prend $\alpha = u_m$ pour la contrainte supplémentaire $\alpha^T u_{m-1} = 0$.

2.6.2 Analyse en composantes principales pour les données fonctionnelles

L'analyse en composante principale fonctionnelle est une extension de l'analyse en composante principale en dimension infinie. Elle favorise la conversion des données fonctionnelles infinies en un vecteur de dimension finie afin d'y appliquer les méthodes de l'analyse de données multivariées. Le but est donc de réduire une dimension infinie en une dimension finie en perdant le moins d'infomations. La notion de projection est alors souvent utilisée en ACP fonctionnelle. En pratique, il s'agit de projeter les fonctions originales sur un sous-espace engendré par les composantes principales. Cette projection permet de représenter chaque fonction en utilisant un nombre réduit de composantes principales plutôt que la totalité des informations initiales.

Soit S_p un sous-espace engendré par $\{u_1, ..., u_p\}$ un système orthonormal de L^2 ($p \geq 1$ fixé). Notons $Proj_{S_p}(X)$ la projection orthogonale de la fonction aléatoire X sur S_p. Le système $\{u_1, ..., u_p\}$ est optimal si $\mathbb{E}[||X - Proj_{S_p}(X)||^2]$ est minimale, avec $Proj_{S_p}(X)$ exprimé en terme de moyenne quadratique. Détaillons brièvement.

$$\begin{aligned}\mathbb{E}[||X - Proj_{S_p}(X)||^2 &= \mathbb{E}[||X - \sum_{j=1}^{p} < X, u_j > u_j||^2], \\ &= \mathbb{E}[||X||^2] - \sum_{j=1}^{p} \mathbb{E}[< X, u_j >^2], \\ &= \mathbb{E}[||X||^2] - \sum_{j=1}^{p} < \mathcal{K}(u_j), u_j >,\end{aligned}$$

avec $\mathcal{K}$ l'opérateur de covariance de X. Pour plus de détails, consulter la source [Atto, 2021].

Nous souhaitons donc maximiser $\sum_{j=1}^{p} < \mathcal{K}(u_j), u_j >$. Comme $\mathcal{K}$ est semi-défini positif, nous voulons maximiser chaque terme $< \mathcal{K}(u_j), u_j >$ avec u_j des vecteurs orthogonaux et de norme 1. Autrement dit, on veut maximiser $< \mathcal{K}(x), x >$ avec $||x|| = 1$.

Ensuite, définissons les opérateurs compacts afin de déterminer ce qu'est un opérateur de Hilbert-Schmidt.

Définition 2.6.1. *Soient $(H, < ., . >)$ un espace de Hibert. Soient v_j et f_j deux bases orthonormales et λ_j une suite réelle positive convergent vers 0. Un opérateur ψ est compact (complètement continu) si sa décomposition en valeurs singulières est*

$$\psi(X) = \sum_{j=1}^{\infty} \lambda_j < x, v_j > f_j, \qquad \forall x \in H.$$

Définition 2.6.2. *Un opérateur de Hilbert-Schmidt est un opérateur compact tel que* $\sum_{j=1}^{\infty} \lambda_j^2 < \infty$.

Ces défintions nous amène à la proposition suivante.

Proposition 2.6.1. *Tout opérateur ψ de Hilbert-Schmidt, symétrique, défini positif sur un espace de Hilbert $(H, < ., . >)$, admet une décomposition*

$$\psi(X) = \sum_{j=1}^{\infty} \lambda_j < x, v_j > v_j, \qquad \forall x \in H,$$

avec v_j les fonctions propres de ψ telles que $\psi(v_j) = \lambda_j v_j$.

Comme $\mathcal{K}$ est un opérateur de Hilbert-Schmidt, symétrique, défini positif de L^2, par la proposition 2.6.1, on a :

$$\begin{aligned}< \mathcal{K}(x), x > &= < \sum_{j=1}^{\infty} \lambda_j < x, \xi_j > \xi_j, x > \\ &= \sum_{j=1}^{\infty} \lambda_j < x, \xi_j >^2, \quad \text{détails [Atto, 2021]},\end{aligned}$$

avec λ_j les valeurs propres telles que $\lambda_1 > ... > \lambda_p$ et ξ_j les fonction propres unitaires de $\mathcal{K}$, $\forall j$.

Cela revient à maximiser $\sum_{j=1}^{\infty} \lambda_j < x, \xi_j >^2$ sous la contrainte $\sum_{j=1}^{\infty} < x, \xi_j >^2 = 1$.

Nous pouvons désormais définir x :

— Etape 1 : comme λ_1 est le maximum des valeurs propres, on prend $x = \xi_1$ (λ_1 est atteint quand $x = \xi_1$).

— Etape 2 : comme λ_2 est le deuxième maximum des valeurs propres, on prend $x = \xi_2$ pour la contrainte supplémentaire $< x, \xi_1 >= 0$.

⋮

— Etape m : comme λ_m est le m-ième maximum des valeurs propres, on prend $\alpha = u_m$ pour la contrainte supplémentaire $< x, \xi_{m-1} >= 0$.

Nous pouvons désormais définir l'analyse en composante principale fonctionnelle (ACPF) à partir d'un échantillon de courbes $X_1, ..., X_n$. L'objectif est de parcourir tous les systèmes orthonormaux de L^2 composés de p fonctions afin de trouver une base $u_1..., u_p$ $(1 \leq p \leq n)$ telle que l'expression $\sum_{i=1}^{n} ||X_i - Proj_{S_p}(X_i)||^2$ soit minimisée.

En suivant le raisonnement ci-dessus, on remarque que les fonctions $u_1, ..., u_p$ correspondent à $\xi_1, ..., \xi_p$, les fonctions propres unitaires de $\hat{\mathcal{K}}_n$, l'opérateur de covariance empirique avec $\hat{\lambda}_1 > \cdots > \hat{\lambda}_p$ ses valeurs propres. Ces fonctions propres sont appelées Composantes Principales Fonctionnelles Empiriques. Ainsi, chaque courbe X_i de dimension infinie peut être remplacées par des vecteurs de dimension p :

$$X_i = \sum_{j=1}^{p} < X_i, u_j > u_j, \tag{2.4}$$

c'est-à-dire $X_i = [< X_i, u_1 >, < X_i, u_2 >, ..., < X_i, u_p >]^T$.

2.6.3 ACPF exemple de données réelles

Reprenons l'exemple du jeu de données présenté à la section 2.1.1 sur les indices boursiers et procédons à l'analyse en composantes principales. Projetons les données initiales sur les trois pemières fonctions propres. Celles-ci ont représentées à la FIGURE 2.8. Le premier score principal ainsi obtenu explique 83,91% de la variance totale des données, le second 15,83% et le troisième 0,26%.

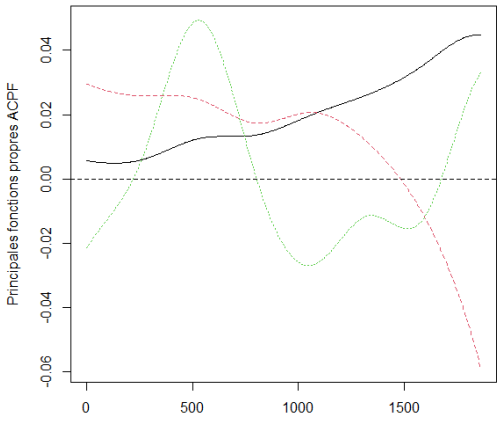

FIGURE 2.8 – Trois pemières fonctions propres principales (noir puis rouge puis vert) : ACPF.

2.7 Fonction noyau et matrice de Gram

Les noyaux (kernels) permettent de représenter des échantillons de données de façon flexible afin de pouvoir les analyser, les comparer. En d'autres termes, les fonctions noyaux permettent de gérer des données non linéaires en les projetant dans un espace de dimension plus grand où les données deviennent plus séparables linéairement.

Définition 2.7.1. *Une fonction noyau $\mathcal{N} : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ permet de quantifier la similarité entre deux objets (courbes,...) dans un espace $\mathcal{X}$.*

Notons qu'une fonction noyau est souvent symétrique ($\forall x, x' \in \mathcal{X}, \mathcal{N}(x, x') = \mathcal{N}(x', x)$) et non négative ($\forall x, x' \in \mathcal{X}, \mathcal{N}(x, x') \geq 0$).

Il existe une matrice qui résume les relations de similarité entre les données, telles que mesurées par la fonction noyau. En effet, lorsqu'on a un ensemble de données X et qu'on calcule la fonction noyau pour chaque paire de données dans cet ensemble, on obtient une matrice de Gram. Cette matrice peut être interprétée comme une matrice de covariance lorsque les données sont centrées. Elle capture la covariance entre les valeurs de la fonction noyau pour différentes paires de données dans l'espace transformé. C'est pourquoi elle est appelée la matrice de covariance des fonctions noyaux. C'est une matrice symétrique, dont la définition est la 2.7.2.

Définition 2.7.2. *$N(X) \in \mathbb{R}^{n \times n}$ est appelée matrice de Gram de X si $N_{ij} = \mathcal{N}(x_i, x_j)$ avec $\mathcal{N}$ la fonction noyau.*

Deux théorèmes importants découlent de ces notions. Il s'agit du théorème de Mercer et du théorème du représentant (*representer theorem*). Le théorème de Mercer permet de vérifier si une fonction est un noyau valide, ce qui est essentiel pour garantir la convergence et la stabilité des algorithmes.

Théorème 2.7.1. ***Théorème de Mercer*** *Soient N la matrice de Gram définie positive et λ_i la i-ème valeur propre de N. La matrice de Gram admet une décomposition en vecteurs propres telle que*

$$N = U^T \Lambda U, \quad \text{avec } \Lambda = diag(\lambda_1, ..., \lambda_n).$$

L'élément (i, j) de la matrice N est donné par

$$N_{ij} = \left(\Lambda^{1/2} U_{.,i}\right) \left(\Lambda^{1/2} U_{.,j}\right).$$

Notons $N_{ij} = \phi(x_i)\phi(x_j)$ avec $\phi(x_i) = \Lambda^{1/2} U_{.,i}$.

Si la fonction noyau est définie positive et satisfait la condition de Mercer, elle permet de calculer le produit intérieur suivant

$$\mathcal{N}(x, x') =< \phi(x), \phi(x') > . \tag{2.5}$$

Un autre théorème intéressant et le théorème du représentant. Celui-ci énonce que, sous certaines conditions, une fonction donnée peut être représentée de manière exacte ou approximative par une combinaison linéaire de fonctions de base, pouvant être des fonctions noyau.

Théorème 2.7.2. ***Théorème du représentant*** *Soient $x_1, .., x_n$ les points d'apprentissage d'une fonction f. Il est possible d'exprimer f comme combinaison linéaire de produit de fonctions noyaux $\mathcal{N}$*

$$f(x) = \sum_{i=1}^{n} a_i \mathcal{N}(x, x_i).$$

Chapitre 3

Classification de données fonctionnelles

Dans le domaine de l'analyse de données, la classification de données fonctionnelles représente une avancée majeure dans la compréhension et l'exploitation des informations contenues dans des courbes et des fonctions. La classification de données fonctionnelles est alors très intéressante pour les prédictions concernant toutes sortes de courbes. Ce chapitre explore les défis de la classification de données fonctionnelles, examinant comment les caractéristiques de ces données nécessitent des approches spécifiques pour une modélisation efficace.

Ce chapitre présentent différents types de classificateurs fonctionnels (sections 3.1 à 3.5). Pour l'ensemble de ce chapitre, les données sont fonctionnelles et sont donc des courbes $X_i(t)$ avec $t \in T \in \mathbb{R}$ (i =1,...,n). Les méthodes expliquées dans ce chapitre sont la classification à l'aide des k plus proches voisins, la classification basée sur la notion de profondeur, les arbres de décisions, la classification binaire par projection à deux dimensions ou encore la classification par la méthode de la machine à vecteur de support des moindres carrés. Pour beaucoup de ces méthodes, nous faisons référence à des notions du chapitre 2. Nous nous efforçons également de présenter des avantages et inconvénients pour chaque méthode.

Par la suite, nous explorons les applications pratiques de la classification de données fonctionnelles dans des contextes réels afin d'approfondir la compréhension des méthodes associées à la classification de données fonctionnelles.

3.1 Méthode k-NN pour les données fonctionnelles

Dans le chapitre 1, nous avons présenté la méthode des k plus proches voisins car celle-ci peut s'adapter aux données fonctionnelles. Deux sources ont permis de réaliser cette section. Il s'agit de [Fuchs et al., 2015] et [Fuchs et al., 2017].

La méthode k-NN est utilisée pour sélectionner des caractéristiques sur les covariables fonctionnelles afin de les interpréter. L'idée est d'utiliser un ensemble de semi-métriques correctement choisies qui se concentrent chacune sur une caractéristique précise de la courbe.

Considérons $x_i^{(a)}(t)$ la différentiation d'ordre a de la courbe $x_i(t)$. La semi-métrique des dérivées de deux covariables fonctionnelles $x_i^{(a)}(t)$ et $x_j^{(a)}(t)$ avec $i \neq j$ est donnée par $d(x_i^{(a)}(t), x_j^{(a)}(t))$.

Nous commençons par présenter cette méthode pour classifier une certaine observation $x^*(t)$ avec d'abord une puis plusieurs semi-métriques. Ensuite, nous généralisons la méthode pour toutes les données $x_i(t)$ d'un jeu de données.

3.1.1 Classification k-NN d'une donnée x^* avec une semi-métrique

Supposons que pour un jeu de données $x_i(t)$ $(i = 1, ..., n)$, il existe G classes différentes. Chaque courbe $x_i(t)$ appartient alors à l'une de ces G classes, notée par $y_i \in \{1, ..., G\}$.

Soit $x^*(t)$, une nouvelle observation de classe inconnue y^*. Nous allons classer les $x_i^{(a)}(t)$ observations de l'échantillon de la plus proche de $x^{*(a)}$ à la plus éloignée en considérant une semi-métrique d.

$$d(x^{*(a)}(t), x_{(1)}^{(a)}(t)) \leq ... \leq d(x^{*(a)}(t), x_{(k)}^{(a)}(t)) \leq ... \leq d(x^{*(a)}(t), x_{(n)}^{(a)}(t)).$$

Nous pouvons à présent définir les k plus proches voisins de $x^*(t)$. Ceux-ci constituent le voisinage

$$N(x^{*(a)}(t)) = \{x_j^{(a)}(t) \mid d(x^{*(a)}(t), x_j^{(a)}(t)) \leq d(x^{*(a)}(t), x_{(k)}^{(a)}(t))\}.$$

On peut donc ainsi trouver la classe y^* de l'observation $x^*(t)$. Pour ce faire, nous allons estimer la probabilité π_g que l'observation $x^*(t)$ soit de la classe $g \in \{1, ..., G\}$ par l'estimateur $\hat{\pi}_g$. Cet estimateur est donnée par

$$\hat{\pi}_g = \frac{1}{k} \sum_{j: x_j^{(a)}(t) \in N(x^{*(a)}(t))} \mathbb{1}(y_j = g),$$

avec $\mathbb{1}(y_j = g)$ la fonction indicatrice qui vaut 1 si $y_j = g$ et 0 sinon.

L'observation $x^*(t)$ est assignée à la classe ayant la plus haute probabilité $\hat{\pi}_g$.

$$y^* = \underset{g}{\operatorname{argmax}}(\hat{\pi}_g). \tag{3.1}$$

3.1.2 Classification k-NN d'une donnée x^* avec plusieurs semi-métriques

Considérons toujours une nouvelle observation $x^*(t)$ dont on souhaite trouver la classe. Comme expliqué précédemment, nous souhaitons plusieurs semi-métriques telles que chaque semi-métrique soit liée à une caractéristique de la courbe. Nous allons donc considérer plusieurs semi-métriques pour appliquer la méthode des k plus proches voisins. On les notes $d_l(x_i^{(a)}(t), x_j^{(a)}(t))$ pour les dérivées de deux covariables fonctionnelles $x_i^{(a)}(t)$ et $x_j^{(a)}(t)$ avec $i \neq j$. Des exemples de semi-métriques que l'on peut utiliser sont repris à la TABLE 2.1. Ce sont les plus utilisées et les plus connues, il pourrait y en avoir d'autres.

