

Faculté des sciences

KERNEL PRINCIPAL COMPONENT ANALYSIS

Auteur : Abdouraman

Promoteur : Johan Segers

Lecteur : Anouar El Gouch

Année académique 2023-2024

Master [120] en sciences des données, orientation statistique

Table des matières

Table des figures	iv
Liste des tableaux	vi
Remerciements	vii
Résumé	ix
Abstract	x
Introduction	1
1 Analyse en Composantes Principales (ACP)	3
1.1 Description de l'ACP	3
1.2 Modèle sous-jacent des données de l'ACP	4
1.3 Variance maximale	4
1.3.1 Cas d'une seule dimension	4
1.3.2 Cas de r dimensions	5
1.4 Erreur minimale	6
1.4.1 Cas d'une seule dimension	6
1.4.2 Cas de r dimensions	8
1.5 Limitations de l'ACP	8
2 Changement d'espace : de l'espace d'origine vers l'espace d'attributs	10
2.1 Correspondance non linéaire	10
2.2 La méthode du noyau	11
2.2.1 Propriétés d'un noyau	12
2.2.2 Astuce de la méthode du noyau : le "kernel trick"	12
2.3 Exemples des noyaux	13
2.3.1 Noyaux polynomiaux	13
2.3.2 Noyau gaussien	15
2.3.3 Autres noyaux	15
2.3.4 Construction des noyaux	16
2.4 Quelques opérations dans l'espace d'attributs à partir du noyau	18
2.4.1 Norme d'un point dans l'espace d'attributs	19
2.4.2 Distance entre deux points dans l'espace d'attributs	19
2.4.3 Moyenne et variance dans l'espace d'attributs	19

2.4.4	Centrer dans l'espace d'attributs	20
2.4.5	Normaliser dans l'espace d'attributs	21
2.4.6	Centrer et réduire dans l'espace d'attributs	21
3	Analyse en Composantes Principales par la méthode du noyau (KACP)	22
3.1	Méthodes spectrales	22
3.2	Description du KACP	24
3.3	L'ACP dans l'espace d'attributs	25
3.3.1	Utilisation explicite de la fonction de correspondance ϕ	25
3.3.2	Utilisation du "kernel trick"	26
3.3.3	Normalisation des axes principaux	27
3.4	Projection des individus sur les composantes principales de l'espace d'attributs	27
3.4.1	Approche Alternative : utilisation de la dualité	28
3.5	Coordonnées factorielles des variables et corrélation	28
3.6	Reconstruction d'erreur dans l'espace d'attributs	29
3.6.1	Erreur de projection	29
3.6.2	Erreur de distance entre les points	29
3.7	Observation supplémentaire	30
3.8	Centralisation et standardisation dans l'espace d'attributs	30
3.9	Propriétés des axes principaux dans l'espace d'attributs	30
3.10	Décompositions d'inertie	31
3.10.1	Qualité de représentation globale	31
3.10.2	Inertie conservée par un axe	31
3.11	Informations auxiliaires	32
3.11.1	Qualité de représentation des variables	32
3.11.2	Contributions	32
3.12	Avantages et désavantages du KACP	33
3.12.1	Comparaison entre l'ACP et le KACP	33
3.13	Algorithme du KACP	34
3.13.1	Étapes à suivre pour la réalisation du KACP	34
3.13.2	Pseudo algorithme du KACP	35
4	Expérience numérique : Données simulées	36
4.1	Application dans le domaine du Clustering	36
4.1.1	Cercles concentriques	36
4.1.2	Demi-lune	42
4.2	Application dans la préservation de la forme d'origine en dimension inférieure	45
4.2.1	"Swiss Manifold"	46
5	Cas pratique : données réelles	48
5.1	Application du KACP en Classification	48
5.1.1	Description de la base de données	48
5.1.2	Visualisation des classes en 3D	49
5.1.3	Pouvoir prédictif	51
5.2	Application du KACP en Régression	55
5.2.1	Description de la base de données	55
5.2.2	Visualisation de la variable réponse en 3D	56

5.2.3 Pouvoir prédicteur	57
Conclusion	61
A Code Python	63
Bibliographie	78

Table des figures

1.1	Illustration de l'ACP en dimension 2 où la première composante principale capture la quasi-totalité de la variance de données projetées.	7
1.2	Illustration de l'ACP en dimension 2 où les données sont non linéaires et l'ACP n'arrive pas à faire une réduction à une dimension optimale.	8
4.1	Données simulées sous forme de deux cercles concentriques.	37
4.2	Utilisation de l'ACP dans l'espace d'origine.	37
4.3	Transformation de la base de données en utilisant la fonction de correspondance $\phi(x, y) = (x, y, 2x^2 + 2y^2)$	38
4.4	Utilisation de l'ACP dans l'espace d'attributs.	39
4.5	Utilisation du KACP avec un kernel linéaire.	39
4.6	Utilisation du KACP avec un kernel polynomial de degré 2 et un terme indépendant de 0.	40
4.7	Utilisation du KACP avec un kernel gaussien de paramètre gamma égale à 1.	40
4.8	Utilisation du KACP avec un kernel gaussien de paramètre gamma égale à 10.	41
4.9	Reconstruction des données des 2 cercles concentriques	42
4.10	Données simulées sous forme de deux demi-lunes distinctes.	42
4.11	Utilisation de l'ACP.	43
4.12	Utilisation du KACP avec un kernel linéaire.	43
4.13	Utilisation du KACP avec un kernel polynomial de degré 2 et un terme indépendant de 1.	44
4.14	Utilisation du KACP avec un kernel gaussien de paramètre gamma égale à 0.385.	44
4.15	Utilisation du KACP avec un kernel gaussien de paramètre gamma égale à 15.	45
4.16	Reconstruction des données des 2 demi-lunes.	45
4.17	Données simulées sous forme de swiss manifold.	46
4.18	Forme d'origine du swiss manifold préservée dans le plan principal.	46
4.19	Forme d'origine du swiss manifold non préservée par le KACP dans le plan principal.	47
5.1	Visualisation des classes de la base de données <i>breast cancer</i> à partir de la projection des individus dans l'espace formé par les trois premières composantes principales de l'ACP classique	49
5.2	Visualisation des classes de la base de données <i>breast cancer</i> à partir de la projection des individus dans l'espace formé par les trois premières composantes principales du KACP utilisant un noyau polynomial.	50

5.3	Visualisation des classes de la base de données <i>breast cancer</i> à partir de la projection des individus dans l'espace formé par les trois premières composantes principales du KACP utilisant un noyau gaussien de paramètre $\gamma = 100$	50
5.4	Visualisation des classes de la base de données <i>breast cancer</i> à partir de la projection des individus dans l'espace formé par les trois premières composantes principales du KACP utilisant un noyau gaussien de paramètre $\gamma = 7 \times 10^{-7}$	51
5.5	Comparaison des Courbes ROC sur la base de données <i>Breast cancer</i> en utilisant l'ACP standard et le KACP avec différents noyaux et différentes valeurs de l'hyperparamètre γ	54
5.6	Visualisation de la tendance dans la base de données <i>white wine</i> à partir de la projection des individus dans l'espace formé par les trois premières composantes principales de l'ACP standard.	56
5.7	Visualisation de la tendance dans la base de données <i>white wine</i> à partir de la projection des individus dans l'espace formé par les trois premières composantes principales du utilisant un noyau quadratique.	57
5.8	Visualisation de la tendance dans la base de données <i>white wine</i> à partir de la projection des individus dans l'espace formé par les trois premières composantes principales du KACP utilisant un noyau gaussien de paramètre $\gamma = 0.18$	58
5.9	Visualisation de la tendance dans la base de données <i>white wine</i> à partir de la projection des individus dans l'espace formé par les trois premières composantes principales du KACP en utilisant un noyau gaussien de paramètre $\gamma = 0.001$	59

Liste des tableaux

5.1	Score moyen de prédiction obtenu en fonction du paramètre γ en faisant une classification en composantes principales (du noyau) sur la base de données breast cancer, en utilisant le nombre optimal des composantes principales (CP=3) et une validation croisée avec $K = 5$.	52
5.2	Score moyen de prédiction obtenu en fonction du nombre de composantes principales en faisant une classification en composantes principales (du noyau) sur la base de données breast cancer, en utilisant le paramètre optimal $\gamma = 7 \times 10^{-6}$ et une validation croisée avec $K = 5$.	52
5.3	Scores de prédiction obtenus en faisant une classification en composantes principales (du noyau) sur la base de données breast cancer en utilisant une validation croisée avec $K = 5$ et $K = 10$ échantillons et 3 composantes principales.	52
5.4	Matrices de confusion (où la première ligne représente la classe réelle M et la deuxième, la classe réelle B alors que la première colonne représente la classe prédite M et la deuxième, la classe prédite B) obtenues en faisant une classification en composantes principales (du noyau) sur la base de données breast cancer en utilisant une validation croisée avec $K = 5$ échantillons.	53
5.5	Scores de prédiction obtenus en faisant une classification en composantes principales (du noyau) sur la base de données breast cancer en utilisant une validation croisée avec $K = 5$ et $K = 10$ échantillons et les paramètres optimaux pour chaque méthode/noyau.	55
5.6	Moyenne des MSE (MMSE) en fonction du nombre de composantes principales en faisant une régression en composantes principales sur la base de données <i>white wine</i> .	58
5.7	Minimum des MMSE (ensemble avec la valeur correspondante du paramètre γ du noyau gaussien) en fonction du nombre de composantes principales en faisant une régression en composantes principales du noyau sur la base de données <i>white wine</i> .	59
5.8	Minimum des MMSE (ensemble avec la valeur correspondante du paramètre γ du noyau gaussien) en fonction du nombre de composantes principales en faisant une régression en composantes principales du noyau sur la base de données <i>white wine</i> .	60
5.9	Minimum des MMSE (ensemble avec la valeur correspondante du paramètre γ du noyau quadratique) en fonction du nombre de composantes principales en faisant une régression en composantes principales du noyau sur la base de données <i>white wine</i> .	60

Remerciements

Ce mémoire a été réalisé sous la supervision du professeur **Johan Segers**. C'est donc en priorité à lui que j'adresse mes plus vifs remerciements non seulement pour le temps qu'il a consacré malgré ses multiples occupations mais aussi pour l'attention particulière qu'il a portée à ce travail. Il a fait preuve d'une réactivité remarquable à chaque fois que je lui envoyais un e-mail pour le rencontrer (durant le premier quadrimestre) ou bien que je remplissais le fichier Excel de prise de rendez-vous (durant le second quadrimestre), je lui en suis vraiment reconnaissant.

L'Université catholique de Louvain (UCLouvain) à travers l'institution qu'elle représente, l'Ecole de Statistique, Biostatistique et Sciences Actuarielles (LSBA) et tous ses personnels en charge de cette formation en data science, sans vous rien de tout ça ne se serait réalisé, je ne peux que vous dire merci. Durant ces années de Master, ma formation scientifique en sciences des données s'est construite à travers des enseignements de qualité reçus grâce à des professeurs dynamiques et pédagogiques. A vous tous mes chers formateurs je tiens à vous exprimer ma profonde gratitude. C'est également l'occasion pour moi de remercier les membres du jury pour la lecture attentive et les commentaires constructifs qu'ils ne manqueront pas d'apporter sur mon travail.

A mes camarades de formation auprès desquels j'ai trouvé refuge lorsque la fatigue se faisait ressentir et lorsque le doute commençait à s'installer, une grande partie de l'achèvement de ce master est rendu possible grâce à vous, je pense notamment à **Keline Laure Meniodem Deliota** avec qui j'ai réalisé la quasi totalité de mes projets de groupe. A travers toi **Laure**, je vous en suis tous reconnaissant.

A ma tendre épouse **Khadyjat** je ne sais comment t'exprimer ma gratitude pour le soutien infaillible, la patience sans limite et l'endurance remarquable dont tu as fait preuve dans la gestion du quotidien social de notre petite famille pendant mes périodes d'absence. A ma fille **Raaniyah** et mes deux fils **Helwane** et **Yazid** tous trois, vous avez été sans le savoir une source de courage et d'inspiration, une lumière qui a guidé mes choix lors des moments difficiles. Il suffisait de penser à vous trois pour réaliser à quel point je devrais impérativement aller au bout de ce que j'avais commencé.

A mon défunt père **Bakari Adama** qui nous a malheureusement quitté en septembre dernier. Papa je sais que tu aurais été une fois de plus fier de moi. Du haut de là où tu te trouves je te dis merci pour le modèle que tu as été et pour l'homme que tu m'as fait devenir. Je te dédie entièrement ce mémoire. A ma tendre mère **Aminatou Oumarou** qui demande sans relâche quand est ce que ces études vont finir. Maman je te dis aussi merci, mon courage prend source de premiers pas d'école qu'on esquissait sous ton accompagnement, je m'en souviens encore comme si c'était aujourd'hui. A mes deux frères cadets **Younoussa** et **Ismailou** je vous dis aussi merci d'avoir épaulé mon épouse pendant tout ce temps quand elle commençait à ressentir le poids de mon indisponibilité.

Et enfin à vous tous qui avez contribué de près ou de loin à la réussite de ce master je vous dis merci. Pour ne pas tous vous citer je pense particulièrement à **NJAPA Christelle Modibbo Ismaila** et j'en passe.

Résumé

Ce travail porte sur l'utilisation de la méthode du noyau/kernel dans l'analyse en composantes principales (KACP). Il vise deux objectifs principaux. Le premier consiste à comprendre et à s'imprégner de la théorie et des fondements mathématiques derrière cette méthode. Il s'agit en effet de montrer pas à pas comment l'artifice du KACP est construit et contenu dans la simple matrice du noyau et ses vecteurs propres. À partir de cette matrice, toutes les analyses pertinentes dans l'espace d'attributs peuvent être effectuées sans faire des transformations lourdes et complexes vers cet espace. À travers le “kernel trick”, nous allons voir qu'il est possible d'opérer dans l'espace d'attributs (de dimension plus élevée) sans pour autant quitter l'espace d'origine (de dimension plus petite), et par conséquent de réduire la complexité informatique liée au temps de calcul engendré par la très grande dimension de cet espace. Le second objectif quant à lui consiste à appliquer la théorie largement revisitée, dans les domaines du clustering, de la classification et de la régression en utilisant non seulement des données simulées mais aussi des cas réelles. Il s'agit de mettre en exergue la capacité de cette méthode à ressortir la structure non linéaire et complexe cachée dans la base des données et le pouvoir prédictif qu'elle offre par rapport à l'ACP classique.

Abstract

This work focuses on the use of kernel component analysis (KPCA). It has two main objectives. The first one is to understand the theory and mathematical foundations behind that method. It is in fact about showing step by step how the backbone of the KPCA is constructed and contained in the simple kernel matrix and its eigenvectors. From this matrix, all relevant analyses in the feature space can be performed without making heavy and complex transformations to this space. Through the kernel trick, we will see that it is possible to operate in the feature space (of higher dimension) without leaving the original space (of smaller dimension), and hence reduce the computational complexity linked to the calculation time generated by the very large dimension of that space. The second objective consists of applying the theory widely revised in the fields of clustering, classification and regression using not only simulated data but also real cases. This is to highlight the capacity of this method to capture the non linear and complex structure hidden in the database and the predictive power it offers compared to the classic PCA.

Introduction

Dans le vaste paysage du machine learning, l'analyse en composantes principales (ACP) s'impose comme une méthode fondamentale dans la réduction de dimension et la visualisation des données. Cependant, sa version traditionnelle se limite aux transformations linéaires et ne parvient pas à capturer les relations non linéaires complexes inhérentes à de nombreux ensembles de données du monde réel.

Le noyau ACP (KACP) est une extension révolutionnaire qui surmonte cette limitation en exploitant la puissance offerte par les fonctions du noyau. Il rend possible l'application de l'ACP classique sur des données à structure non linéaire grâce à la transformation préalable de l'espace d'origine en un nouvel espace appelé espace d'attributs, dans lequel les données transformées deviennent à structure quasi linéaire et où l'ACP classique est à nouveau réalisable. Grâce à l'astuce du kernel connue communément sous le nom de "kernel trick", une transformation explicite des données d'origine point par point n'est même pas nécessaire et est plutôt remplacée de manière équivalente par une matrice du noyau sur laquelle toutes les analyses pertinentes peuvent y être effectuées. Le choix de cette matrice détermine implicitement la fonction de transformation non linéaire et induit par conséquent l'espace d'attributs ainsi que sa dimension. Lorsque l'on connaît ou définit l'expression du noyau, alors il est possible d'effectuer toutes les opérations nécessaires dans l'espace d'attributs, peu importe l'étendue de sa dimension, à partir de l'espace d'origine.

Notre objectif dans ce travail est tout d'abord d'élucider et de présenter la théorie sous-jacente ainsi que les fondements mathématiques liés à la méthode du KACP. Ensuite nous l'appliquerons dans différents domaines tels que le clustering, la visualisation, la classification et la régression afin de faire ressortir de manière concrète sur des données réelles et/ou simulées le potentiel énorme dont elle dispose par rapport à la méthode de l'ACP classique.

Ce mémoire est structuré comme suit. Dans le chapitre 1, nous avons revisité les éléments essentiels de l'ACP classique et avons mis en lumière les limites auxquelles cette méthode fait face. Le chapitre 2 a été l'occasion de s'attaquer à la problématique de la recherche de la représentation adéquate à l'analyse de notre problème et par conséquent au problème de changement d'espace et l'inconvénient majeur qui y est lié. Nous avons introduit dans ce chapitre la méthode du noyau et l'avantage qu'elle offre à travers le "kernel trick". Certaines classes de noyaux éligibles ont été présentés ensemble avec leur propriétés et l'espace d'attributs qu'ils induisent. Pour terminer ce chapitre, nous avons passé en revue les opérations utiles auxquelles nous aurons à faire face dans l'espace d'attributs et qui sont cruciales pour la suite de ce travail. Dans le chapitre 3, notre dévolu a été jeté sur la méthode du KACP proprement dit. Après avoir situé cette dernière parmi les différentes méthodes existantes, nous avons passé au peigne fin la théorie sous-jacente à cette méthode, ses fondements mathématiques, son approche étape par étape, ses avantages

et ses inconvénients. Quant aux chapitres 4 et 5, ils ont été le théâtre d'un vaste champ d'application de la technique du KACP. Dans le premier, nous avons simulé de manière académique des données dans le but de leur octroyer une orientation précise adaptée à la mise en lumière de la problématique et/ou de l'avantage que nous voudrions mettre en exergue et les avons ensuite appliquées dans le domaine du clustering. Dans le second, nous avons à travers des données réelles, appliqué le KACP dans les domaines de la classification et de la régression et avons ressorti le pouvoir prédictif dont dispose cette méthode à travers les multiples choix des noyaux et de leurs hyperparamètres.

Chapitre 1

Analyse en Composantes Principales (ACP)

Lorsque l'on a à faire à des données de grande dimension, il est tout à fait naturel d'envisager la possibilité de les projeter sur un espace de dimension plus petite tout en gardant l'essentiel de l'information qu'elles contiennent. Le processus par lequel ces données sont projetées sur un espace de dimension inférieure est appelé la réduction de dimension.

Les toutes premières méthodes de réduction de dimension qui ont vu le jour étaient des méthodes linéaires. Et la principale d'entre elles est sans doute l'analyse en composantes principales en abrégé ACP.

1.1 Description de l'ACP

L'ACP est la méthode la plus ancienne et la plus connue des méthodes de réduction de dimension. Elle a été introduite dans les années 1900 par K. Pearson [1] puis reprise et développée par H. Hotelling [2]. Elle fait référence au processus par lequel les composantes principales sont calculées et par la même occasion leurs utilisations au service d'une bonne compréhension de données. L'une des utilisations phares dans la compréhension de ces dernières est inéluctablement la visualisation de celles ci dans un espace de dimension raisonnable à la perception humaine. Rappelons que les composantes principales permettent de résumer un large éventail de variables corrélés par un plus petit nombre de variables représentatives qui expliquent collectivement la plupart de la variabilité au sein d'une base de données.

L'ACP est une approche non supervisée qui est donc basée sur la recherche d'une base de dimension r ($r \ll p$, où p est la dimension réelle de la base des données) totalement décorrélée qui capture le mieux la variance des données. La direction avec la plus grande projection de la variance (aussi celle qui minimise l'erreur quadratique moyenne) est appelée la première composante principale et la direction orthogonale à la première (aussi celle qui minimise l'erreur quadratique moyenne y exclue la première), qui capture la deuxième plus grande variance projetée est appelée la deuxième composante principale, ainsi de suite.

Avant toute chose rappelons que les détails sur les développements mathématiques de cette section 1.1 se trouvent dans le livre *Data Mining and Analysis* de M. J. Zaki et M. Wagner [3]. Ce livre constitue l'outil de référence pour les deux premiers chapitres de ce

travail.

1.2 Modèle sous-jacent des données de l'ACP

Le modèle d'ACP suppose essentiellement que les p variables observées, regroupées dans le vecteur aléatoire $\mathbf{x} = (x_1, x_2, \dots, x_p)^T$, résulte d'une transformation linéaire $\mathbf{U}$ de r variables latentes inconnues, écrites sous la forme $\mathbf{y} = (y_1, y_2, \dots, y_r)^T$, où r est la dimension intrinsèque de la base de données. L'on peut alors écrire :

$$\mathbf{x} = \mathbf{U}\mathbf{y},$$

où $\mathbf{U}$ est de dimension $p \times r$. Toutes les variables latentes $\mathbf{y}$ sont supposées avoir **une distribution gaussienne**. De plus, la transformation $\mathbf{U}$ est contrainte à être un axe de changement, ce qui signifie que ses colonnes sont orthogonales les unes aux autres et de norme unitaire. Autrement dit $\mathbf{U}^T \mathbf{U} = \mathbf{I}_r$. Comme nous le verrons dans la suite la dimension r et la matrice $\mathbf{U}$ pourront être déterminées à travers la méthode ACP.

Une autre hypothèse importante de l'ACP est que les deux variables observées $\mathbf{x}$ et latente $\mathbf{y}$ doivent être toutes deux centrées, c'est à dire qu'elles doivent être de moyenne nulle. Par ailleurs les composantes du vecteur observé $\mathbf{x}$ peuvent provenir d'origines très différentes. Il est alors utile dans ce cas de les standardiser c'est-à-dire de diviser chaque composante du vecteur $\mathbf{x}$ par son écart type après centrage, ceci dans le but de les rendre toutes au même pied d'égalité avant l'application de l'ACP.

1.3 Variance maximale

1.3.1 Cas d'une seule dimension

En ACP, il est question de trouver la direction $\mathbf{u}$, vecteur unitaire qui maximise la variance de la projection des points.

Soit $\mathbf{X}$ la matrice $n \times p$ des données centrées, Σ sa matrice de variance-covariance et $\mathbf{x}_i^T$ un point/individu de coordonnées $(x_{i1}, x_{i2}, \dots, x_{ip})$ dans la base standard $(\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_p)$.

La projection d'un point $\mathbf{x}_i$ sur le vecteur unitaire $\mathbf{u}$ est donnée par :

$$\begin{aligned} \mathbf{x}'_i &= \left(\frac{\mathbf{u}^T \mathbf{x}_i}{\mathbf{u}^T \mathbf{u}} \right) \mathbf{u} \\ &= (\mathbf{u}^T \mathbf{x}_i) \mathbf{u} \\ &= a_i \mathbf{u}, \end{aligned}$$

avec $a_i = \mathbf{u}^T \mathbf{x}_i$.

La variance projetée de tous les points sur l'axe unitaire $\mathbf{u}$ est donnée par :

$$\begin{aligned}
\sigma_{\mathbf{u}}^2 &= \frac{1}{n} \sum_{i=1}^n (a_i - \mu_{\mathbf{u}}) \\
&= \frac{1}{n} \sum_{i=1}^n (\mathbf{u}^T \mathbf{x}_i)^2 \\
&= \frac{1}{n} \sum_{i=1}^n \mathbf{u}^T (\mathbf{x}_i \mathbf{x}_i^T) \mathbf{u} \\
&= \frac{1}{n} \mathbf{u}^T \left(\sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^T \right) \mathbf{u} \\
&= \mathbf{u}^T \Sigma \mathbf{u},
\end{aligned}$$

où $\mu_{\mathbf{u}} = 0$ est la coordonnée de la moyenne $\mu = 0$ des données centrées sur l'axe $\mathbf{u}$. Le vecteurs $\mathbf{u}$ qui maximise la variance $\sigma_{\mathbf{u}}^2$ sous la contrainte $\mathbf{u}^T \mathbf{u} = 1$ est donnée en introduisant le Lagrangien $J(\mathbf{u})$ et le multiplicateur de Lagrange α , par :

$$\max_{\mathbf{u}} J(\mathbf{u}) = \max_{\mathbf{u}} (\mathbf{u}^T \Sigma \mathbf{u} - \alpha (\mathbf{u}^T \mathbf{u} - 1)).$$

En dérivant $J(\mathbf{u})$ par rapport à $\mathbf{u}$, puis en égalant le résultat à 0, l'on obtient :

$$\Sigma \mathbf{u} = \alpha \mathbf{u}, \tag{1.1}$$

ce qui est une équation aux valeurs propres, où α est la valeur propre de la matrice variance-covariance Σ associée au vecteur propre $\mathbf{u}$. De plus en prenant le produit scalaire de l'équation (1.1) par $\mathbf{u}^T$ de deux cotés de l'égalité, l'on obtient :

$$\sigma_{\mathbf{u}}^2 = \alpha.$$

Autrement dit maximiser la variance $\sigma_{\mathbf{u}}^2$ revient tout simplement à prendre **la valeur propre λ_1 la plus grande** de la matrice de variance-covariance Σ . Ainsi donc cette plus grande valeur propre λ_1 représente la variance maximale de la projection des points et son vecteur propre associé $\mathbf{u}_1$, vecteur propre dominant, représente la direction recherchée de cette variance maximale.

1.3.2 Cas de r dimensions

De la même manière qu'en dimension 1, pour trouver la dimension r qui représenterait au mieux la base de données $\mathbf{X}$, il suffira de calculer les p valeurs propres de la matrice de variance-covariance Σ et d'en choisir les r premières et leur vecteur propre correspondant.

En ordonnant de manière décroissante les différentes valeurs propres λ_i de la matrice de variance-covariance Σ :

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_r \geq \lambda_p \geq 0,$$

et en notant par $\mathbf{X}'$ la base de données formée par les coordonnées des points projetés, la

variance projetée des points sur les r premières composantes est donnée par :

$$\begin{aligned}\text{Var}(\mathbf{X}') &= \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{P}_r \mathbf{x}_i) \\ &= \sum_{i=1}^r (\mathbf{u}_i^T \boldsymbol{\Sigma} \mathbf{u}_i) \\ &= \sum_{i=1}^r \lambda_i,\end{aligned}$$

avec $\mathbf{P}_r = \mathbf{u}_r \mathbf{u}_r^T = \sum_{i=1}^r \mathbf{u}_i \mathbf{u}_i^T$, la matrice de projection et $\mathbf{u}_r = (\mathbf{u}_1 \quad \dots \quad \mathbf{u}_r)$, la matrice des vecteurs de la base de dimension r .

Quant à la proportion de la variance capturée par ces r premières composantes, elle est donnée par :

$$\begin{aligned}f(r) &= \frac{\lambda_1 + \dots + \lambda_r}{\lambda_1 + \lambda_2 + \dots + \lambda_r + \lambda_{r+1} + \dots + \lambda_p} \\ &= \frac{\sum_{i=1}^r \lambda_i}{\sum_{i=1}^p \lambda_i} \\ &= \frac{\text{Var}(\mathbf{X}')}{\text{Var}(\mathbf{X})},\end{aligned}$$

où

$$\begin{aligned}\text{Var}(\mathbf{X}) &= \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - \mu\|^2 \\ &= \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i\|^2 \\ &= \text{tr}(\boldsymbol{\Sigma}) \\ &= \sum_{i=1}^p \sigma_i^2 \\ &= \sum_{i=1}^p \lambda_i,\end{aligned}$$

est la variance totale de la base de données et est invariant par changement de base. Le choix de r s'effectue lorsqu'une baisse importante se produit dans les valeurs propres à partir de la *rème*.

Illustrons ce concept d'ACP par des données en dimension 2 représentées sur la figure 1.1 où la première composante principale capture la quasi-totalité de la variance (99,26%) permettant ainsi de réduire la dimension de la base de données à 1.