Pour chaque semi-métrique d_l, on classe à nouveau les observations de la plus proche à la plus éloignée de l'observation $x^{*(a)}(t)$.

$$d_l(x^{*(a)}(t), x_{(1)}^{(a)}(t)) \leq ... \leq d_l(x^{*(a)}(t), x_{(k)}^{(a)}(t)) \leq ... \leq d_l(x^{*(a)}(t), x_{(n)}^{(a)}(t)).$$

On peut alors déterminer un voisinage des k plus proches voisins pour chaque semi-métrique d_l. Celui-ci est noté

$$N_l(x^{*(a)}(t)) = \{x_j^{(a)}(t) \mid d_l(x^{*(a)}(t), x_j^{(a)}(t)) \leq d_l(x^{*(a)}(t), x_{(k)}^{(a)}(t))\}. \tag{3.2}$$

Les probabilités qu'une nouvelle observation $x^*(t)$ appartienne à une classe $g \in \{1, ..., G\}$ sont calculées séparément pour chacune des semi-métriques

$$\hat{\pi}_{gl} = \frac{1}{k} \sum_{j: x_j^{(a)}(t) \in N_l(x^{*(a)}(t))} \mathbb{1}(y_j = g).$$

Ces probabilités $\hat{\pi}_{gl}$ sont propres à chaque distance d_l. Cela ne nous aide pas vraiment pour classifier $x^*(t)$. On peut à présent calculer la probabilité globale que l'observation $x^*(t)$ soit de la classe $g \in \{1, ..., G\}$. On note cette probabilité

$$\hat{\pi}_g = \sum_{l=1}^{p} c_l \hat{\pi}_{gl},$$

avec $c_l \geq 0$ et $\sum_{l=1}^{p} c_l = 1$.

Les poids c_l doivent être estimés ou choisis par l'utilisateur. En effet, ils permettent de déterminer quelles semi-métriques, et donc quelles caractéristiques de la courbe ont le plus de poids. C'est-à-dire que le poids indique celles qui donnent le plus d'informations pour la distinction des classes. L'observation $x^*(t)$ est alors assignée à la classe de plus haute probabilité comme pour l'équation (3.1).

3.1.3 Classification k-NN de toutes les données avec plusieurs semi-métriques

Nous avons expliqué comment classifier une certaine observation $x^*(t)$ avec la méthode des k plus proches voisins. Il est à présent possible de déterminer ces probabilités pour chacune des observations $x_i^{(a)}(t)$ $(i = 1, ..., n)$ d'un jeu de données par rapport aux $n - 1$ autres observations. Les k plus proches voisins d'une observations $x_i(t)$ sont déterminés comme à l'équation (3.2). Ce voisinage est noté $N_l(x_i^{(a)}(t))$.

La probabilité que l'observation $x_i(t)$ appartienne à une classe $g \in \{1, ..., G\}$ selon une semi-métrique d_l est

$$\hat{\pi}_{gli} = \frac{1}{k} \sum_{j \neq i : x_j^{(a)}(t) \in N_l(x_i^{(a)}(t))} \mathbb{1}(y_j = g).$$

La probabilité globale que l'observation $x_i(t)$ soit de la classe $g \in \{1, ..., G\}$ est alors donnée par

$$\hat{\pi}_{gi} = \sum_{l=1}^{p} c_l \hat{\pi}_{gli}.$$

Dans ce cas, chaque observation $x_i(t)$ est assignée à la classe de plus haute probabilité $\hat{\pi}_{gi}$. On note la classe de l'observation i

$$y_i = \underset{g}{\operatorname{argmax}}(\hat{\pi}_{gi}).$$

Pour chacune des semi-métriques utilisée, nous ne considérons pas toujours le même degré de différentiation a, ni le même nombre k de plus proches voisins. On détermine alors $a \in \{a_1, ..., a_q\}$ et $k \in \{k_1, ..., k_m\}$. Dans ce cas, l'indice l ne correspond plus uniquement à une semi métrique d_l mais à un triplet $\{d_l, a, k\}$.

Le fonctionnement de k-NN est intuitif, ce qui facilite la compréhension et l'interprétation des résultats. De plus, dans le cas des k plus proches voisins pour les données fonctionnelles, si on choisit correctement les semi-métriques, la méthode reste efficace même lorsque l'ensemble des données est très grand. En revanche, la classification k-NN nécessite le calcul de distances entre chaque paire d'observations, ce qui peut devenir coûteux en terme de simulation pour un échantillon de grande taille. Un autre inconvénient de la méthode est le fait que le choix du nombre de voisins (k) peut influencer les performances du modèle. Une valeur de k trop petite ou trop grande ne sera pas significative du modèle.

3.2 Classification de données fonctionnelles basée sur la profondeur

Un autre classificateur fonctionnel est celui basé sur la profondeur. Nous avons explicité cette notion à la section 2.3. Le but est de déterminer une distance entre une nouvelle donnée et les observations considérées comme étant les plus représentatives de chaque groupe, c'est-à-dire les plu profondes. A présent, nous allons présenter deux méthodes basées sur la profondeur, très utiles pour la classification de données fonctionnelles. Cette section a été rédigée à l'aide de la source [Lopez-Pintado and Romo, 2007].

3.2.1 Méthode 1 : distance par rapport à la moyenne tronquée

Pour commencer, considérons un ensemble de courbes x_i $(i = 1, ..., n)$ continues et définies sur un intervalle compact (fermé borné). Considérons également un ensemble de naturels $y_i \in \{1, ..., G\}$ indiquant la classe de x_i. Le principe de la méthode est de calculer la distance entre une nouvelle observation et la moyenne tronquée de chaque groupe.

Définition 3.2.1. *Soit $x_{(i)}$ $(i = 1, ..., n)$ un échantillon ordonné de l'observation la plus profonde à la moins profonde. La moyenne tronquée de cet échantillon est donnée par*

$$m^{\alpha} = \frac{\sum_{i=1}^{n-\lceil n\alpha \rceil} x_{(i)}}{n - \lceil n\alpha \rceil},$$

avec $\lceil n\alpha \rceil$ le plafond de $n\alpha$ et α un paramètre $\in [0, 1[$.

Nous pouvons donc calculer la moyenne tronquée m_g^{α} de chaque groupe $g \in \{1, ..., G\}$ pour un α donné. Ensuite, nous pouvons calculer la distance entre m_g^{α} et la nouvelle observation x^*. On la note $d(x^*, m_g^{\alpha})$ avec d une distance quelconque dans l'intervalle de définition des courbes. Lorsque $\alpha = 0$, cela revient à calculer la distance à la moyenne et lorsque $\alpha = \dfrac{n-1}{n}$, cela revient à calculer la distance par rapport à l'observation la plus profonde.

Il reste alors à classer x^* dans le groupe k tel que

$$d(x^*, m_k^{\alpha}) = \min_{g=1,...,G} \left\{ d(x^*, m_g^{\alpha}) \right\}.$$

3.2.2 Méthode 2 : distance moyenne pondérée

Pour chaque groupe, la seconde méthode consiste à considérer la moyenne pondérée des distances d'une nouvelle observation à chaque élément du groupe. La moyenne pondérée permet de tenir compte de l'importantce de chacun des éléments d'un ensemble. La profondeur de chaque observation sera notre système de pondération, c'est-à-dire que plus une observation est profonde, plus elle aura du poids dans le calcul de la distance moyenne. Notons $S(x)$ la profondeur de l'observation x et $A_g = \{x_1, ..., x_{n_g}\}$ le groupe g.

La distance moyenne pondérée entre une nouvelle observation x^* et le groupe A_g, avec d une distance quelconque, est donnée par

$$AD(x^*, A_g) = \frac{\sum_{i=1}^{n_g} d(x^*, x_i) S(x_i)}{\sum_{i=1}^{n_g} S(x_i)}.$$

Malheureusement, le résultat dépend du nombre n_g d'observations de chaque groupe g. Si le nombre d'observations est très différent d'un groupe à l'autre, les résultats peuvent être faussés. Il est donc préférable de considérer une distance moyenne pondérée limitée. En effet, on ne considère plus toutes les observations de chaque groupe mais seulement les m observations les plus profondes. Cette distance est notée

$$TAD(x^*, A_g) = \frac{\sum_{i=1}^{m} d(x^*, x_i) S(x_i)}{\sum_{i=1}^{m} S(x_i)}.$$

Le principe est alors le même que pour la méthode précédente, nous souhaitons classer x^* dans le groupe k tel que

$$TAD(x^*, A_k) = \min_{g=1,...,G} \{TAD(x^*, A_g)\}.$$

Le plus gros avantage de ces deux méthodes est le fait qu'elles soient robustes aux valeurs aberrantes. La profondeur est moins sensible aux observations extrêmes, ce qui peut améliorer la performance du modèle dans des situations où les données sont sujettes à des valeurs aberrantes, d'autant plus que les valeurs aberrantes ne sont pas toujours facilement identifiables dans un ensemble de données fonctionnelles. D'autre part, comme de nombreuses méthodes, la performance de la classification fonctionnelle basée sur la profondeur peut être affectée si la dimension du jeu de données est élevé.

3.3 Arbre de décision (decision tree)

Les arbres de décision constituent un autre méthode fondamentale pour la classification de données fonctionnelles. Pour réaliser cette section, nos inspirations sont les sources [Navlani, 2023], [Balakrishnan and Madigan,] et [Corporation,]. Le principe d'un arbre de décision est d'utiliser des tests sur les variables prédictives pour établir des règles et diviser les données de façon récursive en un certains nombre de régions.

L'idée est de partir d'un noeud racine contenant l'ensemble complet des données x_j ($j = 1, ..., n$). Ce noeud racine correspond au premier noeud de décision. En effet, nous allons analyser les données selon une première caractéristique r_1 de la courbe permettant la division des observations en différents groupes. Les branches qui découlent des noeuds de décision représentent les règles de décision. Par exemple, on pourrait prendre r_1 comme étant le maximum de la courbe. Les branches de décision pourrait alors être une indication sur la valeur du maximum, par exemple, $r_1 > 4$ et $r_1 \leq 4$. On pourrait aussi simplement utiliser la notion de distance entre deux courbes présentée au chapitre 2. Ensuite, chaque nouvel ensemble de données ainsi créé devient à son tour noeud de décision pour une nouvelle caractéristique r_k avec $k = 1, ..., K$ et K le nombre de caractéristiques observées. Lorsque toutes les caractéristiques des courbes ont été explorées, l'arbre est terminé. Les derniers noeuds obtenus, appelés noeuds feuilles, contiennent chacun un certain nombre n_i d'obesvations ($\sum_i n_i = n$) et correspondent aux π_i classes obtenues par arbre de décision ($i = 1, ..., m$). Un exemple d'arbre de décision ainsi construit est illustré à la FIGURE 3.1.

Pour construire les arbres de décision, il existe plusieurs algorithmes. Les deux algorithmes les plus connus sont le CART, Classification And Regression Tree de Breiman ([Breiman et al., 2017]) et le ID3, Iterative Dichotomiser 3 de Ross Quinlan ([Quinlan, 1986]). L'idée générale de ces algorithmes analogues est décrite ci-dessous.

1. Utiliser une mesure de sélection d'attribut afin de déterminer quelle caractéristique r_k est la meilleure pour en faire un noeud de décision en premier.
2. Diviser ce noeud de décision en sous-ensembles relatifs aux valeurs possibles pour l'attribut considéré.
3. Répéter ce processus pour chaque nouveau noeud jusqu'à atteindre uniquement des noeuds feuilles.

L'algorithme CART et l'algorithme ID3 utilisent cependant des mesures de sélection d'attribut (ASM) différentes. Nous les détaillons dans les sections 3.3.1 et 3.3.2. Une extension de l'algorithme CART est présentée à la section 3.3.3. Il s'agit des forêts aléatoires.

3.3.1 ID3 : Iterative Dichotomiser 3

Cet algorithme d'arbre de décisions se base sur l'ASM *gain d'information*. Ce critère mesure les changements d'entropie, après la division d'un ensemble de données selon un attribut. L'attribut choisit comme critère de division est celui qui maximise le gain d'information et ce jusqu'à atteindre un certain seuil ou jusqu'à ce que tous les attributs aient été exploités.

Un inconvénient des arbres de décisions pour les données fonctionnelles est le fait que les données fonctionnelles peuvent souvent avoir une dimension élevée Cela peut conduire à des arbres complexes et surajustés, en particulier avec ID3, qui a une tendance à créer des arbres profonds.

3.3.2 CART : Classification And Regression Tree

Le deuxième algorithme présenté est l'algorithme CART. Celui-ci traite uniquement les arbres de décisions binaire, c'est-à-dire qu'il y a maximum deux branches par noeuds. Le critère de division de tels arbres est l'*indice de Gini*, il s'agit d'une mesure de l'impureté.

L'attribut choisi est celui qui minimise cette variation de l'indice de Gini. Lorsqu'il n'y a plus aucun changement dans la diminution de l'impurté ou qu'elle est inférieure à un certain seuil, on considère que l'arbre de décision arrête d'évoluer. Les feuilles de l'arbre sont alors atteintes.

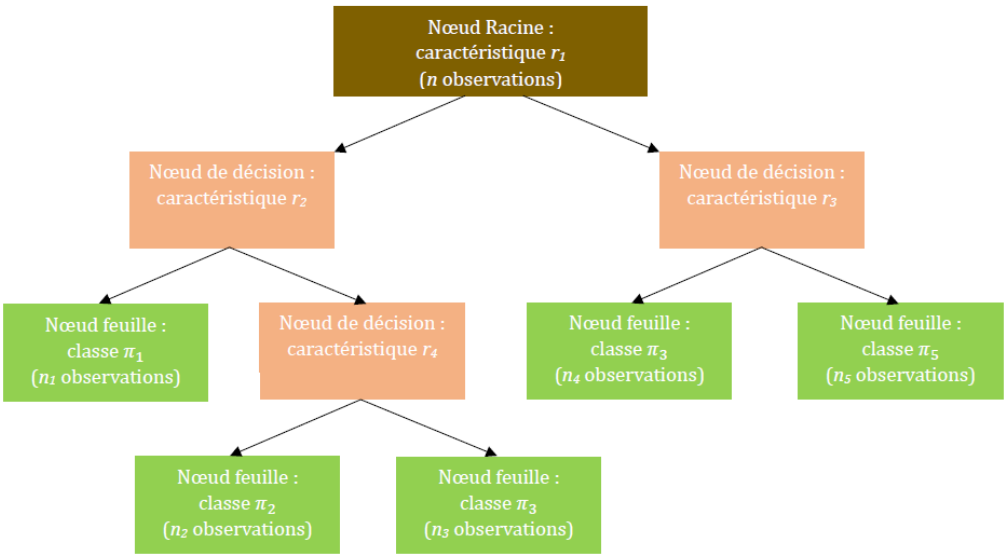

FIGURE 3.1 – Exemple d'arbre de décision où on considère $K = 4$ caractéristiques pour les courbes observées et où on obtient $m = 5$ classes.

Le problème est que les arbres sont construits pour certaines données mais parfois, ils dépassent ces données. La solution à ce problème est l'élagage. Le principe est de simplifier la structure de l'arbre en supprimant certaines noeuds. En effet, on va évaluer les risques et coûts de l'ajout d'une variable à l'arbre et vérifier si c'est plus intéressant avec ou sans. La validation croisée est alors utilisée pour sélectionner le sous arbre qui engendre le moins de risques.

Un inconvénient de l'algorithme CART est le fait que la variance est élevée, ce qui suggère une grande dispersion des données. Il n'est donc pas toujours applicable à un grand nombre de données. La solution est d'utiliser un autre algorithme appelé Random Forests, de Breiman également. Celui-ci est une extension de CART.