1.4 Erreur minimale

1.4.1 Cas d'une seule dimension

Il s'avère que la direction qui maximise la variance est aussi celle qui minimise la moyenne des erreurs quadratiques. Ce qui est d'ailleurs une très bonne chose dans la

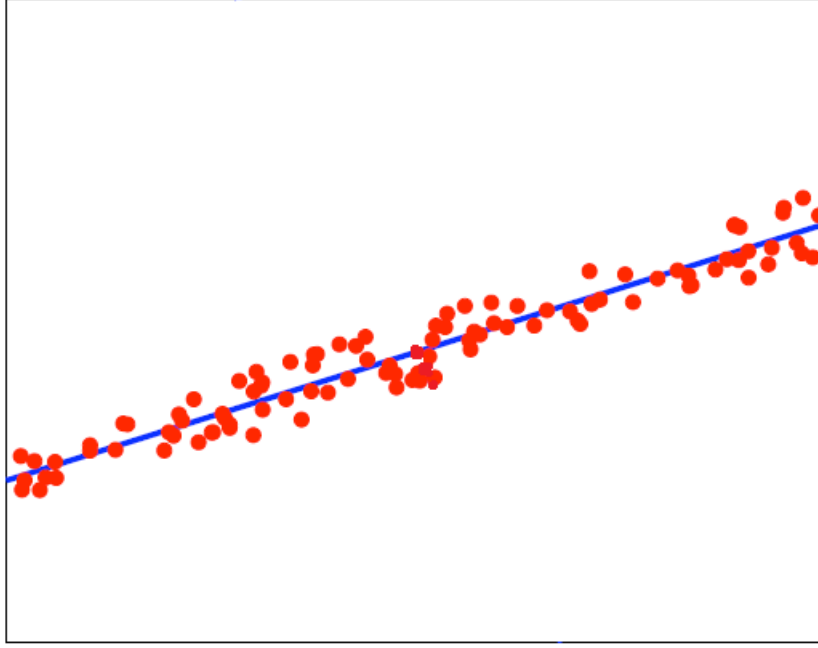


FIGURE 1.1 – Illustration de l’ACP en dimension 2 où la première composante principale capture la quasi-totalité de la variance de données projetées.

mesure où la projection sur cette direction représenterait le mieux la base de données par rapport à toute projection sur une autre direction. En effet, la moyenne du carré des erreurs, le MSE est donnée par :

$$\begin{aligned}
 \text{MSE}(\mathbf{u}) &= \frac{1}{n} \sum_{i=1}^n \epsilon_i^2 \\
 &= \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{x}'_i\|^2 \\
 &= \sum_{i=1}^n \frac{\|\mathbf{x}_i\|^2}{n} - \mathbf{u}^\top \Sigma \mathbf{u} \\
 &= \text{Var}(\mathbf{X}) - \mathbf{u}^\top \Sigma \mathbf{u}.
 \end{aligned} \tag{1.2}$$

Le premier terme de droite de l’égalité $\text{Var}(\mathbf{X})$ étant fixe, l’équation (1.2) montre que le MSE varie inversement à la variance des données projetées sur l’axe $\mathbf{u}$, $\mathbf{u}^\top \Sigma \mathbf{u}$. Ainsi donc la direction $\mathbf{u}$ qui maximise cette variance projetée est bien celle qui minimise le MSE. Pour le vecteur propre dominant u_1 nous avons donc :

$$\text{MSE}(\mathbf{u}_1) = \text{Var}(\mathbf{X}) - \lambda_1.$$

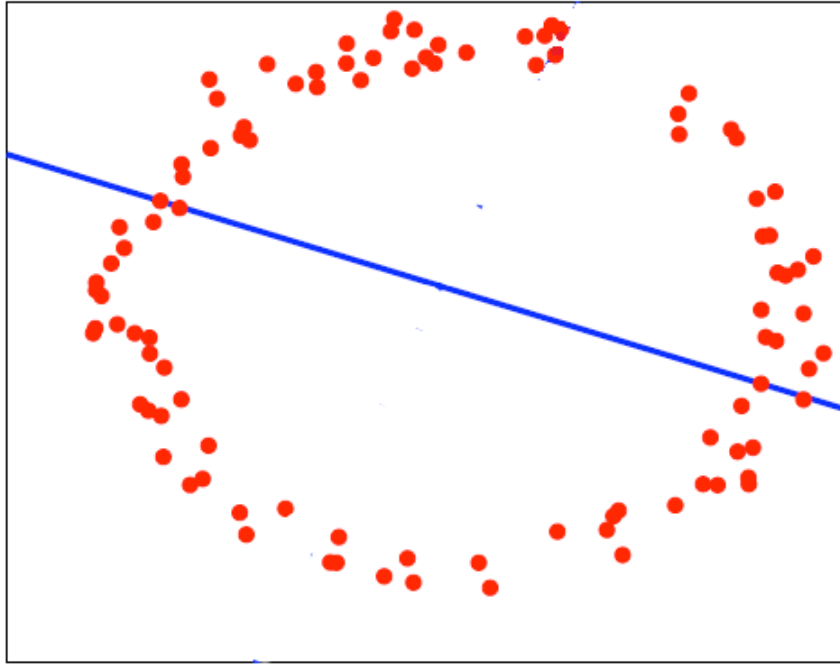


FIGURE 1.2 – Illustration de l’ACP en dimension 2 où les données sont non linéaires et l’ACP n’arrive pas à faire une réduction à une dimension optimale.

1.4.2 Cas de r dimensions

De la même manière qu’en dimension 1, l’erreur quadratique moyenne est donnée par :

$$\begin{aligned} \text{MSE} &= \frac{1}{n} \sum_{i=1}^n \|\mathbf{x}_i - \mathbf{x}'_i\|^2 \\ &= \text{Var}(\mathbf{X}) - \text{Var}(\mathbf{A}) \\ &= \sum_{i=1}^p \lambda_i - \sum_{i=1}^r \lambda_i. \end{aligned}$$

Comme dans le cas d’une dimension, le premier terme (la variance totale) étant constant, les r premières composantes qui maximisent la variance projetée sont bien celles qui minimisent l’erreur quadratique moyenne.

1.5 Limitations de l’ACP

Il est important de souligner que la réduction de dimension par la méthode de l’ACP marche bien à condition que la relation contenue dans les données mis en jeu soit linéaire, ce qui est d’ailleurs le cas des données de la figure 1.1. Lorsque cette dernière ne l’est pas, alors la situation est toute autre. Dans leur livre intitulé *Non Linear Dimensionality Reduction* [4], John A. Lee et Michel Verleysen ont discuté à la section 2.4.5, des limitations de l’ACP que nous conseillons au lecteur intéressé de jeter un coup d’œil.

Considérons en guise d’exemple la base de données en dimension 2 illustrée sur la figure 1.2. La première composante principale ne capture que la moitié (50%) de la variance

projetée. ce qui vraisemblablement ne permet pas de réduire de manière optimale la dimension de cette base de données. Intuitivement l'on sent qu'une **direction circulaire** qui traverserait le nuage des points en son sein capterait la quasi-totalité de cette variance, contrairement à une projection linéaire comme l'impose l'ACP.

Si l'on souhaite malgré tout utiliser l'ACP même dans ce genre de cas où la relation au sein de la base de données est non linéaire, alors il est impérative de faire au préalable une transformation non linéaire de cette dernière de sorte à ce qu'elle soit dans une nouvelle représentation adaptée à l'application de l'ACP. Ainsi l'on passe d'un espace de données non transformées appelé espace d'entrée à l'espace des données transformées appelés espace d'attributs.

Chapitre 2

Changement d'espace : de l'espace d'origine vers l'espace d'attributs

En analyse de données, trouver une représentation d'une base de données qui soit appropriée à un problème donné est aussi importante que trouver la méthode d'analyse en elle-même, qui soit convenable à l'analyse de ce problème. Quelques fois il est alors impérative d'effectuer une transformation de données d'origine à travers une correspondance non linéaire afin d'espérer atterrir dans la représentation appropriée.

2.1 Correspondance non linéaire

Dans la réalité, la représentation d'origine d'une base de données est le plus souvent non appropriée à un meilleur traitement de cette dernière. Ainsi la correspondance non linéaire permet de trouver pour tout vecteur $\mathbf{x}$ de la base de données $\mathbf{X}$ une fonction de correspondance non-linéaire ϕ de sorte que $\phi(\mathbf{x})$ soit la représentation adéquate de $\mathbf{x}$ dans la nouvelle représentation. Si la fonction de correspondance ϕ est judicieusement choisie, alors elle est en mesure de ressortir la relation non linéaire contenue dans la base de données ou mieux encore elle permettra de projeter la base de données dans un espace où **la nouvelle relation est simplement linéaire**. Ainsi au lieu de concevoir des algorithmes non linéaires complexes dans l'espace d'entrée/origine, l'on pourra simplement appliquer des « algorithmes linéaires » habituels à l'instar de l'algorithme de l'ACP, dans l'espace des attributs/caractéristiques à une base de données qui intrinsèquement est non linéaire.

Dans le domaine du SVM (Support Vector Machine) par exemple, comment ne pas mentionner l'assertion de Simon Haykin [5], conséquence du théorème de Cover [6] sur la séparabilité, qui dit que l'on peut avec une forte probabilité, transformer une base de données non linéairement séparable en une base de données linéairement séparable en le projetant dans un espace de dimension supérieure (espace de dimension suffisamment élevée) via une transformation non linéaire.

L'inconvénient avec la correspondance non linéaire est justement qu'elle requiert le plus souvent (si ce n'est toujours le cas) des fonctions de correspondance de **dimension très élevée** voire même **infinie**. Ceci a pour conséquence d'engendrer un espace d'attributs de dimension largement supérieure à l'espace d'origine. Ce qui paraît non seulement contre-intuitive du point de vue de la réduction de dimension (où l'objectif est plutôt de réduire la dimension actuelle), mais aussi très coûteux en terme de temps calcul, car on

l'aura bien compris il faudra faire correspondre explicitement chaque point $\mathbf{x}$ de la base de données par cette fonction. Ainsi il serait souhaitable de pouvoir faire une analyse dans la nouvelle représentation (où la base de données est adéquatement représentée) mais sans pour autant forcément passer explicitement par la fonction de correspondance (qui est réputée de dimension très élevée) : c'est là qu'intervient alors la méthode du noyau.

2.2 La méthode du noyau

Comme nous l'avons souligné à la section 2.1, l'espace d'attributs est généralement de très grande dimension et voire même de dimension infinie. Ainsi transformer explicitement tous les points de l'espace d'entrée vers l'espace d'attributs peut être très coûteux en terme de temps de calcul dans le cas où la dimension est très grande et juste impossible dans le cas où elle est infinie.

La méthode du noyau a été introduite récemment par Schölkopf, Smola et Müller [7] afin de permettre d'éviter une telle transformation explicite point par point lorsque l'on désire passer de l'espace d'entrée vers l'espace d'attributs. Elle utilise plutôt une représentation des points dans l'espace d'attributs par une matrice $n \times n$ de paires de similarités appelée matrice du noyau $\mathbf{K}$ qui s'écrit :

$$\mathbf{K} = \begin{pmatrix} K(\mathbf{x}_1, \mathbf{x}_1) & K(\mathbf{x}_1, \mathbf{x}_2) & \cdots & K(\mathbf{x}_1, \mathbf{x}_n) \\ K(\mathbf{x}_2, \mathbf{x}_1) & K(\mathbf{x}_2, \mathbf{x}_2) & \cdots & K(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ K(\mathbf{x}_n, \mathbf{x}_1) & K(\mathbf{x}_n, \mathbf{x}_2) & \cdots & K(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}.$$

Les éléments $K(\mathbf{x}_i, \mathbf{x}_j)$ de cette matrice appelés noyaux [8] où fonctions de similarité sont quant à eux donnés par :

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j).$$

Le noyau ainsi défini n'est rien d'autre qu'une généralisation du produit scalaire entre deux points/individus $\mathbf{x}_i$ et $\mathbf{x}_j$. Entant que telle, il permet de calculer dans n'importe quel espace engendré par la fonction ϕ , le cosinus de l'angle entre ces deux points (en supposant qu'ils soient normalisés à l'unité) et la distance qui les sépare (à partir de la différence de leur norme individuel). Par conséquent, il peut être considéré comme une fonction permettant de calculer **une mesure de similarité** entre les deux points. Contrairement au produit scalaire classique, la liberté qu'il offre de choisir la fonction de correspondance ϕ lui permet de concevoir une grande variété de mesure de similarité. Vu sous un autre angle, son choix détermine implicitement la fonction de transformation non linéaire ϕ et induit par conséquent l'espace d'attributs ainsi que sa dimension m . La bonne nouvelle c'est qu'une fois son expression définie, il peut être calculé sans nécessairement connaître ou bien construire explicitement la fonction de correspondance ϕ . Autrement dit, lorsque l'on connaît l'expression du noyau, alors il est possible d'effectuer toutes les opérations nécessaires dans l'espace d'attributs, **peu importe l'étendue de sa dimension**, à partir de la représentation d'origine $\mathbf{x}$ dans l'espace d'entrée.

2.2.1 Propriétés d'un noyau

Étant donné que la fonction du noyau K représente un produit scalaire, elle ne peut être choisie arbitrairement et pour être valide elle doit remplir les conditions suivantes :

- Elle doit être **symétrique** :

$$K(\mathbf{x}_i, \mathbf{x}_j) = K(\mathbf{x}_j, \mathbf{x}_i).$$

En effet, de part sa définition un produit scalaire étant symétrique, la fonction du noyau K donné par :

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) \\ &= \phi(\mathbf{x}_j)^\top \phi(\mathbf{x}_i) \\ &= K(\mathbf{x}_j, \mathbf{x}_i), \end{aligned}$$

l'est aussi.

- Sa matrice associée $\mathbf{K}$ doit être **semi-définie positive** :

$$\mathbf{a}^\top \mathbf{K} \mathbf{a} \geq 0, \quad \forall \text{ vecteur } \mathbf{a} \in \mathbb{R}^n.$$

En effet,

$$\begin{aligned} \mathbf{a}^\top \mathbf{K} \mathbf{a} &= \sum_{i=1}^n \sum_{j=1}^n a_i a_j K(\mathbf{x}_i, \mathbf{x}_j) \\ &= \sum_{i=1}^n \sum_{j=1}^n a_i a_j \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) \\ &= \left(\sum_{i=1}^n a_i \phi(\mathbf{x}_i) \right)^\top \left(\sum_{j=1}^n a_j \phi(\mathbf{x}_j) \right) \\ &= \left\| \sum_{i=1}^n a_i \phi(\mathbf{x}_i) \right\|^2 \geq 0, \quad \forall a_i \in \mathbb{R}, i \in [1:n], \end{aligned}$$

est bien semi-définie positive.

2.2.2 Astuce de la méthode du noyau : le “kernel trick”

L'astuce de la méthode du noyau communément appelée le “**kernel trick**”, est cette astuce qui permet de sauter par-dessus des calculs explicites liés à la correspondance des coordonnées lorsque l'on souhaite passer d'une représentation de dimension faible à une représentation de dimension plus élevée, et de trouver une fonction appelée noyau qui renvoie le même résultat, comme si nous avions, en fait, calculé l'expansion exacte des termes alors que nous n'en avons pas. Cette astuce nous permet alors d'opérer dans un espace de dimension élevée sans quitter l'espace d'origine de dimension plus petite, et par conséquent, de réduire la complexité informatique liée au temps de calcul engendré par la très grande dimension du nouvel espace. Outre cela, elle offre aussi une grande flexibilité au niveau des algorithmes dans la mesure où à partir d'un algorithme construit pour un noyau K donné l'on peut basculer à un algorithme alternatif pour un noyau K^* en

remplaçant simplement K par K^* . Cela revient implicitement à changer à la fois l'espace d'attributs et la fonction de correspondance ϕ équivalente.

Ainsi comme nous l'avons souligné précédemment, au lieu de faire correspondre chaque point de l'espace d'entrée à un point de l'espace d'attributs de grande dimension, les données sont plutôt représentées via la matrice du noyau $\mathbf{K}$ et toutes les analyses pertinentes peuvent y être effectuées à travers cette dernière. Mieux encore, une fois que la matrice du noyau de dimension $n \times n$ est calculée, nous n'avons même plus besoin des points dans l'espace d'entrée, car toutes les opérations impliquant uniquement des produits scalaires dans l'espace d'attributs peuvent être effectuées sur la matrice $\mathbf{K}$.

Lorsque le noyau K satisfait les conditions mentionnées à la sous section 2.2.1, Schölkopf, Smola et Müller détaillent dans l'annexe C de leur article sur l'analyse des composants non linéaires [9], qu'il existe toujours une fonction de correspondance ϕ dont K représente le produit scalaire dans un espace. Autrement dit qu'il existe un espace (de dimension plus élevée) où l'on peut tirer profit de l'astuce de la méthode du noyau.

En analyse des données plusieurs méthodes peuvent être “kernelisées” à l'instar du SVM, des moyens mobiles et de l'ACP. Dans le cas du premier l'on parle du noyau SVM [10, 11], dans le cas du deuxième, du noyau K-means [7] et dans le cas du dernier, du noyau ACP [9]. Le **noyau ACP** en abrégé **KACP**, qui fait l'objet de ce mémoire, sera discuté en détail dans la section 3.2.

2.3 Exemples des noyaux

Du fait que le “kernel trick” ne fonctionne qu'avec des noyaux qui remplissent des propriétés bien définies (voire sous section 2.2.1), seules quelques fonctions sont habituellement utilisées comme noyaux. Les plus connus sont :

- les noyaux polynomiaux,
- et les noyaux gaussiens.

2.3.1 Noyaux polynomiaux

De toutes les familles des noyaux, les noyaux polynomiaux [12, 13] sont sans doute les plus simples. La forme générale des noyaux polynomiaux de degré q s'écrit :

$$K_q(\mathbf{x}_i, \mathbf{x}_j) = \left(c + \gamma \mathbf{x}_i^T \mathbf{x}_j \right)^q,$$

où $c \geq 0$ et $\gamma \geq 0$ sont des constantes.

Lorsque $\gamma = 1$, on montre [3] qu'à ces noyaux, correspondent des fonctions de correspondance non linéaires ϕ de l'espace d'entrée de dimension p vers l'espace d'attributs de dimension m , données par :

$$\phi(\mathbf{x}) = \left(\dots, \sqrt{\binom{q}{\mathbf{n}} c^{n_0}} \prod_{k=1}^p x_k^{n_k}, \dots \right)^T$$

où $\binom{q}{\mathbf{n}}$ est le coefficient binomial et $\mathbf{n} = (n_0, n_1, \dots, n_p)$ prend toutes les affectations possibles de sorte que $|\mathbf{n}| = n_0 + n_1 + \dots + n_p = q$. La dimension m de l'espace d'attributs quant à elle, est donnée par $\binom{p+q}{q}$, qui peut prendre vite de très grandes valeurs lorsque

p et/ou q sont élevés. En guise d'exemple pour un noyau de degré $q = 3$ et une base de donnée de dimension $p = 1\,000$ dans l'espace d'origine l'on passerait à un espace d'attributs de dimension $m = 167\,668\,501$.

Parmi les noyaux polynomiaux, les plus répandus sont :

- le noyau linéaire
- et le noyau quadratique.

Noyau linéaire

Lorsque $q = 1$, le noyau correspondant est dit linéaire et s'écrit dans le cas $\gamma = 1$:

$$K(\mathbf{x}_i, \mathbf{x}_j) = c + \mathbf{x}_i^T \mathbf{x}_j.$$

Ce dernier n'est rien d'autre qu'un produit scalaire classique dans l'espace d'entrée.

Si $c = 0$ alors la fonction de correspondance se réduit simplement à l'identité :

$$\phi(\mathbf{x}) = \mathbf{x},$$

et la dimension m de l'espace d'attributs est dans ce cas égal à la dimension de l'espace d'entrée, c'est à dire $m = p$.

Noyau quadratique

Lorsque $q = 2$, le noyau correspondant est dit quadratique et s'écrit dans le cas $\gamma = 1$:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (c + \mathbf{x}_i^T \mathbf{x}_j)^2. \quad (2.1)$$

Dans le cas particulier où l'espace d'entrée est de dimension 2 ($p = 2$) et que la constante c est nulle ($c = 0$), l'équation (2.1) devient :

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\sqrt{2}x_{i1}x_{i2}, x_{i1}^2, x_{i2}^2) (\sqrt{2}x_{j1}x_{j2}, x_{j1}^2, x_{j2}^2)^T.$$

Ce qui donne lieu à une fonction de correspondance :

$$\phi(\mathbf{x}) = (\sqrt{2}x_1x_2, x_1^2, x_2^2)^T,$$

et donc à un espace d'attributs de dimension $m = 3$.

Pour la même dimension $p = 2$, lorsque l'on choisit plutôt la constante $c = 1$, l'équation (2.1) devient :

$$K(\mathbf{x}_i, \mathbf{x}_j) = (1, \sqrt{2}x_{i1}, \sqrt{2}x_{i2}, \sqrt{2}x_{i1}x_{i2}, x_{i1}^2, x_{i2}^2) (1, \sqrt{2}x_{j1}, \sqrt{2}x_{j2}, \sqrt{2}x_{j1}x_{j2}, x_{j1}^2, x_{j2}^2)^T.$$

Ce qui donne lieu à une fonction de correspondance :

$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)^T,$$

et donc à un espace d'attributs de dimension $m = 6$.

2.3.2 Noyau gaussien

Le noyau gaussien [12, 13] encore appelé noyau de base radiale est sans doute le noyau le plus utilisé en pratique. Il s'écrit comme suit :

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \left\{ -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2} \right\} = \exp \left\{ -\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2 \right\},$$

avec $\gamma = \frac{1}{2\sigma^2}$ et où $\sigma > 0$ est le paramètre de dispersion qui joue le même rôle que l'écart type dans une fonction de densité normale.

Il est important de noter que lorsque deux points $\mathbf{x}_i$ et $\mathbf{x}_j$ sont très proches alors $K(\mathbf{x}_i, \mathbf{x}_j) \simeq 1$, par contre lorsqu'ils sont très distants, $K(\mathbf{x}_i, \mathbf{x}_j) \simeq 0$. Ce noyau s'apparente donc à une sorte de **mesure de similarité** entre les points et prend une valeur quasi nulle lorsque deux points sont éloignés dans l'espace d'attributs et une valeur proche de 1 lorsque les deux points sont très proches dans cet espace.

On montre [3] qu'à ce noyau, correspond une fonction de correspondance non linéaire ϕ donnée par :

$$\phi(\mathbf{x}) = \left(\dots, \sqrt{\frac{(2\gamma)^q}{q!} \binom{q}{\mathbf{n}}} \exp \{ -\gamma \|\mathbf{x}\|^2 \} \prod_{k=1}^p x_k^{n_k}, \dots \right)^T, \quad (2.2)$$

avec des dimensions qui s'étendent sur tous les degrés $q \in \{0, 1, 2, \dots\}$ et avec $\mathbf{n} = (n_0, \dots, n_p)$ qui prend toutes les affectations possibles de sorte que $|\mathbf{n}| = q$ pour chaque valeur de q .

Il est intéressant de noter que la fonction ϕ ainsi définie fait alors correspondre l'espace d'entrée de dimension p à un espace d'attributs de **dimension infini** ($m = \infty$) à cause des degrés q qui s'étendent à l'infini dans son expression donnée par l'équation (2.2). Ainsi, du point de vue du théorème de Cover [6], ce noyau est particulièrement d'une grande pertinence dans la mesure où, associé à une fonction de correspondance non linéaire ϕ de dimension infinie, serait le cas idéal pour rendre une base de données séparable avec une probabilité proche de 1. Cela justifierait sans doute son utilisation fréquente en pratique.

2.3.3 Autres noyaux

Parmi les autres noyaux symétriques, nous retrouvons :

- le noyau exponentiel
- le noyau sigmoïde
- et le noyau B_n -splines

Noyau exponentiel. Nous distinguons le noyau exponentiel [12, 13] :

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(\gamma \mathbf{x}_i^T \mathbf{x}_j) - 1,$$

et le noyau exponentiel de base radiale [12, 13]

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp \left\{ -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|}{\kappa} \right\} = \exp \{ -\gamma \|\mathbf{x}_i - \mathbf{x}_j\| \},$$

avec $\gamma = 1/\kappa$ et où κ est une constante positive.

Noyau sigmoïde

Le noyau sigmoïde ou tangente hyperbolique [12, 13] quant à lui s'écrit :

$$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + \nu),$$

où γ est une constante positive et ν une constante négative. Ce noyau trouve plus son application dans le réseau de neurones qui utilise traditionnellement la fonction sigmoïde comme fonction d'activation. Il n'est pas toujours garanti que la matrice associée K de ce noyau soit semi-définie positive et par conséquent il ne correspond donc parfois pas à la fonction noyau d'une fonction de correspondance implicite.

Noyau B_n-splines

Et enfin le noyau B_n-splines [14] qui s'écrit :

$$K(\mathbf{x}_i, \mathbf{x}_j) = B_{2p+1}(\|\mathbf{x}_i - \mathbf{x}_j\|),$$

avec

$$\begin{aligned} B_0 &= I_{[-\frac{1}{2}, \frac{1}{2}]} \\ B_n &= B_n \otimes B_0 \\ &= \bigotimes_{i=1}^n I_{[-\frac{1}{2}, \frac{1}{2}]}, \end{aligned}$$

où I_X représente la fonction indicatrice et $\otimes$ représente l'opération de convolution définie par :

$$(f \otimes g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t - \tau)d\tau = \int_{-\infty}^{+\infty} f(t - \tau)g(\tau)d\tau.$$

2.3.4 Construction des noyaux

Il est possible de construire des noyaux à partir des noyaux valides existants.

Soit α_l des constantes positives et K_l des noyaux valides pour tout $l \in \{1, \dots, L\}$.

Addition par un scalaire

Tout noyau

$$K(\mathbf{x}_i, \mathbf{x}_j) = K_l(\mathbf{x}_i, \mathbf{x}_j) + \alpha_l,$$

résultant de l'addition d'un noyau valide avec une constante positive est aussi un noyau valide. En effet

$$\begin{aligned} K(\mathbf{x}_i, \mathbf{x}_j) &= K_l(\mathbf{x}_i, \mathbf{x}_j) + \alpha_l \\ &= \phi_1(\mathbf{x}_i)^T \phi_1(\mathbf{x}_j) + \alpha_l \\ &= (\phi_1(\mathbf{x}_i), \sqrt{\alpha_l})^T (\phi_1(\mathbf{x}_j), \sqrt{\alpha_l}) \\ &= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j), \end{aligned}$$

est bien le produit scalaire des deux points $\mathbf{x}_i$ et $\mathbf{x}_j$ dans l'espace d'attributs et par conséquent est un noyau valide, avec ϕ et ϕ_1 les fonctions de correspondance associées respectivement aux noyaux K et K_1 .

Multiplication par un scalaire

Tout noyau

$$K(\mathbf{x}_i, \mathbf{x}_j) = \alpha_l \times K_l(\mathbf{x}_i, \mathbf{x}_j),$$

résultant de la multiplication d'un noyau valide avec une constante positive est aussi un noyau valide.

En effet soit $\mathbf{K}$ et $\mathbf{K}_l$ les matrices de noyau associées respectivement aux noyaux K et K_l . Pour tout $\mathbf{a} \in \mathbb{R}^n$

$$\mathbf{a}^T \mathbf{K} \mathbf{a} = \alpha_l \mathbf{a}^T \mathbf{K}_l \mathbf{a} \geq 0,$$

est bien semi-définie positive et par conséquent le noyau K est un noyau valide.

Combinaison linéaire des noyaux

Tout noyau

$$K(\mathbf{x}_i, \mathbf{x}_j) = \sum_{l=1}^L \alpha_l K_l(\mathbf{x}_i, \mathbf{x}_j),$$

résultant de la combinaison linéaire des noyaux valides où tous les coefficients du développement sont des constantes positives est aussi un noyau valide.

En effet, pour tout $\mathbf{a} \in \mathbb{R}^n$

$$\mathbf{a}^T \mathbf{K} \mathbf{a} = \sum_{l=1}^L \alpha_l \mathbf{a}^T \mathbf{K}_l \mathbf{a} \geq 0,$$

est bien semi-définie positive et par conséquent le noyau K est un noyau valide.

Produit des deux noyaux

Tout noyau

$$K(\mathbf{x}_i, \mathbf{x}_j) = K_l(\mathbf{x}_i, \mathbf{x}_j) \times K_{l'}(\mathbf{x}_i, \mathbf{x}_j),$$

résultant de la multiplication de deux noyaux valides est aussi un noyau valide.

En effet, l'on peut réécrire les matrices symétriques et semi-définie positives $\mathbf{K}_l$ et $\mathbf{K}_{l'}$ en leur décomposition en valeurs et vecteurs propres :

$$\mathbf{K}_l = \sum_{i=1}^n \lambda_i \mathbf{u}_i \mathbf{u}_i^T \text{ et } \mathbf{K}_{l'} = \sum_{j=1}^n \mu_j \mathbf{v}_j \mathbf{v}_j^T,$$

où $\lambda_i \geq 0$ et $\mu_j \geq 0$. La matrice $\mathbf{K}$ peut s'écrire :

$$\mathbf{K} = \mathbf{K}_l \odot \mathbf{K}_{l'} = \sum_{k=1}^{n^2} \gamma_k \mathbf{w}_k \mathbf{w}_k^T,$$

avec $\gamma_k = \lambda_{\lfloor k/n \rfloor} \mu_{k \bmod n} \geq 0$, $\mathbf{w} = \mathbf{u}_{\lfloor k/n \rfloor} \odot \mathbf{v}_{k \bmod n}$ et où le symbol $\odot$ représente le produit matriciel de Hadamard (produit terme à terme des deux matrices). Et pour tout $\mathbf{a} \in \mathbb{R}^n$

$$\begin{aligned} \mathbf{a}^T \mathbf{K} \mathbf{a} &= \sum_{k=1}^{n^2} \gamma_k \mathbf{a}^T \mathbf{w}_k \mathbf{w}_k^T \mathbf{a} \\ &= \sum_{i=1}^{n^2} \gamma_k (\mathbf{w}_k^T \mathbf{a})^2 \geq 0, \end{aligned}$$

est bien semi-définie positive et par conséquent le noyau K est un noyau valide.