3.3.3 Random Forest

L'idée fondamentale derrière les forêts aléatoires est de construire un grand nombre d'arbres de décision individuels et d'agréger leurs prédictions pour obtenir un modèle robuste et précis. L'algorithme des forêts aléatoires, illustré à la FIGURE 3.2 est construit en 3 étapes :

1. La première étape consiste à sélectionner L échantillons aléatoires de k observations à partir de l'ensemble de données. Les forêts aléatoires utilisent une technique appelée *bagging*, qui consiste à créer plusieurs échantillons *bootstrap*. Il s'agit de sous-ensembles d'observations tirées avec remplacement. Cela signifie qu'une même observation peut être sélectionnée plusieurs fois, tandis que d'autres peuvent ne pas être sélectionnées du tout.
2. Pour chacun de ces l échantillons *bootstrap* $(l = 1, ..., L)$, on construit un arbre de décision qui permet de faire des prédictions sur les données x_i $(i = 1, ..., n)$.
3. Pour déterminer la classe d'une certaine observation x_i, on procède à un vote majoritaire. Cela signifie qu'on prédit la classe de x_i comme étant celle prédite par le plus d'arbres parmis les L arbres construits.

En construisant de nombreux arbres avec des échantillons de données différents, l'algorithme des forêts aléatoires est moins sensible au surajustement par rapport à un seul arbre de décision. Cela contribue à une meilleure généralisation des prédictions.

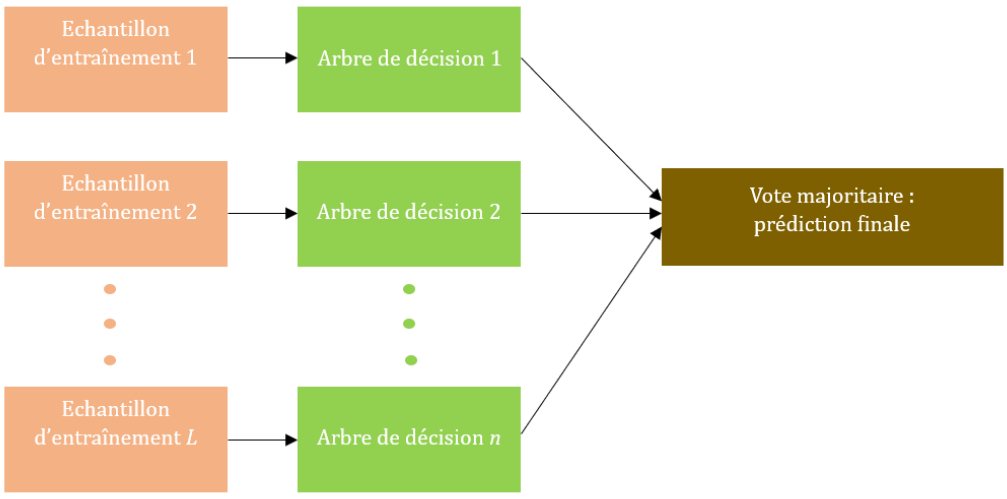

FIGURE 3.2 – Schéma de forêts aléatoires.

3.3.4 Arbre de décision pour des données fonctionnelles "multiples"

Pour former des arbres de décision pour les données fonctionnelles, nous souhaitons traiter des vecteurs dont les valeurs représentent différents scores sur une même courbe à des moments différents. Ces scores correspondent alors aux critères de divisions. Néanmoins, il existe également des arbres de décisions pour traiter des jeux de données contenant plusieurs courbes informatives par variable. Un exemple d'abre de décision pour traiter ce type de problème est expliqué dans [Möller and Gertheiss, 2018].

Par exemple, nous essayons de prédire s'il va pleuvoir un jour en particulier en analysant les courbes des températures, des précipitations et de l'humidité en fonction du temps pour la journée choisie. L'idée est de comparer la courbe des températures pour une nouvelle observation avec la courbe moyenne des températures pour les jours de pluie (classe A) et la courbe moyenne des températures pour les jours secs (classe B). Puis, faire de même avec les courbes de précipitaons et d'humidité.

Pour créer l'arbre de décision, il faut analyser la proximité de chaque courbe d'une nouvelle observation $x'_j(t)$ avec les courbes moyennes des classes A ($\overline{x}_j^A(t)$) et celles de la classe B ($\overline{x}_j^B(t)$) pour chaque prédicteurs j. La courbe $x'_j(t)$ est alors envoyée vers le noeud relatif à la moyenne la plus proche.

3.4 Classification binaire par projection à deux dimensions

Cette section est, tout comme la section 2.6, issue de la source [Atto, 2021]. Considérons des données d'entraînement (X_1, Y_1), ..., (X_n, Y_n) indépendantes et identiquement distribuées avec $X_i \in L^2(\tau)$ des fonctions aléatoires pour tout i. Pour la classifiation binaire, nous considérons uniquement deux groupes. Soient n_0 le nombre de courbes de la population π_0 et n_1 le nombre de courbes de la population π_1. On dit que X_i provient de π_k si $Y_i = k$ avec $k \in \{0, 1\}$ $\forall i = 1, ..., n$. Les courbes des populations π_k se distinguent par leurs fonctions moyenne μ_k, leurs fonctions de covariance K_k et leurs opérateurs de covariance $\mathcal{K}_k$.

Soient $\theta_{k1}, \theta_{k2}, ...$ les valeurs propres de $\mathcal{K}_k$, l'opérateur de covariance de la population π_k et $\phi_{k1}, \phi_{k2}, ...$ les fonctions propres associées avec $\theta_{k1} > \theta_{k2} > \dots$ et $K_k \in \{0, 1\}$. On définit alors la décomposition spectrale suivante

$$K_k(u, v) = \sum_{j=1}^{\infty} \theta_{kj} \phi_{kj}(u) \phi_{kj}(v).$$

Ensuite, nous pouvons définir une estimation $\overline{X}_k$ de la moyenne μ_k de X en considérant X_{ki} la i-ème courbe provenant de la population π_k

$$\overline{X}_k(t) = \frac{1}{n_k} \sum_{i=1}^{n_k} X_{ik}(t). \tag{3.3}$$

3.4.1 Classificateur centroïde

La méthode de classification utilisée est la méthode du classificateur centroïde. On définit alors le centre de chaque classe k par $\overline{X}_k$, son estimation de la moyenne donnée à à l'équation (3.3). Une donnée X^* est classifiée en π_k lorsque $\overline{X}_k$ est le centre le plus proche de X^* selon une métrique D de L^2.

Soit le classificateur centroïde $T(X^*) = D(X, \overline{X}_0) - D(X, \overline{X}_1)$. On note

$$X^* \in \begin{cases} \pi_0 & \text{si } T(X^*) < 0 \\ \pi_1 & \text{si } T(X^*) > 0 \end{cases}$$

Notons que si $T(X^*) = 0$, la décision de la classe de X^* est aléatoire.

3.4.2 Classificateur centroïde dans le cas bidimensionnel

L'objectif est alors de classer une nouvelle courbe $X^* \in L^2(\tau)$ dans la population π_0 ou π_1. Nous souhaitons projeter X^* dans un sous espace à deux dimensions $L^2(\tau)$ par la direction $\vec{\psi} = (\psi_1, \psi_2)$ telle que l'erreur de classification soit minimum, avec ψ_1, ψ_2 deux vecteurs orthonormés de $L^2(\tau)$. Ce projeté est noté

$$X(\vec{\psi}) = \begin{pmatrix} \int_\tau X(t)\psi_1(t)dt \\ \int_\tau X(t)\psi_2(t)dt \end{pmatrix} = \begin{pmatrix} < X, \psi_1 > \\ < X, \psi_2 > \end{pmatrix}$$

Soient $X, Y \in L^2(\tau)$ et $||.||_2$ la norme euclidienne sur $\mathbb{R}^2$, la métrique considérée pour la classification centroïde est

$$\begin{aligned} D_\psi &= ||(X-Y)(\vec{\psi})||_2, \\ &= \sqrt{< X-Y, \psi_1 >^2 + < X-Y, \psi_2 >^2}. \end{aligned}$$

Ainsi, le classificateur centroïde est

$$\begin{aligned} T(X) &= D^2_\psi(X, \overline{X}_1) - D^2_\psi(X, \overline{X}_0), \\ &= (< X-\overline{X}_1, \psi_1 >^2 + < X-\overline{X}_1, \psi_2 >^2) - (< X-\overline{X}_0, \psi_1 >^2 + < X-\overline{X}_0, \psi_2 >^2) \\ &= (< X-\overline{X}_1, \psi_1 >^2 - < X-\overline{X}_0, \psi_1 >^2) + (< X-\overline{X}_1, \psi_2 >^2 - < X-\overline{X}_0, \psi_2 >^2). \end{aligned}$$

3.4.3 Classificateur centroïde dans le cas tri-dimensionnel

Cette sous-section est développée à la suite des sections précédentes en guise de supplément mais ne figure dans aucune source. L'objectif est identique à celui en deux dimensions mais, à présent, nous souhaitons projeter X^* dans un sous espace à trois dimensions par la direction $\vec{\psi} = (\psi_1, \psi_2, \psi_3)$ telle que l'erreur de classification soit minimum. Ce projeté est noté :

$$X(\vec{\psi}) = \begin{pmatrix} \int_\tau X(t)\psi_1(t)dt \\ \int_\tau X(t)\psi_2(t)dt \\ \int_\tau X(t)\psi_3(t)dt \end{pmatrix} = \begin{pmatrix} < X, \psi_1 > \\ < X, \psi_2 > \\ < X, \psi_3 > \end{pmatrix}$$

Soient $X, Y \in L^2(\tau)$ et $||.||_2$ la norme euclidienne sur $\mathbb{R}^2$. Pour la classification centroïde en trois dimensions, la métrique considérée est la suivante

$$D_\psi = \sqrt{< X-Y, \psi_1 >^2 + < X-Y, \psi_2 >^2 + < X-Y, \psi_3 >^2}.$$

Ce qui signifie que le classificateur centroïde est donné par

$$\begin{aligned} T(X) =& D^2_\psi(X - \overline{X}_1) - D^2_\psi(X - \overline{X}_0) \\ =& (< X-\overline{X}_1, \psi_1 >^2 + < X-\overline{X}_1, \psi_2 >^2 + < X-\overline{X}_1, \psi_3 >^2) - (< X-\overline{X}_0, \psi_1 >^2 \\ &+ < X-\overline{X}_0, \psi_2 >^2 + < X-\overline{X}_0, \psi_3 >^2) \\ =& (< X-\overline{X}_1, \psi_1 >^2 - < X-\overline{X}_0, \psi_1 >^2) + (< X-\overline{X}_1, \psi_2 >^2 - < X-\overline{X}_0, \psi_2 >^2) \\ &+ (< X-\overline{X}_1, \psi_3 >^2 - < X-\overline{X}_0, \psi_3 >^2) \end{aligned}$$

3.5 Classification à l'aide du noyau

Le dernier classificateur présenté dans ce chapitre est celui basé sur le noyau. Nous avons consulté les sources [Wu et al., 2013] et [Andrecut, 2020] pour réaliser cette section. L'objectif d'une telle classification est d'utiliser un classificateur linéaire (plus simple à utiliser) pour résoudre un problème linéaire à l'aide d'une fonction noyau $\mathcal{N}$. En effet, les fonctions noyaux peuvent capturer des caractéristiques importantes des courbes, facilitant ainsi la discrimination entre différentes classes de formes fonctionnelles. Une des méthodes de classification à l'aide du noyau les plus utilisées est la méthode de la machine à vecteur de support des moindres carrés (Least Squared Support Vector Machine), notée LS-SVM. La méthode LS-SVM est conçue pour résoudre des problèmes de classification en utilisant une approche basée sur la régression. Elle est particulièrement utile lorsque la séparation parfaite des classes n'est pas possible et lorsqu'une certaine tolérance aux erreurs est acceptable.

Pour cette classification, considérons un jeu de données X divisé en K classes. Une observation $x_i \in \mathbb{R}^M$ de la classe C_k $(k = 1, ..., K)$ se traduit par un vecteur $y_i \in \mathbb{R}^K$ tel que chaque élément de y_i vaut 0 sauf le k-ème qui vaut 1. On, $\forall i = 1, ..., n$ et $\forall j = 1, ..., K$

$$x_i \in C_k \Longleftrightarrow y_{ij} = \begin{cases} 1 & \text{si } j = k \\ 0 & \text{sinon.} \end{cases} \tag{3.4}$$

De façon général, maximiser la probabilité qu'une observation appartienne à une certaine classe. Ainsi, la classification d'une nouvelle observation x est déterminée par

$$x \in C_k \Longleftrightarrow k = arg \max_{l=1,...,K} g_l(x),$$

Avec g_l une fonction softmax, appelée aussi fonction exponentielle normalisée. Ce type de fonction est utilisé en machine learning pour convertir un score en probabilité pour la classification. Le but est donc de transformer un vecteur $z = (z_1, ..., z_K)$ de K dimensions correspondant aux scores de classes en une distribution de probabilité, c'est à dire en un autre vecteur $g_l(z)$ de K nombres réels strictement positifs et de somme valant 1. On note

$$g_l(z) = \frac{exp(z_l)}{\sum_{k=1}^{K} exp(z_k)}.$$

Pour cette classification, on veut regarder à quel point x est proche des autres observations grâce à une fonction de qualité g_l qui est une fonction noyau. Comme chaque fonction f peut être écrite comme une combinaison linéaire des fonctions propres de $\mathcal{N}$ (voir 2.5), on prend

$$g_l(x) = \frac{\sum_{i=1}^{n} exp(\mathcal{N}(x, x_i)a_{il} + b_l)}{\sum_{j=1}^{K}\sum_{i=1}^{n} exp(\mathcal{N}(x, x_i)a_{ij} + b_j)}.$$

Les a_{ij} correspondent aux estimateurs de Lagrange permettant de trouver les points stationnaires (minimum/maximum) d'une fonction dérivable et b_j correspond au biais, c'est à dire à la différence entre une estimation et la vraie valeur du paramètre estimé. L'objectif est alors de déterminer les paramètres optimaux a_{ij} et b_j $(\forall i = 1, ..., n$ et $\forall j = 1, ..., K)$. Pour ce faire, nous devons résoudre le problème d'optimisation suivant ([Andrecut, 2020])

$$W = arg \min_{W} ||\Phi W - Y||^2 + \epsilon ||W||^2,$$

avec les matrices Φ, W et Y définies comme ci-dessous.

Soient $\epsilon > 0$ un paramètre de régularisation, $u = [1, ..., 1]^T$ un vecteur de dimension n, $I_{n\times n}$ la matrice identité de dimensions $n \times n$ et N la matrice du noyau qui respecte le théorème 2.7.1. On définit Φ la matrice du noyau étendu

$$\Phi = \begin{pmatrix} 0 & u^T \\ u & N + \epsilon I_{n\times n} \end{pmatrix}$$

Soient a_{ij} les estimateurs de Lagrange et b_j le biais. On définit W la matrice $(n+1) \times K$ telle que

$$W = \begin{pmatrix} b_1 & \dots & b_K \\ a_{11} & \dots & a_{1K} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nK} \end{pmatrix}$$

Soient y_{ij} les éléments des vecteurs y_i définis en 3.4. On définit la matrice Y de dimension $(n+1) \times K$ telle que :

$$Y = \begin{pmatrix} 0 & \dots & 0 \\ y_{11} & \dots & y_{1K} \\ \vdots & \ddots & \vdots \\ y_{n1} & \dots & y_{nK} \end{pmatrix}$$

Pour une telle classification, l'utilisation de noyaux permet de modéliser des relations non linéaires entre les variables fonctionnelles, offrant ainsi une grande flexibilité dans la modélisation des données. En revanche, le choix du noyau approprié peut avoir un impact significatif sur la performance du modèle. Sélectionner le bon noyau dépend souvent de la nature des données fonctionnelles, ce qui peut être un défi.

Chapitre 4

Evaluations numériques

Pour procéder aux simulations des différents classificateurs étudiés dans le chapitre 3, nous utilisons le logiciel RStudio et plus particulièrement le package FDA.USC. Dans ce chapitre, nous nous penchons sur l'évaluation comparative de quatre méthodes de classification avec cross-validation leave-one-out. Ces méthodes incluent la méthode k-NN, une approche basée sur la profondeur, une méthode de classification à l'aide de noyaux, et la méthode de l'arbre de décision. Nous ne faisons pas la classification binaire par projection à deux dimensionss car celle-ci est présentée, avec détails des simutions, à la référence [Atto, 2021]. Nous avons choisi de réaliser les simulations des différents classificateurs sur plusieurs jeux de données différents afin d'offrir une perspective plus complète sur la performance des classificateurs et de favoriser la généralisation des résultats. En effet, cela permet de déterminer si les performances observées sont spécifiques à un ensemble de données particulier ou si la bonne performance du classificateur est plus largement vérifiée.

Chaque méthode est soumise à une série de tests, où divers paramètres seront explorés pour mesurer l'impact de ces variations sur les performances de classification. Pour la méthode k-NN, nous examinons différentes valeurs de k, déterminant ainsi le nombre optimal de voisins à considérer. Dans le cas de l'approche basée sur la profondeur, différentes profondeurs sont explorées pour évaluer leur influence sur la qualité des résultats. La méthode à l'aide du noyau est testée avec différents types de noyaux, offrant une perspective sur l'effet de ces choix sur la classification. Enfin, pour la méthode de l'arbre de décision, divers scores d'évaluation seront considérés dans le contexte d'une analyse en composantes principales fonctionnelles (ACPF).

Les jeux de données, avec chacun des caractéristiques uniques, sont un terrain d'expérimentation pour évaluer la robustesse et l'efficacité des méthodes de classification dans des contextes variés. La diversité des paramètres testés permet une exploration approfondie des performances de chaque classificateur, et les conclusions tirées de ces analyses fournissent des informations précieuces sur la sélection appropriée des méthodes de classification en fonction des caractéristiques des données et des objectifs spécifiques.

4.1 Jeux de données et lissage des données

Les simulations sont réalisées sur trois jeux de données : `Growth` et `Phoneme` disponibles sur R et `Rainfall`, un jeu de données qui vient de la source [Archive, 1992]. Premièrement, `Growth` est un jeu de données concernant la croissance. Il représente l'évolution de la taille de 39 garçons et 54 filles sur trente-et-un âges différents entre un an et dix-huit ans. Ensuite, un phonème est la plus petite unité sonore qui peut entraîner un changement de sens dans une langue donnée. Les phonèmes ne sont pas les sons physiques eux-mêmes, mais plutôt des entités abstraites représentant des catégories de sons qui sont perçues comme équivalentes par les locuteurs d'une langue. Le jeu de données `Phoneme` correspond aux log-périodogrammes de 250 phonèmes discrétisés sur cent-cinquante fréquences. Cela correspond à une estimation de la densité spectrale de puissance

d'un signal selon la fréquence d'un son, c'est-à-dire selon le nombre de cycles d'oscillations par seconde. La fréquence est directement liée à la hauteur perçue d'un son. Les phonèmes de ce jeu de données sont relatifs au langage anglophone et sont classés selon cinq types avec 50 phonèmes par type. Ceux de la classe 1 correspondent au son /sh/, ceux de la classe 2 au son /iy/, ceux de la classe 3 au son /dcl/, ceux de la classe 4 au son /aa/ et enfin ceux de la classe 5 au son /ao/. La source [Thomas and Raskutti, 1998] donne des informations supplémentaires à propos des phonèmes. Le dernier jeu de données utilisé pour tester les classificateurs est `Rainfall`. Il comprend des données concernant les précipitations relevées sur 365 jours pour 190 stations météorologiques en Australie. L'objectif est de classer ces stations météorologiques en deux groupes. Nous prenons le groupement effectué avec le quatrième score FOBI dans la source [Critchley et al., 2019]. Ce *working paper* fournit une séparation Est-Ouest. A l'Est, les stations météorologique sont connues pour subir de fortes pluies en automne et au printemps, voir même des cyclones.

Ces jeux de données sont discrétisés. Afin de les rendre fonctionnels, nous allons créer une base de fonctions et lisser les données par le principe du lissage de données de la section 2.1.1. Pour chacun des jeux de données, nous considérons des splines cubiques. En ce qui concerne le nombre de fonctions de base et le paramètre de lissage, nous testons différentes valeurs. Pour le nombre de fonction de base, nous testons toutes les valeurs entre 5 et 30. En ce qui concerne le paramètre de rugosité, nous testons les valeurs suivantes : $1.10^i, 1.10^i$ et 1.10^i pour $i = \{-3, ..., 3\}$. Pour chacun des tests, nous observons les résultats de classifications correctes pour le classificateur k-NN. Ces résultats correspondent à des tableaux de dimensions 26×21 pour les trois jeux de données, cela serait trop lourd et difficile à lire. De ce fait, prenons uniquement l'exemple pour le jeu de données `Growth` pour lequel le tableau est plus facile à représenter. La TABLE 4.1 reprend les propabilités de classifier correctement les observtaions du jeu de données `Gowth` grâce au classificateur k-NN par validation croisée leave-one-out, selon le nombre de fonctions de base et le paramètre de rugosité λ utilisé. Premièrement, on remarque que les paramètres de rugosité λ de 0.001 à 500 donne la même probabilité de classifier correctement les observations. Le deuxième constat est le fait que, mis à par pour $\lambda = 5000$, la probabilité de classifications correctes est identique peu importe le nombre de fonctions de base. L'observation générale est le fait que le nombre de fonctions de base a moins d'importance que le paramètre de rugosité.

Nombre de fonctions de base	$\lambda =$ **0.001 ... 500**	$\lambda =$ **1000**	$\lambda =$ **2000**	$\lambda =$ **5000**
5/6	0.968	0.957	0.946	0.925
7 ⋮ **21**	0.968	0.957	0.946	0.914
22 ⋮ **30**	0.968	0.957	0.946	0.903

TABLE 4.1 – Table des probabilités de classifications correctes à l'aide du classificateur k-NN pour les données `Growth` selon le nombre de fonctions de base et le paramètre de rugosité λ utilisés.

Pour présenter chacun des classificateurs, nous utilisons le même lissage par jeu de données. Nous prenons celui avec les paramètres qui engendrent le moins d'erreurs de classification avec le classificateur k-NN par cross-validation leave-one-out. Par exemple, pour le jeu de données `Growth`, selon la TABLE 4.1, la meilleure probabilité de classifier correctement les observations est de 0.968. Cette dernière apparait dans plusieurs cas. Nous choisissons donc celui avec le moins de fonctions de base, soit 5, et un paramètre de rugosité λ entre 0.001 et 500. Ce paramètre ne doit pas être trop faible ni trop élevé pour avoir un compromis entre l'ajustement des données et le lissage, prenons $\lambda = 10$. Nous avons fait des tests similaires pour les jeux de données `Phoneme` et `Rainfall`. On

considère les valeurs de la TABLE 4.2. Le jeu de données `Growth` et le lissage correspondant se trouvent aux FIGURES 4.1 et 4.2. Le jeu de données `Phoneme` est représenté à la FIGURE 4.3 et le lissage à la FIGURE 4.4. Enfin, les FIGURES 4.5 et 4.6 représentent le jeu de données `Rainfall` ainsi que les données lissés. Pour `Phoneme` et `Rainfall`, la grande quantité d'observations ne permet pas la distinction directe des classes. Celles-ci sont disponibles aux annexes A et B pour `Phoneme` et `Rainfall` respectivement.

	`Growth`	`Phoneme`	`Rainfall`
Nombre de fonctions de base	5	11	7
Paramètre de rugosité	10	200	10

TABLE 4.2 – Nombre de fonctions de base et valeur du paramètre de rugosité qui engendrent le moins d'erreurs de classification avec le classificateur k-NN par cross-validation leave-one-out.

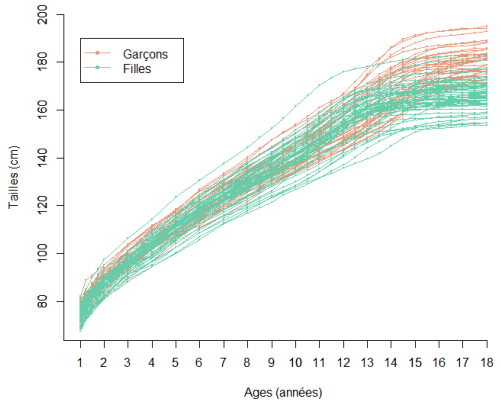

FIGURE 4.1 – Representation du jeu de données `Growth` (données discrètes).

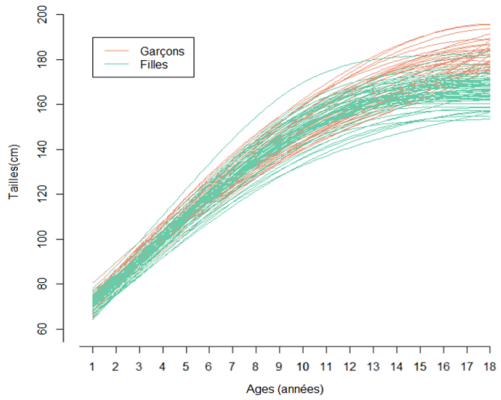

FIGURE 4.2 – Representation du jeu de données `Growth` (données lissées).

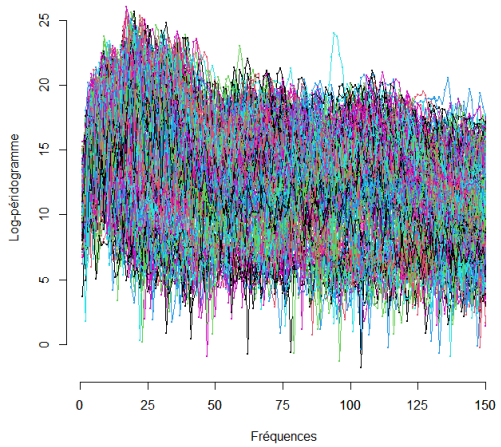

FIGURE 4.3 – Representation du jeu de données `Phoneme` (données discrètes).

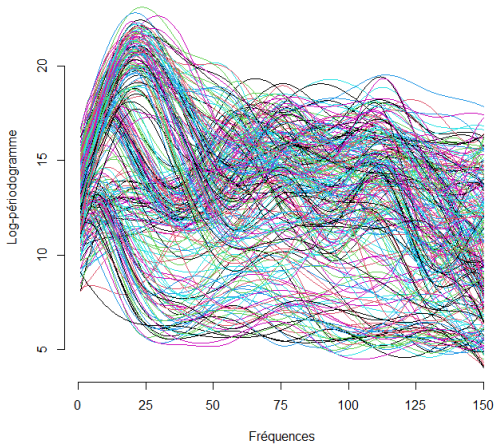

FIGURE 4.4 – Representation du jeu de données `Phoneme` (données lissées).

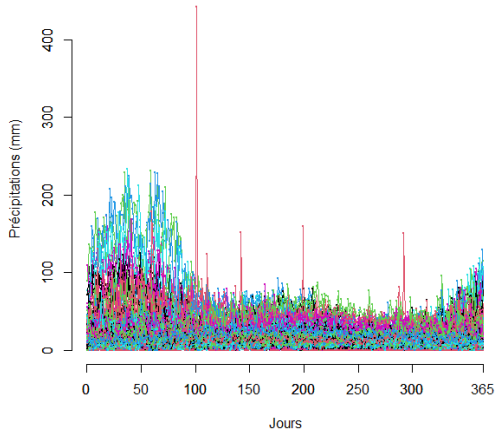

FIGURE 4.5 – Representation du jeu de données `Rainfall` (données discrètes).

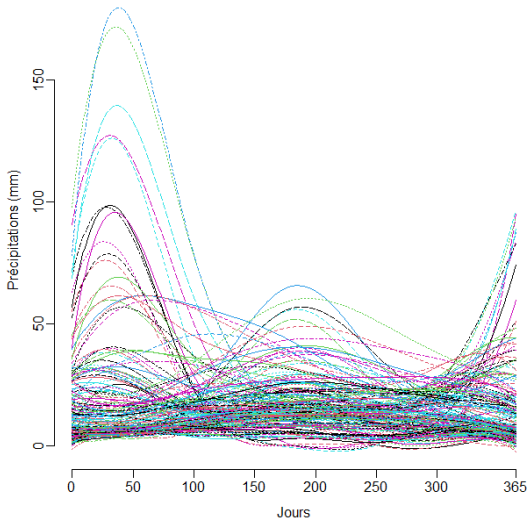

FIGURE 4.6 – Representation du jeu de données `Rainfall` (données lissées).

4.2 Classification k-NN

Pour chacun des jeux de données présentés à la section 4.1, nous cherchons le k qui minimise les erreurs de classification, c'est-à-dire, le k qui donne la meilleure *accuracy*, qui correspond à la probabilité d'obtenir une classification correcte à l'aide du classificateur k-NN, pour une validation croisée leave-one-out. La métrique considérée est sur L^2. Ces différentes probabilités sont données à la TABLE 4.3 pour les k de 1 à 15 pour les trois jeux de données considérés. Nous avons fait les tests pour tous les k possibles mais nous n'affichons pas tous les résultats.

k	`Growth`	`Phoneme`	`Rainfall`
1	0.936	0.872	0.937
2	0.957	0,760	0.953
3	**0.968**	0.860	0.942
4	0.968	0.816	0.958
5	0.957	0.876	0.948
6	0.968	0.848	**0.963**
7	0.958	0.892	0.911
8	0.958	0.856	0.921
9	0.957	**0.912**	0.921
10	0.957	0.888	0.932
11	0.957	0.904	0.921
12	0.957	0.880	0.926
13	0.957	0.892	0.916
14	0.957	0.884	0.921
15	0.957	0.888	0.921

TABLE 4.3 – Table des probabilités de classifications correctes à l'aide du classificateur k-NN pour les données `Growth, Phoneme` et `Rainfall` avec k de 1 à 15.

4.2.1 `Growth`

Le nombre de voisins qui donne la meilleure probabilité de classer une nouvelle observation pour le jeu de données `Growth` par la méthode k-NN est 3. Cette probabilité est de 0.968. Nous avons testé tous les k supérieurs à 15 jusqu'à considérer l'entièreté du jeu de données et c'est toujours

$k = 3$ qui donne la meilleure classification. La table de contingence pour la classification k-NN, pour $k = 3$, du jeu de données `Growth` est la TABLE 4.4. Cela signifie qu'aucune erreur de classification n'a été faites concernant les garçons et que les courbes de trois filles ont été classifiées comme étant celles de garçons.

	Garçons	Filles
Garçons	39	0
Filles	3	51

TABLE 4.4 – Table de contingence de la classification k-NN pour le jeu de données `Growth`.

4.2.2 `Phoneme`

Selon la TABLE 4.3, considérer 9 voisins permet de meilleurs résultats de classification pour le jeu de données `Phoneme`. La table de contingence correspondante est donnée à la TABLE 4.5. Cette table de contingence nous apprend que toutes les observations des phonèmes 1 et 2 ont correctement été classifiées. En ce qui concerne le phonème 3, trois observations ont été mal classifiées. Enfin, en ce qui concerne les phonèmes 4 et 5, les missclassifications sont plus importantes. En effet, on compte neufs observations mal classifiées pour le phonème 4 et dix pour le phonème 5. Mis à part une observations du phonème 5 classée dans le phonème 3, les missclassifications des deux dernières classes sont l'une avec l'autre. Elles sont plutôt nombreuses mais cela ne veut pas dire que le classificateur est mauvais. Cela peut simplement signifier que ces deux classes sont très proches, c'est à dire que les sons /aa/ et /ao/ sont proches. En effet, le son /aa/ est le symbole phonétique représentant le son de la voyelle "a" dans des mots tels que "car" en anglais. En ce qui concerne le son /ao/, c'est le symbole phonétique représentant le son "au" dans des mots tels que "caught" en anglais. La proximité sonore donne donc effectivement du sens à ses missclassifications.

	Phoneme 1	Phoneme 2	Phoneme 3	Phoneme 4	Phoneme 5
Phoneme 1	50	0	0	0	0
Phoneme 2	0	50	0	0	0
Phoneme 3	1	2	47	0	0
Phoneme 4	0	0	0	41	9
Phoneme 5	0	0	1	9	40

TABLE 4.5 – Table de contingence de la classification k-NN pour le jeu de données `Growth`.