Fonction polynomiale d'un noyau

Tout noyau

$$K(\mathbf{x}_i, \mathbf{x}_j) = P(K_l(\mathbf{x}_i, \mathbf{x}_j)),$$

résultant du polynôme $P : \mathbb{R} \rightarrow \mathbb{R}$ d'un noyau valide avec des coefficients de développement positifs est aussi un noyau valide.

En effet, le polynôme P étant une combinaison linéaire des puissances du noyau K_l avec des coefficients positifs et que les puissances de K_l sont des produits de K_l par lui-même, c'est à dire des noyaux valides (voire section 2.3.4), alors leur combinaison linéaire est bien un noyau valide (voire section 2.3.4).

Fonction exponentielle d'un noyau

Tout noyau

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(K_l(\mathbf{x}_i, \mathbf{x}_j)),$$

résultant de la fonction exponentielle d'un noyau valide est aussi un noyau valide.

En effet, en développant en série de Taylor la fonction exponentielle :

$$\exp(K_l(\mathbf{x}_i, \mathbf{x}_j)) = \sum_{n=0}^{\infty} \frac{K_l^n(\mathbf{x}_i, \mathbf{x}_j)}{n!},$$

qui est la limite d'un polynôme P d'un noyau valide avec des coefficients positifs, est bien par conséquent lui aussi un noyau valide (voire sous-sous-section 2.3.4)

2.4 Quelques opérations dans l'espace d'attributs à partir du noyau

Une fois de plus nous renvoyons le lecteur dans le livre *Data Mining and Analysis* [3] pour plus des détails et/ou exemples numériques sur les différentes opérations dans l'espace d'attributs qui sont repris dans cette section. Le but étant ici de pouvoir montrer qu'à partir de la méthode du noyau, il est possible de travailler avec des données centrées et réduites dans l'espace des attributs comme l'exige le KACP.

2.4.1 Norme d'un point dans l'espace d'attributs

La norme au carré d'un point dans l'espace d'attributs s'écrit :

$$\|\phi(\mathbf{x})\|^2 = \phi(\mathbf{x})^T \phi(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}).$$

Ce qui permet de déduire la norme du point $\mathbf{x}$ dans l'espace d'attributs :

$$\|\phi(\mathbf{x})\| = \sqrt{K(\mathbf{x}, \mathbf{x})}.$$

2.4.2 Distance entre deux points dans l'espace d'attributs

La norme au carré de la distance entre deux points $\mathbf{x}_i$ et $\mathbf{x}_j$ dans l'espace d'attributs s'écrit :

$$\begin{aligned} \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2 &= \|\phi(\mathbf{x}_i)\|^2 + \|\phi(\mathbf{x}_j)\|^2 - 2\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \\ &= K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j). \end{aligned}$$

On en déduit donc la distance entre les deux points dans l'espace d'attributs :

$$\|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\| = \sqrt{K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j)}.$$

2.4.3 Moyenne et variance dans l'espace d'attributs

Dans l'espace d'attributs, l'expression de la moyenne μ_ϕ s'écrit :

$$\mu_\phi = \frac{1}{n} \sum_{i=1}^n \phi(\mathbf{x}_i).$$

Étant donné que les fonctions de correspondance ne sont pas toujours accessibles, cette norme dans l'espace d'attributs n'est pas directement calculable.

Par contre il est possible de calculer la norme au carré de cette moyenne :

$$\begin{aligned} \|\mu_\phi\|^2 &= \mu_\phi^T \mu_\phi \\ &= \left(\frac{1}{n} \sum_{i=1}^n \phi(\mathbf{x}_i) \right)^T \left(\frac{1}{n} \sum_{j=1}^n \phi(\mathbf{x}_j) \right) \\ &= \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \\ &= \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n K(\mathbf{x}_i, \mathbf{x}_j). \end{aligned}$$

Et donc la norme de la moyenne elle-même :

$$\|\mu_\phi\| = \frac{1}{n} \sqrt{\sum_{i=1}^n \sum_{j=1}^n K(\mathbf{x}_i, \mathbf{x}_j)}.$$

Quant à la variance totale dans l'espace d'attributs, elle est donnée par :

$$\begin{aligned}
\sigma_\phi^2 &= \frac{1}{n} \sum_{i=1}^n \|\phi(\mathbf{x}_i) - \mu_\phi\|^2 \\
&= \frac{1}{n} \sum_{i=1}^n \left(\|\phi(\mathbf{x}_i)\|^2 - 2\phi(\mathbf{x}_i)^\top \mu_\phi + \|\mu_\phi\|^2 \right) \\
&= \frac{1}{n} \sum_{i=1}^n \left[\|\phi(\mathbf{x}_i)\|^2 - 2\phi(\mathbf{x}_i)^\top \left(\frac{1}{n} \sum_{j=1}^n \phi(\mathbf{x}_j) \right) + \left(\frac{1}{n} \sum_{k=1}^n \phi(\mathbf{x}_k) \right)^\top \left(\frac{1}{n} \sum_{l=1}^n \phi(\mathbf{x}_l) \right) \right] \\
&= \frac{1}{n} \sum_{i=1}^n \left(\|\phi(\mathbf{x}_i)\|^2 - \frac{2}{n} \sum_{j=1}^n \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) + \frac{1}{n^2} \sum_{k=1}^n \sum_{l=1}^n \phi(\mathbf{x}_k)^\top \phi(\mathbf{x}_l) \right) \\
&= \frac{1}{n} \sum_{i=1}^n \left(K(\mathbf{x}_i, \mathbf{x}_i) - \frac{2}{n} \sum_{j=1}^n K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{n^2} \sum_{k=1}^n \sum_{l=1}^n K(\mathbf{x}_k, \mathbf{x}_l) \right) \\
&= \frac{1}{n} \sum_{i=1}^n K(\mathbf{x}_i, \mathbf{x}_i) - \frac{2}{n^2} \sum_{i=1}^n \sum_{j=1}^n K(\mathbf{x}_i, \mathbf{x}_j) + \frac{n}{n^3} \sum_{i=1}^n \sum_{j=1}^n K(\mathbf{x}_i, \mathbf{x}_j) \\
&= \frac{1}{n} \sum_{i=1}^n K(\mathbf{x}_i, \mathbf{x}_i) - \frac{1}{n^2} \sum_{i=1}^n \sum_{j=1}^n K(\mathbf{x}_i, \mathbf{x}_j).
\end{aligned}$$

2.4.4 Centrer dans l'espace d'attributs

Bien que l'on ne puisse pas calculer explicitement chaque point centré

$$\phi_c(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \mu_\phi,$$

dans l'espace d'attributs, l'on peut néanmoins calculer la matrice du noyau associée aux données centrées dans cet espace :

$$\mathbf{K}_c = \{K_c(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1}^n.$$

Les éléments $\{K_c(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1}^n$ de cette matrice sont donnés par :

$$\begin{aligned}
K_c(\mathbf{x}_i, \mathbf{x}_j) &= \phi_c(\mathbf{x}_i)^\top \phi_c(\mathbf{x}_j) \\
&= (\phi(\mathbf{x}_i) - \mu_\phi)^\top (\phi(\mathbf{x}_j) - \mu_\phi) \\
&= \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) - \phi(\mathbf{x}_i)^\top \mu_\phi - \phi(\mathbf{x}_j)^\top \mu_\phi + \mu_\phi^\top \mu_\phi \\
&= \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) - \frac{1}{n} \sum_{k=1}^n \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_k) - \frac{1}{n} \sum_{k=1}^n \phi(\mathbf{x}_j)^\top \phi(\mathbf{x}_k) + \frac{1}{n^2} \sum_{l=1}^n \sum_{p=1}^n \phi(\mathbf{x}_l)^\top \phi(\mathbf{x}_p) \\
&= K(\mathbf{x}_i, \mathbf{x}_j) - \frac{1}{n} \sum_{k=1}^n K(\mathbf{x}_i, \mathbf{x}_k) - \frac{1}{n} \sum_{k=1}^n K(\mathbf{x}_j, \mathbf{x}_k) + \frac{1}{n^2} \sum_{l=1}^n \sum_{p=1}^n K(\mathbf{x}_l, \mathbf{x}_p).
\end{aligned}$$

Ce qui permet d'écrire de manière compacte la matrice $\mathbf{K}_c$:

$$\begin{aligned}
\mathbf{K}_c(\mathbf{x}_i, \mathbf{x}_j) &= \mathbf{K} - \frac{1}{n} \mathbf{1}_{n \times n} \mathbf{K} - \frac{1}{n} \mathbf{K} \mathbf{1}_{n \times n} + \frac{1}{n^2} \mathbf{1}_{n \times n} \mathbf{K} \mathbf{1}_{n \times n} \\
&= (\mathbf{I} - \frac{1}{n} \mathbf{1}_{n \times n}) \mathbf{K} (\mathbf{I} - \frac{1}{n} \mathbf{1}_{n \times n}),
\end{aligned} \tag{2.3}$$

où $\mathbf{1}_{n \times n}$ est la matrice singulière dont les éléments sont tous des 1.

2.4.5 Normaliser dans l'espace d'attributs

En utilisant la fonction de correspondance normalisé $\phi_n(\mathbf{x}_i) = \frac{\phi(\mathbf{x}_i)}{\|\phi(\mathbf{x}_i)\|}$, il est possible de calculer le noyau des données normalisées dans l'espace d'attributs $K_n(\mathbf{x}_i, \mathbf{x}_j)$. Ce dernier est donné par :

$$\begin{aligned} K_n(\mathbf{x}_i, \mathbf{x}_j) &= \phi_n(\mathbf{x}_i)^T \phi_n(\mathbf{x}_j) \\ &= \frac{\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)}{\|\phi(\mathbf{x}_i)\| \|\phi(\mathbf{x}_j)\|} \\ &= \frac{K(\mathbf{x}_i, \mathbf{x}_j)}{\sqrt{K(\mathbf{x}_i, \mathbf{x}_i) K(\mathbf{x}_j, \mathbf{x}_j)}}. \end{aligned}$$

Soit sous forme matricielle compacte :

$$\mathbf{K}_n = \mathbf{W}^{-1/2} \mathbf{K} \mathbf{W}^{-1/2},$$

où $\mathbf{W}^{-1/2}$ est une matrice diagonale dont les éléments diagonaux sont donnés par $\mathbf{W}^{-1/2}(\mathbf{x}_i, \mathbf{x}_i) = \frac{1}{\sqrt{K(\mathbf{x}_i, \mathbf{x}_i)}}$ et les éléments non diagonaux par $\mathbf{W}^{-1/2}(\mathbf{x}_i, \mathbf{x}_j) = 0$ pour tout $i \neq j$.

2.4.6 Centrer et réduire dans l'espace d'attributs

Il est finalement possible de travailler sur des données centrées et réduites dans l'espace d'attributs directement à partir du noyau des données centrées et réduites sur cet espace. Ce noyau est donné par :

$$\begin{aligned} K_{cn}(\mathbf{x}_i, \mathbf{x}_j) &= \phi_{cn}(\mathbf{x}_i)^T \phi_{cn}(\mathbf{x}_j) \\ &= \frac{\phi_c(\mathbf{x}_i)^T \phi_c(\mathbf{x}_j)}{\|\phi_c(\mathbf{x}_i)\| \|\phi_c(\mathbf{x}_j)\|} \\ &= \frac{K_c(\mathbf{x}_i, \mathbf{x}_j)}{\sqrt{K_c(\mathbf{x}_i, \mathbf{x}_i) K_c(\mathbf{x}_j, \mathbf{x}_j)}}. \end{aligned}$$

Soit sous forme matricielle compacte :

$$\mathbf{K}_{cn} = \mathbf{W}_c^{-1/2} \mathbf{K}_c \mathbf{W}_c^{-1/2}, \quad (2.4)$$

où $\mathbf{W}_c^{-1/2}$ est comme précédemment une matrice diagonale dont les éléments diagonaux sont donnés par $\mathbf{W}_c^{-1/2}(\mathbf{x}_i, \mathbf{x}_i) = \frac{1}{\sqrt{K_c(\mathbf{x}_i, \mathbf{x}_i)}}$ et les éléments non diagonaux par $\mathbf{W}_c^{-1/2}(\mathbf{x}_i, \mathbf{x}_j) = 0$ pour tout $i \neq j$.

Comme dans le cadre de l'ACP où il est impératif de centrer et réduire les points dans l'espace d'input, le noyau ACP utilisera aussi la matrice du noyau centré et réduit $\mathbf{K}_{cn}$ afin de permettre de travailler avec l'équivalent **des données centrées et réduites** dans l'espace d'attributs.

Chapitre 3

Analyse en Composantes Principales par la méthode du noyau (KACP)

La réduction de dimension linéaire vue au chapitre 1 n'a de sens que lorsque la relation existante au sein de la base des données est linéaire ou du moins lorsqu'elle l'est approximativement. Lorsque cette relation n'est pas linéaire, alors l'on fait plutôt recours à la réduction de dimension non linéaire. Les méthodes de réduction de dimensions non linéaires ont été largement étudiées dans [4]. Ces méthodes ont été classées en plusieurs catégories :

- les méthodes spectrales,
- les méthodes de réseau de neurones,
- les méthodes de reconstruction d'erreur,
- les méthodes de préservation de distance,
- les méthodes de préservation de similarité.

Le KACP, centre d'intérêt de ce travail fait partie des méthodes spectrales.

3.1 Méthodes spectrales

Comme leur nom l'indique, les méthodes spectrales sont des méthodes qui font appel à la décomposition spectrale des matrices. Cette dernière consiste en la factorisation sous une forme canonique d'une matrice diagonalisable $\mathbf{M}$. Cette matrice sera représentée en termes de ses valeurs propres et vecteurs propres de la manière suivante :

$$\mathbf{M} = \mathbf{U} \mathbf{D} \mathbf{U}^T,$$

où $\mathbf{D}$ est la matrice diagonale des valeurs propres et $\mathbf{U}$ une matrice dont les colonnes sont les vecteurs propres correspondants.

Les méthodes spectrales sont subdivisées en deux groupes :

- Matrices denses sémi-définies positives de dissimilarités
- Matrices creuses sémi-définies positives de similarités

Matrices denses sémi-définies positives de dissimilarités

En ce qui concerne le premier groupe, une transformation préalable via un opérateur ϕ est d'abord appliquée sur les données initiales de grande dimensions afin d'obtenir une matrice dense de dissimilarités. Une décomposition de ladite matrice est alors effectuée à partir de la méthode spectrale et les valeurs propres dominants ainsi que les vecteurs propres correspondants sont considérés dans le calcul de la réduction de dimensions. Dans ce groupe l'on retrouve les méthodes suivantes :

- l'ACP,
- le *Classical Multidimensional Scaling* (CMDS),
- l'Isomap,
- et le **KACP**.

Les deux méthodes classiques les plus connues à savoir l'ACP et le CMDS [15] sont basées sur la notion de distance euclidienne par paire de points. Elles correspondent toutes deux à une transformation identité ($\phi = \text{identité}$), autrement dit une transformation à la sortie de laquelle les données initiales en grande dimension sont laissées telles quelles. Ce qui ne permet donc naturellement pas à ces deux méthodes de réaliser une réduction de dimension de manière non linéaire. La seule différence entre ces deux méthodes est que l'ACP utilise la matrice de variance-covariance $\Sigma = \mathbf{X}^T \mathbf{X}$ alors que le CMDS utilise la matrice de Gram $\mathbf{G} = \mathbf{X} \mathbf{X}^T$. Ces deux méthodes constituent donc le même problème, le même modèle et conduisent à la même solution en terme de valeurs propres. En ce qui concerne les vecteurs propres on peut exprimer ceux donnés par l'une de méthode en fonction des vecteurs propres de l'autre en utilisant la **dualité**.

Quant à la méthode Isomap [16], elle utilise à la place de la distance euclidienne, une distance graphique, plus précisément une distance géodésique comme transformation. Ce qui lui permet alors de dérouler les données non linéaires en forme de rouleaux avant de faire une réduction de dimensions. Elle est donc une réduction de dimension non linéaire au sens propre du terme. La seule différence entre cette méthode et la méthode CMDS est qu'elle utilise la distance graphique comme métrique dans l'espace des données alors que la méthode CMDS utilise la distance euclidienne. Excepté cela la procédure spectrale utilisée sur la matrice de Gram dans la réduction de dimension reste la même pour les deux méthodes.

Enfin pour la méthode KACP, l'opérateur de transformation ϕ est comme nous l'avons vu n'importe quelle fonction de correspondance qui donnerait lieu à un noyau remplissant les propriétés définies à la section 2.2.1. Ce noyau représente un produit scalaire comme dans le cas du CMDS excepté que ce produit scalaire est dans l'espace d'attributs. Une fois la matrice de Gram formée dans cet espace ; la même procédure spectrale est appliquée comme dans le cas du CMDS. Ainsi le KACP n'est rien d'autre que du CMDS réalisé dans l'espace d'attributs. Contrairement à ce que son nom laisse paraître, cette méthode est plus étroitement liée au CMDS qu'à l'ACP [17] elle même. Le KACP fera l'objet d'une attention particulière à la section 3.2.

Matrices denses sémi-définies positives de similarités

Le deuxième groupe applique aussi une transformation préalable des données initiales de grande dimension via un opérateur ϕ afin d'obtenir une matrice mais plutôt creuse de

similarités. Une décomposition de ladite matrice est alors effectuée à partir de la méthode spectrale et les valeurs propres non dominantes (c’est à dire de fin) ainsi que les vecteurs propres correspondants sont considérés dans le calcul de la réduction de dimensions. Ce groupe est constitué des méthodes suivantes : le Locally Linear Embedding [18] et le Laplacien Eigenmap [19]. Le premier approxime Chaque donnée par une combinaison linéaire (régularisée) de ses K voisins les plus proches, puis essaye de reproduire cette combinaison linéaire dans un espace de dimension plus petite. Quant au second, il s’appuie sur des concepts de la théorie des graphes comme l’opérateur laplacien sur un graphe et est basé sur la minimisation des distances locales, c’est-à-dire les distances entre des points voisins. Ces deux méthodes ont été mentionnées dans ce travail juste à titre indicatif comme faisant partie des méthodes spectrales au même titre que le KACP.

3.2 Description du KACP

L’ACP vue au chapitre 1 est principalement utilisée pour réduire la dimension d’une base de données à structure linéaire de grande dimension en les projetant sur un sous-espace de dimension inférieure tout en préservant le maximum de variance. Le KACP introduit tout récemment (en 1998) par Schölkopf, Smola et Müller [9], étend cette technique aux données à structure non linéaire. Il y parvient en appliquant une correspondance non linéaire, connue sous le nom de fonction de noyau permettant alors de transformer les données d’origine avant d’appliquer (sur ces données transformées) une ACP linéaire. A cause de cette transformation non linéaire (appliquée préalablement aux données d’origine), les composantes principales linéaires obtenues dans l’espace d’attributs correspondent aux composantes non linéaires dans l’espace d’entrée. Ainsi nous pouvons dire que le KACP ou l’ACP non linéaire est une méthode qui permet en utilisant la méthode du noyau, de trouver des directions non linéaires au sein d’une base de données.

Bien que plus proche du CMDS, le KACP est historiquement considéré comme la version non linéaire de l’ACP. Comme toute méthode spectrale de réduction de dimension, il est basé sur une décomposition spectrale mais plutôt dans l’espace d’attributs où les données d’origine initialement non linéaires sont d’abord transformées de manière à être (quasi) linéaires. Pour que cette linéarisation des données marche, la fonction de transformation appliquée doit être de dimension plus élevée. Cette fonction de transformation/correspondance n’est pas toujours connue, tant bien que même si elle l’est, elle introduit des calculs fastidieux liés à sa très grande dimension. Un aspect remarquable de la méthode KACP est que les vecteurs d’attributs ne sont justement pas nécessaires pour calculer les scores des composantes principales. Grâce au “kernel trick” introduit au chapitre 2, une transformation explicite des données d’origine point par point est évitée. Elle est plutôt remplacée de manière équivalente par une matrice du noyau qui calcule le produit scalaire (des données transformées) dans l’espace d’attributs à partir d’une fonction noyau (choisie selon la nature du problème à résoudre) et les données de l’espace d’entrée.

3.3 L'ACP dans l'espace d'attributs

3.3.1 Utilisation explicite de la fonction de correspondance ϕ

Soit ϕ la fonction de correspondance qui permet de transformer chaque donnée $\mathbf{x}$ de dimension p dans l'espace d'entrée à sa donnée correspondante $\phi(\mathbf{x})$ de dimension m dans l'espace d'attributs. Si l'on considère des données centrées $\phi_c(\mathbf{x})$ dans l'espace d'attributs, la matrice de variance de ces données s'écrit :

$$\Sigma_{\phi_c} = \frac{1}{n} \sum_{i=1}^n \phi_c(\mathbf{x}_i) \phi_c(\mathbf{x}_i)^T. \quad (3.1)$$

Afin d'effectuer l'ACP dans l'espace d'attributs de sorte à mimer le traitement standard de l'ACP tel qu'effectué dans l'espace d'entrée, nous devons trouver les valeurs propres $\lambda_\alpha \geq 0$ et les vecteurs propres non nuls $\mathbf{u}_\alpha$ correspondants de la matrice de variance-covariance Σ_{ϕ_c} :

$$\Sigma_{\phi_c} \mathbf{u}_\alpha = \lambda_\alpha \mathbf{u}_\alpha. \quad (3.2)$$

Il est important de noter à ce stade que contrairement au $\alpha = 1, \dots, p$ qui est **fini** lorsque l'on réalise l'ACP dans l'espace d'entrées, le $\alpha = 1, \dots, m$ peut être **infini** dans l'espace d'attributs lorsque l'on a à faire à des fonctions de correspondance de dimension infinie. Cependant la **dualité** nous enseigne que le nombre de valeurs propres **non nulles** est de $\tilde{m} = \text{rank}(\phi) \leq \min(n, m)$ qui lui est bien **fini** (avec ϕ la matrice $n \times m$ des données dans l'espace d'attributs). Étant donné que l'objectif en ACP est de retenir les composantes principales $\mathbf{u}_\alpha$ dominantes (qui contiennent le maximum d'information/variance) correspondantes aux valeurs propres $\lambda_\alpha > 0$ les plus élevées, le fait que m puisse prendre une valeur infinie n'est donc pas un problème en soit.

En remplaçant Σ_{ϕ_c} par son expression donnée par l'équation (3.1), l'équation (3.2) aux valeurs propres devient :

$$\begin{aligned} \left(\frac{1}{n} \sum_{i=1}^n \phi_c(\mathbf{x}_i) \phi_c(\mathbf{x}_i)^T \right) \mathbf{u}_\alpha &= \lambda_\alpha \mathbf{u}_\alpha \iff \\ \frac{1}{n} \sum_{i=1}^n \phi_c(\mathbf{x}_i) (\phi_c(\mathbf{x}_i)^T \mathbf{u}_\alpha) &= \lambda_\alpha \mathbf{u}_\alpha \iff \\ \sum_{i=1}^n \frac{(\phi_c(\mathbf{x}_i)^T \mathbf{u}_\alpha)}{n \lambda_\alpha} \phi_c(\mathbf{x}_i) &= \mathbf{u}_\alpha, \quad \forall \lambda_\alpha \neq 0. \end{aligned}$$

Ce qui conduit à :

$$\mathbf{u}_\alpha = \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i), \quad (3.3)$$

avec $v_{\alpha i} = (n \lambda_\alpha)^{-1} \phi_c(\mathbf{x}_i)^T \mathbf{u}_\alpha$. Il existe donc des coefficients scalaire $v_{\alpha i}$, $i = 1, \dots, n$ tels que chaque vecteur propre $\mathbf{u}_\alpha$ associé à la valeur propre λ_α puisse être écrite comme une combinaison linéaire des points transformés. Les scalaires v_i montrent l'importance de chaque point dans la direction $\mathbf{u}_\alpha$ associée à la variance λ_α .

Étant donné que la fonction de correspondance ϕ n'est pas toujours connu, du moins même si elle l'est, la transformation explicite des points vers l'espace d'attributs n'est pas

toujours évidente, il n'est donc pas possible de calculer directement la direction principale $\mathbf{u}_\alpha$ telle qu'exprimée en fonction des $\phi_c(\mathbf{x}_i)$ par l'équation (3.3).

En réintroduisant le vecteur propre donné par l'équation (3.3) dans l'équation aux valeurs propres (3.2), l'on obtient :

$$\begin{aligned} \left(\frac{1}{n} \sum_{i=1}^n \phi_c(\mathbf{x}_i) \phi_c(\mathbf{x}_i)^T \right) \left(\sum_{j=1}^n v_{\alpha j} \phi_c(\mathbf{x}_j) \right) &= \lambda_\alpha \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i) \\ \frac{1}{n} \sum_{i=1}^n \left(\phi_c(\mathbf{x}_i) \sum_{j=1}^n \phi_c(\mathbf{x}_i)^T v_{\alpha j} \phi_c(\mathbf{x}_j) \right) &= \lambda_\alpha \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i) \\ \sum_{i=1}^n \left(\phi_c(\mathbf{x}_i) \sum_{j=1}^n v_{\alpha j} \phi_c(\mathbf{x}_i)^T \phi_c(\mathbf{x}_j) \right) &= n \lambda_\alpha \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i). \end{aligned} \quad (3.4)$$

Nous pouvons réécrire l'équation (3.4) uniquement en fonction du noyau afin de profiter de l'avantage offert par le “kernel trick”.

3.3.2 Utilisation du “kernel trick”

Rappelons que le “kernel trick” permet de réaliser toutes les opérations en termes de fonction du noyau dans l'espace d'entrée, sans avoir à transformer explicitement les données dans l'espace d'attributs. Afin de s'assurer que tous les calculs soient faits à partir du produit scalaire dans l'espace d'attributs et donc du noyau, nous allons multiplier l'équation (3.4) par $\phi_c(\mathbf{x}_k)^T$ de deux côtés de l'égalité. Ceci conduit à :

$$\begin{aligned} \sum_{i=1}^n \left(\phi_c(\mathbf{x}_k)^T \phi_c(\mathbf{x}_i) \sum_{j=1}^n v_{\alpha j} \phi_c(\mathbf{x}_i)^T \phi_c(\mathbf{x}_j) \right) &= n \lambda_\alpha \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_k)^T \phi_c(\mathbf{x}_i) \\ \sum_{i=1}^n \left(K_c(\mathbf{x}_k, \mathbf{x}_i) \sum_{j=1}^n v_{\alpha j} K_c(\mathbf{x}_i, \mathbf{x}_j) \right) &= n \lambda_\alpha \sum_{i=1}^n v_{\alpha i} K_c(\mathbf{x}_k, \mathbf{x}_i) \\ \sum_{i=1}^n K_c(\mathbf{x}_k, \mathbf{x}_i) \mathbf{K}_{ci}^T \mathbf{v}_\alpha &= n \lambda_\alpha \mathbf{K}_{ck}^T \mathbf{v}_\alpha, \end{aligned} \quad (3.5)$$

avec $\mathbf{K}_{ci} = (K_c(\mathbf{x}_i, \mathbf{x}_1), \dots, K_c(\mathbf{x}_i, \mathbf{x}_n))^T$, le vecteur ligne de la matrice du noyau centré $\mathbf{K}_c$ et $\mathbf{v}_\alpha = (v_{\alpha 1}, \dots, v_{\alpha n})$ le vecteur colonne des coefficients.

Étant donné que k prend toutes les valeurs possibles de 1 à n , l'équation (3.5) est un système de n équations qui peut s'écrire de manière matricielle sous forme compact :

$$\mathbf{K}_c^2 \mathbf{v}_\alpha = n \lambda_\alpha \mathbf{K}_c \mathbf{v}_\alpha. \quad (3.6)$$

En multipliant l'équation (3.6) par $\mathbf{K}_c^{-1}$ de deux cotés de l'égalité, on obtient :

$$\begin{aligned} \mathbf{K}_c \mathbf{v}_\alpha &= n \lambda_\alpha \mathbf{v}_\alpha \\ \mathbf{K}_c \mathbf{v}_\alpha &= \nu_\alpha \mathbf{v}_\alpha \end{aligned} \quad (3.7)$$

où $\nu_\alpha = n \lambda_\alpha$ et $\alpha = 1, \dots, n$ cette fois ci. Nous retombons ainsi sur un problème aux valeurs propres classique où ν_α est la valeur propre et $\mathbf{v}_\alpha$ le vecteur propre correspondant.