4.2.3 `Rainfall`

Pour le jeu de données `Rainfall`, sept observations sont missclassifiées par rapport à la classification déterminée par IC4. La table de contingence correspondante est la TABLE 4.6. Ces résultats sont obtenus avec 6 voisins pour une probabilité de classifications correctes de 0.963. Sept observations sont considérées comme des villes de l'Ouest (en gris) alors qu'il s'agit de villes classifiées initilement comme étant de l'Est (en noir). En revanche, aucune ville de l'Est n'a été missclassifiée. Les missclassifications sont entourées en rouge à la FIGURE 4.7. La séparation Ouest-Est initiale ne correspond pas à une moitié de pays mais plutôt à la côte (Sud) Est contre le reste du pays. Les missclassifications semblent donc cohérentes, en particulier pour les quatre observations en rouge les plus à l'Ouest. Celles-ci étaient initialement considérées comme villes de l'Est par IC4 probablement parce que les tempêtes tropicales et les systèmes dépressionnaires peuvent entraîner des pluies abondantes le long des côtes.

	Est	Ouest
Est	66	7
Ouest	0	117

TABLE 4.6 – Table de contingence de la classification k-NN pour le jeu de données `Rainfall`.

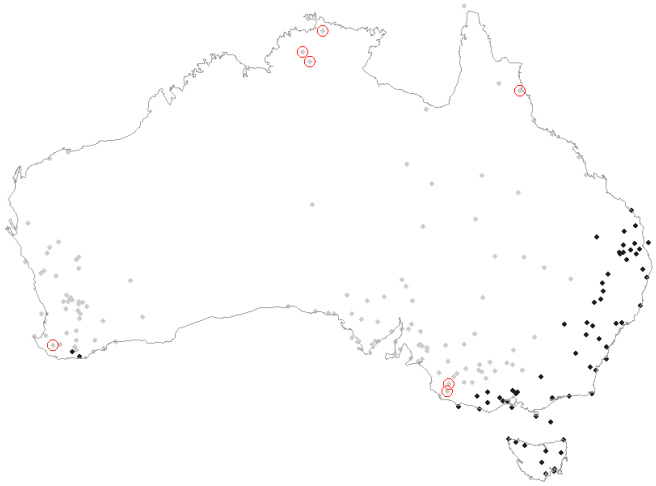

FIGURE 4.7 – Representation des missclassifications par k-NN du jeu de données `Rainfall`.

4.2.4 Conclusion sur le classificateur k-NN

Le classificateur k-NN donne des résultats satisfaisants pour les trois jeux de données. En analysant ces résultats, on peut observer que plus le nombre de données est élevé, moins la probabilité de correctes classifications est élevée. En effet, le jeu de données `Growth` comporte 93 observations et le classificateur k-NN a fournit une probabilité de classifier correctement les observations de 0.968. Cette probabilité est de 0.912 pour le jeu de données `Phoneme` qui comporte 250 observations. Pour le jeu de données `Rainfall`, de 190 observations, cette probabilité est de 0.963. Les trois résultats restent néanmoins très proches. On peut donc supposer que la complexité du jeu de données, en terme de quantité de données, influence légèrement les résultats de la classifcation k-NN.

Ensuite, plus le jeu de données comporte d'observations, plus le nombre de voisins optimal sera élevé. En effet, la classification k-NN optimale nécessite trois voisins le jeu de données `Growth`, neuf pour le jeux de données `Phoneme` et six pour le jeu de données `Rainfall`. Cela confirme le résultat théorique sur le consitance universelle de k-NN présenté à la section 1.5.1.

De plus, les résultats (non présentés dans la TABLE 4.3) pour des k bien supérieurs à 15 prouvent que des valeurs de k trop élevées donnent des résultats de plus en plus faibles. Cela est logique car prendre $k = n$ voudrait dire que chaque point dans l'ensemble de données est considéré comme un voisin lors de la classification. Cela peut conduire à une sensibilité excessive aux valeurs aberrantes qui peuvent fortement influencer la décision pour chaque point à classifier. De plus, cela peut conduire à un surajustement, où le modèle s'adapte trop précisément aux données d'entraînement et ne se généralise pas bien aux nouvelles données. Ajoutons que prendre $k = n$ augmente la complexité de la classification sans nécessairement apporter d'avantages significatifs en terme de performance prédictive.

4.3 Classification par arbre de décision

Une autre méthode de classification est celle des arbres de décision. Pour appliquer une telle méthode aux données fonctionnelles, nous réduisons la dimension des données fonctionnelles par ACPF et faisons un arbre de décision sur l'approximation en dimension finie, soit sur les K premiers scores principaux. Plusieurs K sont testés pour chaque jeu de données. La mesure de sélection d'attribut considrée est l'indice de Gini.

4.3.1 `Growth`

Pour le jeu de données `Growth`, nous avons lissé les données à l'aide de cinq fonctions de base. De ce fait, nous avons pris en compte les scores 1 à 5 pour tester l'arbre de décision. Les résultats de ces différents tests sont représentés à la TABLE 4.7. A savoir que le pourcentage de variance expliqué est de 84.01% pour le premier score, 12.45% pour le deuxième, 2.42% pour le troisième, 0.91% pour le quatrième et 0.21% pour le cinquième.

Modèles	1	2	3	4
Nombre de scores	1	2	3	4 ou 5
Nombre de noeuds intermédiaires	1	3	3	3
Nombre de noeuds finaux	2	4	5	4
Taux de classifications correctes	68.89%	94.62%	94.62%	94.62%

TABLE 4.7 – Modèles obtenus par arbre de décision selon le nombre de scores principaux pour le jeu de données `Growth`.

Nous pouvons en déduire qu'utiliser un seul score est insuffisant pour trouver une classification satisfaisante. En revanche, considérer deux à 5 scores donne la même probabilté de classifier correctement les observations, soit 94.62%. Nous choisissons le modèle 2 comme meilleure modèle car il utilise moins de scores principaux pour un même résultats que les modèles 3 à 5. Ce modèle est présenté à la FIGURE 4.8. Dans les modèles 2 à 5, les observations missclassifiées sont identiques. Il s'agit des observations 27, 47, 64, 68 et 77. La table de contingence correspondante est donnée à la TABLE 4.8. Celle-ci indique que quatre filles sont missclassifiées et seulement un garçon n'est pas correctement classifié. Les noeuds feuilles (3, 4, 6 et 7) de la FIGURE4.8 sont représentés graphiquement à l'annexe C.

L'arbre de décision de la FIGURE 4.8 peut être analysé grâce aux deux fonctions propres qui ont permi sa division. Ces dernières sont représentées à la FIGURE 4.9. La première division est liée à la fonction propre $X2$. Celle-ci souligne la plus grande différence entre les courbes des deux classes, à savoir le pique de croissance dans la deuxième moitié des courbes chez les garçons. Pour les filles, la deuxième moitié des courbes s'apparente plus à une phase de stabilisation. La première fonction propre, $X1$ permet une seconde division. Cette dernière a l'allure générale de l'ensemble des observations du jeu de données `Growth`. Elle correspond d'ailleurs à 84.01% de la variance expliquée.

	Garçons	Filles
Garçons	38	1
Filles	4	50

TABLE 4.8 – Table de contingence de la classification par arbre de décision pour le jeu de données `Growth` et le modèle 2.

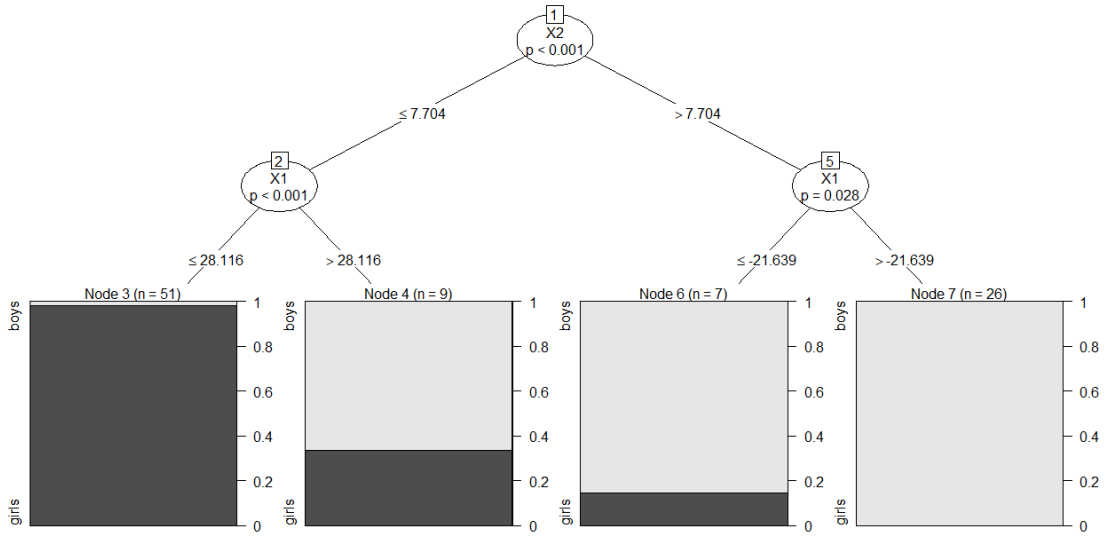

FIGURE 4.8 – Arbre de décision pour le jeu de données `Growth` et le modèle 2.

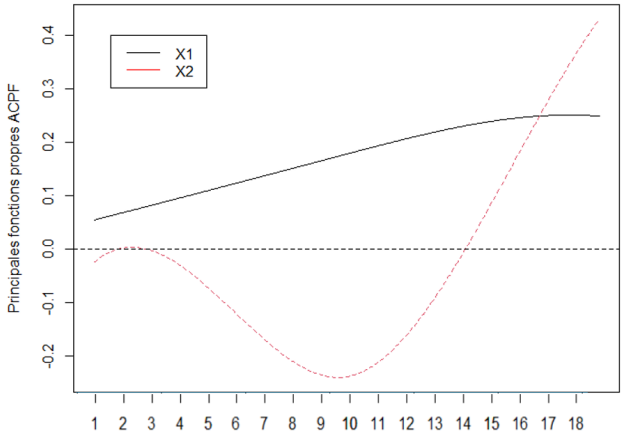

FIGURE 4.9 – Deux premières fonctions propres relatives à l'arbre de décision de la FIGURE 4.8.

4.3.2 `Phoneme`

Nous avons choisi d'utiliser onze fonctions de base pour lisser les données `Phoneme`. De ce fait, nous pouvons tester tous les modèles prenant en compte 1 à 11 composantes principales afin de déterminer lequel engendre le moins d'erreurs de classification. Notons que certains modèles sont identiques pour un nombre de composantes principales considérées différent. La TABLE 4.9 liste ces modèles.

Modèles	1	2	3	4	5	6
Nombre de scores	1	2	3	4	5 à 8	9 à 11
Nombre de noeuds intermédiaires	4	7	8	10	9	9
Nombre de noeuds finaux	5	8	9	11	10	10
Taux de classifications correctes	49,60%	84,80%	88.80%	91.60%	89,20%	88,40%

TABLE 4.9 – Modèles obtenus par arbre de décision selon le nombre de scores principaux pour le jeu de données `Phoneme` et le modèle 4.

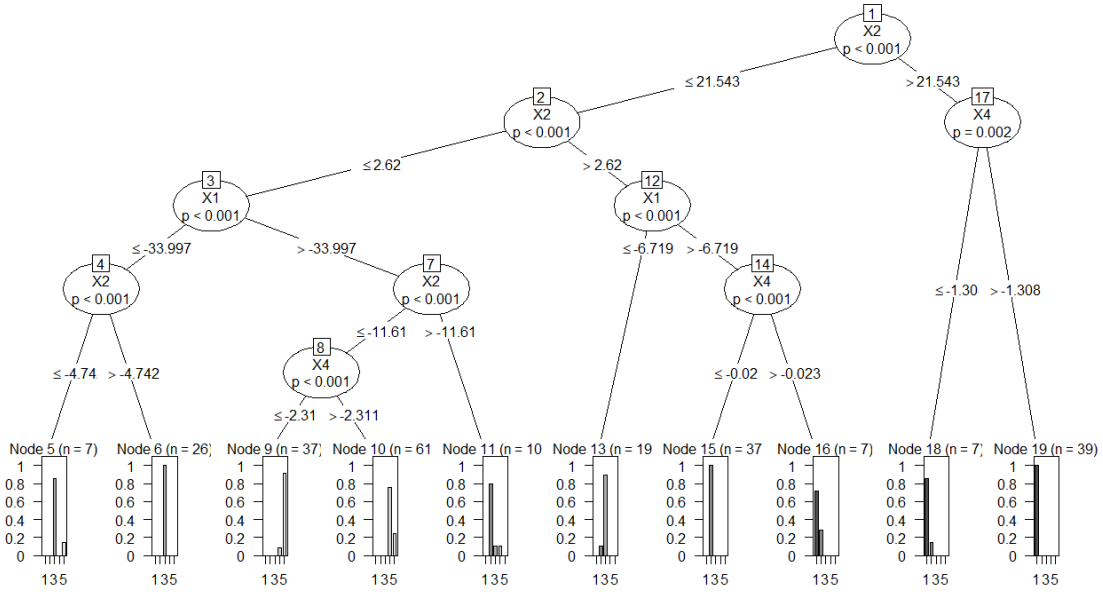

FIGURE 4.10 – Decision tree pour le jeu de données `Phoneme` et le modèle 4.

Le modèle donnant la meilleure probabilité de classer correctement une nouvelle observation est le modèle 4. Pour ce modèle, 21 observations sont missclassifiées sur 250, soit un taux d'erreur de 8.4%. De façon générale, utiliser 2 à 11 scores engendre des résultats plus ou moins proches pour le taux de classifications correctes. En revanche, comme attendu, utiliser un unique score donne un résultat très mauvais avec plus de 50% d'erreur de classification. Ceci s'explique par le fait que la première fonction propre correspond à une variance expliquée de seulement 67.43%. La table de contingence correspondant au modèle 4 est la TABLE 4.10 et ce modèle est représenté à la FIGURE 4.10. Cet arbre étant assez complexe, avec beaucoup de noeuds, nous ne le détaillons pas à l'aide de ses fonctions propres. En revanche, nous pouvons dire que la variance expliquée est de 67.43%, 21,55%, 3,69% et 2.85% pour les fonctions propres 1, 2, 3 et 4 respectivement. Les onze noeuds feuilles de la FIGURE 4.10 sont représentés graphiquement à l'annexe D.

	Phoneme 1	Phoneme 2	Phoneme 3	Phoneme 4	Phoneme 5
Phoneme 1	50	0	0	0	0
Phoneme 2	3	45	2	0	0
Phoneme 3	0	2	48	0	0
Phoneme 4	0	1	0	41	8
Phoneme 5	0	0	0	5	45

TABLE 4.10 – Table de contingence de la classification par arbre de décision pour le jeu de données `Phoneme` et le modèle 4.

4.3.3 `Rainfall`

Le jeu de données `Rainfall` a été lissé à l'aide de sept fonctions de base. De ce fait, sept modèles sont étudiés au maximum. A nouveau, un nombre différent de scores principaux peut engendrer un même modèle. La TABLE 4.11 présente les informations sur ces modèles. Le taux de classifications correctes est toujours en comparaison aux résultats obtenus avec IC4.

Modèles	1	2	3	4
Nombre de scores	1	2	3 à 6	7
Nombre de noeuds intermédiaires	2	4	4	5
Nombre de noeuds finaux	3	5	5	6
Taux de classifications correctes	88.95%	94.21%	95.26%	95.26%

TABLE 4.11 – Modèles obtenus par arbre de décision selon le nombre de scores principaux pour le jeu de données `Rainfall`.