Remarquons au passage que ce problème est bien semblable à celui du CMDS où la matrice de Gram $\mathbf{G} = \mathbf{X}\mathbf{X}^T$ dans l'espace d'entrée est remplacée par son équivalent $\mathbf{K}_c = \mathbf{\Phi}_c\mathbf{\Phi}_c^T$ dans l'espace d'attributs, où $\mathbf{\Phi}_c$ est la matrice de données dans l'espace d'attributs.

Une fois le vecteur propre $\mathbf{v}_\alpha$ trouvé, nous pouvons le réintroduire dans l'expression de $\mathbf{u}_\alpha$ donnée par l'équation (3.3) afin d'obtenir la α ème composante principale dans l'espace d'attributs.

3.3.3 Normalisation des axes principaux

Étant donné que les vecteurs propres $\mathbf{u}_\alpha$ sont orthogonaux, on a $\mathbf{u}_\alpha^T \mathbf{u}_\beta = 1$ si $\alpha = \beta$ et 0 sinon. Cela conduit en remplaçant $\mathbf{u}_\alpha$ par son expression donnée par l'équation (3.3) :

$$\sum_{i=1}^n \sum_{j=1}^n v_{\alpha i} v_{\alpha j} \phi_c(\mathbf{x}_i)^T \phi_c(\mathbf{x}_j) = 1,$$

soit sous forme compact

$$\mathbf{v}_\alpha^T \mathbf{K}_c \mathbf{v}_\alpha = 1.$$

Et en utilisant l'équation aux valeurs propres (3.7), nous pouvons écrire :

$$\mathbf{v}_\alpha^T \mathbf{K}_c \mathbf{v}_\alpha = \mathbf{v}_\alpha^T \nu_\alpha \mathbf{v}_\alpha = \nu_\alpha \mathbf{v}_\alpha^T \mathbf{v}_\alpha = 1,$$

soit

$$\|\mathbf{v}_\alpha\|^2 = 1/\nu_\alpha.$$

Le vecteur $\mathbf{v}_\alpha$ lui-même étant un vecteur propre et donc orthogonal de norme égale à l'unité, pour garantir que $\mathbf{u}$ soit un vecteur unitaire, nous devons mettre à l'échelle le vecteur $\mathbf{v}$ de sorte que sa norme soit $\|\mathbf{v}_\alpha\| = \sqrt{1/\nu_\alpha}$. Pour se faire, il faudra multiplier le vecteur $\mathbf{v}_\alpha$ par $\sqrt{1/\nu_\alpha}$.

3.4 Projection des individus sur les composantes principales de l'espace d'attributs

Comme nous le savons bien, ce qui est important ce n'est pas les composantes principales elles mêmes mais plutôt la projection des individus/points dans l'espace engendré par ces composantes. A cet égard, un point existant $\mathbf{x}_j$ de l'espace d'entrée peut être projeté sur la composante principale $\mathbf{u}_\alpha$ de l'espace d'attributs comme suit :

$$y_{\alpha j} = \mathbf{u}_\alpha^T \phi(\mathbf{x}_j) = \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i)^T \phi_c(\mathbf{x}_j) = \sum_{i=1}^n v_{\alpha i} K_c(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{K}_{cj}^T \mathbf{v}_\alpha,$$

avec $\mathbf{K}_{cj}$ le j ème vecteur colonne de la matrice noyau $\mathbf{K}_c$. Ceci ne dépend que du kernel K_c . In fine nous voyons qu'il n'est même pas nécessaire de déterminer les directions principales $\mathbf{u}_\alpha$, seuls le kernel K_c et ses vecteurs/valeurs propres comptent.

De manière générale, nous pouvons déduire les nouvelles variables $\mathbf{y}_\alpha$, constituées des n coordonnées des observations par rapport à un axe principal α :

$$\mathbf{y}_\alpha = \mathbf{K}_c^T \mathbf{v}_\alpha : n \times 1, \quad \alpha = 1, \dots, \min(n, m).$$

3.4.1 Approche Alternative : utilisation de la dualité

Selon la dualité les axes principaux sont parallèles aux vecteurs propres $\mathbf{v}_\alpha$ de la matrice de Gram qui est la matrice du noyau $\mathbf{K}_c$ dans notre cas spécifique. Nous pouvons écrire :

$$\frac{\mathbf{y}_\alpha}{\sqrt{n}} = \sqrt{\lambda_\alpha} \mathbf{v}_\alpha$$

soit

$$\begin{aligned} \mathbf{y}_\alpha &= \sqrt{n\lambda_\alpha} \mathbf{v}_\alpha \\ &= \sqrt{\nu_\alpha} \mathbf{v}_\alpha \\ &= \sqrt{1/\nu_\alpha} \mathbf{v}_\alpha \\ &= \sqrt{1/\nu_\alpha} \mathbf{K}_c^\top \mathbf{v}_\alpha \\ &= \mathbf{K}_c^\top \left[\sqrt{1/\nu_\alpha} \mathbf{v}_\alpha \right] \end{aligned}$$

On retrouve ainsi, le résultat trouvé à la section 3.4, où le vecteur $\mathbf{v}_\alpha$ a été directement mise à l'échelle comme discuté à la section 3.3.3

3.5 Coordonnées factorielles des variables et corrélation

Rappelons que l'analyse de la dualité entre les vecteurs propres des matrices Σ_{ϕ_c} et $\mathbf{K}_c$ correspondant à des valeurs propres non nulles, nous enseigne que les vecteurs coordonnées factorielles de colonnes/variables $\boldsymbol{\varphi}_\alpha$ sont parallèles aux vecteurs propres de lignes $\mathbf{u}_\alpha$ (c'est à dire les vecteurs propres de la matrice Σ_{ϕ_c}). Ils sont reliés par :

$$\boldsymbol{\varphi}_\alpha = \sqrt{\lambda_\alpha} \mathbf{u}_\alpha.$$

En remplaçant $\mathbf{u}_\alpha$ par son expression donnée par l'équation (3.3), nous obtenons :

$$\boldsymbol{\varphi}_\alpha = \sqrt{\lambda_\alpha} \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i).$$

Nous voyons malheureusement que le calcul des coordonnées factorielles des variable $\boldsymbol{\varphi}_\alpha$ nécessitent la connaissance des $\phi_c(\mathbf{x}_i)$, c'est à dire de la transformation explicite des variables d'origine. Cette transformation n'étant pas toujours connue ou souhaitée, il en est alors de même pour les coordonnées factorielles des variables.

Étant donné que les $\boldsymbol{\varphi}_\alpha$ ne sont pas toujours connus à cause des variables transformées $\phi_c(\mathbf{x}_i)$ qui ne le sont pas toujours non plus et dont ils dépendent, la corrélation entre une variable transformée j et une composante principale α donnée par $\varphi_{\alpha j} = \text{cor}(\phi_c^j, \mathbf{y}_\alpha)$ n'est pas toujours connue non plus. Par conséquent la représentation du cercle de corrélation comme on a coutume de le faire dans le cadre de l'ACP linéaire n'est pas forcément envisageable.

3.6 Reconstruction d'erreur dans l'espace d'attributs

La projection d'un point $\phi(\mathbf{x}_j)$ sur l'espace engendré par les r premières composantes principales est donnée par :

$$\begin{aligned}\Pi\phi_c(\mathbf{x}_j) &= \sum_{\alpha=1}^r \mathbf{y}_{\alpha} \mathbf{u}_{\alpha} \\ &= \sum_{\alpha=1}^r \mathbf{y}_{\alpha j} \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i)\end{aligned}\tag{3.8}$$

3.6.1 Erreur de projection

L'erreur de projection d'un point $\phi(\mathbf{x}_j)$ est donnée par :

$$\begin{aligned}\|\phi_c(\mathbf{x}_j) - \Pi\phi_c(\mathbf{x}_j)\|^2 &= \|\phi_c(\mathbf{x}_j)\|^2 - \phi_c(\mathbf{x}_j)^T \Pi\phi_c(\mathbf{x}_j) - (\Pi\phi_c(\mathbf{x}_j))^T \phi_c(\mathbf{x}_j) \\ &\quad + \|\Pi\phi_c(\mathbf{x}_j)\|^2.\end{aligned}$$

En remplaçant $\Pi\phi_c(\mathbf{x}_j)$ par son expression donnée par l'équation (3.8), cette erreur devient :

$$\begin{aligned}\|\phi_c(\mathbf{x}_j) - \Pi\phi_c(\mathbf{x}_j)\|^2 &= \|\phi_c(\mathbf{x}_j)\|^2 - \phi_c(\mathbf{x}_j)^T \sum_{\alpha=1}^r \mathbf{y}_{\alpha j} \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i) \\ &\quad - \left(\sum_{\alpha=1}^r \mathbf{y}_{\alpha j} \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i) \right)^T \phi_c(\mathbf{x}_j) \\ &\quad + \left(\sum_{\alpha=1}^r \mathbf{y}_{\alpha j} \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i) \right)^T \left(\sum_{\alpha=1}^r \mathbf{y}_{\alpha j} \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i) \right) \\ &= \|\phi_c(\mathbf{x}_j)\|^2 - \sum_{\alpha=1}^r \mathbf{y}_{\alpha j} \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_j)^T \phi_c(\mathbf{x}_i) \\ &\quad - \sum_{\alpha=1}^r \mathbf{y}_{\alpha j} \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i)^T \phi_c(\mathbf{x}_j) \\ &\quad + \sum_{\alpha=1}^r \sum_{\alpha'=1}^r \mathbf{y}_{\alpha j} \mathbf{y}_{\alpha' j} \sum_{i=1}^n \sum_{j=1}^n v_{\alpha i}^T v_{\alpha' j} \phi_c(\mathbf{x}_i)^T (\phi_c(\mathbf{x}_j)),\end{aligned}$$

soit

$$\|\phi_c(\mathbf{x}_j) - \Pi\phi_c(\mathbf{x}_j)\|^2 = K_c(\mathbf{x}_j, \mathbf{x}_j) - 2 \sum_{\alpha=1}^r \mathbf{y}_{\alpha j} \mathbf{K}_{c j}^T \mathbf{v}_{\alpha} + \sum_{\alpha=1}^r \sum_{\alpha'=1}^r \mathbf{y}_{\alpha j} \mathbf{y}_{\alpha' j} \mathbf{v}_{\alpha}^T \mathbf{K}_c \mathbf{v}_{\alpha'}.$$

Une fois de plus cette erreur est exprimée uniquement à partir de la matrice du noyau et de ses vecteurs propres. Il est à noter que cette erreur est quasi nulle si l'on retient suffisamment des composantes principales.

3.6.2 Erreur de distance entre les points

Une autre façon de mesurer la performance du KACP est d'examiner dans quelle mesure il préserve les distances entre les points de données. A cet égard l'on peut calculer

le delta entre la distance euclidienne d_{ij} entre les points $\phi_c(\mathbf{x}_i)$ et $\phi_c(\mathbf{x}_j)$ et $\tilde{d}_{ij}$ entre leurs projections $\Pi\phi_c(\mathbf{x}_i)$ et $\Pi\phi_c(\mathbf{x}_j)$:

$$\begin{aligned} d_{ij} - \tilde{d}_{ij} &= \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\| - \|\Pi\phi(\mathbf{x}_i) - \Pi\phi(\mathbf{x}_j)\| \\ &= \sqrt{K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_j, \mathbf{x}_j) - 2K(\mathbf{x}_i, \mathbf{x}_j)} \\ &\quad - \sqrt{\sum_{\alpha=1}^r \sum_{\alpha'=1}^r \mathbf{y}_{\alpha i} \mathbf{y}_{\alpha' i} \mathbf{v}_{\alpha}^T \mathbf{K}_c \mathbf{v}_{\alpha} + \sum_{\alpha=1}^r \sum_{\alpha'=1}^r \mathbf{y}_{\alpha j} \mathbf{y}_{\alpha' j} \mathbf{v}_{\alpha}^T \mathbf{K}_c \mathbf{v}_{\alpha} - 2 \sum_{\alpha=1}^r \sum_{\alpha'=1}^r \mathbf{y}_{\alpha i} \mathbf{y}_{\alpha' j} \mathbf{v}_{\alpha}^T \mathbf{K}_c \mathbf{v}_{\alpha}}. \end{aligned}$$

Ce delta ne déroge pas à la règle car il est aussi exprimé uniquement à partir de la matrice du noyau et de ses vecteurs propres. La moyenne des deltas sur toutes les paires de points constitue la mesure de l'erreur de reconstruction.

3.7 Observation supplémentaire

En KACP il est aussi possible de projeter tout autre point n'appartenant pas à la base de données initiales. Un nouveau point $\phi(\mathbf{x})$ peut être projeté sur la composante principale $\mathbf{u}_{\alpha}$ comme suit :

$$\mathbf{u}_{\alpha}^T \phi(\mathbf{x}) = \sum_{i=1}^n v_{\alpha i} \phi_c(\mathbf{x}_i)^T \phi_c(\mathbf{x}) = \sum_{i=1}^n v_{\alpha i} K_c(\mathbf{x}_i, \mathbf{x}).$$

La base de données d'origine est considérée comme active et permet de calculer les différents axes principaux. Une fois ces axes déterminés, l'on peut projeter des données supplémentaires sur ces axes sans les recalculer.

3.8 Centralisation et standardisation dans l'espace d'attributs

Rappelons que comme l'exige l'ACP, les calculs menés jusqu'à présent supposent que les données soient centrés dans l'espace d'attributs. Comme nous l'avons vu à la section 2.4.4, cette centralisation passe par le calcul de la matrice du noyau centrée $\mathbf{K}_c$ donné par l'équation (2.3).

Par ailleurs il se peut qu'en fonction du problème à résoudre, la standardisation des données soit souhaitable en plus de l'exigence de la centralisation de celles ci. Dans ce cas les calculs menés jusqu'ici restent valables en remplaçant tout simplement la fonction de correspondance centrée ϕ_c par la fonction de correspondance centrée et réduite ϕ_{cn} et le kernel centré $\mathbf{K}_c$ par le kernel centré et réduit $\mathbf{K}_{cn}$. La standardisation des données dans l'espace d'attributs a été abordée à la section 2.4.6 et le kernel centré et réduit correspondant est donné par l'équation (2.4).

3.9 Propriétés des axes principaux dans l'espace d'attributs

Le KACP étant de l'ACP classique réalisé dans l'espace d'attributs, toutes les propriétés mathématiques et statistiques de l'ACP (voir Chapitre 2 du livre de Jolliffe [20]) sont

applicables au KACP mais dans l'espace d'attributs. Ainsi on peut donc affirmer que le KACP est une base orthogonale munie des propriétés suivantes :

- Les r premières composantes principales comportent plus de variance que toutes les autres directions orthogonales r .
- L'erreur quadratique moyenne d'approximation dans la représentation des observations par les r premières composantes principales est minimale.
- Les composantes principales sont décorréées.
- Les r premières composante principales ont une information mutuelle maximale vis à vis des données d'entrée (cela est valable sous des hypothèses gaussiennes, et dépend donc des données et du noyau choisi).

3.10 Décompositions d'inertie

3.10.1 Qualité de représentation globale

Comme dans le cas de l'ACP :

- L'inertie totale est donnée par :

$$I = \sum_{\alpha=1}^{\min(n,m)} \lambda_{\alpha} = \frac{1}{n} \sum_{\alpha=1}^{\min(n,m)} \nu_{\alpha}.$$

- La proportion de variance expliquée par la composante α est donnée par :

$$\frac{\lambda_{\alpha}}{\sum_{\alpha=1}^{\min(n,m)} \lambda_{\alpha}} = \frac{\nu_{\alpha}}{\sum_{\alpha=1}^{\min(n,m)} \nu_{\alpha}}.$$

- La proportion de variance expliquée par les r premières composantes principale est donnée par :

$$\tau_r = \frac{\sum_{\alpha=1}^r \lambda_{\alpha}}{\sum_{\alpha=1}^{\min(n,m)} \lambda_{\alpha}} = \frac{\sum_{\alpha=1}^r \nu_{\alpha}}{\sum_{\alpha=1}^{\min(n,m)} \nu_{\alpha}}, \quad r = 1, \dots, \min(n, m).$$

De nouveau, toutes ces proportions sont calculables uniquement à partir des valeurs propres de la matrice du noyau sans explicitement avoir à faire appel à la fonction de correspondance.

3.10.2 Inertie conservée par un axe

La décomposition de l'inertie λ_{α} conservée par l'axe principal α en termes d'observations est donnée par :

$$\begin{aligned} \lambda_{\alpha} &= \frac{1}{n} \sum_{i=1}^n y_{\alpha i}^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\mathbf{K}_{ci}^{\top} \mathbf{v}_{\alpha})^2 \\ &= \text{Var}(\mathbf{y}_{\alpha}). \end{aligned}$$

Cette inertie s'écrit uniquement en fonction du noyau et de ses vecteurs propres.

Quant à la décomposition de l'inertie conservée en termes de variables donnée par :

$$\begin{aligned}\lambda_\alpha &= \sum_{j=1}^m \varphi_{\alpha j}^2 \\ &= \sum_{j=1}^m \text{cor}(\phi_c^j, \mathbf{y}_\alpha)^2,\end{aligned}$$

elle n'est pas forcément envisageable à cause des raisons évoquées à la section 3.5.

3.11 Informations auxiliaires

Qualité de représentation des observations

Le Cosinus carré de l'angle entre $\phi_c(\mathbf{x}_i)$ et sa projection sur l'axe α est donné par :

$$\frac{y_{\alpha i}^2}{y_{1i}^2 + \dots + y_{\tilde{m}i}^2} = \frac{(\mathbf{K}_{ci}^\top \mathbf{v}_\alpha)^2}{(\mathbf{K}_{ci}^\top \mathbf{v}_1)^2 + \dots + (\mathbf{K}_{ci}^\top \mathbf{v}_{\tilde{m}})^2} \in [0, 1],$$

avec rappelons le $\tilde{m} = \text{rank}(\phi) \leq \min(n, m)$.

Quant au cosinus carré de l'angle entre $\phi_c(\mathbf{x}_i)$ et sa projection sur les r premières composantes, elle est donné par :

$$\frac{y_{1i}^2 + \dots + y_{ri}^2}{y_{1i}^2 + \dots + y_{\tilde{m}i}^2} = \frac{(\mathbf{K}_{ci}^\top \mathbf{v}_1)^2 + \dots + (\mathbf{K}_{ci}^\top \mathbf{v}_r)^2}{(\mathbf{K}_{ci}^\top \mathbf{v}_1)^2 + \dots + (\mathbf{K}_{ci}^\top \mathbf{v}_{\tilde{m}})^2} \in [0, 1].$$

Tout ceci est une fois de plus fonction uniquement du noyau et de ses vecteurs propres.

3.11.1 Qualité de représentation des variables

Le cosinus carré de l'angle entre ϕ_{cn}^j et sa projection sur l'axe α donné par $\varphi_{\alpha j}^2 = \text{cor}(\phi_{cn}^j, \mathbf{y}_\alpha)^2$, ainsi que le cosinus carré de l'angle entre ϕ_{cn}^j et sa projection sur les r premières composantes donné par $\text{cor}(\phi_{cn}^j, \mathbf{y}_1)^2 + \dots + \text{cor}(\phi_{cn}^j, \mathbf{y}_r)^2$ ne sont pas forcément envisageables non plus à cause des raisons évoquées à la section 3.5.

3.11.2 Contributions

La contribution de l'observation i à l'axe α est donnée par :

$$\frac{y_{\alpha i}^2}{n\lambda_\alpha} = \frac{(\mathbf{K}_{ci}^\top \mathbf{v}_\alpha)^2}{n\lambda_\alpha}.$$

Cette contribution dépend aussi bien évidemment uniquement de la matrice du noyau et de ses vecteurs propres.

Quant à la contribution de la variable j à l'axe α donnée par :

$$\frac{\varphi_{\alpha j}^2}{\lambda_\alpha} = \frac{\text{cor}(\phi_c^j, \mathbf{y}_\alpha)^2}{\lambda_\alpha},$$

elle n'est malheureusement pas forcément envisageable à cause des raisons évoquées à la section 3.5.

3.12 Avantages et désavantages du KACP

L'avantage principal du KACP est sans doute sa capacité à pouvoir capturer des structures complexes non linéaires dans les données, permettant ainsi une meilleure représentation et une bonne compréhension de celles-ci. Outre cela elle permet une réduction efficace de la dimensionnalité tout en préservant les caractéristiques importantes des données, conduisant ainsi à de meilleures performances dans les tâches. Enfin le KACP offre un outil puissant pour visualiser des données de grande dimension dans des dimensions inférieures, facilitant ainsi l'analyse exploratoire des données et la reconnaissance de formes.

Par ailleurs, le revers de la médaille du KACP réside sans doute dans la perte de l'information liée aux variables. En effet, comme nous l'avons vu dans les sections précédentes, la mise à l'écart de la fonction de correspondance ϕ dont prend avantage les méthodes du noyau, rend inaccessible les coordonnées factorielles des variables, le cercle de corrélation, la qualité de représentation des variables et leurs contributions aux axes principaux. Ces informations sur les variables constituent l'identité même d'une ACP et sont primordiales dans la compréhension et l'interprétation de ses résultats. Outre cela, l'on peut souligner la grande exigence du KACP en termes de calcul pour des grosses bases de données. Le calcul de la matrice du noyau associée évolue généralement en fonction du nombre de points de données plutôt que du nombre de dimensions de données. En plus choisir la fonction du noyau appropriée peut être difficile, car différents noyaux peuvent conduire à des résultats variables. Cela demande une connaissance à priori du domaine de l'expérimentation.

3.12.1 Comparaison entre l'ACP et le KACP

Bien que les deux méthodes sont toutes utilisées dans la réduction de dimension et l'extraction de variables, elles présentent néanmoins quelques différences :

- *Transformation linéaire contre transformation non linéaire* : En effet, l'ACP est basée sur des **transformations linéaires** et s'avère efficace lorsque les données sous-jacentes présentent des relations linéaires. Alors que le KACP est conçu pour faire face à des **relations non linéaires** dans les données. Il utilise une correspondance non linéaire (fonction noyau) pour transformer les données initialement non linéaires à des données linéaires dans un espace de dimension plus élevée, où l'ACP linéaire peut être appliquée. Cela lui confère la capacité de capturer des structures complexes non linéaires que l'ACP classique n'arrive pas.
- *Type d'axes principaux* : Comme nous le savons déjà, les composantes principales linéaires obtenues dans l'espace d'attributs correspondent aux composantes non linéaires dans l'espace d'entrée. Ainsi pendant que l'ACP détermine des **axes principaux linéaires** à travers les données dans l'espace d'entrée, le KACP détermine plutôt des axes principaux linéaires dans l'espace d'attributs qui correspondent à des **axes principaux non linéaires** dans l'espace d'origine.
- *Flexibilité* : L'ACP suppose la linéarité des données, ce qui limite sa capacité à capturer des modèles complexes dans des ensembles de données non linéaires. En revanche, le KACP peut gérer un large éventail de relations non linéaires, ce qui le rend plus flexible et adapté à un plus large éventail d'ensembles de données.
- *Espace d'action* : L'ACP réalise la réduction de dimension en opérant dans **l'espace d'entrée**. Elle sélectionne les r composantes principales qui expliquent la majeure

partie de la variance des données. Ces composants forment dans l'espace d'entrée un nouveau sous-espace où les données peuvent être projetées. Le KACP suit le même concept, mais opère plutôt dans **l'espace d'attributs** de dimension supérieure. Il sélectionne aussi les r composantes principales dans cet espace d'attributs afin de former un nouveau sous-espace où les données peuvent être projetées.

- *Complexité liée au temps de calcul* : L'ACP implique le calcul de la matrice de covariance de dimensions $p \times p$, qui peut être facilement décomposable en valeurs et vecteurs propres. En revanche, Le KACP nécessite le calcul de la matrice du noyau $n \times n$, ce qui peut être coûteux en termes de calcul lors de la décomposition spectrale. Ceci rend alors le KACP très coûteux en termes de complexité informatique contrairement à l'ACP qui est généralement plus rapide et plus efficace. La complexité du KACP augmente avec **la taille des observations** alors que celle de l'ACP augmente avec **le nombre de dimensions de l'espace d'entrée**.
- *Choix du noyau* : Un aspect critique du KACP est la sélection d'une fonction de noyau appropriée. Le choix du noyau détermine le type de relations non linéaires qui peuvent être capturées. La sélection du bon noyau est cruciale pour le succès du KACP et nécessite souvent des connaissances à priori dans le domaine alors que l'ACP ne requiert pas l'utilisation d'un noyau pour son fonctionnement
- *Domaine d'application* : Le choix entre l'ACP et le KACP dépend des caractéristiques de données et du problème spécifique à résoudre. Alors l'ACP est conseillé pour des domaines dont les données à traiter présentent des relations linéaires, le KACP lui se veut d'être utiliser pour les domaines présentant une base de données présentant des relations non linéaires.

Nous reprenons dans le tableau ci dessous le résumé de ces différences :

	KACP	ACP
Transformation	non linéaire	linéaire
axes principaux	non linéaires	linéaires
Flexibilité	modèles complexes	modèles simples
Espace d'action	espace d'origine	espace d'attributs
Complexité	taille d'observation n	dimension des variables p
Choix du noyau	plusieurs noyaux possibles	noyau identité
Domaine d'application	plus étendu	plus restreint

3.13 Algorithme du KACP

3.13.1 Étapes à suivre pour la réalisation du KACP

Afin de réaliser correctement le KACP, les étapes suivantes sont nécessaires :

1. **Préparer des données** : commencez par organiser l'ensemble de données sous forme matricielle, où les lignes représentent les observations et les colonnes représentent les caractéristiques.
2. **Sélectionner le noyau** : sélectionnez une fonction de noyau appropriée en fonction des caractéristiques des données. Les choix courants incluent le noyau gaussien (RBF), le noyau polynomial, le noyau sigmoïde, etc.

3. **Calculer la matrice du noyau** : calculez la matrice du noyau, qui mesure la similarité entre les paires de points dans l'espace transformé d'attribut.
4. **Centrer la matrice du noyau** : soustrayez la moyenne de chaque colonne et ligne de la matrice du noyau pour la centrer.
5. **Réduire la matrice du noyau centrée** : si nécessaire, diviser par la racine du produit des éléments diagonaux correspondants de la matrice du noyau centré pour standardiser.
6. **Décomposer la matrice du noyau en valeurs et vecteurs propres** : effectuez une décomposition des valeurs propres sur la matrice du noyau centrée pour obtenir les vecteurs propres et les valeurs propres.
7. **Extraire les composantes principales** : sélectionnez les r vecteurs propres dominant correspondant aux plus grandes valeurs propres, formant les composantes principales.
8. **Projeter les observations dans l'espace des axes principaux** : projetez les données originales sur le sous-espace couvert par les composants principaux sélectionnés pour obtenir les données transformées.

3.13.2 Pseudo algorithme du KACP

Ci dessous le pseudo algorithme permettant d'implémenter le KACP.

Algorithme du KACP

- KACP**($\mathbf{X}$, $\mathbf{K}$, m , τ)
- 1 $\mathbf{K} = \{K\}_{i,j=1,\dots,n}$, # Calcul de la matrice du noyau $n \times n$.
 - 2 $\mathbf{K}_c = (\mathbf{I} - \frac{1}{n}\mathbf{1}_{n \times n})\mathbf{K}(\mathbf{I} - \frac{1}{n}\mathbf{1}_{n \times n})$, # Centrer la matrice du kernel.
 - 3 $\mathbf{K}_{cn} = \mathbf{W}_c^{-1/2}\mathbf{K}_c\mathbf{W}_c^{-1/2}$, # Puis réduire la matrice du kernel centré si nécessaire,
sinon passé directement à 4
 - 4 $\mathbf{K}_{c/cn} = \mathbf{v}^T N \mathbf{v}$, # Diagonaliser la matrice centré ou centré réduite
afin d'extraire les valeurs et vecteurs propre
 - 5 $\tau_r = \frac{\sum_{\alpha=1}^r \nu_\alpha}{\sum_{\alpha=1}^{\min(n,m)} \nu_\alpha}$, $r = 1, \dots, \min(n, m)$, # Calculer les proportions expliquées
par les r premières composantes.
 - 6 $r_0 = \min(r)$ tel que $\tau_r \geq \tau$, # Choisir la dimension de l'espace principal réduit
 - 7 $\mathbf{V}_{r_0} = (\mathbf{v}_1, \dots, \mathbf{v}_{r_0})$, # Constituer la matrice des vecteurs réduits
 - 8 $\mathbf{Y} = \mathbf{K}_{c/cn}^T \mathbf{V}_{r_0}$, # Matrice des données projetées dans l'espace réduit
-

Chapitre 4

Expérience numérique : Données simulées

Dans ce chapitre, nous allons simuler de manière académique des données dans le but de leur octroyer une orientation précise qui sera adaptée à la mise en lumière de la problématique et/ou de l'avantage que nous voudrions mettre en exergue. Pour se faire nous simulerons des données simples à structure/relation non linéaire mettant en difficulté la méthode de l'ACP standard et montrerons dans la suite que les limites affichées par l'ACP classique peuvent être surmontées lorsque l'on fait appel au KACP.

Avant d'aller plus loin, il est important de signaler que les simulations et applications réalisées dans ce chapitre (chapitre 4) et le chapitre suivant (chapitre 5) ont été faites à partir du langage de programmation **Python** en utilisant principalement les fonctions **KernelPCA** et **PCA** de la librairie **sklearn.decomposition**.

4.1 Application dans le domaine du Clustering

Dans le domaine du clustering par exemple, il est très fréquent d'effectuer d'abord une réduction de dimension avant d'appliquer le clustering, ceci dans le but non seulement d'éviter le phénomène de la malédiction de dimension (lorsqu'on a faire à des données de très grande dimension) mais aussi de faciliter la visualisation des résultats. Cependant la méthode de réduction de dimension à choisir est d'importance capitale afin de garder au maximum les informations essentielles et dont la structure des clusters que l'on a dans l'espace d'origine.

4.1.1 Cercles concentriques

Nous simulons sur la figure 4.2 deux clusters en dimension 2 sous forme de deux cercles concentriques de rayons distincts. Il est clair que ces deux clusters ne sont pas linéairement séparables, ce qui signifie qu'aucune ligne droite ne peut séparer les deux classes.

Utilisation de l'ACP dans l'espace d'origine

Dans un premier temps nous utilisons l'ACP dans l'espace d'origine afin de projeter les données sur le premier plan principal -figure 4.2a- et ensuite sur le premier axe principal

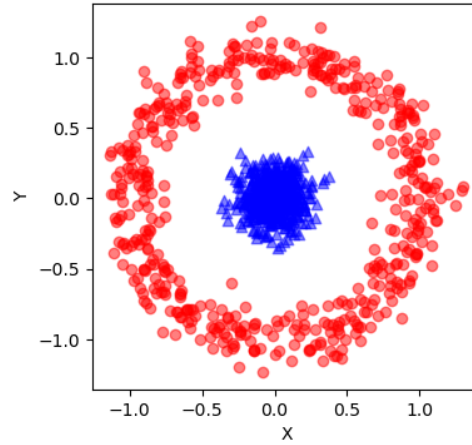
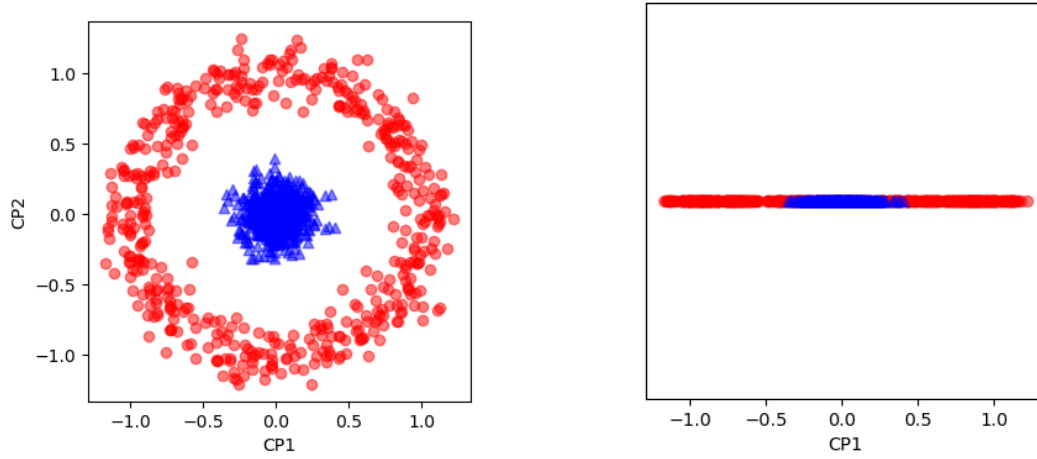


FIGURE 4.1 – Données simulées sous forme de deux cercles concentriques.



(a) Projection sur le premier plan principal. (b) Projection sur le premier axe principal.

FIGURE 4.2 – Utilisation de l'ACP dans l'espace d'origine.

-figure 4.2b. L'objectif étant ici de réduire la base de données de dimension 2 en dimension 1 à partir de l'ACP, tout en conservant les deux classes séparées.

Nous constatons clairement que les points sont désormais mélangés et la projection sur le premier axe principal ne permet plus de séparer les deux clusters comme c'était le cas dans l'espace d'origine et dans le plan principal. Cette réduction de dimension à partir de l'ACP altère donc la structure des clusters et les réduit en un seul cluster indissociable.

Utilisation de l'ACP dans l'espace d'attributs

Suivant la logique du KACP, transformons à présent -figure 4.3- notre base de données dans une dimension supérieure -dimension 3- en utilisant une fonction de correspondance très simple

$$\phi(x, y) = (x, y, 2x^2 + 2y^2),$$

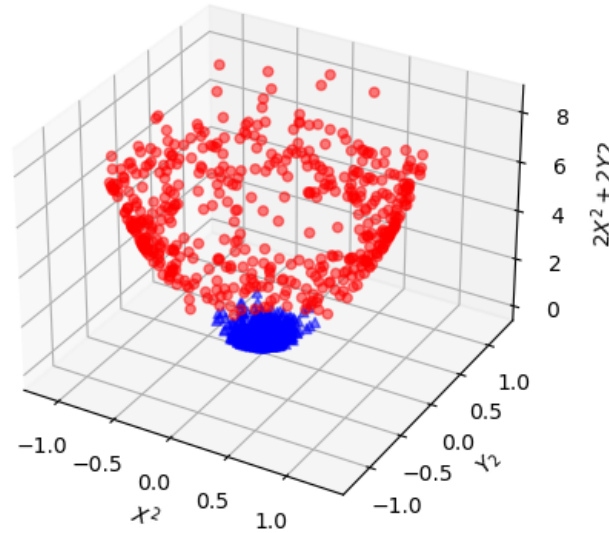


FIGURE 4.3 – Transformation de la base de données en utilisant la fonction de correspondance $\phi(x, y) = (x, y, 2x^2 + 2y^2)$.

choisie de sorte que le plus grand cercle soit shifté vers le haut dans la direction verticale, étant donné que la somme de ses composantes au carré est plus grande.

Puis appliquons l’ACP dans le nouveau espace de dimension 3 et gardons dans un premier temps le premier plan principal -figure 4.5 et dans un second temps le premier axe principal -figure 4.5.

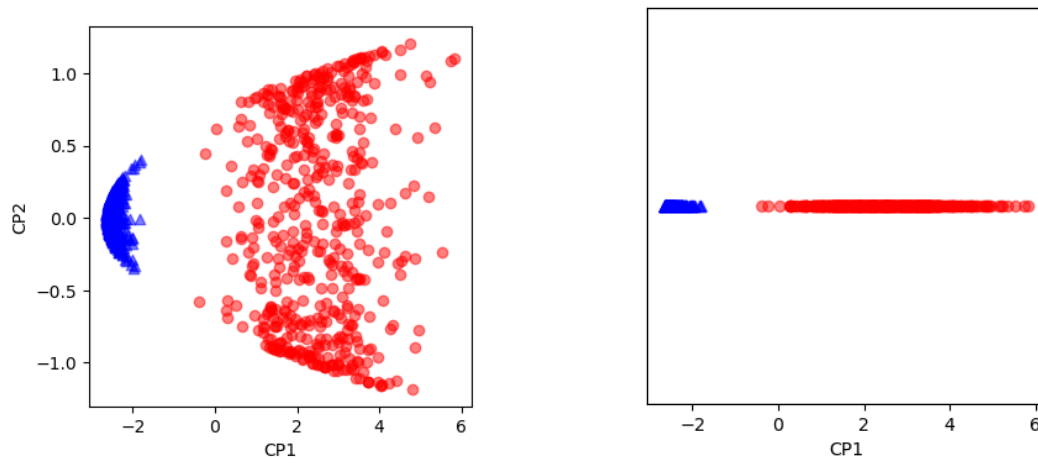
Contrairement à ce qui est observé dans l’espace d’origine, nous constatons que la projection sur le premier plan rend les deux clusters linéairement séparables et la projection sur le premier axe principal permet désormais de séparer efficacement les deux classes comme c’était le cas dans l’espace d’origine.

Utilisation du KACP

Au lieu de passer via une transformation explicite de la fonction de correspondance, mettons en application le “kernel trick” vu au chapitre 2. pour se faire, testons d’abord le kernel linéaire -figure 4.5. Nous retrouvons bien évidemment les résultats produits par l’ACP dans l’espace d’origine. Ensuite essayons le kernel polynomial de degré 2 avec un terme indépendant égale à 0 -figure 4.6- puis le kernel gaussien avec un coefficient gamma égale à 1 -figure 4.7- puis égale à 10 -figure 4.8.

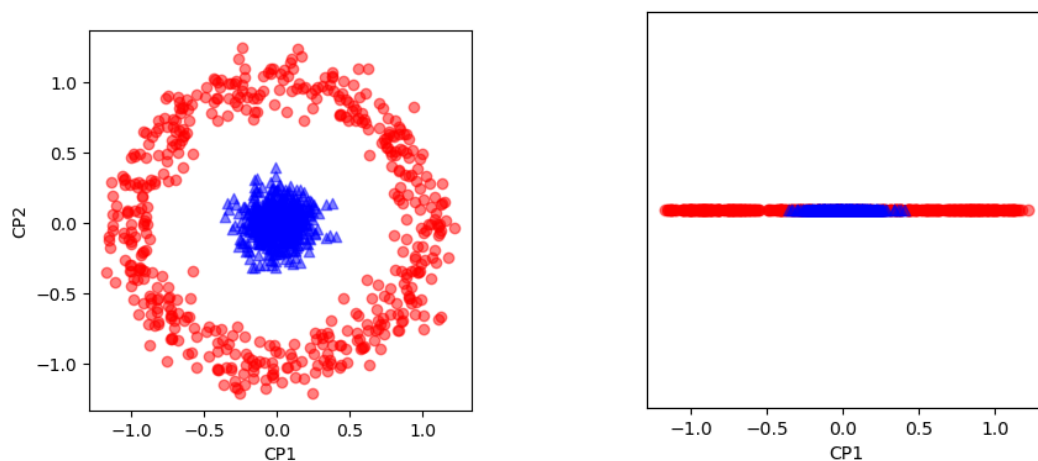
Nous constatons que la projection sur le plan principal par le kernel polynomial -figure 4.6a- arrive à rendre les deux classes linéairement séparables. Par contre la réduction de dimension sur le premier axe principal -figure 4.6b- par ce kernel ne fait pas mieux que le résultat obtenu à partir de l’ACP dans l’espace d’origine.

Quant au **kernel gaussien avec le paramètre gamma égale à 1**, il conduit à un résultat presque similaire à celui obtenu en appliquant l’ACP dans **l’espace d’attributs**, confirmant ainsi la non nécessité de passer dans l’espace d’attributs via une fonction de correspondance de grande dimension ou de dimension infinie et d’appliquer l’ACP dans cet espace, mais plutôt de travailler directement dans l’espace d’origine via le “kernel



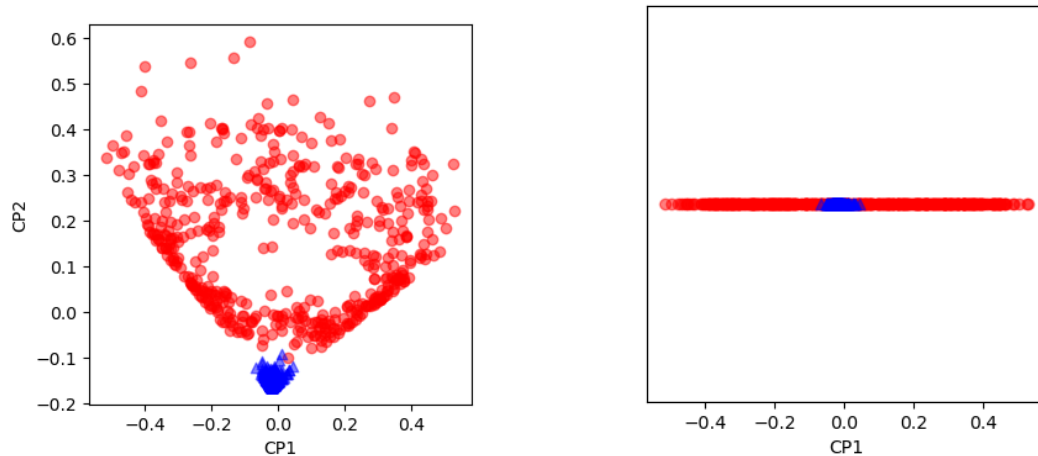
(a) Projection sur le premier plan principal dans l'espace d'attributs (b) Projection sur le premier axe principal dans l'espace d'attributs.

FIGURE 4.4 – Utilisation de l'ACP dans l'espace d'attributs.



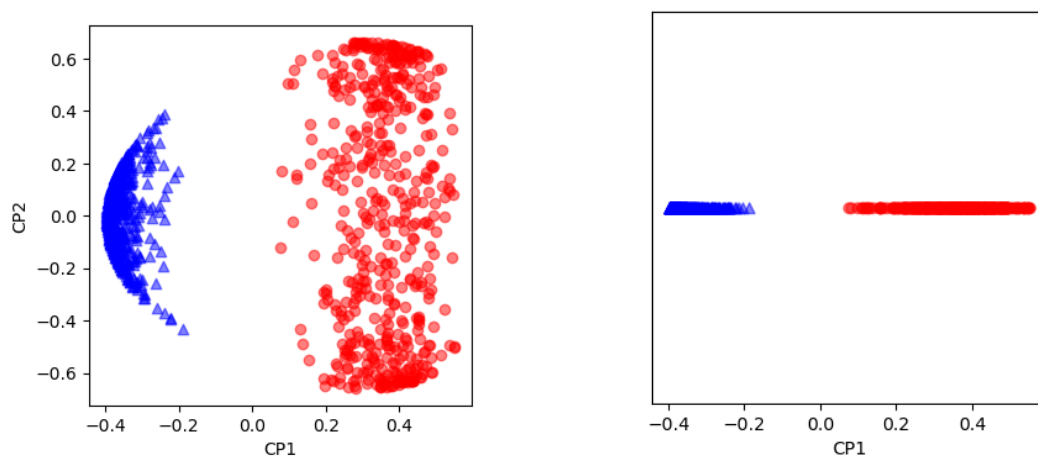
(a) Projection sur le premier plan principal (b) Projection sur le premier axe principal

FIGURE 4.5 – Utilisation du KACP avec un kernel linéaire.



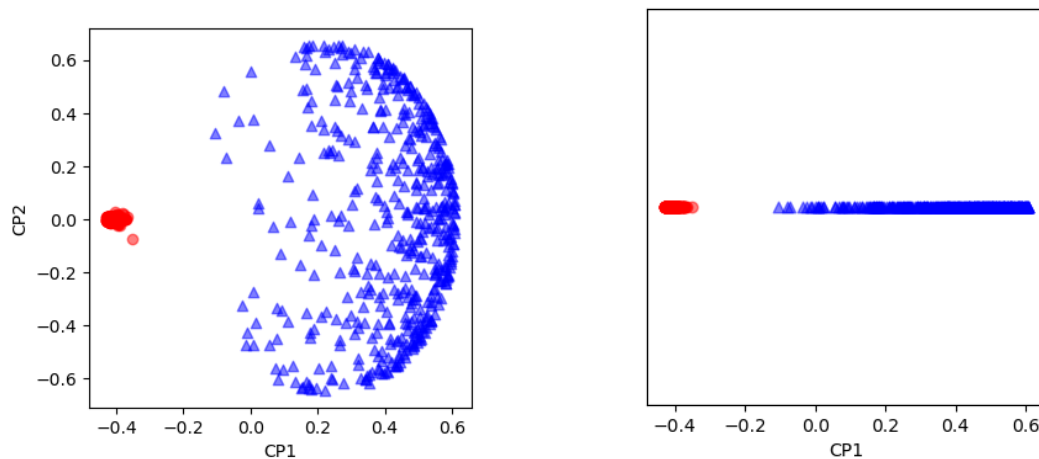
(a) Projection sur le premier plan principal (b) Projection sur le premier axe principal

FIGURE 4.6 – Utilisation du KACP avec un kernel polynomial de degré 2 et un terme indépendant de 0.



(a) Projection sur le premier plan principal (b) Projection sur le premier axe principal

FIGURE 4.7 – Utilisation du KACP avec un kernel gaussien de paramètre gamma égale à 1.



(a) Projection sur le premier plan principal (b) Projection sur le premier axe principal

FIGURE 4.8 – Utilisation du KACP avec un kernel gaussien de paramètre gamma égale à 10.

trick”. Ainsi ce kernel sépare complètement les 2 clusters à la fois dans le plan principal -figure 4.7a- que sur l’axe principal -figure 4.7b- en projetant le cluster formé par le cercle intérieur vers la gauche et celui formé par le cercle extérieur vers la droite.

Le kernel gaussien avec un paramètre gamma égale à 10 sépare lui aussi les 2 clusters projetés autant dans le plan principal -figure 4.9c- que sur l’axe principal -figure 4.8b, à la seule différence que cette fois c’est plutôt le cluster formé par le cercle extérieur qui est projeté vers la gauche et celui formé par le cercle intérieur vers la droite.

Cette première simulation nous montre que lorsque le kernel et ses hyperparamètres sont judicieusement choisis, le KACP peut contrairement à l’ACP, ressortir la structure complexe et non linéaire cachée dans la base de données. Les résultats ci dessus mettent en évidence le véritable potentiel du Kernel PCA par rapport au PCA standard.

Afin de rester dans la logique du KACP, nous ne ferons plus dans la suite des transformations explicites de la base de donnée en utilisant une fonction de correspondance, mais prendrons plutôt avantage du “kernel trick” et nous nous limiterons par conséquent uniquement à l’utilisation des fonctions du noyau dans l’espace d’origine.

Reconstruction

Bien que le KACP ait permis de séparer les deux clusters comme illustré à la section précédente, il présente néanmoins l’inconvénient de ne pas toujours être capable de reconstruire à l’identique les données d’origine lorsque le nombre de composantes principales retenues est égale à la dimension de l’espace d’origine. En effet, contrairement à l’ACP qui conduit à une reconstruction quasi exacte des données d’origine lorsque le nombre des composantes principales est égal à la dimension de l’espace d’origine, le KACP ne couvrant pas réellement la dimension de l’espace d’attributs mais s’étendant uniquement sur un sous-espace de celle ci, conduit plutôt à une reconstruction approximative. Nous illustrons cela sur la figure 4.9

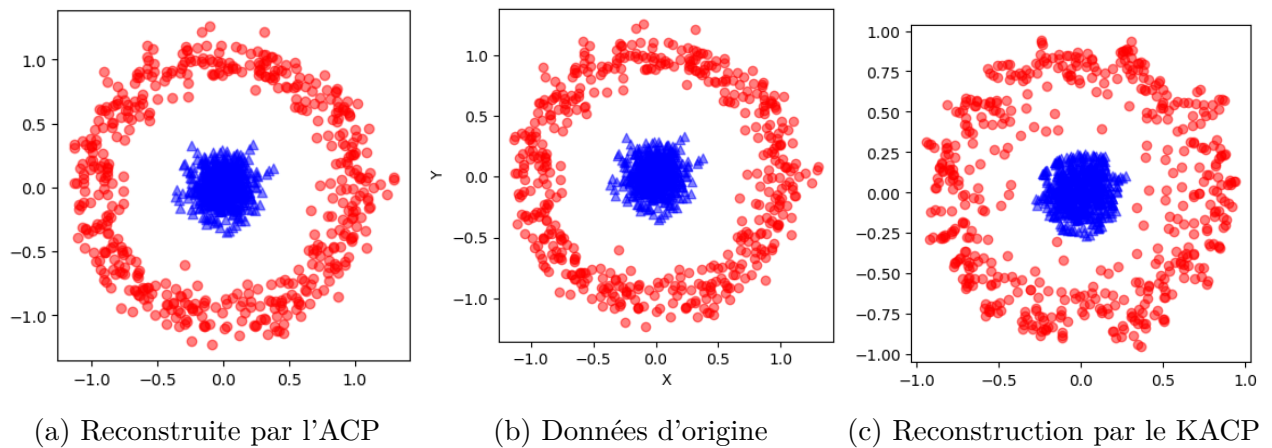


FIGURE 4.9 – Reconstruction des données des 2 cercles concentriques

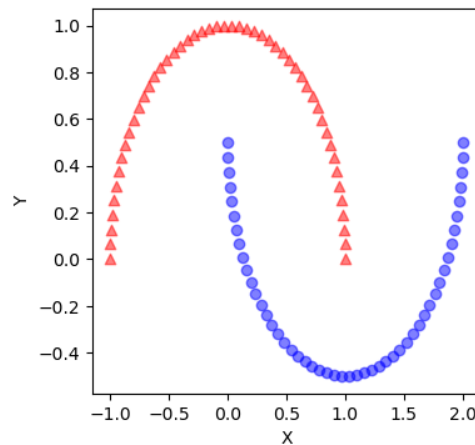


FIGURE 4.10 – Données simulées sous forme de deux demi-lunes distinctes.

4.1.2 Demi-lune

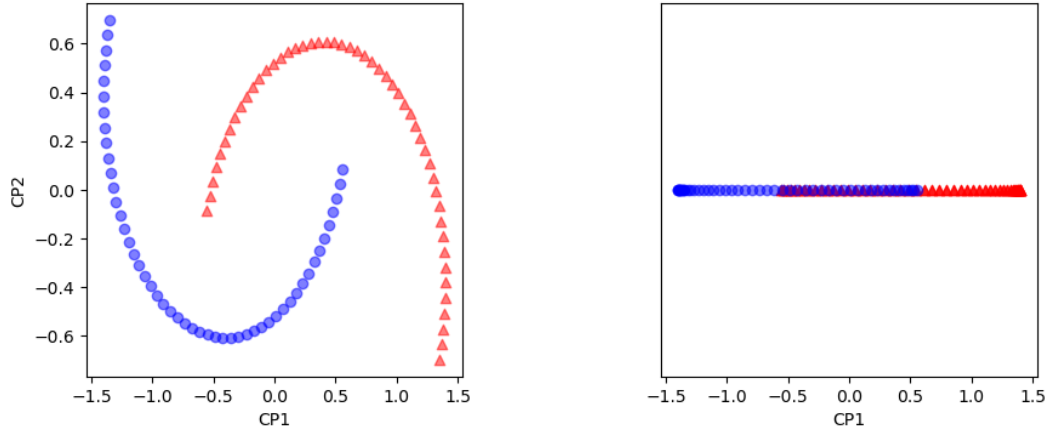
Comme à la section précédente, nous simulons sur la figure 4.10 deux clusters en dimension 2 en forme de demi-lune entrelacées et bien distinctes.

Utilisation de l'ACP

Nous utilisons comme précédemment l'ACP pour projeter les données sur le premier plan principal -figure 4.11a- et ensuite sur le premier plan principal, figure 4.11b. Le même constat que dans le cas des cercles concentriques est observé : le premier axe principal ne permet plus de séparer les deux clusters comme c'était le cas dans l'espace d'origine et dans le premier plan principal.

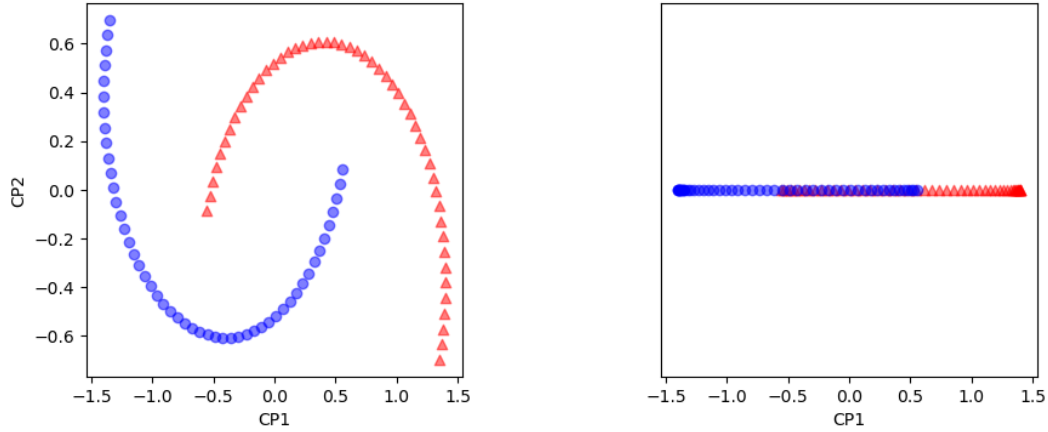
Utilisation du KACP

Dans un deuxième temps nous refaisons le même exercice en utilisant cette fois ci le KACP. Nous testons d'abord le kernel linéaire, figure 4.12 et retrouvons bien les résultats produits par l'ACP. Ensuite nous essayons à tour de rôle le kernel polynomial de degré 2



(a) Projection sur le premier plan principal. (b) Projection sur le premier axe principal.

FIGURE 4.11 – Utilisation de l'ACP.



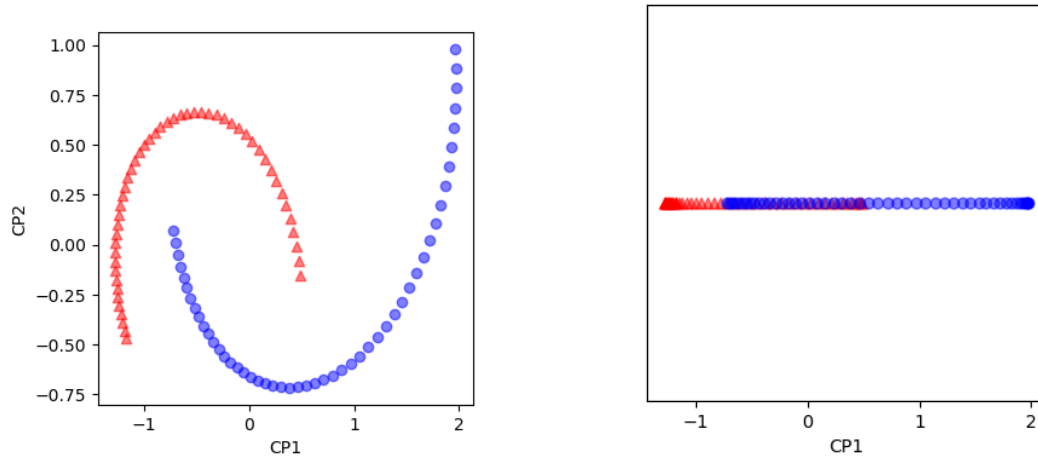
(a) Projection sur le premier plan principal (b) Projection sur le premier axe principal

FIGURE 4.12 – Utilisation du KACP avec un kernel linéaire.

avec un terme indépendant égale à 1, figure 4.13, puis le kernel gaussien avec un coefficient gamma égale à 0.385, figure 4.14, puis égale à 15, figure 4.15.

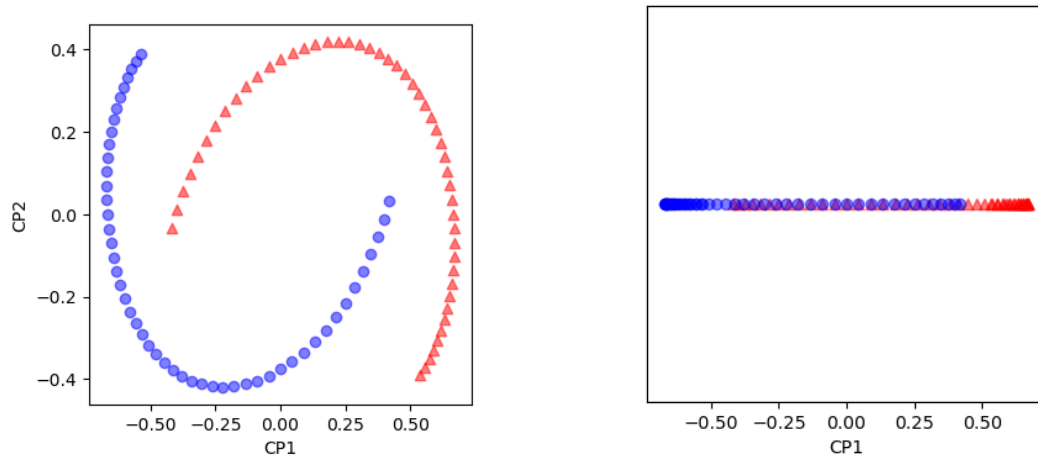
Nous constatons que le kernel polynomial non seulement n'améliore pas le résultat obtenu par l'ACP ou le kernel linéaire mais curieusement permute aussi la position de deux clusters dans le premier plan principal, figure 4.13a. Quant au kernel gaussien avec le paramètre gamma égale à 0.385, bien qu'il ne sépare pas les 2 clusters en dimension 1, figure 4.15b, on observe que la projection sur le premier plan principale, figure 4.14a a tendance à déplier les demi-lunes et à les décaler l'une de l'autre. Lorsque l'on passe à gamma égale à 15, figure 4.15, la projection sur le premier plan principal, figure 4.15a montre deux demi-lunes torpillées mais complètement décaler l'une de l'autre et la projection sur le premier axe principale conduit à 2 clusters complètement séparés, figure 4.15b comme dans l'espace d'origine.