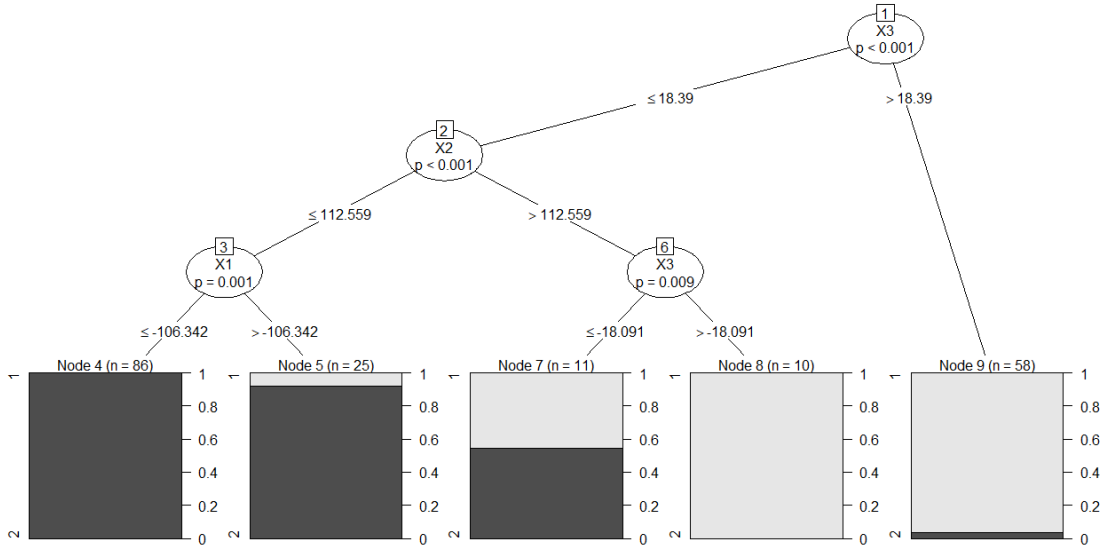

FIGURE 4.11 – Arbre de décision pour le jeu de données `Rainfall` et le modèle 3.

Pour une même probabilité de classer correctement une nouvelle observation, nous choisissons le modèles 3 comme modèle final. En effet, le modèle 4 donne le même taux de correctes classifications mais pour plus de noeuds finaux. De plus, considérer 3 à 6 scores principaux engendre le même modèle donc prenons celui le plus simple, avec 3 scores principaux. Ceux-ci sont liés à une variance expliquée de 73.75% pour le premier, 21.38% pour le second et 3.10% pour le troisième. L'arbre de décision correspondant est représenté à la FIGURE 4.11 et la table de contingence correspondante est la TABLE 4.12.

En ce qui concerne la FIGURE 4.11, nous affichons les trois premières fonctions propres à la FIGURE 4.12. Le premier neoud de décision concerne la troisième fonction propre ($X3$), qui permet de distinguer les observations pour lesquelles on constate plus de grosses pluies en fin d'année. Le deuxième noeud de décision, en lien avec $X2$ provoque une séparation pour les villes qui ont de fortes pluies en milieu d'année. Enfin, la première fonction propre $X1$ permet un troisième noeud de décision pour les données de fortes pluies plutôt en début d'année. C'est d'ailleurs cette première fonction qui explique le plus la variance des données avec 73.75%.

	Est	Ouest
Est	66	7
Ouest	2	115

TABLE 4.12 – Table de contingence de la classification par arbre de décisions pour le jeu de données `Rainfall` et le modèle 3.

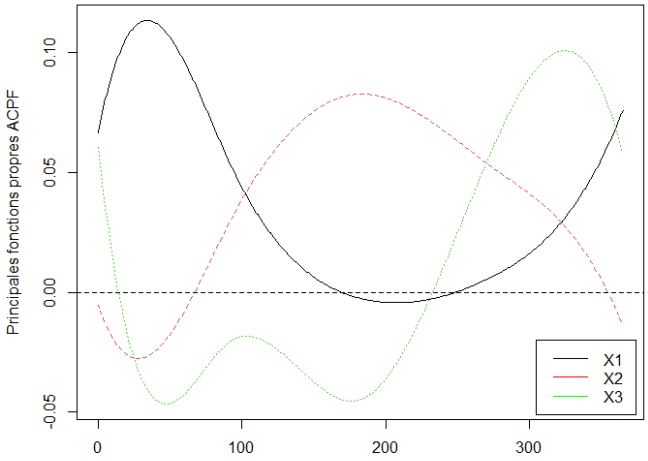

FIGURE 4.12 – Trois premières fonctions propres relatives à l'arbre de décision de la FIGURE 4.11.

Comme nous présentons les missclassifications sur une carte, il est plus représentatif de les montrer toutes en même temps que d'illustrer un à un les noeuds de la FIGURE 4.11. Néanmois, nous pouvons affirmer que le noeud 4 de la FIGURE 4.11 donne une probabilité de 1 d'être à l'Est, le noeud 5 donne une probabilité de 0.920 d'être à l'Est également et pour le noeud 7, cette probabilité est de 0.545. En ce qui concerne la probabilité de classifier les observations à l'Ouest, elle est de 1 pour les observations du noeuds 8 et de 0.966 pour celles du noeud 9. Les observations qui ne sont pas correctement classifiées selon cet arbre de décision sont les observations 88, 93, 21, 22, 25, 106, 132, 147 et 149. Elles sont entourées en rouge à la FIGURE 4.13. On remarque d'allieurs que les missclassifications sont plutôt cohérentes mais on se questionne sur les deux observations entourée en rouge les plus à l'Est. En effet, elle sont classifiée comme étant de l'Ouest par le classificateur d'arbre de décision alors que ce sont les deux villes les plus à l'Est du pays et qu'elles sont entourées uniquement de villes considérées comme de l'Est.

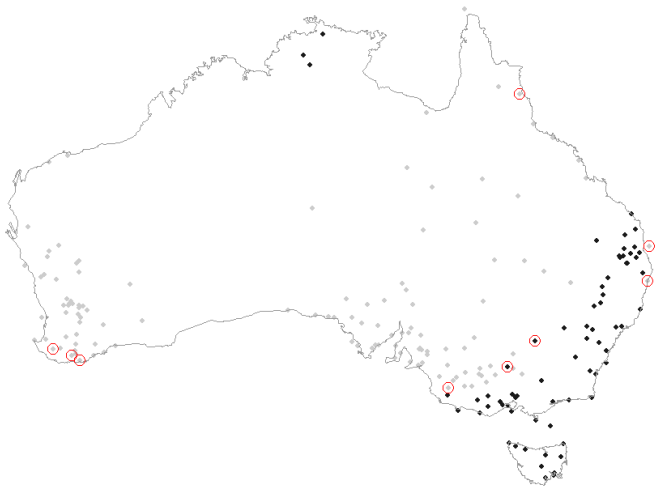

FIGURE 4.13 – Représentation des missclassifications par arbre de décisions pour le jeu de données `Rainfall` et le modèle 3.

4.3.4 Conclusion sur le classificateur par arbre de décision

La classification par arbre de décision est une méthode également satisfaisante avec des probabilités finales de 0.946, 0.916 et 0.953 de classer correctement les observations. A nouveau, le taux de missclassification du jeu de données `Phoneme` est plus élevé que les deux autres. Aussi, pour ce jeu de données, l'arbre est bien plus complexe que pour les données `Growth` ou `Rainfall`. Comme ce n'est pas la première fois qu'on fait ce constat, on peut supposer que le jeu de données `Phoneme` comporte des classes assez proches, ce qui rend la tâche plus difficile au classificateur.

Comme nous avons fait l'ACPF, nous pouvons faire une analyse sur le nombre de scores considérés. Lorsqu'on considère un seul score, les résultats sont très mauvais pour les trois jeux de données. Notamment pour le jeu de données `Phoneme` ou la probabilité de classifier correctement les observations est seulement de 0.496. Notons également que certains scores différents engendre les mêmes modèles. Un dernière remarque est le fait que plus le nombre de fonctions de base nécessaire au lissage des données est élevé, plus le nombre de scores donnant la meilleure classification est élevé aussi. En effet, on a 2 scores pour les données `Growth` lissées par 5 fonctions de bases, 4 scores pour les données `Phoneme` dont le lissage a été effectué par 11 fonctions de bases et enfin 3 scores pour les données `Rainfall` lissées par 7 fonctions de bases.

4.4 Classification basée sur la profondeur

La classification basée sur la profondeur présentée à la 3.2 fait référence à la profondeur de Tukey. Ce n'est pourtant pas la seule que l'on peu appliquer. Pour cette simulation nous nous intéressons égalment à une autre mesure de la profondeur statistique : la profondeur de Fraiman et Muniz (FM). Elle évalue la profondeur d'un point en fonction de la proportion d'observations qui le dominent parmi toutes les combinaisons possibles d'observations de l'ensemble de données. Contrairement à la profondeur de Tukey, la profondeur FM prend en compte toutes les combinaisons possibles, ce qui peut fournir une mesure plus complète de la localisation d'un point. Nous comparons alors les résultats des classifications effectuées en tenant compte de ces deux profondeurs. Ceux-ci sont repris à la Table 4.13. De façon générale, la classification basée sur la profondeur semble donner des résultats moins convainquant que les autres classificateurs.

	`Growth`	`Phoneme`	`Rainfall`
Profondeur de Tukey	0.763	0.376	0.832
Profondeur de Fraiman et Muniz	0.720	0.752	0.937

Table 4.13 – Table des probabilités de classifications correctes basées sur la profondeur pour les données `Growth, Phoneme` et `Rainfall` avec la profondeur de Tukey et la profondeur de Fraiman et Muniz.

4.4.1 `Growth`

Pour le jeu de données `Growth`, retenons la profondeur de Tukey qui donne des résultats légèrement meilleurs que celle de Fraiman et Muniz avec une probabilité de 0.763 de classifier correctement les observations. Nous observons les missclassificatons à la Table 4.14. Au total, aucune observation n'a été missclassifiée pour les garçons. Pour les filles, 22 observations on été missclassifiées, soit environs 40% des filles, ce qui est très élevé comme taux d'erreur.

	Garçons	Filles
Garçons	39	0
Filles	22	32

TABLE 4.14 – Table de contingence de la classification basée sur la profondeur de Tukey pour le jeu de données `Growth`.

4.4.2 `Phoneme`

Pour le jeu de données `Phoneme`, la classification basée sur la profondeur de Fraiman et Muniz donne des résultats deux fois plus satisfaisant que celle de Tukey. En effet, la profondeur de Fraiman et Muniz donne une probabilité de correctes classifications de 0.752 contre 0.376 pour celle de Tukey. La différence est tellement importante qu'il est intéressant de voir les tables de contingence liées aux deux profondeurs. Les résultats de la classification basée sur la profondeur de Fraiman et Muniz sont présentés à la TABLE 4.15. On remarque, comme pour les classificateurs précédents, qu'il y a l'air d'avoir à nouveau une confusion entre les classes 4 et 5. Aussi, c'est la première fois que l'on rencontre des erreurs de classification pour le phonème 1 (10% d'erreurs). Ensuite, les résultats de la classification basée sur la profondeur de Tukey sont présentés à la TABLE 4.16. Cela signifie qu'aucune observation de la classe 1 n'est missclassifiée. En renvanche, plus de 60% des observations de chaque autre classe sont considérees comme des observations de la classe 1, soit au total 202 observations classifiées comme étant des phonèmes 1 contre 50 en réalité, c'est aberrant. En effet, cela voudrait dire que les sons /iy/, /dcl/, /aa/ et /ao/ seraient tous très proches du son /sh/ mais absolument pas proches les uns avec les autres. Cela n'a pas de sens.

	Phoneme 1	Phoneme 2	Phoneme 3	Phoneme 4	Phoneme 5
Phoneme 1	40	9	0	0	1
Phoneme 2	5	40	0	0	5
Phoneme 3	0	0	47	0	3
Phoneme 4	1	7	1	31	0
Phoneme 5	0	2	4	14	30

TABLE 4.15 – Table de contingence de la classification basée sur la profondeur de Fraiman et Muniz pour le jeu de données `Phoneme`.

	Phoneme 1	Phoneme 2	Phoneme 3	Phoneme 4	Phoneme 5
Phoneme 1	50	0	0	0	0
Phoneme 2	43	7	0	0	0
Phoneme 3	38	0	12	0	1
Phoneme 4	38	0	0	11	1
Phoneme 5	33	0	0	3	14

TABLE 4.16 – Table de contingence de la classification basée sur la profondeur de Tukey pour le jeu de données `Phoneme`.

4.4.3 `Rainfall`

Appliquer le classificateur basé sur la profondeur au jeu de donnée `Rainfall` donne de meilleurs propabilités de correctes classifications pour la profondeur de Fraiman et Muniz que pour celle de Tukey. La table de contingenge relative aux classifications basées sur la profondeur de Fraiman et Muniz est la TABLE 4.17, toujours par rapport à IC4. Les données missclassifiées sont en rouge à la FIGURE 4.14. Il y a 8 observations classifiées comme étant des villes de l'Est alors qu'elles sont de l'Ouest selon IC4 et 4 observations inversément. Une bonne partie de ces missclassifications

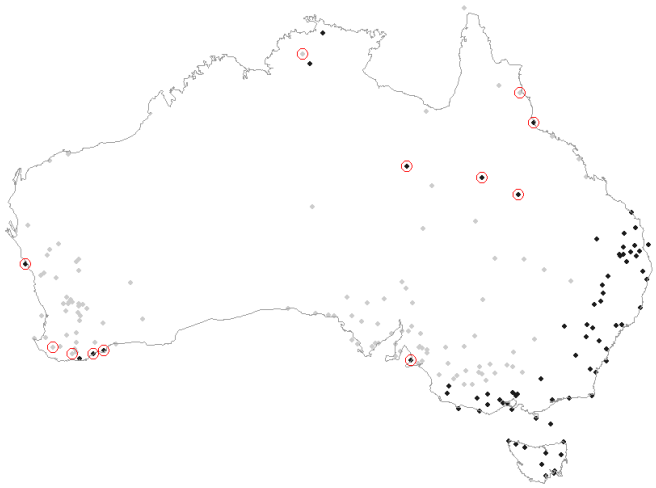

FIGURE 4.14 – Representation des missclassifications du jeu de données `Rainfall` par classification basée sur la profondeur de Fraiman et Muniz.

sont des observations sur les côtes (Sud-Ouest et Noed-Est). Certaines sont condsidérées de l'Est alors qu'elles sont initalement classifiées à l'Ouest et inversément pour des villes très proches les unes des autres, sur les mêmes côtes. Cela peut s'expliquer par les climats ressemblant, tropical au Nord-Est et subtropical au Sud-Ouest.En revanche, les trois onservations noires entourées en rouge (ne se trouvant pas sur les côtes) nous questionnent. En effet, elles sont considérées de l'Est par cette classification alors que toutes les observtaions aux alentours (hors côtes) sont classifiées comme étant de l'Ouest. Cela nous interroge une fois de plus sur la qualité de cette classification.

	Est	Ouest
Est	69	4
Ouest	8	109

TABLE 4.17 – Table de contingence de la classification basée sur la profondeur de Fraiman et Muniz pour le jeu de données `Rainfall`.

4.4.4 Conclusion sur le classificateur basé sur la profondeur

De façon générale, pour l'une ou l'autre profondeur, les résultats sont moins satisfaisants que ceux des autres classificateurs. De plus, la fourchette de probabilités de classifications correctes est très large (entre 0.376 et 0.937), ce qui indique le caractère instable de ce classificateur.

En ce qui concerne la profondeur de Tukey, elle donne des résultats de classifications médiocre avec le jeu de données `Phoneme`. Et même lorsque ces résutats sont plus satisfaisant que Fraiman et Muniz, notamment pour le jeu de données `Growth`, ils restent faibles. On remarque pour les trois jeux de données que, en général, la profondeur de Tukey n'engendre aucune missclassification de la première classe rencontrée mais classifie la plupart des observations des autres classes dans cette première classe. Du point de vue de l'algorithme, les critères de classification de la première classe semble être trop larges, ce qui inclu un nombre excessif et inaproprié d'observations. On peut donc conclure que la classification basée sur la profondeur de Fraiman et Muniz est meilleure que celle basée sur le profondeur de Tukey, bien que celle de Fraiman et Muniz soit loin d'être excellente. Notons qu'il existe d'autres profondeurs fonctionnelles non étudiées dans le cadre de ce mémoire mais qui pourraient peut-être être plus adaptées pour la classification.