Comme dans le cas précédent, Le KACP montre une fois de plus, contrairement à l'ACP, le potentiel énorme dont il dispose à ressortir les structures complexes et non linéaires cachées dans la base de données lorsque la fonction du noyau est judicieusement



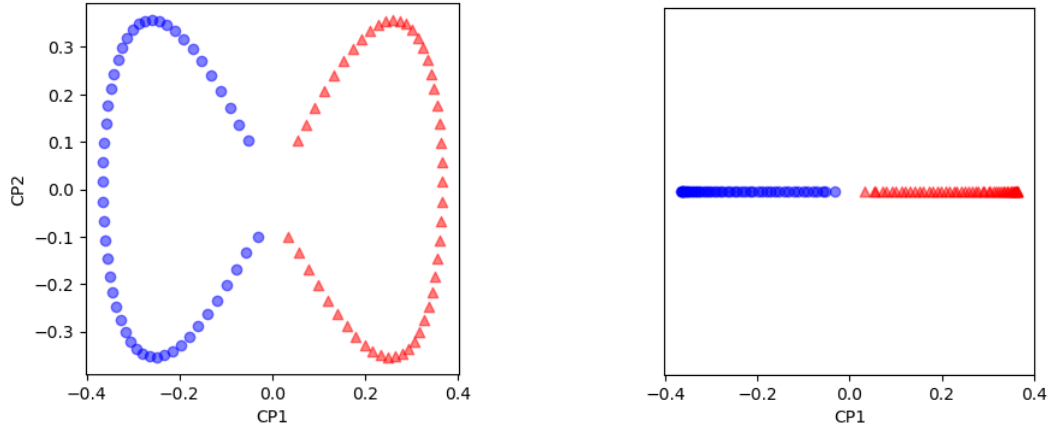
(a) Projection sur le premier plan principal (b) Projection sur le premier axe principal

FIGURE 4.13 – Utilisation du KACP avec un kernel polynomial de degré 2 et un terme indépendant de 1.



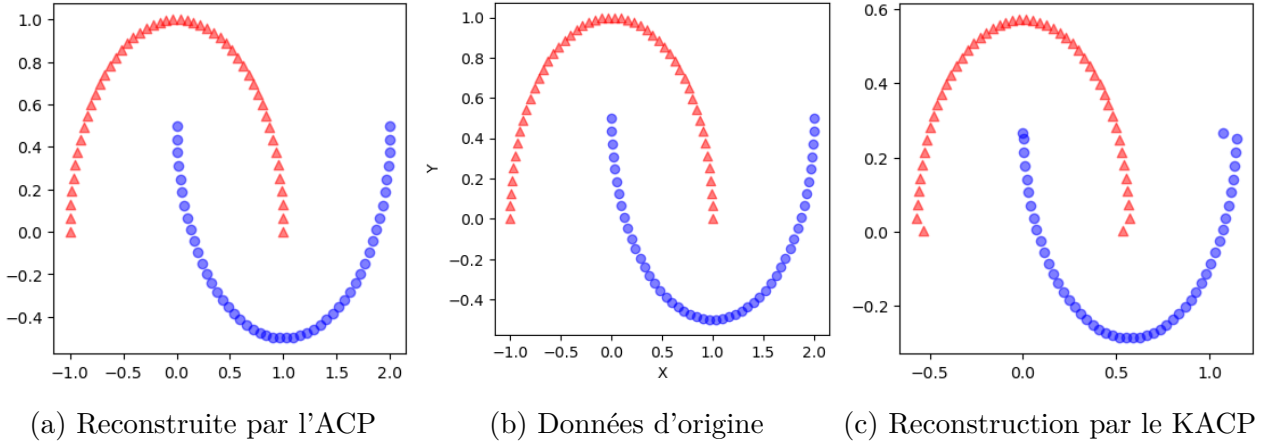
(a) Projection sur le premier plan principal. (b) Projection sur le premier axe principal.

FIGURE 4.14 – Utilisation du KACP avec un kernel gaussien de paramètre gamma égale à 0.385.



(a) Projection sur le premier plan principal. (b) Projection sur le premier axe principal.

FIGURE 4.15 – Utilisation du KACP avec un kernel gaussien de paramètre gamma égale à 15.



(a) Reconstituée par l'ACP (b) Données d'origine (c) Reconstruction par le KACP

FIGURE 4.16 – Reconstruction des données des 2 demi-lunes.

choisie. Dans le cas contraire, il peut plutôt empirer les résultats donnés par l'ACP linéaire.

Reconstruction

Comme dans le cas précédente nous montrons sur la figure 4.16, la capacité de reconstruction par les deux méthodes. Une fois de plus on observe que contrairement à l'ACP, le KACP montre sa limite au niveau de la reconstruction à l'identique des données d'origine.

4.2 Application dans la préservation de la forme d'origine en dimension inférieure

Un autre critère d'évaluation d'une méthode de réduction de dimension est aussi sa capacité à pouvoir conserver la forme d'origine d'un objet aussi complexe qu'il soit lorsque l'on passe à une dimension plus petite.

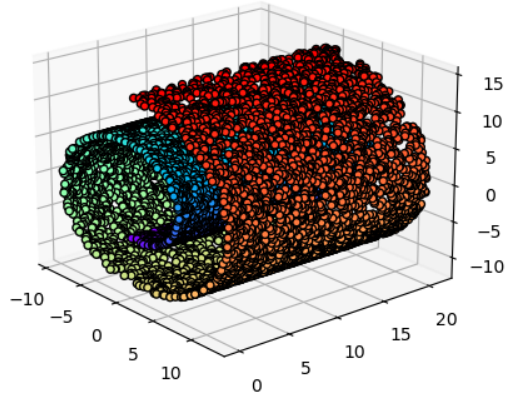


FIGURE 4.17 – Données simulées sous forme de swiss manifold.

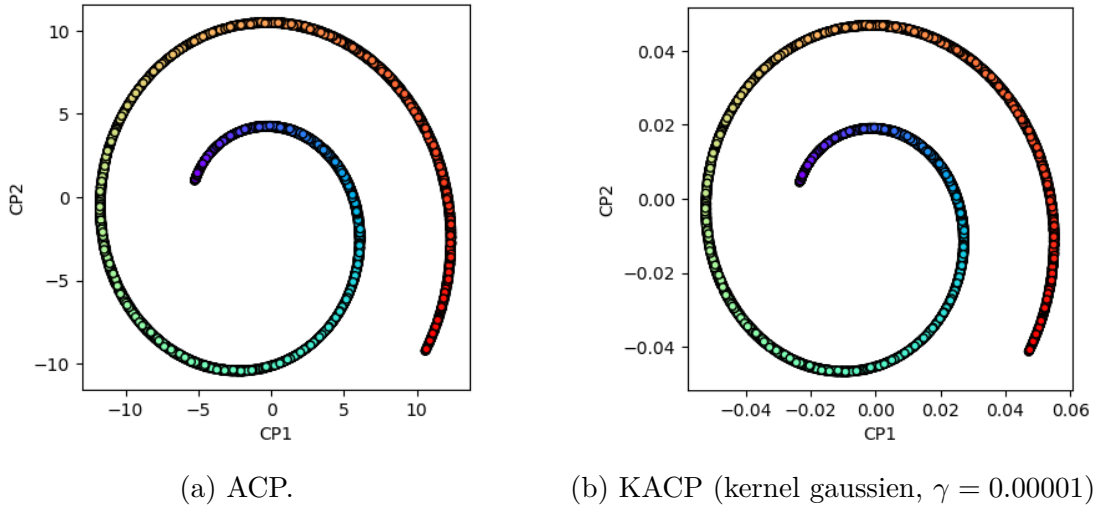


FIGURE 4.18 – Forme d’origine du swiss manifold préservée dans le plan principal.

Dans cette section nous simulerons un objet de forme particulière, à savoir le “swiss Manifold”. Une description et/ou les équations de cet objet se trouve dans le chapitre 6 de [4].

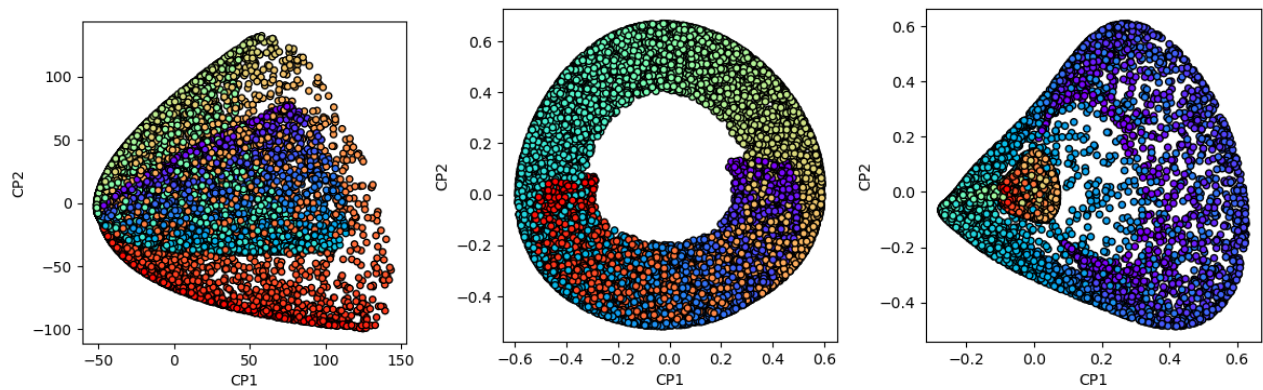
4.2.1 “Swiss Manifold”

Sur la figure 4.17, nous représentons en dimension 3, le “Swiss Manifold” et le représentons à nouveau en dimension 2 dans le plan principal obtenu par l’ACP -figure 4.18a- puis dans le plan principal obtenus par le KACP en utilisant le kernel gaussien de paramètre $\gamma = 0.0001$ -figure 4.18b.

4.10 deux clusters en dimension 2 en forme de demi-lune entrelacées et bien distinctes.

Nous remarquons que l’ACP linéaire donne un résultat aussi satisfaisant que le KACP avec un kernel et un paramètre γ adéquatement choisi.

Par ailleurs lorsqu’on choisi un noyau aléatoire on voit sur la figure 4.19 que le résultat obtenu s’écarte de la forme d’origine escomptée en dimension 3. Ceci nous montre que



(a) Noyau quadratique (b) Noyau gaussien ($\gamma = 0.01$) (c) Noyau gaussien ($\gamma = 0.05$)

FIGURE 4.19 – Forme d'origine du swiss manifold non préservée par le KACP dans le plan principal.

dans certains cas l'ACP linéaire fait bien le Job et que l'on n'a pas forcément besoin de la recherche d'un noyau adapté qui peut s'avérer fastidieux

Chapitre 5

Cas pratique : données réelles

En plus de permettre une bonne visualisation des données comme illustré au chapitre 4, le KACP trouve aussi son application dans le domaine de la classification et de la régression. En effet, comme dans le cadre de la classification en composantes principales (CCP) et de la régression en composantes principale (RCP), lorsque l'on a à faire à une base de données de dimension élevée, au lieu de faire directement une classification ou une régression sur les variables explicatives d'origine, le KACP peut d'abord être utilisé afin de réduire la dimension de la base de données tout en gardant le maximum d'information puis alors faire la classification ou la régression plutôt sur les composantes principales les plus pertinentes.

Dans ce chapitre il sera question de mettre en lumière le pouvoir prédictif des composantes principales résultant du KACP en comparant les résultats obtenus à ceux de l'ACP standard. Ainsi nous effectuerons à tour de rôle une **classification en composantes principales du noyau (CCPN)** (voir section 5.1) puis une **régression en composantes principales du noyau (RCPN)** (voir section 5.2) [21] sur des bases de données réelles.

5.1 Application du KACP en Classification

Afin d'évaluer comment le KACP performe par rapport à l'ACP standard lorsqu'il est appliqué dans le domaine de la classification, nous allons effectuer une CCPN sur un jeu de données bien connu à savoir le *Breast Cancer data*.

L'objectif est ici dans un premier temps de regarder à quel point les observations appartenant aux différentes classes ont été séparées après réduction de dimension par la méthode du KACP et dans un second temps d'évaluer la capacité de prédiction des classes du test set lorsqu'une méthode de prédiction est entraînée sur le training set.

5.1.1 Description de la base de données

La base de données *Breast Cancer data* (de la librairie `sklearn.datasets`) compte 569 observations sur 31 attributs dont l'une représente la variable réponse binaire (diagnosis, M = malignant (37%) et B = benign(63%)) et 30 autres représentent des variables explicatives numériques constituées de 3 mesures sur 10 caractéristiques de la tumeur. Nous allons ainsi effectuer la classification d'une tumeur (1 = cancéreuse ou 0 = non cancéreuse) en fonction des composantes principales résultant d'un (K)ACP sur les différentes mesures (moyenne, écart type, minimum) effectuées sur les caractéristiques propres à chaque tumeur, à l'instar

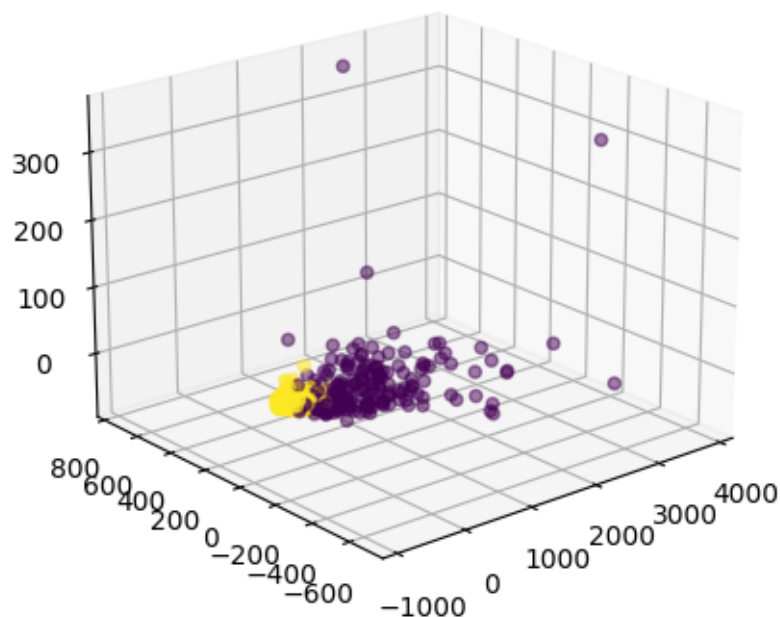


FIGURE 5.1 – Visualisation des classes de la base de données *breast cancer* à partir de la projection des individus dans l'espace formé par les trois premières composantes principales de l'ACP classique

de la concavité de la tumeur, la surface des lobes de la tumeur et bien d'autres. Cette base de données fait partie des multiples données disponibles dans la librairie **sklearn.datasets** de python et a donc l'avantage d'être utilisable directement sans être téléchargée.

5.1.2 Visualisation des classes en 3D

Dans un premier temps utilisons l'ensemble de la base de données afin de réaliser d'abord une ACP puis un KACP en utilisant différents noyaux. Gardons les 3 premières composantes principales afin de visualiser quelles méthodes/noyaux permettent-ils de mieux séparer les deux classes distinguées par deux couleurs différentes.

Nous présentons dans la figure 5.1 le résultat donné par l'ACP, puis dans la figure 5.2, celui donné par le KACP en utilisant le noyau polynomial et dans les figures 5.3 et 5.4 ceux donnés par le KACP en utilisant les noyaux gaussiens de paramètres γ respectivement égaux à 100 et 7×10^{-7} .

Le constat immédiat qui saute à l'oeil est que le KACP utilisant le noyau polynomial et le noyau gaussien avec un paramètre $\gamma = 100$ mélange les deux classes. Tandis que le premier concentre les deux classes non séparées, le deuxième tend plutôt à les disperser. Quant à l'ACP, il concentre les deux classes comme dans le cas du noyau polynomial mais arrive à séparer une bonne partie des deux classes. Enfin le KACP utilisant le noyau gaussien de paramètre $\gamma = 7 \times 10^{-7}$ disperse largement les deux classes et présente une séparation des classes plus prononcée que dans le cas de l'ACP malgré que certaines observations des classes différents semblent se chevaucher. Par conséquent on s'attendrait

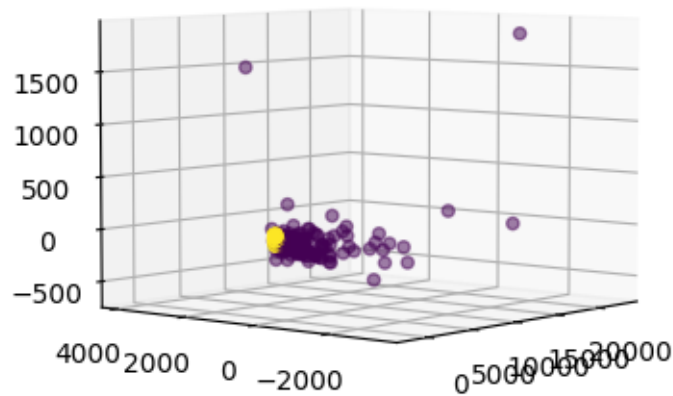


FIGURE 5.2 – Visualisation des classes de la base de données *breast cancer* à partir de la projection des individus dans l'espace formé par les trois premières composantes principales du KACP utilisant un noyau polynomial.

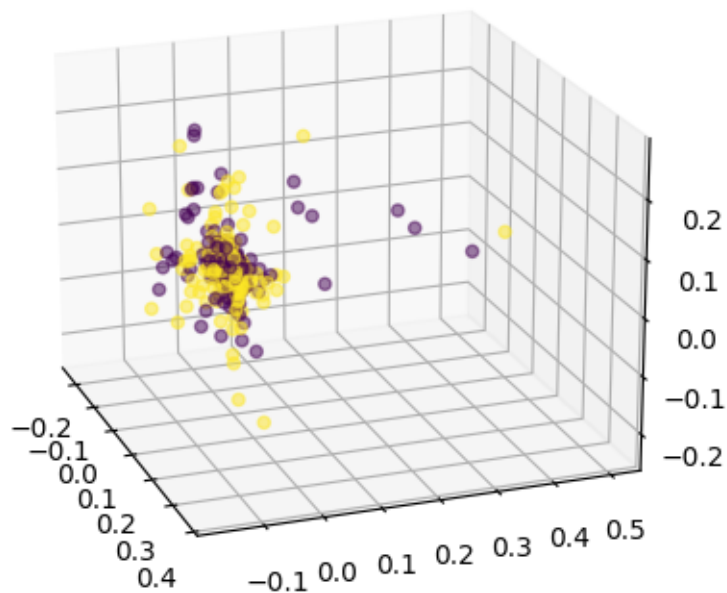


FIGURE 5.3 – Visualisation des classes de la base de données *breast cancer* à partir de la projection des individus dans l'espace formé par les trois premières composantes principales du KACP utilisant un noyau gaussien de paramètre $\gamma = 100$.

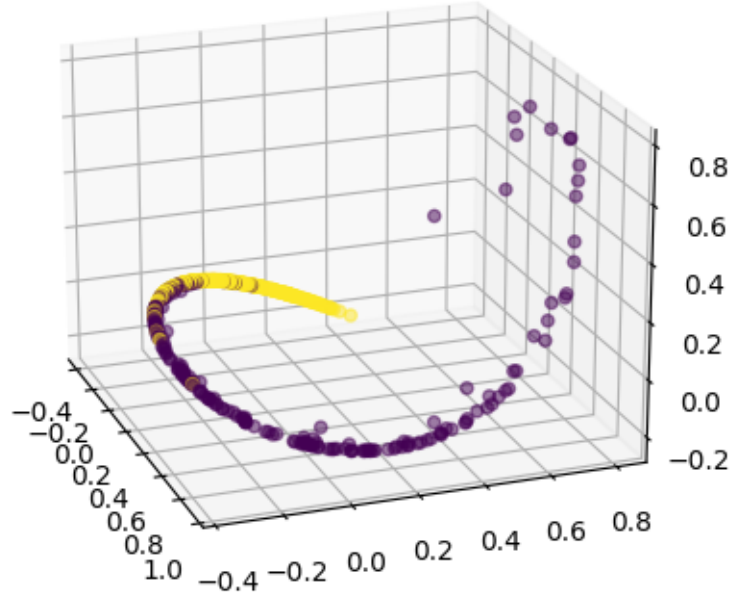


FIGURE 5.4 – Visualisation des classes de la base de données *breast cancer* à partir de la projection des individus dans l’espace formé par les trois premières composantes principales du KACP utilisant un noyau gaussien de paramètre $\gamma = 7 \times 10^{-7}$.

à ce que le KACP utilisant le noyau gaussien de paramètre $\gamma = 7 \times 10^{-7}$ ait un pouvoir prédictif de classe plus élevés que l’ACP lorsque l’on retient les 3 premières composantes principales pour effectuer la classification. C’est ce que nous allons vérifier dans la suite.

5.1.3 Pouvoir prédictif

A présent nous allons faire de la classification en utilisant comme prédicteurs les 3 premières composantes principales issues de l’ACP puis du KACP et comme méthode de classification, la régression logistique avec comme seuil de classification la valeur 0.5 (valeur par défaut de la fonction `LogisticRegression()` de la librairie `Sklearn` de python). Pour se faire, nous splittons la base de données en K échantillons stratifiés de sorte à garder dans chaque échantillon quasi la même distribution pour la variable réponse que dans la base de données globale. Nous faisons une validation croisée où à tour de rôle ($K - 1$) des K échantillons seront utilisés pour entraîner/apprendre le modèle et l’échantillon restant pour évaluer la performance de ce dernier.

Afin de déterminer le nombre des composantes principales (CP) et le paramètre γ optimaux à utiliser, nous effectuons d’abord sur le KACP utilisant le kernel gaussien (étant donné que ce dernier contient les deux paramètres) un *grid search* intelligent en faisant varier le nombre de composantes principales de 1 à 10 par pas de 1 et le paramètre γ de 10^{-1} à 10^{-10} par pas de -1 sur l’exposant de 10. Nous identifions que la valeur de l’hyperparamètre γ qui maximise le score moyen (*mean accuracy*, défini comme le rapport entre le nombre de prédictions correctes et le nombre total de prédictions) sur les

échantillons se trouve autour de 10^{-6} pour un nombre de CP égal à 3. Après nous faisons alors varier γ de 3×10^{-6} à 4×10^{-7} en fixant le nombre de CP à 3 et présentons dans le tableau 5.1 les scores moyens obtenus, puis pour la valeur optimale de γ trouvée, nous faisons varier de nouveau le nombre de CP de 1 à 10 (tableau 5.2) afin de s'assurer que la valeur optimale de ce nombre reste inchangée pour la valeur de γ retournée par le *grid search*. Dans un soucis de faire une comparaison utilisant un nombre identique des CP pour l'ACP et pour le KACP, nous gardons le même nombre de composantes principales obtenu précédemment par le *grid search*, c'est à dire 3.

γ	3×10^{-6}	2×10^{-6}	1×10^{-6}	9×10^{-7}	8×10^{-6}	7×10^{-6}	6×10^{-6}	5×10^{-7}	4×10^{-7}
Score moyen %	89.807	89.986	91.392	91.568	91.568	91.568	91.392	91.392	91.217

TABLE 5.1 – Score moyen de prédiction obtenu en fonction du paramètre γ en faisant une classification en composantes principales (du noyau) sur la base de données breast cancer, en utilisant le nombre optimal des composantes principales (CP=3) et une validation croisée avec $K = 5$.

nombre des CP	1	2	3	4	5	6	7	8	9
Score moyen %	90.686	91.392	91.568	91.568	91.392	91.392	91.392	91.392	91.392

TABLE 5.2 – Score moyen de prédiction obtenu en fonction du nombre de composantes principales en faisant une classification en composantes principales (du noyau) sur la base de données breast cancer, en utilisant le paramètre optimal $\gamma = 7 \times 10^{-6}$ et une validation croisée avec $K = 5$.

Nous présentons dans le tableau 5.3 le maximum, le minimum et la moyenne des scores de prédiction ainsi que l'écart type (sd) obtenus pour $K = 5$ et $K = 10$ échantillons croisés par l'ACP et par le KACP en utilisant différents noyaux et un nombre des CP égal à 3.

Méthode	Noyau	K=5				K=10			
		max(%)	min(%)	moy.(%)	sd	max(%)	min(%)	moy.(%)	sd
ACP		94.73	75.43	85.76	0.071	96.49	73.68	86.99	0.071
KACP	quadratique	89.47	85.96	87.34	0.015	92.85	75.43	88.05	0.053
	cubique	85.08	77.19	81.72	0.032	87.5	73.68	80.32	0.043
	gaussien ($\gamma = 1 \times 10^{-2}$)	63.15	62.28	62.74	0.004	63.15	61.40	62.74	0.007
	gaussien ($\gamma = 7 \times 10^{-7}$)	95.61	86.84	91.57	0.035	94.74	82.46	91.39	0.039

TABLE 5.3 – Scores de prédiction obtenus en faisant une classification en composantes principales (du noyau) sur la base de données breast cancer en utilisant une validation croisée avec $K = 5$ et $K = 10$ échantillons et 3 composantes principales.

Comme attendu, on observe que les meilleurs scores moyens sont obtenus (que ce soit pour $K = 5$ que pour $K = 10$) par la méthode KACP utilisant le noyau gaussien avec le paramètre $\gamma = 7 \times 10^{-7}$. Les deuxièmes meilleurs scores moyens sont obtenus par le KACP utilisant le noyau quadratique. L'ACP quant à elle vient en troisième position suivie du KACP utilisant le noyau cubique. Les pires scores moyens sont obtenus par le KACP utilisant le noyau gaussien mais avec un paramètre $\gamma = 0.01$ plus grand. Ceci illustre l'importance capitale que joue ce paramètre γ dans le choix du noyau.

Lorsque l'on utilise le maximum des scores comme paramètre de performance, alors ce sont le KACP utilisant le noyau gaussien de paramètre $\gamma = 7 \times 10^{-7}$ et l'ACP qui sortent les meilleurs, suivis du KACP utilisant respectivement le noyau quadratique, le noyau cubique et le noyau gaussien de paramètre $\gamma = 0.1$. Par contre lorsque l'on utilise le minimum des scores comme paramètre de performance, c'est toujours le KACP utilisant le noyau gaussien de paramètre $\gamma = 7 \times 10^{-7}$ qui est le meilleur, suivis du KACP utilisant le noyau quadratique, puis de l'ACP, puis du KACP utilisant respectivement le noyau cubique et le noyau gaussien de paramètre $\gamma = 0.1$.

En somme on voit que, peu importe le critère de performance retenu, le KACP utilisant le noyau gaussien avec le paramètre $\gamma = 7 \times 10^{-7}$ fait mieux que l'ACP. Par ailleurs lorsque ce paramètre γ n'est pas bien choisi, ce noyau conduit à des pires résultats. Cependant, il est quand même intéressant de noter que le KACP avec noyau gaussien de paramètre $\gamma = 0.01$, malgré sa mauvaise performance, donne des résultats moins volatiles que les autres noyaux, lorsque l'on passe d'un échantillon à un autre.

Afin de voir plus en profondeur la classification obtenue par classe, nous avons présenté dans le tableau 5.4 pour le cas $K = 5$, la matrice de confusion obtenue sur les 5 échantillons.

		Échantillon				
Méthode	Noyau	1	2	3	4	5
ACP		$\begin{bmatrix} 34 & 9 \\ 19 & 52 \end{bmatrix}$	$\begin{bmatrix} 40 & 3 \\ 3 & 68 \end{bmatrix}$	$\begin{bmatrix} 33 & 9 \\ 8 & 64 \end{bmatrix}$	$\begin{bmatrix} 39 & 3 \\ 9 & 63 \end{bmatrix}$	$\begin{bmatrix} 37 & 5 \\ 13 & 58 \end{bmatrix}$
KACP	quadratique	$\begin{bmatrix} 20 & 23 \\ 0 & 71 \end{bmatrix}$	$\begin{bmatrix} 25 & 18 \\ 2 & 69 \end{bmatrix}$	$\begin{bmatrix} 16 & 26 \\ 0 & 72 \end{bmatrix}$	$\begin{bmatrix} 25 & 17 \\ 0 & 72 \end{bmatrix}$	$\begin{bmatrix} 27 & 17 \\ 1 & 70 \end{bmatrix}$
	cubique	$\begin{bmatrix} 20 & 15 \\ 1 & 70 \end{bmatrix}$	$\begin{bmatrix} 30 & 13 \\ 2 & 69 \end{bmatrix}$	$\begin{bmatrix} 29 & 13 \\ 3 & 69 \end{bmatrix}$	$\begin{bmatrix} 31 & 11 \\ 1 & 71 \end{bmatrix}$	$\begin{bmatrix} 31 & 11 \\ 2 & 69 \end{bmatrix}$
	gaussien ($\gamma = 1 \times 10^{-2}$)	$\begin{bmatrix} 0 & 43 \\ 0 & 71 \end{bmatrix}$	$\begin{bmatrix} 0 & 43 \\ 0 & 71 \end{bmatrix}$	$\begin{bmatrix} 0 & 42 \\ 0 & 72 \end{bmatrix}$	$\begin{bmatrix} 0 & 42 \\ 0 & 72 \end{bmatrix}$	$\begin{bmatrix} 0 & 42 \\ 0 & 72 \end{bmatrix}$
	gaussien ($\gamma = 7 \times 10^{-7}$)	$\begin{bmatrix} 30 & 13 \\ 2 & 69 \end{bmatrix}$	$\begin{bmatrix} 33 & 10 \\ 2 & 69 \end{bmatrix}$	$\begin{bmatrix} 35 & 7 \\ 2 & 70 \end{bmatrix}$	$\begin{bmatrix} 38 & 4 \\ 1 & 71 \end{bmatrix}$	$\begin{bmatrix} 37 & 5 \\ 2 & 69 \end{bmatrix}$

TABLE 5.4 – Matrices de confusion (où la première ligne représente la classe réelle M et la deuxième, la classe réelle B alors que la première colonne représente la classe prédite M et la deuxième, la classe prédite B) obtenues en faisant une classification en composantes principales (du noyau) sur la base de données breast cancer en utilisant une validation croisée avec $K = 5$ échantillons.

Nous remarquons globalement que le KACP conduit à une meilleure classification pour la classe la plus peuplée, ce qui n'est pas le cas de l'ACP. Le KACP utilisant le noyau gaussien classe tous les individus dans la classe majoritaire expliquant ainsi sa mauvaise performance dans les scores et sa faible volatilité à travers les échantillons. Seul le KACP utilisant le noyau gaussien de paramètre $\gamma = 7 \times 10^{-7}$ fait aussi bien que l'ACP dans la

classification de la classe minoritaire expliquant ainsi les bons résultats des scores dont il dispose, puisqu'à la fois il classe bien les individus de la classe majoritaire comme les autres noyaux et en plus il classe aussi bien les individus de la classe minoritaire comme l'ACP.

Afin de rendre l'analyse non seulement indépendante du seuil de classification choisi mais aussi de l'asymétrie qui peut exister entre les deux classes, nous réalisons les courbes ROC (Receiver Operating Characteristic) [22] des différents modèles mentionnés ci dessus en splittant cette fois ci la base de données en 80% pour entraîner le modèle et puis réaliser le ROC à partir des 20% restant. Le score AUC (Area Under the Curve) qui n'est rien d'autre que la surface sous la courbe ROC fournit une mesure quantitative des performances de notre classificateur, avec une valeur de 1 indiquant une classification parfaite et une valeur de 0,5 indiquant une classification qui ne fait pas mieux qu'une estimation aléatoire. Nous présentons sur la figure 5.5 les ROC obtenus par les différentes méthodes.

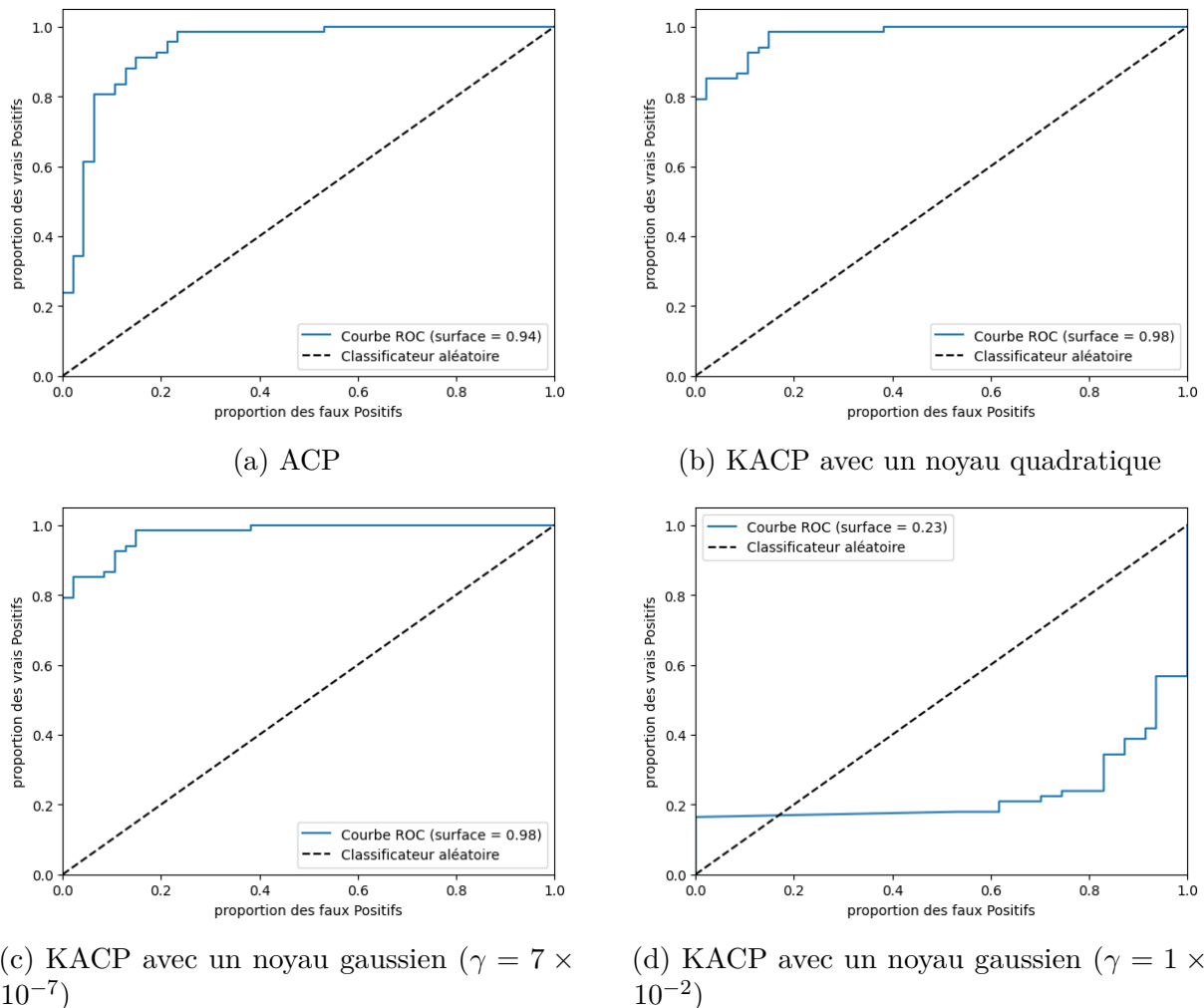


FIGURE 5.5 – Comparaison des Courbes ROC sur la base de données *Breast cancer* en utilisant l'ACP standard et le KACP avec différents noyaux et différentes valeurs de l'hyperparamètre γ .

Avec un AUC de 0.94, l'ACP classique performe moins bien que le KACP utilisant

le noyau quadratique et le noyau gaussien de paramètre $\gamma = 6 \times 10^{-7}$ qui ont tous deux un AUC de 0.98. Avec un AUC de 0.23, le KACP utilisant le noyau gaussien avec un paramètre $\gamma = 1 \times 10^{-2}$ a une performance très médiocre (plus médiocre que même un estimateur aléatoire). Tout ceci est en parfait accord avec les conclusions que nous avons tirées plus haut.

Dans le but d'être juste envers l'ACP et le KACP utilisant le kernel polynomial, nous faisons aussi *un grid search* et trouvons un nombre de CP optimal égal à 1 pour l'ACP et le KACP utilisant un noyau quadratique et un paramètre $\gamma = 7 \times 10^{-7}$ pour ce dernier. Nous présentons dans le tableau 5.5 les résultats avec les paramètres optimaux de chaque méthode/noyau.

Méthode	Noyau	K=5				K=10			
		max(%)	min(%)	moy.(%)	sd	max(%)	min(%)	moy.(%)	sd
ACP		95.61	85.09	91.22	0.041	94.74	82.46	90.69	0.037
KACP	quadratique ($\gamma = 7 \times 10^{-7}$)	93.86	85.09	90.69	0.033	94.73	80.70	90.69	0.043
	gaussien ($\gamma = 7 \times 10^{-7}$)	95.61	86.84	91.57	0.035	94.74	82.46	91.39	0.039

TABLE 5.5 – Scores de prédiction obtenus en faisant une classification en composantes principales (du noyau) sur la base de données breast cancer en utilisant une validation croisée avec $K = 5$ et $K = 10$ échantillons et les paramètres optimaux pour chaque méthode/noyau.

Bien que les résultats de l'ACP et du KACP utilisant le noyau quadratique ont été améliorés, le KACP utilisant le noyau gaussien reste légèrement plus performant que les deux précédemment cités lorsque l'on regarde le score moyen sur les échantillons.

5.2 Application du KACP en Régression

Après avoir montré la capacité de prédiction du KACP dans le domaine de la classification, il est à présent grand temps de tester sa capacité de prédiction dans le domaine de la régression. Pour se faire nous allons nous servir de la base de données *white wine* (provenant du site Kaggle).

Comme dans le cadre de la classification, nous procéderons en deux étapes. Dans un premier temps nous visualiserons les données en les superposant avec un panel de couleurs (variant selon les valeurs continues de la variable à prédire) de la variable réponse, ce qui permettra de repérer éventuellement s'il y a des patterns dans la base de données et par conséquent d'aider à distinguer les clusters susceptibles d'être présents. Dans un second temps nous évaluerons la capacité de prédiction de la variable cible sur le test set lorsqu'une régression linéaire est entraînée sur le training set.

5.2.1 Description de la base de données

La base de données *White wine* compte 4898 observations sur 12 attributs dont l'une (quality) mesurant la qualité du vin sur une échelle de 1 à 9 représente la variable réponse. Les 11 autres variables, à l'instar du PH du vin, de sa densité, de sa teneur en acide sulfurique et bien d'autres représentent des variables explicatives numériques qui donnent

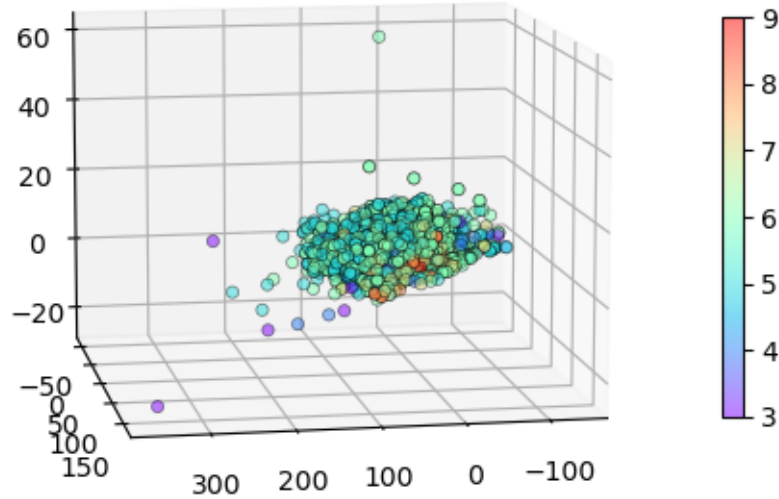


FIGURE 5.6 – Visualisation de la tendance dans la base de données *white wine* à partir de la projection des individus dans l’espace formé par les trois premières composantes principales de l’ACP standard.

quelque peu les caractéristiques du vin étudié. Nous allons effectuer une régression linéaire afin d’essayer d’expliquer la qualité du vin en fonction des composantes principales issues d’un (K)ACP sur les variables explicatives.

5.2.2 Visualisation de la variable réponse en 3D

Dans un premier temps utilisons l’ensemble de la base de données afin de réaliser d’abord une ACP puis un KACP en utilisant différents noyaux. Gardons les 3 premières composantes principales afin de visualiser quelles méthodes/noyaux permettent-ils de desceller une tendance de la variable réponse dont les valeurs sont discriminées par un ensemble de couleurs variant selon les valeurs de cette dernière.

Nous présentons dans la figure 5.6 le résultat donné par l’ACP, puis dans la figure 5.7, celui donné par le KACP en utilisant le noyau quadratique et dans les figures 5.8 et 5.9 ceux donnés par le KACP en utilisant les noyaux gaussiens de paramètres γ respectivement égaux à 0.18 et 0.001.

Contrairement à ce qui avait été observé dans le cadre de la classification, aucune méthode et aucun noyau n’arrive à effectivement montrer un pattern satisfaisant où l’on verrait les vins de haute qualité d’un côté et ceux de basse qualité de l’autre. Néanmoins la méthode du KACP utilisant le noyau gaussien avec un paramètre γ petit semble plus disperser les observations qui sont réparties sous forme d’un rouleau avec les qualités intermédiaires occupant le pourtour et les hautes et basses qualités projetées vers l’intérieur du rouleau. A priori cela ne semble pas distinguer grande chose mais peut toute fois faire

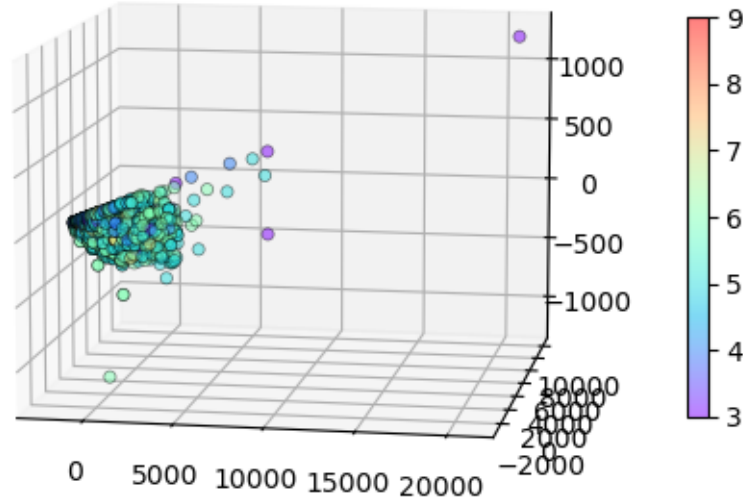


FIGURE 5.7 – Visualisation de la tendance dans la base de données *white wine* à partir de la projection des individus dans l'espace formé par les trois premières composantes principales du utilisant un noyau quadratique.

la différence. Quant au KACP utilisant le noyau gaussien avec un paramètre $\gamma = 0.18$, bien que les vins de toute qualité soient mélangés, l'on observe qu'ils sont répartis le long de 4 axes principaux, faisant peut-être ressortir la tendance d'une autre caractéristique du vin que l'on n'analyse pas en tant que variable réponse.

5.2.3 Pouvoir prédicteur

Nous utiliserons la même approche que dans le cadre de la Classification, c'est à dire la base de donnée sera splittée en $K = 10$ échantillons et une validation croisée sera utilisée pour entraîner et évaluer le modèle. L'erreur quadratique moyenne (MSE) sera utilisée comme critère de performance de notre RCP et RCPN.

Dans un premier temps nous effectuons une régression linéaire en utilisant les variables explicatives d'origine et trouvons une moyenne des MSE (MMSE) sur les 10 échantillons de **0.433**, ce qui représente environ 10% de la valeur moyenne de la variable réponse (5.64).

Puis dans un second temps nous effectuons une RCP en faisant varier le nombre de composantes principales de 1 à 9 et présentons dans le tableau 5.6, les MMSE obtenues pour chaque nombre de composantes principales.

La plus petite MMSE, **0.627**, est obtenue pour un nombre de composantes principales égale à **2**. Cette MMSE est légèrement supérieure à celle obtenue lorsque l'on utilise l'ensemble des variables d'origine comme variables explicatives, ce qui peut paraître logique car la dernière utilise l'ensemble d'information disponible alors que la première n'utilise que 2 variables explicatives. Mais toute fois cette valeur de la MMSE est acceptable au

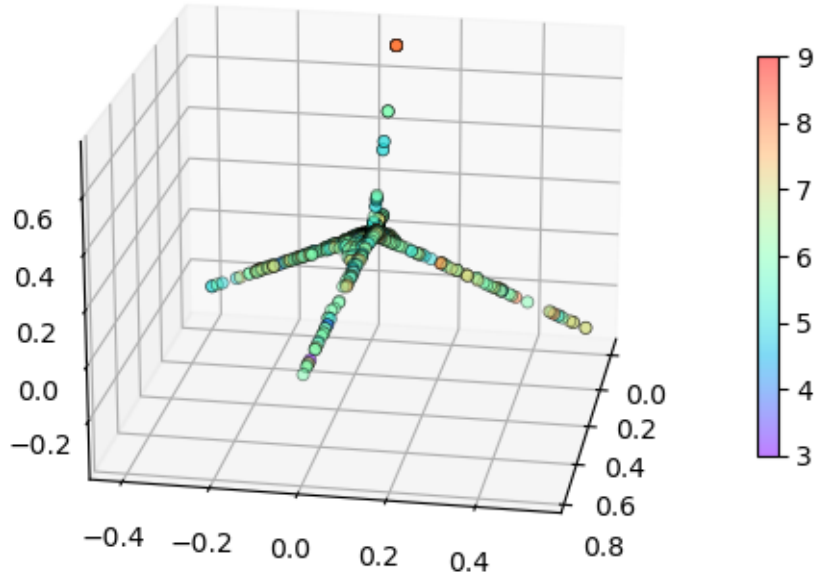


FIGURE 5.8 – Visualisation de la tendance dans la base de données *white wine* à partir de la projection des individus dans l’espace formé par les trois premières composantes principales du KACP utilisant un noyau gaussien de paramètre $\gamma = 0.18$.

nombre des CP	1	2	3	4	5	6	7	8	9
MMSE	0.632	0.627	0.630	0.629	0.741	0.785	0.786	0.790	0.790

TABLE 5.6 – Moyenne des MSE (MMSE) en fonction du nombre de composantes principales en faisant une régression en composantes principales sur la base de données *white wine*.

vue du rapport entre la réduction de dimension considérable de la base de données et la très faible perte en prédiction que l’on a.

Enfin nous effectuons un RCPN utilisant le noyau gaussien, en faisant varier non seulement le nombre de composantes principales de 1 à 9 mais aussi le paramètre γ de 0.000001 à 1000000 par multiple de 10. Nous présentons dans le tableau 5.7, le minimum des MMSE obtenues pour chaque nombre de composantes principales ensemble avec les valeurs correspondantes de γ .

Puis pour chaque nombre de composantes principales, nous faisons varier le paramètre γ autour de la valeur de γ conduisant au minimum des MMSE. Etant donné que c’est la même valeur de γ à savoir 10^{-4} qui conduit au minimum des MMSE pour tous les nombres de composantes principales choisies, nous faisons donc varier γ de 5×10^{-5} à 5×10^{-4} . Nous présentons ainsi dans le tableau 5.8, le minimum des MMSE obtenues pour chaque nombre de composantes principales ensemble avec les valeurs correspondantes de γ .

Nous constatons donc que la plus petite MMSE, **0.616**, est obtenue pour un nombre des composantes principales égale à **3** avec une valeur du paramètre γ de 10^{-4} . Cette

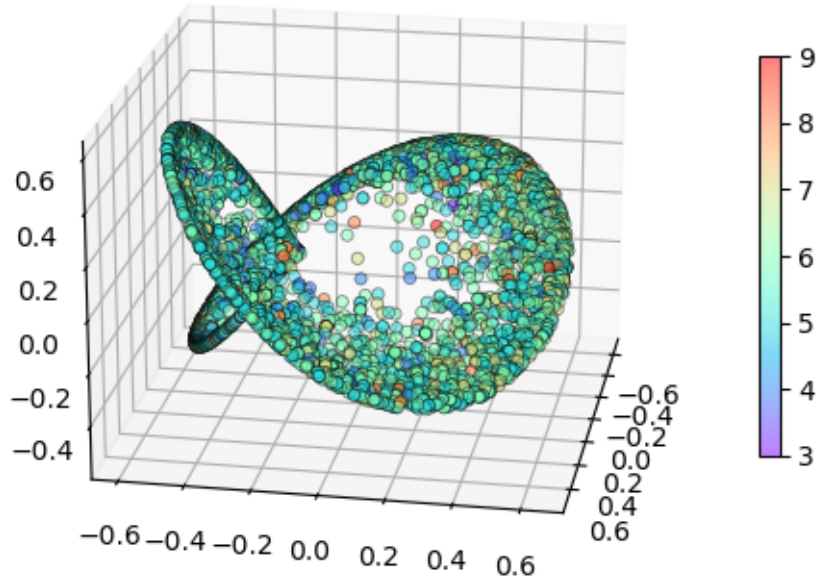


FIGURE 5.9 – Visualisation de la tendance dans la base de données *white wine* à partir de la projection des individus dans l’espace formé par les trois premières composantes principales du KACP en utilisant un noyau gaussien de paramètre $\gamma = 0.001$.

valeur de la MMSE est légèrement plus petite que celle obtenue avec la RCP (**0.627**), ce qui confère donc un pouvoir prédictif légèrement plus élevé pour la RCPN si l’on prend le soin de bien choisir l’hyperparamètre γ . Par ailleurs dans un soucis purement comparatif, si l’on se limitait au même nombre optimal (c’est à dire 2) des composantes principales retenues dans le cadre de la RCP, nous avons une MMSE (obtenue avec la valeur optimale de γ) de **0.626**, ce qui est plus où moins égale à la valeur optimale de la MMSE obtenue dans le cas de la RCP.

Nous refaisons le même exercice mais cette fois en utilisant un kernel quadratique. Nous présentons dans le tableau 5.9, le minimum des MMSE obtenues pour chaque composante principale ensemble avec les valeurs correspondantes de γ .

Nous constatons donc que la plus petite MMSE, **0.620**, est obtenue pour un nombre des composantes principales égale à **4** avec une valeur du paramètre γ de 8×10^{-7} . Une

nombre des CP	1	2	3	4	5	6	7	8	9
γ	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}
MMSE	0.627	0.626	0.616	0.618	0.622	0.636	0.629	0.616	0.616

TABLE 5.7 – Minimum des MMSE (ensemble avec la valeur correspondante du paramètre γ du noyau gaussien) en fonction du nombre de composantes principales en faisant une régression en composantes principales du noyau sur la base de données *white wine*.

nombre des CP	1	2	3	4	5	6	7	8	9
γ	8×10^{-5}	10^{-4}	10^{-4}	10^{-4}	2×10^{-4}	2×10^{-4}	2×10^{-4}	10^{-4}	10^{-4}
MMSE	0.626	0.626	0.616	0.618	0.617	0.616	0.616	0.616	0.616

TABLE 5.8 – Minimum des MMSE (ensemble avec la valeur correspondante du paramètre γ du noyau gaussien) en fonction du nombre de composantes principales en faisant une régression en composantes principales du noyau sur la base de données *white wine*.

nombre des CP	1	2	3	4	5	6	7	8	9
γ	10^{-6}	10^{-7}	3×10^{-7}	8×10^{-7}	8×10^{-7}	10^{-4}	2×10^{-4}	2×10^{-4}	2×10^{-4}
MMSE	0.632	0.627	0.627	0.620	0.634	0.650	0.663	0.663	0.663

TABLE 5.9 – Minimum des MMSE (ensemble avec la valeur correspondante du paramètre γ du noyau quadratique) en fonction du nombre de composantes principales en faisant une régression en composantes principales du noyau sur la base de données *white wine*.

fois de plus, cette valeur de la MMSE est légèrement plus petite que celle obtenue avec la RCP (**0.627**), mais plus grande que celle obtenue avec la RCPN utilisant le noyau gaussien (**0.616**). Comme précédemment, si l'on se limitait au même nombre optimal (c'est à dire 2) des composantes principales retenues dans le cadre de la RCP, nous avons une MMSE (obtenu avec la valeur optimale de γ) de **0.627**, ce qui est plus ou moins égale à la valeur optimale de la MMSE obtenue dans le cas de la RCP.

Contrairement à la régression linéaire utilisant comme prédicteurs les variables d'origine, la RCPN ne permet pas de ressortir le rôle joué par les variables d'origine dans la détermination de la qualité du vin. En effet puisque les variables d'origine sont projetées dans un espace de dimensions plus grands et dans la plupart des cas inconnus, la corrélation entre les composantes principales issues de cet espace et les variables d'origine reste non déterminable. Ce qui par conséquent ne permet pas de ressortir le rôle effectif joué par les différentes variables d'origine dans la prédiction de la qualité du vin. Ceci est d'ailleurs l'un des inconvénients majeurs de cette méthode du KACP car cela rend très difficile voire impossible l'interprétabilité des résultats.

Conclusion

Au terme de ce travail, nous avons dans un premier temps passé en revue la théorie essentielle et les fondements mathématiques permettant une bonne compréhension et une bonne imprégnation de la méthode du KACP. Nous avons montré qu'à partir uniquement de la matrice du noyau et de ses vecteurs propres, nous pouvons déduire certains éléments essentiels permettant d'effectuer une analyse en composantes principales dans l'espace d'attributs, notamment la projection des individus sur les composantes principales dans ledit espace, la qualité de représentation globale et la reconstruction d'erreur dans cet espace.

Dans un second temps, nous avons pu appliquer cette méthode à des données simulées et à des données réelles. A travers les données simulées nous avons montré notamment dans le domaine du clustering que lorsque le kernel et ses hyperparamètres sont judicieusement choisis, le KACP peut (contrairement à l'ACP classique) ressortir la structure non linéaire et complexe cachée dans la base de données. Ce qui permet de mettre ainsi en évidence le véritable potentiel dont dispose cette méthode par rapport à l'ACP standard. A travers les données réelles nous avons montré que le KACP (comme son homologue l'ACP classique) peut trouver son application dans le domaine de la classification et de la régression. Nous avons à travers des exemples concrets fait ressortir le pouvoir prédictif qu'offre cette méthode (par rapport à l'ACP classique) grâce au panel du choix des hyperparamètres dont il jouit.

Toutefois il est à noter que le KACP n'est pas sans inconvénient. Le principal est sans doute la perte de l'information liée aux variables. La mise à l'écart de la fonction de correspondance dont prend avantage les méthodes du noyau, rend inaccessible les coordonnées factorielles des variables, le cercle de corrélation, la qualité de représentation des variables et leurs contributions aux axes principaux. Ces informations sur les variables constituent l'identité même d'une ACP et sont primordiales dans la compréhension et l'interprétation de ses résultats. Outre cela, l'on peut souligner la grande exigence du KACP en termes de calcul pour des grosses bases de données et le choix de la fonction du noyau appropriée qui peut s'avérer difficile et peut souvent requérir une connaissance à priori du domaine de l'expérimentation.

Enfin, nous pensons que ce travail peut dans le futur être enrichi non seulement en mettant sur place une méthode efficace de recherche des noyaux et hyperparamètres adaptés à chaque base de données mais aussi en explorant la méthode du KACP clairsemé mettant en profit les innombrables zéros pouvant figurer dans la matrice du noyau. Pour rappel lorsque l'on a à faire à une base de données qui contient un grand nombre d'observations, les opérations sur la matrice du noyau (dont la dimension est égale au nombre d'observations) peuvent s'avérer fastidieux. Outre cela, nous pensons qu'une piste d'investigation profonde de comment est ce que certains noyaux combinés à des valeurs

spécifiques des hyperparamètres conduisent à une forme assez remarquable du nuage des points issus de la projection des individus dans l'espace engendré par les axes principaux (voir par exemple figure 5.4 et 5.8), pourrait apporter un tonus à ce travail.