4.5 Classification à l'aide du noyau

La dernière classification effectuée est celle à l'aide du noyau présentée à la section 3.5. Nous pouvons tester plusieurs types de noyau pour cette méthode. Ceux que nous allons tester sont les noyaux gaussien, uniforme et d'Epanechnikov. Le noyau gaussien est souvent utilisé lorsque l'on souhaite attribuer plus de poids aux observations proches du point d'estimation alors que le noyau uniforme attribue un poids constant à toutes les observations dans une fenêtre définie. Quand au noyeau d'Epanechnikov, il est utilisé lorsque l'on souhaite un compromis entre le noyau gaussien et le noyau uniforme. Il réduit la variance par rapport au noyau uniforme tout en étant moins sensible aux valeurs extrêmes que le noyau gaussien. Les probabilités de classifier correctement les observations selon le type de noyau sont reprises dans la Table 4.18.

	`Growth`	`Phoneme`	`Rainfall`
Noyau gaussien	0.957	0.892	0.937
Noyau uniforme	0.957	0.9	0.937
Noyau d'Epanechnikov	0.968	0.9	0.937

Table 4.18 – Table des probabilités de classifications correctes à l'aide du noyau pour les données `Growth, Phoneme` et `Rainfall` avec les noyaux gaussien, uniforme et d'Epanechnikov.

4.5.1 `Growth`

Pour cette la classification, le jeu de données `Growth` a de meilleurs résultats avec le noyau d'Epanechnikov, bien que les trois résultats soient proches. Nous avons une probabilité de 0.968 de classifier correctement les observations. La table de contingence correspondante est la Table 4.19. On remarques ainsi que 3 observations sont missclassifiées pour les filles et aucune pour les garçons.

	Garçons	Filles
Garçons	39	0
Filles	3	51

Table 4.19 – Table de contingence de la classification à l'aide du noyau pour le jeu de données `Growth`.

4.5.2 `Phoneme`

Le jeu de données `Phoneme` auquel on applique la classification à l'aide du noyau donne des résultats identiques pour le noyau uniforme et le noyau d'Epanechnikov en terme de probabilités. En revanche, les observations missclassifiées par l'un et l'autre sont légèrements différentes. Nous pouvons le voir grâce aux Table 4.20 et 4.21. Pour les deux méthodes, on remarque encore et toujours une confusion des classes 4 et 5 pour environ un cinquième de leurs observations. Cela reste cohérent étant donnée qu'il s'agit des sons /aa/ et /ao/.

	Phoneme 1	Phoneme 2	Phoneme 3	Phoneme 4	Phoneme 5
Phoneme 1	50	0	0	0	0
Phoneme 2	2	47	1	0	0
Phoneme 3	0	1	49	0	0
Phoneme 4	0	0	0	39	11
Phoneme 5	1	0	0	9	40

Table 4.20 – Table de contingence de la classification à l'aide du noyau uniforme pour le jeu de données `Phoneme`.

	Phoneme 1	Phoneme 2	Phoneme 3	Phoneme 4	Phoneme 5
Phoneme 1	50	0	0	0	0
Phoneme 2	1	47	2	0	0
Phoneme 3	1	2	47	0	0
Phoneme 4	0	0	0	39	11
Phoneme 5	0	0	0	10	40

TABLE 4.21 – Table de contingence de la classification à l'aide du noyau d'Epanechnikov pour le jeu de données `Phoneme`.

4.5.3 `Rainfall`

Nous comparons toujours la classification avec celle obtenue par IC4. Les trois noyaux donnent des probabilités de correctes classifications similaires mais pas les mêmes erreurs de classification. Nous pouvons le voir aux TABLES 4.22. Pour le noyeau gaussien, 41,67% des observations missclassifiées viennent le l'Est et 58,33% de l'Ouest. C'est l'inverse pour le noyau d'Epanechnikov. En ce qui concerne le noyau uniforme, il entraîne cinq fois plus de missclassification de villes de l'Est que de l'Ouest, soit 83,33% des villes de l'Est sont mal classifiées contre 16,66% pour les villes de l'Ouest.

Gaussien	Est	Ouest
Est	68	5
Ouest	7	110

Uniforme	Est	Ouest
Est	63	10
Ouest	2	115

Epanechnikov	Est	Ouest
Est	66	7
Ouest	5	112

TABLE 4.22 – Tables de contingence de la classification à l'aide des noyaux gaussien, uniforme et d'Epanechnikov pour le jeu de données `Rainfall`.

4.5.4 Conclusion sur le classificateur à l'aide du noyau

Ce classificateur donne des résultats de classifications plutôt bons. On remarque à nouveau que plus l'ensemble de données comporte d'observations, moins la probabilité de classifier correctement les observations est élevée. On note les meilleures probabilités à 0.968 pour les 93 données `Growth`, 0.9 pour les 250 observations du jeu de données `Phoneme` et 0.937 pour `Rainfall` comportant 190 observations.

De plus, d'un jeu de données à l'autre, l'application des différents noyaux n'a pas le même impact. En effet, les probabilités de classifier correctement les observations sont identiques pour les trois noyaux pour le jeux de données `Rainfall` mais pas pour les deux autres jeu de données. Le noyau gaussien et le noyau uniforme donne des probabilités de correctes classification similaires pour le jeu de données `Growth`, alors que pour le jeu de données `Phoneme`, ce sont les probabilités basées sur le noyau uniforme et celui d'Epanechnikov qui sont identiques. De façon générale, on peut remarquer que pour chacun des jeux de données, le noyau d'Epanechnikov est ou fait partie des méthodes qui donnent le meilleur résultat de classification. Cela semble cohérent avec sa définition de compromis entre le noyau gaussien et le noyau uniforme.

4.6 Observations générales des missclassifications pour `Growth` et `Phoneme`

Ces analyses se basent sur les classificateurs k-NN, par arbre de décision et à l'aide du noyau. En effet, on ne tient pas compte des missclassifications obenues par le classificateur basé sur la profondeur étant donné les résultats non satisfaisants.

Pour le jeux de données `Growth`, les observations missclassifiées selon les différents classificateurs sont reprises à la TABLE 4.23. Au total, six observations sont mal classifiées pour toutes les classifications cumulées et les observations 47 et 64 sont communes aux trois méhodes. Nous affichons les cinq observations filles classifiées comme étant garçons à la FIGURE 4.15. Les observations 47, 64, 77 et 88 sembles toutes cohérentes car elles font partie des observations les plus élevées parmi toutes les observations filles. En revanche, l'observation 68 chez les filles est étonnament basse pour être missclassifiée. Quand à l'observation 27 (garçon) missclassifiée à la FIGURE 4.16, elle peut être étonnante car on s'attend à ce qu'elle soit plus basse. Finalement, les observations suspectes sont la 27 et la 68 obtenue uniquement par arbre de décision.

Classificateurs	Missclassifications filles	Missclassifications garçons
k-NN	**47, 64**, 88	0
Arbre de décision	**47, 64**, 68, 77	27
A l'aide du noyau	**47, 64**, 88	0

TABLE 4.23 – Observations missclassifiées lors des classification k-NN, arbre de décision et à l'aide du noyau pour le jeu de données `Growth`.

En ce qui concerne les données missclassifiées du jeu de données `Phoneme`, on remarque que pour aucune des trois méthodes de classification il n'y a d'erreur pour les phonèmes de la classe 1. Pour les classes 2 et 3, il y a peu d'erreurs de classification. En revanche, il est intéressant d'afficher les observations mal classifiées pour les phonèmes 4 et 5 car ces derniers ont l'air d'être souvent confondus. Pour rappel, il s'agit des sons /aa/ pour le phonème 4 et /ao/ pour le phonème 5. Ces observations sont affichées aux FIGURES 4.17 et 4.18. En voyant les données côte à côte, on comprend pourquoi les différents classificateurs commettent des erreurs. En effet les deux classes sont proches. Cependant, on remarque quand même que les classificateurs ont tendance à consi-

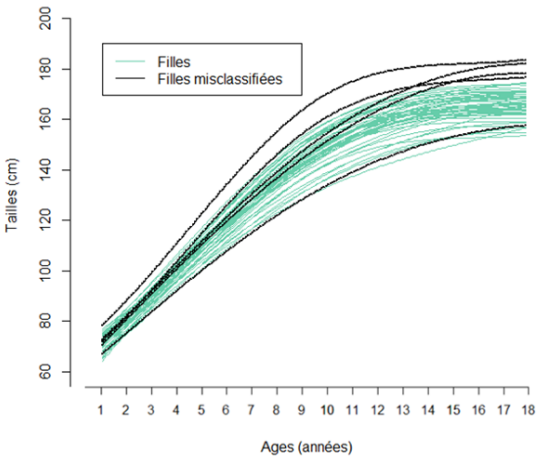

FIGURE 4.15 – Observations filles missclassifiées lors des classification k-NN, arbre de décision et à l'aide du noyau pour le jeu de données `Growth`.

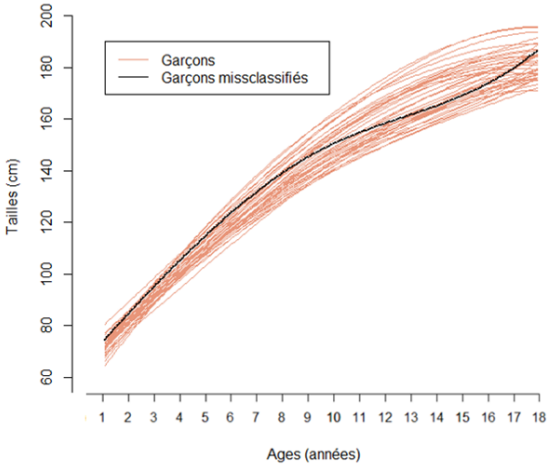

FIGURE 4.16 – Observations garçons missclassifiées lors des classification k-NN, arbre de décision et à l'aide du noyau pour le jeu de données `Growth`.

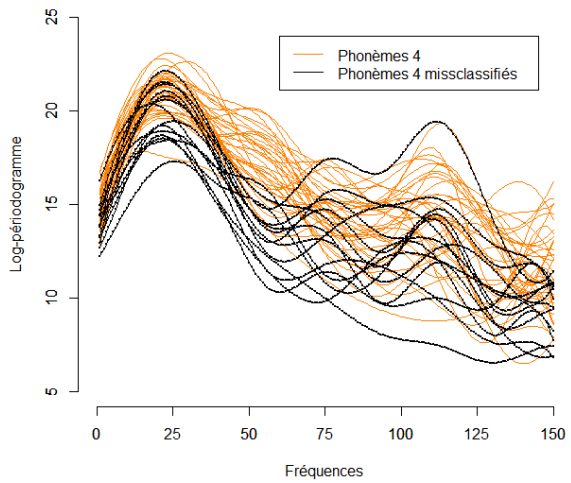

FIGURE 4.17 – Observations phonèmes 4 missclassifiées lors des classification k-NN, arbre de décision et à l'aide du noyau pour le jeu de données `Phoneme`.

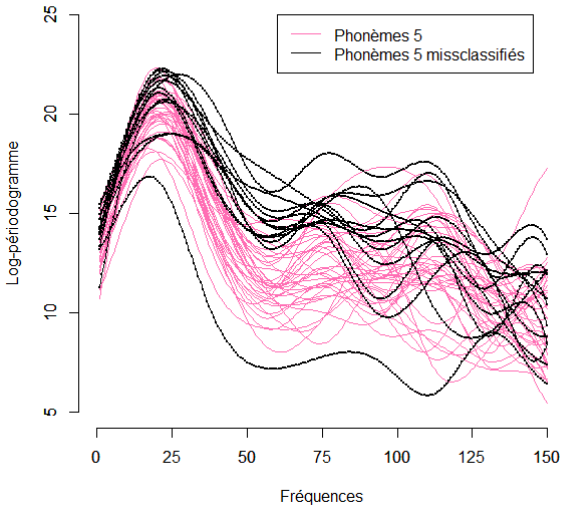

FIGURE 4.18 – Observations phonèmes 5 missclassifiées lors des classification k-NN, arbre de décision et à l'aide du noyau pour le jeu de données `Phoneme`.

dérer les observations de plus faibles log-périodogrammes (plus faible puissance puissance) de la classe 4 (phonème /aa/) comme étant de la classe 5 (phonème /ao/) et les observations avec un des log-périodogrammes plus élevées de la classe 5 comme étant celles de la classe 4. Seule une observation est surprenante, il sagit de l'observation 218 représentée par la courbe noire la plus basse à la FIGURE 4.18 et obtenue uniquement avec la méthode de classification k-NN.

Conclusion

En conclusion de ce mémoire, nous avons parcouru un voyage passionnant à travers l'univers complexe de la classification de données, explorant diverses approches et méthodes pour traiter des données fonctionnelles. Notre objectif principal était de présenter, comprendre et mettre en œuvre des classificateurs de données fonctionnelles, et nous y sommes parvenus (présentation de cinq classificateurs et implémentation avec succès de quatre d'entre eux.)

Les trois premiers chapitres ont posé les bases de notre exploration à travers les principes fondamentaux de la classification, l'analyse de données fonctionnelles, et les différents classificateurs fonctionnels. Cependant, c'est dans le quatrième chapitre que nous avons donné vie à ces concepts théoriques en les confrontant à la réalité des simulations comparatives. L'implémentation de quatre des cinq classificateurs prévus a été une étape cruciale, nous permettant de mesurer leur performance dans des conditions contrôlées.

Les résultats obtenus ont révélé des distinctions intéressantes entre les classificateurs. Les méthodes basées sur le noyau, les k plus proches voisins et l'arbre de décision ont présenté des performances notables, laissant place à des réflexions approfondies sur leur pertinence dans différents contextes. En revanche, le classificateur basé sur la profondeur a montré des résultats plus mitigés, suscitant ainsi un questionnement sur les aspects spécifiques de sa conception ou de son application qui pourraient expliquer ses performances moyennes.

Les perspectives sont vastes et stimulantes. Il serait particulièrement intéressant d'approfondir notre compréhension du classificateur basé sur la profondeur en explorant ses limites et en ajustant ses paramètres pour en améliorer la performance, notamment en testant d'autres profondeurs que celle de Tukey et celle de Fraiman et Muniz. L'exploration de nouveaux classificateurs ainsi que des ajustements de nouveaux paramètres pour les méthodes existantes ouvrent la voie à des découvertes encore plus riches et à des applications plus spécifiques.

Au-delà des aspects techniques, ce mémoire reflète également un parcours personnel. Les moments de doute et de démotivation ont été des épreuves que j'ai réussi à surmonter avec détermination. Aujourd'hui, j'ai beaucoup appris et je suis fière d'avoir relevé ce défi.

Ce mémoire ne marque pas seulement la fin d'un projet académique, mais il représente également l'aboutissement d'un apprentissage de longues années. Les leçons apprises, les compétences acquises et les découvertes réalisées seront des atouts précieux pour les évènements à venir. Merci à tous ceux qui ont contribué, de près ou de loin, à cette aventure.

Annexe A

Représentation des classes du jeu de données `Phoneme`

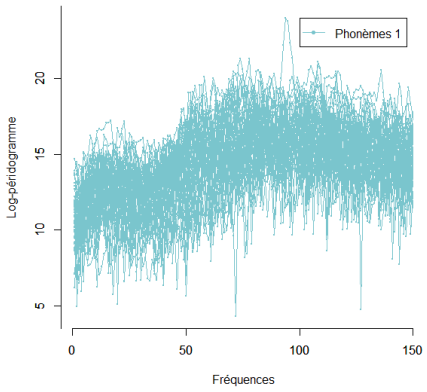

FIGURE A.1 – Représentation du jeu de données `Phoneme` pour la classe 1.

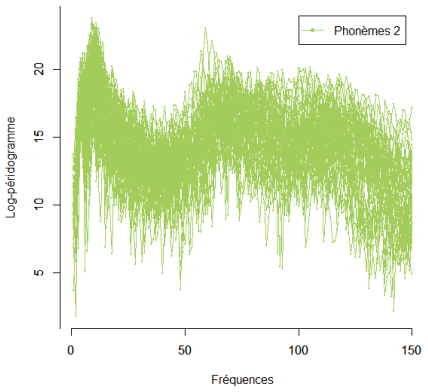

FIGURE A.2 – Représentation du jeu de données `Phoneme` pour la classe 2.

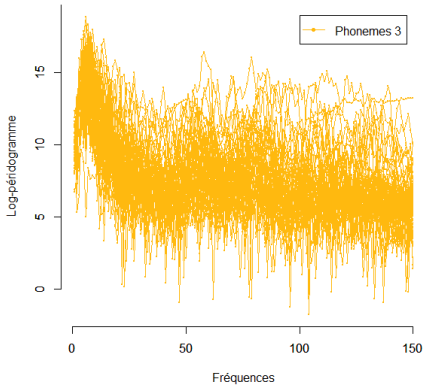

FIGURE A.3 – Représentation du jeu de données `Phoneme` pour la classe 3.

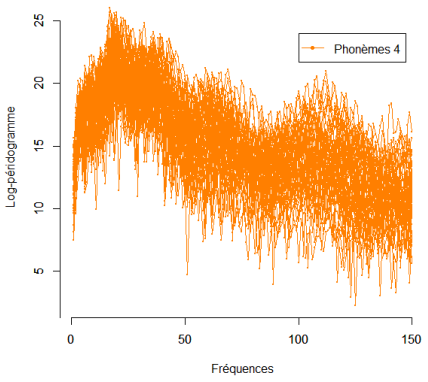

FIGURE A.4 – Représentation du jeu de données `Phoneme` pour la classe 4.

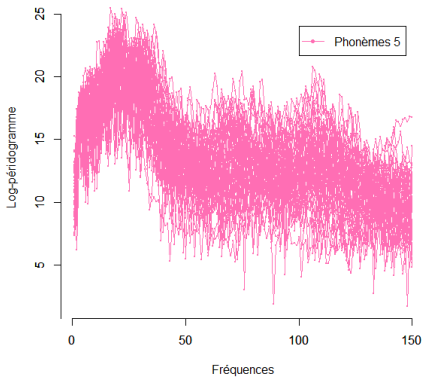

FIGURE A.5 – Représentation du jeu de données `Phoneme` pour la classe 5.

Annexe B

Représentation des classes du jeu de données `rainfall`

FIGURE B.1 – Représentation de la classification du jeu de données `Rainfall` avec IC4.

Annexe C

Représentation des noeuds feuilles de l'arbre de décision pour les données `Growth`

.

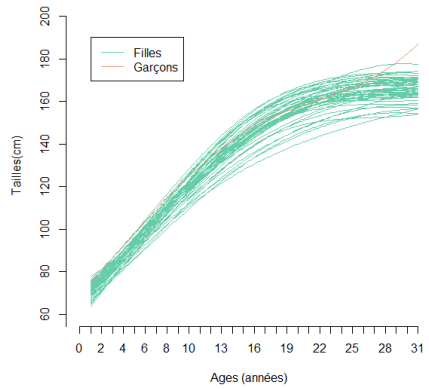

FIGURE C.1 – Représentation du noeud 3 de la FIGURE 4.8 donnant une probabilité de 0.980 d'avoir une fille.

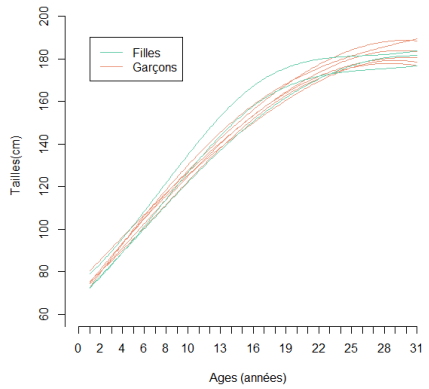

FIGURE C.2 – Représentation du noeud 4 de la FIGURE 4.8 donnant une probabilité de 0.667 d'avoir un garçon.

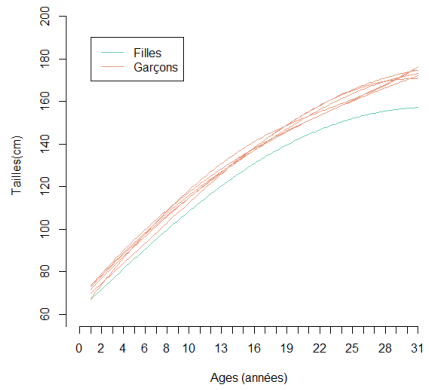

FIGURE C.3 – Représentation du noeud 6 de la FIGURE 4.8 donnant une probabilité de 0.857 d'avoir un garçon.

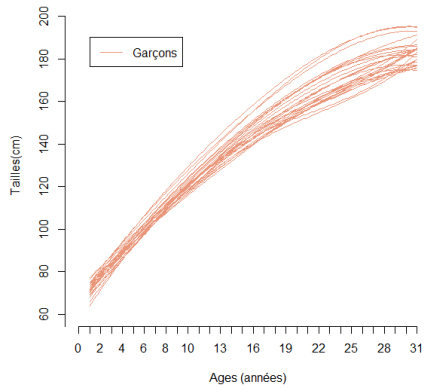

FIGURE C.4 – Représentation du noeud 7 de la FIGURE 4.8 donnant une probabilité de 1 d'avoir un garçon.

Annexe D

Représentation des noeuds feuilles de l'arbre de décision pour les données Phoneme

.

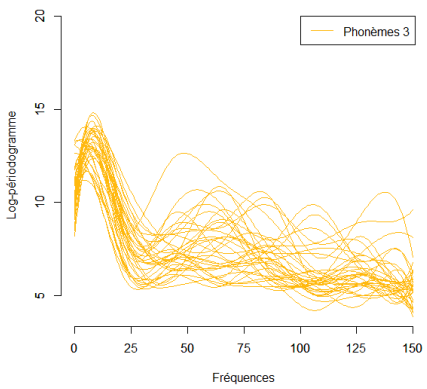

FIGURE D.1 – Représentation du noeud 4 de la FIGURE 4.10 donnant une probabilité de 1 d'avoir le phoneme 3.

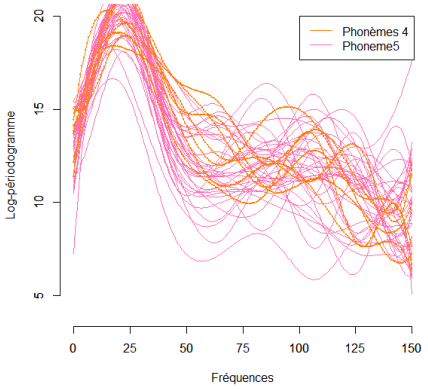

FIGURE D.2 – Représentation du noeud 8 de la FIGURE 4.10 donnant une probabilité de 0.841 d'avoir le phoneme 5.

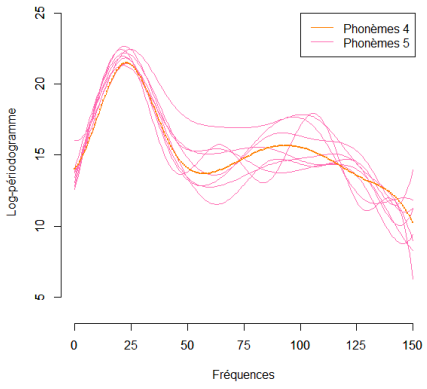

FIGURE D.3 – Représentation du noeud 10 de la FIGURE 4.10 donnant une probabilité de 0.889 d'avoir le phoneme 5.

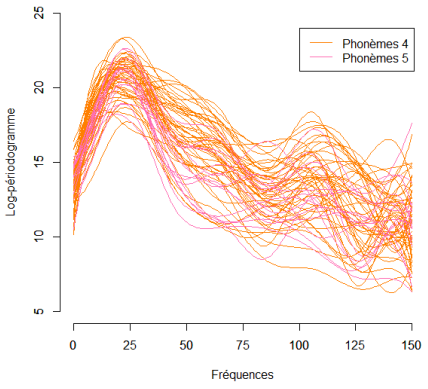

FIGURE D.4 – Représentation du noeud 11 de la FIGURE 4.10 donnant une probabilité de 0.778 d'avoir le phoneme 4.

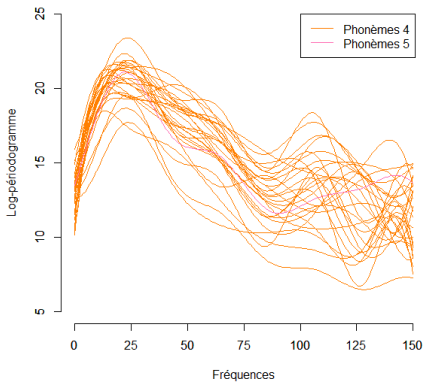

FIGURE D.5 – Représentation du noeud 12 de la FIGURE 4.10 donnant une probabilité de 0.964 d'avoir le phoneme 4.

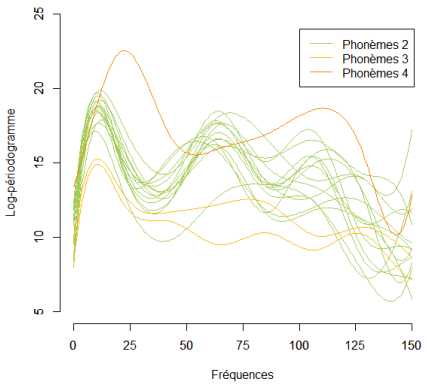

FIGURE D.6 – Représentation du noeud 13 de la FIGURE 4.10 donnant une probabilité de 0.823 d'avoir le phoneme 2.

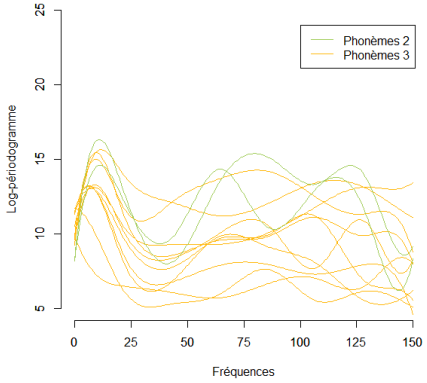

FIGURE D.7 – Représentation du noeud 15 de la FIGURE 4.10 donnant une probabilité de 0.833 d'avoir le phoneme 3.

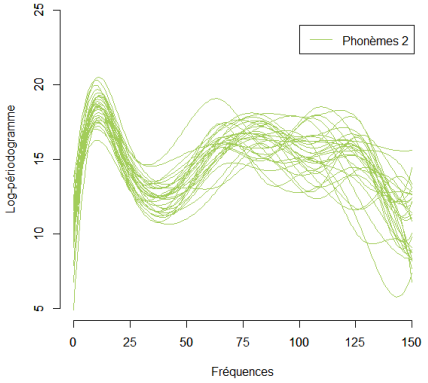

FIGURE D.8 – Représentation du noeud 17 de la FIGURE 4.10 donnant une probabilité de 1 d'avoir le phoneme 2.

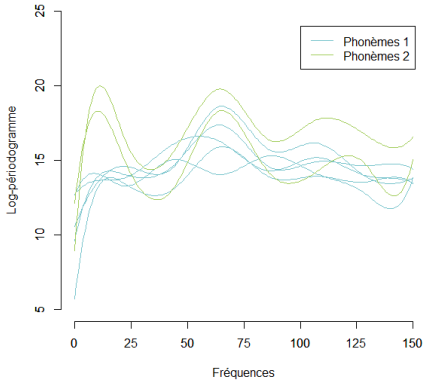

FIGURE D.9 – Représentation du noeud 18 de la FIGURE 4.10 donnant une probabilité de 0.714 d'avoir le phoneme 1.

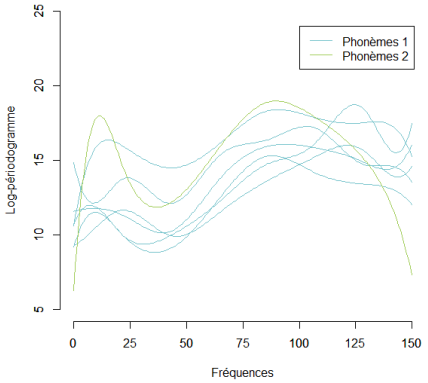

FIGURE D.10 – Représentation du noeud 20 de la FIGURE 4.10 donnant une probabilité de 0.857 d'avoir le phoneme 1.

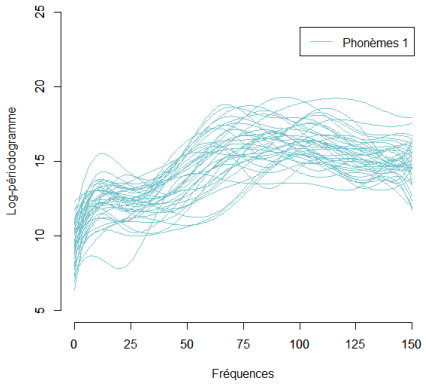

FIGURE D.11 – Représentation du noeud 21 de la FIGURE 4.10 donnant une probabilité de 1 d'avoir le phoneme 1.

Bibliographie

[Andrecut, 2020] Andrecut, M. (2020). K-Means Kernel Classifier. Calgary, Alberta, T3G 5Y8, Canada.

[Archive, 1992] Archive, U. R. D. (1992). NCAR RDA Dataset DS482.1. rda.ucar.edu.

[Atto, 2021] Atto, E. J. (2021). Classification binaire de données fonctionnelles par projection à deux dimensions. Université du Québec à Montréal, pages 22–30, 63–73.

[Balakrishnan and Madigan,] Balakrishnan, S. and Madigan, D. Decision Tree for Functional Variables. Rutgers University.

[Bayart, 2011] Bayart, F. (2011). Splines cubiques.

[Breiman et al., 2017] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (2017). Classification and regression trees.

[Corporation,] Corporation, X. Xoriant Blog on Decision Trees for Classification : A Machine Learning Algorithm.

[Critchley et al., 2019] Critchley, F., Li, B., Oja, H., Sabolová, R., and Van Bever, G. (2019). Functional independent component analysis an extension of the fourth-order blind identification. .

[Ferraty and Vieu, 2006] Ferraty, F. and Vieu, P. (2006). Nonparametric Functional Data Analysis. Springer Science & Business Media.

[Fuchs et al., 2015] Fuchs, K., Gertheiss, J., and Tutz, G. (2015). Nearest neighbor ensembles for functional data with interpretable feature selection. Chemometrics and Intelligent Laboratory Systems, 146 :186–197.

[Fuchs et al., 2017] Fuchs, K., Pöbnecker, W., and Tutz, G. (2017). Classification of functional data with k-nearest-neighbor ensembles by fitting constrained multinomial logit models.

[Gaillard, 2018] Gaillard, P. (2018). Lecture Notes on K-Nearest Neighbors. .

[Lopez-Pintado and Romo, 2007] Lopez-Pintado, S. and Romo, J. (2007). Depth-based classification for functional data.

[Möller and Gertheiss, 2018] Möller, A. and Gertheiss, J. (2018). A classification tree for functional data. ResearchGate.

[Navlani, 2023] Navlani, A. (2023). Decision Tree Classification in Python Tutorial .

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1) :81–106.

[Ramsay et al., 2005] Ramsay, J., Ramsay, J., Silverman, B. W., and Silverman, H. O. W. P. O. M. B. W. (2005). Functional Data Analysis, volume 2. Springer Science & Business Media.

[Sharma, 2021] Sharma, S. (2021). K-Nearest Neighbour : The Distance-Based Machine Learning Algorithm. Analytics Vidhya.

[Thomas and Raskutti, 1998] Thomas, I Zukerman, I. and Raskutti, B. (1998). Extracting Phoneme Pronunciation Information from Corpora.

[Van Bever, 2021] Van Bever, G. (2021). Chapitre 5 : Méthodes classiques de l'analyse multivariée : Classification. (UNamur).

[Wu et al., 2013] Wu, W., Dai, Y., and Lu, L. (2013). Kernels an Kernel Methods.

[Younso and Azhari, 2022] Younso, A Kanaya, Z. and Azhari, N. (2022). Consistency of the k-Nearest Neighbor Classifier for Spatially Dependent Data.