Annexe A

Code Python

```
#chapitre 4: donn es simul es

#Simulation 1: Formes en demi-lune

import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, random_state=123)
plt.figure(figsize=(4,4))
plt.scatter(X[y==0, 0], X[y==0, 1],color='red', marker='^', alpha=0.5)
plt.scatter(X[y==1, 0], X[y==1, 1],color='blue', marker='o', alpha=0.5)
plt.ylabel('Y')
plt.xlabel('X')
plt.show()

import numpy as np
from sklearn.decomposition import PCA
scikit_pca = PCA(n_components=2)
X_spca = scikit_pca.fit_transform(X)

plt.figure(figsize=(4,4))
plt.scatter(X_spca[y==0, 0], X_spca[y==0, 1],
            color='red', marker='^', alpha=0.5)
plt.scatter(X_spca[y==1, 0], X_spca[y==1, 1],
            color='blue', marker='o', alpha=0.5)
plt.ylabel('CP2')
plt.xlabel('CP1')
plt.show()

plt.figure(figsize=(4,4))
plt.scatter(X_spca[y==0, 0], np.zeros((50,1)),
            color='red', marker='^', alpha=0.5)
plt.scatter(X_spca[y==1, 0], np.zeros((50,1)),
            color='blue', marker='o', alpha=0.5)
plt.tick_params(left = False, labelleft = False)
plt.xlabel('CP1')
plt.show()
```

```

#reconstruction
X_hat_pca = scikit_pca.inverse_transform(scikit_pca.transform(X))

plt.figure(figsize=(4,4))
plt.scatter(X_hat_pca[y==0, 0],X_hat_pca[y==0, 1] ,
            color='red', marker='^', alpha=0.5)
plt.scatter(X_hat_pca[y==1, 0],X_hat_pca[y==1, 1] ,
            color='blue', marker='o', alpha=0.5)
plt.show()

#PCA
import numpy as np
from sklearn.decomposition import PCA
scikit_pca = PCA(n_components=2)
X_spca = scikit_pca.fit_transform(X)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(7,3))
ax[0].scatter(X_spca[y==0, 0], X_spca[y==0, 1],
              color='red', marker='^', alpha=0.5)
ax[0].scatter(X_spca[y==1, 0], X_spca[y==1, 1],
              color='blue', marker='o', alpha=0.5)
ax[1].scatter(X_spca[y==0, 0], np.zeros((50,1))+0.02,
              color='red', marker='^', alpha=0.5)
ax[1].scatter(X_spca[y==1, 0], np.zeros((50,1))-0.02,
              color='blue', marker='o', alpha=0.5)
ax[0].set_xlabel('CP1')
ax[0].set_ylabel('CP2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('CP1')
plt.show()

#KPCA
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='rbf', gamma=15)
X_kpca = kpca.fit_transform(X)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(7, 3))
ax[0].scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
              color='red', marker='^', alpha=0.5)
ax[0].scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
              color='blue', marker='o', alpha=0.5)
ax[1].scatter(X_kpca[y==0, 0], np.zeros((50,1)),
              color='red', marker='^', alpha=0.5)
ax[1].scatter(X_kpca[y==1, 0], np.zeros((50,1)),
              color='blue', marker='o', alpha=0.5)
ax[0].set_xlabel('CP1')
ax[0].set_ylabel('CP2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('CP1')
plt.tight_layout()
plt.show()

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='rbf', gamma=0.385)

```

```

X_kpca = kpca.fit_transform(X)
plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
            color='red', marker='^', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
            color='blue', marker='o', alpha=0.5)
plt.ylabel('CP2')
plt.xlabel('CP1')
plt.show()

plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], np.zeros((50,1)),
            color='red', marker='^', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], np.zeros((50,1)),
            color='blue', marker='o', alpha=0.5)
plt.tick_params(left = False, labelleft = False)
plt.xlabel('CP1')
plt.show()

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='rbf', gamma=15, fit_inverse_transform=True)
X_kpca = kpca.fit_transform(X)
plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
            color='red', marker='^', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
            color='blue', marker='o', alpha=0.5)
plt.ylabel('CP2')
plt.xlabel('CP1')
plt.show()

plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], np.zeros((50,1)),
            color='red', marker='^', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], np.zeros((50,1)),
            color='blue', marker='o', alpha=0.5)
plt.tick_params(left = False, labelleft = False)
plt.xlabel('CP1')
plt.show()

#reconstruction
X_hat_kpca = kpca.inverse_transform(kpca.transform(X))

plt.figure(figsize=(4,4))
plt.scatter(X_hat_kpca[y==0, 0], X_hat_kpca[y==0, 1] ,
            color='red', marker='^', alpha=0.5)
plt.scatter(X_hat_kpca[y==1, 0], X_hat_kpca[y==1, 1] ,
            color='blue', marker='o', alpha=0.5)
plt.show()

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='linear', gamma=15)
X_kpca = kpca.fit_transform(X)
plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
            color='red', marker='^', alpha=0.5)

```

```

plt.scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
            color='blue', marker='o', alpha=0.5)
plt.ylabel('CP2')
plt.xlabel('CP1')
plt.show()

plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], np.zeros((50,1)),
            color='red', marker='^', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], np.zeros((50,1)),
            color='blue', marker='o', alpha=0.5)
plt.tick_params(left = False, labelleft = False)
plt.xlabel('CP1')
plt.show()

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='poly', degree=2, coef0=1)
X_kpca = kpca.fit_transform(X)

plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
            color='red', marker='^', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
            color='blue', marker='o', alpha=0.5)
plt.ylabel('CP2')
plt.xlabel('CP1')
plt.show()

plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], np.zeros((50,1)),
            color='red', marker='^', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], np.zeros((50,1)),
            color='blue', marker='o', alpha=0.5)
plt.tick_params(left = False, labelleft = False)
plt.xlabel('CP1')
plt.show()

#Simulation 2: Cercles concentriques

from sklearn.datasets import make_circles

X, y = make_circles(n_samples=200, random_state=1,
                    noise=0.05, factor=0.1)

plt.figure(figsize=(4,4))
plt.scatter(X[y==0, 0], X[y==0, 1], color='red', marker='o', alpha=0.5)
plt.scatter(X[y==1, 0], X[y==1, 1], color='blue', marker='^', alpha=0.5)
# plt.title('Concentric circles')
plt.ylabel('Y')
plt.xlabel('X')
plt.show()

```

```

import numpy as np
from sklearn.decomposition import PCA
scikit_pca = PCA(n_components=2)
X_spca = scikit_pca.fit_transform(X)
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(7,3))
ax[0].scatter(X_spca[y==0, 0], X_spca[y==0, 1],
              color='red', marker='o', alpha=0.5)
ax[0].scatter(X_spca[y==1, 0], X_spca[y==1, 1],
              color='blue', marker='^', alpha=0.5)
ax[1].scatter(X_spca[y==0, 0], np.zeros((100,1))+0.02,
              color='red', marker='o', alpha=0.5)
ax[1].scatter(X_spca[y==1, 0], np.zeros((100,1))-0.02,
              color='blue', marker='^', alpha=0.5)
ax[0].set_xlabel('CP1')
ax[0].set_ylabel('CP2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('CP1')
plt.show()

import numpy as np
from sklearn.decomposition import PCA
scikit_pca = PCA(n_components=2)
X_spca = scikit_pca.fit_transform(X)
plt.figure(figsize=(4,4))
plt.scatter(X_spca[y==0, 0], X_spca[y==0, 1],
            color='red', marker='o', alpha=0.5)
plt.scatter(X_spca[y==1, 0], X_spca[y==1, 1],
            color='blue', marker='^', alpha=0.5)
plt.ylabel('CP2')
plt.xlabel('CP1')
plt.show()

plt.figure(figsize=(4,4))
plt.scatter(X_spca[y==0, 0], np.zeros((100,1)),
            color='red', marker='o', alpha=0.5)
plt.scatter(X_spca[y==1, 0], np.zeros((100,1)),
            color='blue', marker='^', alpha=0.5)
plt.tick_params(left = False, labelleft = False)
plt.xlabel('CP1')
plt.show()

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='rbf', gamma=15)
X_kpca = kpca.fit_transform(X)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(7, 3))

ax[0].scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
              color='red', marker='o', alpha=0.5)
ax[0].scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
              color='blue', marker='^', alpha=0.5)
ax[1].scatter(X_kpca[y==0, 0], np.zeros((100, 1)),
              color='red', marker='o', alpha=0.5)
ax[1].scatter(X_kpca[y==1, 0], np.zeros((100, 1)),

```

```

        color='blue', marker='^', alpha=0.5)

ax[0].set_xlabel('CP1')
ax[0].set_ylabel('CP2')
ax[1].set_ylim([-1, 1])
ax[1].set_yticks([])
ax[1].set_xlabel('PC1')

plt.show()

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='linear', gamma=15)
X_kpca = kpca.fit_transform(X)
plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
            color='red', marker='o', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
            color='blue', marker='^', alpha=0.5)
plt.ylabel('CP2')
plt.xlabel('CP1')
plt.show()

plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], np.zeros((100,1)),
            color='red', marker='o', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], np.zeros((100,1)),
            color='blue', marker='^', alpha=0.5)
plt.tick_params(left = False, labelleft = False)
plt.xlabel('CP1')
plt.show()

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='poly', degree=2, coef0=0)
X_kpca = kpca.fit_transform(X)
plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
            color='red', marker='o', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
            color='blue', marker='^', alpha=0.5)
plt.ylabel('CP2')
plt.xlabel('CP1')
plt.show()

plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], np.zeros((100,1)),
            color='red', marker='o', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], np.zeros((100,1)),
            color='blue', marker='^', alpha=0.5)
plt.tick_params(left = False, labelleft = False)
plt.xlabel('CP1')
plt.show()

from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='rbf', gamma=1)
X_kpca = kpca.fit_transform(X)
plt.figure(figsize=(4,4))

```

```
plt.scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
            color='red', marker='o', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
            color='blue', marker='^', alpha=0.5)
plt.ylabel('CP2')
plt.xlabel('CP1')
plt.show()
```

```
plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], np.zeros((100,1)),
            color='red', marker='o', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], np.zeros((100,1)),
            color='blue', marker='^', alpha=0.5)
plt.tick_params(left = False, labelleft = False)
plt.xlabel('CP1')
plt.show()
```

```
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='rbf', gamma=10)
X_kpca = kpca.fit_transform(X)
plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], X_kpca[y==0, 1],
            color='red', marker='o', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], X_kpca[y==1, 1],
            color='blue', marker='^', alpha=0.5)
plt.ylabel('CP2')
plt.xlabel('CP1')
plt.show()
```

```
plt.figure(figsize=(4,4))
plt.scatter(X_kpca[y==0, 0], np.zeros((100,1)),
            color='red', marker='o', alpha=0.5)
plt.scatter(X_kpca[y==1, 0], np.zeros((100,1)),
            color='blue', marker='^', alpha=0.5)
plt.tick_params(left = False, labelleft = False)
plt.xlabel('CP1')
plt.show()
```

Chapitre 5

```
# Classification (Breast Cancer Data)
from sklearn.decomposition import KernelPCA
from sklearn.decomposition import PCA
#from keras.datasets import mnist
from sklearn.datasets import load_breast_cancer
from numpy import reshape
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import KernelPCA
from statistics import mean, stdev
from sklearn.metrics import classification_report, confusion_matrix
```

```

from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
x = cancer.data
y = cancer.target

scikit_pca = PCA(n_components=3)
z = scikit_pca.fit_transform(x)
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(projection='3d')
ax.scatter(z[:,0], z[:,1], z[:,2], c=y, alpha=0.5)
ax.view_init(azim=-130, elev=20)
plt.show()

kpca = KernelPCA(kernel="rbf", n_components=3, gamma=0.0000007)
z = kpca.fit_transform(x)
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(projection='3d')
ax.scatter(z[:,0], z[:,1], z[:,2], c=y, alpha=0.5)
ax.view_init(azim=-20, elev=20)
plt.show()

kpca = KernelPCA(kernel="rbf", n_components=3, gamma=100)
z = kpca.fit_transform(x)
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(projection='3d')
ax.scatter(z[:,0], z[:,1], z[:,2], c=y, alpha=0.5)
ax.view_init(azim=-20, elev=20)
plt.show()

kpca = KernelPCA(kernel="poly", n_components=3, degree=2, gamma=0.001)
z = kpca.fit_transform(x)
fig = plt.figure(figsize=(4,4))
ax = fig.add_subplot(projection='3d')
ax.scatter(z[:,0], z[:,1], z[:,2], c=y, alpha=0.5)
ax.view_init(azim=-140, elev=5)
plt.show()

# Gread Search for CP
skf = StratifiedKFold(n_splits=10)
#k_pca=KernelPCA(n_components=3,kernel='poly', degree=2)
classifier=LogisticRegression(random_state=0)

CP_score_dict = {}
for i in range(10):
    pca=PCA(n_components=i+1)
    lst_accu_stratified = []
    for train_index, test_index in skf.split(x, y):
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        #x_train=k_pca.fit_transform(x_train)
        #x_test=k_pca.fit_transform(x_test)
        x_train=pca.fit_transform(x_train)
        x_test=pca.fit_transform(x_test)

```

```

        classifier.fit(x_train,y_train)

        lst_accu_stratified.append(classifier.score(x_test, y_test))
    CP_score_dict[i+1]= mean(lst_accu_stratified)*100

print(CP_score_dict)

# grid search for gamma rbf.
skf = StratifiedKFold(n_splits=5)
classifier=LogisticRegression(random_state=0)

gamma_val = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001,
              0.00000001]
gamma_score_dict = {}
for i in gamma_val:
    k_pca=KernelPCA(n_components=3,kernel='rbf', gamma= i, degree=2)
    lst_accu_stratified = []
    for train_index, test_index in skf.split(x, y):
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        x_train=k_pca.fit_transform(x_train)
        x_test=k_pca.fit_transform(x_test)
        #x_train=pca.fit_transform(x_train)
        #x_test=pca.fit_transform(x_test)

        classifier.fit(x_train,y_train)

        lst_accu_stratified.append(classifier.score(x_test, y_test))
    gamma_score_dict[i]= mean(lst_accu_stratified)*100

print(gamma_score_dict)

# grid search for gamma.
skf = StratifiedKFold(n_splits=5)
classifier=LogisticRegression(random_state=0)

gamma_val = [9e-06, 8e-06, 7e-06, 6e-06, 5e-06, 4e-06, 3e-06,
              2e-06,1e-06, 9e-07, 8e-07, 7e-07, 6e-07, 5e-07, 4e-07, 3e-07,
              2e-07,1e-07,]
gamma_score_dict = {}
for i in gamma_val:
    k_pca=KernelPCA(n_components=3,kernel='rbf', gamma= i, degree=2)
    lst_accu_stratified = []
    for train_index, test_index in skf.split(x, y):
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        x_train=k_pca.fit_transform(x_train)
        x_test=k_pca.fit_transform(x_test)
        #x_train=pca.fit_transform(x_train)
        #x_test=pca.fit_transform(x_test)

        classifier.fit(x_train,y_train)

```

```

    lst_accu_stratified.append(classifier.score(x_test, y_test))
gamma_score_dict[i]= mean(lst_accu_stratified)*100

print(gamma_score_dict)

# Gread Search for CP
skf = StratifiedKFold(n_splits=5)
#k_pca=KernelPCA(n_components=3,kernel='poly', degree=2)
classifier=LogisticRegression(random_state=0)

CP_score_dict = {}
for i in range(10):
    k_pca=KernelPCA(n_components=i+1,kernel='rbf', gamma= 0.0000007, degree=2)
    lst_accu_stratified = []
    for train_index, test_index in skf.split(x, y):
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        x_train=k_pca.fit_transform(x_train)
        x_test=k_pca.fit_transform(x_test)
        #x_train=pca.fit_transform(x_train)
        #x_test=pca.fit_transform(x_test)

        classifier.fit(x_train,y_train)

        lst_accu_stratified.append(classifier.score(x_test, y_test))
    CP_score_dict[i+1]= mean(lst_accu_stratified)*100

print(CP_score_dict)

# grid search for gamma noyau quadratic.
skf = StratifiedKFold(n_splits=5)
classifier=LogisticRegression(random_state=0)

gamma_val = [0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001,
              0.00000001]
gamma_score_dict = {}
for i in gamma_val:
    k_pca=KernelPCA(n_components=1,kernel='poly', gamma= i, degree=2)
    lst_accu_stratified = []
    for train_index, test_index in skf.split(x, y):
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        x_train=k_pca.fit_transform(x_train)
        x_test=k_pca.fit_transform(x_test)
        #x_train=pca.fit_transform(x_train)
        #x_test=pca.fit_transform(x_test)

        classifier.fit(x_train,y_train)

        lst_accu_stratified.append(classifier.score(x_test, y_test))
    gamma_score_dict[i]= mean(lst_accu_stratified)*100

print(gamma_score_dict)

```

```

# grid search for gamma noyau quadratique.
skf = StratifiedKFold(n_splits=5)
classifier=LogisticRegression(random_state=0)

gamma_val = [9e-06, 8e-06, 7e-06, 6e-06, 5e-06, 4e-06, 3e-06,
              2e-06,1e-06, 9e-07, 8e-07, 7e-07, 6e-07, 5e-07, 4e-07, 3e-07,
              2e-07,1e-07,]
gamma_score_dict = {}
for i in gamma_val:
    k_pca=KernelPCA(n_components=1,kernel='poly', gamma= i, degree=2)
    lst_accu_stratified = []
    for train_index, test_index in skf.split(x, y):
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        x_train=k_pca.fit_transform(x_train)
        x_test=k_pca.fit_transform(x_test)
        #x_train=pca.fit_transform(x_train)
        #x_test=pca.fit_transform(x_test)

        classifier.fit(x_train,y_train)

        lst_accu_stratified.append(classifier.score(x_test, y_test))
    gamma_score_dict[i]= mean(lst_accu_stratified)*100

print(gamma_score_dict)

# Gread Search for CP noyau quadratic
skf = StratifiedKFold(n_splits=5)
#k_pca=KernelPCA(n_components=3,kernel='poly', degree=2)
classifier=LogisticRegression(random_state=0)

CP_score_dict = {}
for i in range(10):
    k_pca=KernelPCA(n_components=i+1,kernel='poly', gamma= 7e-06, degree=2)
    lst_accu_stratified = []
    for train_index, test_index in skf.split(x, y):
        x_train, x_test = x[train_index], x[test_index]
        y_train, y_test = y[train_index], y[test_index]

        x_train=k_pca.fit_transform(x_train)
        x_test=k_pca.fit_transform(x_test)
        #x_train=pca.fit_transform(x_train)
        #x_test=pca.fit_transform(x_test)

        classifier.fit(x_train,y_train)

        lst_accu_stratified.append(classifier.score(x_test, y_test))
    CP_score_dict[i+1]= mean(lst_accu_stratified)*100

print(CP_score_dict)

# Create StratifiedKFold object.
skf = StratifiedKFold(n_splits=10)
k_pca=KernelPCA(n_components=1,kernel='poly', gamma= 7e-06, degree=2)
pca=PCA(n_components=1)

```

```

classifier=LogisticRegression(random_state=0)
lst_accu_stratified = []

for train_index, test_index in skf.split(x, y):
    x_train, x_test = x[train_index], x[test_index]
    y_train, y_test = y[train_index], y[test_index]

    x_train=k_pca.fit_transform(x_train)
    x_test=k_pca.fit_transform(x_test)
    #x_train=pca.fit_transform(x_train)
    #x_test=pca.fit_transform(x_test)

    classifier.fit(x_train,y_train)
    y_test_predict=classifier.predict(x_test)
    cm=confusion_matrix(y_test,y_test_predict)
    print('Confusion matrix:',cm)

    lst_accu_stratified.append(classifier.score(x_test, y_test))

# Print the output.
print('List of possible accuracy:', lst_accu_stratified)
print('\nMaximum Accuracy That can be obtained from this model is:',
      max(lst_accu_stratified)*100, '%')
print('\nMinimum Accuracy:',
      min(lst_accu_stratified)*100, '%')
print('\nOverall Accuracy:',
      mean(lst_accu_stratified)*100, '%')
print('\nStandard Deviation is:', stdev(lst_accu_stratified))

# matrix:
from sklearn.metrics import classification_report, confusion_matrix
cm=confusion_matrix(y_test,y_pred)
cm
lr_train_predict=logistic_regression.predict(x_train_pca)

# Regression

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import scale
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV, Lasso, LassoCV
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA

from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import KFold
from statistics import mean, stdev

df = pd.read_csv('winequality-red.csv', sep=';')
df.shape
df

```

```

target = 'quality'
x = df.drop(target,axis=1)
y = df[target]
mean(y)

# Create StratifiedKFold object.
skf = StratifiedKFold(n_splits=10)
k_pca=KernelPCA(n_components=5,kernel='rbf', gamma=0.05)
pca=PCA(n_components=4)

lst_accu_stratified = []
for train_index, test_index in skf.split(x,y):
    x_train, x_test = x.iloc[train_index.tolist()], x.iloc[test_index.tolist()]
    y_train, y_test = y[train_index], y[test_index]
    #X_train, X_test = scale(x_train), scale(x_test)
    lr = LinearRegression().fit(x_train, y_train)
    pred = lr.predict(x_test)

    lst_accu_stratified.append(mean_squared_error(y_test, pred))
mean(lst_accu_stratified)

# Gread Search for CP
skf = StratifiedKFold(n_splits=10)
CP_MSE_dict = {}
for i in range(10):
    pca=PCA(n_components=i+1)
    MSE = []
    for train_index, test_index in skf.split(x,y):
        x_train, x_test = x.iloc[train_index.tolist()], x.iloc[test_index.tolist()]
        y_train, y_test = y[train_index], y[test_index]
        #X_train_scaled, X_test_scaled = scale(x_train), scale(x_test)
        x_train=pca.fit_transform(x_train)
        x_test=pca.fit_transform(x_test)

        lr = LinearRegression().fit(x_train, y_train)
        pred = lr.predict(x_test)
        MSE.append(mean_squared_error(y_test, pred))

    CP_MSE_dict[i+1]= mean(MSE)
CP_MSE_dict

# Gread Search for CP
skf = StratifiedKFold(n_splits=10)
CP_MSE_dict = {}
for i in range(10):
    kpca = KernelPCA(kernel="rbf",n_components=i+1, gamma=0.18)
    MSE = []
    for train_index, test_index in skf.split(x,y):
        x_train, x_test = x.iloc[train_index.tolist()], x.iloc[test_index.tolist()]
        y_train, y_test = y[train_index], y[test_index]
        #X_train_scaled, X_test_scaled = scale(x_train), scale(x_test)
        x_train=kpca.fit_transform(x_train)
        x_test=kpca.fit_transform(x_test)

        lr = LinearRegression().fit(x_train, y_train)

```

```

        pred = lr.predict(x_test)
        MSE.append(mean_squared_error(y_test, pred))
    CP_MSE_dict[i+1]= mean(MSE)
CP_MSE_dict

# Gread Search for CP
skf = StratifiedKFold(n_splits=10)
CP_MSE_dict = {}
for i in [1000000,100000,10000,1000,100,10,0.1,0.01,0.001,0.0001, 0.00001, 0.000001]:
    kpca = KernelPCA(kernel="rbf",n_components= 10, gamma=i)
    MSE = []
    for train_index, test_index in skf.split(x,y):
        x_train, x_test = x.iloc[train_index.tolist()], x.iloc[test_index.tolist()]
        y_train, y_test = y[train_index], y[test_index]
        #X_train_scaled, X_test_scaled = scale(x_train), scale(x_test)
        x_train=kpca.fit_transform(x_train)
        x_test=kpca.fit_transform(x_test)

        lr = LinearRegression().fit(x_train, y_train)
        pred = lr.predict(x_test)
        MSE.append(mean_squared_error(y_test, pred))
    CP_MSE_dict[i]= mean(MSE)
CP_MSE_dict

# Gread Search for CP
skf = StratifiedKFold(n_splits=10)
CP_MSE_dict = {}
for i in [0.00005, 0.00006, 0.00007, 0.00008,0.00009,0.0001,0.0002,0.0003,0.0004,0.0005]:
    kpca = KernelPCA(kernel="rbf",n_components= 9, gamma=i)
    MSE = []
    for train_index, test_index in skf.split(x,y):
        x_train, x_test = x.iloc[train_index.tolist()], x.iloc[test_index.tolist()]
        y_train, y_test = y[train_index], y[test_index]
        #X_train_scaled, X_test_scaled = scale(x_train), scale(x_test)
        x_train=kpca.fit_transform(x_train)
        x_test=kpca.fit_transform(x_test)

        lr = LinearRegression().fit(x_train, y_train)
        pred = lr.predict(x_test)
        MSE.append(mean_squared_error(y_test, pred))
    CP_MSE_dict[i]= mean(MSE)
CP_MSE_dict

# Gread Search for KACP
skf = StratifiedKFold(n_splits=10)
CP_MSE_dict = {}
for i in [1e-01,1e-02,1e-03,1e-04, 1e-05, 1e-06, 1e-07,1e-08,1e-09,1e-10, 1e-11]:
    kpca = KernelPCA(kernel="poly", degree=2, n_components= 9, gamma=i)
    MSE = []
    for train_index, test_index in skf.split(x,y):
        x_train, x_test = x.iloc[train_index.tolist()], x.iloc[test_index.tolist()]
        y_train, y_test = y[train_index], y[test_index]
        #X_train_scaled, X_test_scaled = scale(x_train), scale(x_test)
        x_train=kpca.fit_transform(x_train)
        x_test=kpca.fit_transform(x_test)

```

```

        lr = LinearRegression().fit(x_train, y_train)
        pred = lr.predict(x_test)
        MSE.append(mean_squared_error(y_test, pred))
    CP_MSE_dict[i]= mean(MSE)
CP_MSE_dict

# Gread Search for CP
skf = StratifiedKFold(n_splits=10)
CP_MSE_dict = {}
for i in [5e-08,6e-08,7e-08,8e-08,8e-08,1e-07,2e-07,3e-07,4e-07, 5e-07]:
    kpca = KernelPCA(kernel="poly", degree=2, n_components= 1, gamma=i)
    MSE = []
    for train_index, test_index in skf.split(x,y):
        x_train, x_test = x.iloc[train_index.tolist()], x.iloc[test_index.tolist()]
        y_train, y_test = y[train_index], y[test_index]
        #X_train_scaled, X_test_scaled = scale(x_train), scale(x_test)
        x_train=kpca.fit_transform(x_train)
        x_test=kpca.fit_transform(x_test)

        lr = LinearRegression().fit(x_train, y_train)
        pred = lr.predict(x_test)
        MSE.append(mean_squared_error(y_test, pred))
    CP_MSE_dict[i]= mean(MSE)
CP_MSE_dict

kpca = KernelPCA(kernel="rbf",n_components=3, gamma=0.18)
x_scale = x#scale(x)
z = kpca.fit_transform(x_scale)
fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(projection='3d')
sc=ax.scatter(z[:,0], z[:,1], z[:,2], c=y, alpha=0.5,cmap=plt.cm.rainbow, edgecolor=)
fig.colorbar(sc, orientation='vertical', shrink=0.45)
ax.view_init(azim=10, elev=20)
plt.show()

kpca = KernelPCA(kernel="rbf",n_components=3, gamma=0.001)
x_scale = x#scale(x)
z = kpca.fit_transform(x_scale)
fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(projection='3d')
sc=ax.scatter(z[:,0], z[:,1], z[:,2], c=y, alpha=0.5,cmap=plt.cm.rainbow, edgecolor=)
fig.colorbar(sc, orientation='vertical', shrink=0.45)
ax.view_init(azim=10, elev=20)
plt.show()

```

Bibliographie

- [1] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11) :559–572, 1901.
- [2] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6) :417, 1933.
- [3] Mohammed J Zaki and Wagner Meira. *Data mining and analysis : fundamental concepts and algorithms*. Cambridge University Press, 2014.
- [4] John A Lee, Michel Verleysen, et al. *Nonlinear dimensionality reduction*, volume 1. Springer, 2007.
- [5] Simon Haykin. *Neural networks and learning machines, 3/E*. Pearson Education India, 2009.
- [6] Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, (3) :326–334, 1965.
- [7] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels : support vector machines, regularization, optimization, and beyond*. MIT press, 2002.
- [8] J Mercer. Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society of London Series A*, 209 :415–446, 1909.
- [9] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5) :1299–1319, 1998.
- [10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20 :273–297, 1995.
- [11] Nello Cristianini and John Shawe-Taylor. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.
- [12] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.
- [13] Isabelle Guyon, B Boser, and Vladimir Vapnik. Automatic capacity tuning of very large vc-dimension classifiers. *Advances in neural information processing systems*, 5, 1992.
- [14] Alex J Smola et al. *Regression estimation with support vector learning machines*. PhD thesis, Master’s thesis, Technische Universität München, 1996.

- [15] Warren S Torgerson. Multidimensional scaling : I. theory and method. *Psychometrika*, 17(4) :401–419, 1952.
- [16] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500) :2319–2323, 2000.
- [17] Christopher Williams. On a connection between kernel pca and metric multidimensional scaling. *Advances in neural information processing systems*, 13, 2000.
- [18] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500) :2323–2326, 2000.
- [19] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems*, 14, 2001.
- [20] Ian T Jolliffe. Principal component analysis. *Technometrics*, 45(3) :276, 2003.
- [21] Chooleewan Dachapak, Shunshoku Kanae, Zi-Jiang Yang, and Kiyoshi Wada. Kernel principal component regression with application to nonlinear prediction. 2003.
- [22] Kelly H Zou, Aiyi Liu, Andriy I Bandos, Lucila Ohno-Machado, and Howard E Rockette. *Statistical evaluation of diagnostic performance : topics in ROC analysis*. CRC press, 2011.

