Université de Montréal

Réduction de dimensionnalité non linéaire et vorace

par
Marie Ouimet

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maîtrise en Informatique

décembre, 2004

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Réduction de dimensionnalité non linéaire et vorace

présenté par:

Marie Ouimet

a été évalué par un jury composé des personnes suivantes:

Pierre Poulin
(président-rapporteur)

Yoshua Bengio
(directeur de recherche)

Balázs Kégl
(membre du jury)

Mémoire accepté le 3 janvier 2005

Sommaire

Les méthodes spectrales de réduction de dimensionnalité et les méthodes de segmentation spectrale exigent le calcul des vecteurs propres principaux d'une matrice de taille $n \times n$ où n est le nombre d'exemples. Des techniques ont été proposées dans la littérature pour accélérer les méthodes à noyau en se concentrant sur un sous-ensemble de m exemples. Nous proposons une procédure vorace pour la sélection de ce sous-ensemble, qui est basée sur la distance dans l'espace des caractéristiques entre un exemple candidat et le sous-espace généré par les exemples précédemment choisis. Dans le cas de l'ACP à noyau ou de la segmentation spectrale, nous obtenons un algorithme en $O(m^2n)$, où $m \ll n$, qui, contrairement aux techniques précédemment proposées, peut se formuler de façon en-ligne. Pour la même complexité en temps, nous pouvons également calculer la projection des exemples non choisis sur le sous-espace engendré par les exemples choisis dans l'espace des caractéristiques. En représentant ainsi les exemples par leur projection nous obtenons une approximation de plus faible rang de la matrice de Gram sur toutes les données. Nous pouvons également borner l'erreur correspondant à cette approximation de la matrice de Gram.

Mots-clés : apprentissage non-supervisé, réduction de dimensionnalité, méthodes spectrales, noyaux, algorithmes voraces.

Summary

Spectral methods for dimensionnality reduction and methods for spectral clustering require the calculation of the principal eigenvectors of a matrix of size $n \times n$ where n is the number of examples. Techniques were proposed in the literature to accelerate kernel methods by focusing on a subset of m examples. We propose a greedy procedure for the selection of this subset, which is based on the distance in the feature space between a candidate example and the subspace spanned by the previously selected examples. In the case of kernel PCA or spectral clustering, we get an algorithm in $O(m^2 n)$, where $m \ll n$, which can be formulated in an online setting. For the same complexity in time, we can also calculate the projection of the non-selected examples on the subspace spanned by the chosen examples in the feature space. By representing the examples by their projection we obtain a lower rank approximation of the Gram matrix on all the data. We can also bound the error corresponding to this approximation of the Gram matrix.

Keywords : unsupervised learning, dimensionnality reduction, spectral methods, kernels, greedy algorithms.

Table des matières

Liste des figures

Liste des tableaux

Liste des algorithmes

À ma famille
et à Simon

Chapitre 1

Introduction

L'objectif de l'apprentissage statistique est de tirer de l'information d'un ensemble de données. Alors que dans l'apprentissage supervisé à chaque exemple est associée une étiquette qu'on essaye de prédire, dans l'apprentissage non-supervisé (Hinton et Sejnowski 1999) on cherche de façon générale à caractériser les données. On se pose des questions comme : Où se répartissent les données dans l'espace ? Quelle distribution aurait pu générer ces données ? Quelles sont les zones de haute densité ? Les données se retrouvent-elles dans des groupements différents ? On peut aussi chercher à représenter les données de façon plus compacte, par des attributs statistiquement indépendants, à éliminer la redondance dans les variables, à découvrir les véritables degrés de liberté ou à débruiter les données. Dans ce mémoire, nous nous intéressons à un domaine de l'apprentissage non-supervisé : l'apprentissage de variétés. On utilise souvent des exemples tirés du domaine de la vision pour présenter ce type de problèmes. Prenons l'exemple d'une caméra numérique qui photographie un objet fixe sous plusieurs angles. Les images ainsi obtenues sont représentées par quelques milliers de pixels alors qu'on pourrait résumer chacune de ces images simplement par trois valeurs correspondant à la position de la caméra. Ces coordonnées constituent les seuls véritables degrés de

liberté dans ce problème (si on suppose qu'à chaque position, la caméra ne peut avoir qu'une seule orientation, soit de viser l'objet bien au centre de son objectif). Dans l'espace de quelques milliers de dimensions où chaque point correspond à une image possible, les images qui correspondent à cet objet fixé se retrouveront sur une variété repliée dont la dimensionnalité intrinsèque est de 3.

Les méthodes spectrales de réduction de dimensionnalité peuvent découvrir de telles variétés. Elles ont fait l'objet ces dernières années d'un grand intérêt et plusieurs variantes ont été développées. Ces techniques sont toutefois coûteuses à la fois en temps et en mémoire. Or, de nos jours, à l'ère de l'informatisation, nous retrouvons de plus en plus de bases de données de très grande taille. Il est donc important que les algorithmes d'apprentissage soient très efficaces. Dans cette optique, nous avons cherché à développer une technique qui, en un espace mémoire et un temps fixé, obtiendrait de meilleurs résultats que les techniques de réduction non-linéaire de dimensionnalité originales.

1.1 Aperçu du mémoire

Ce mémoire se présente comme suit. Dans un premier temps, au chapitre 2, nous cherchons à motiver l'utilisation d'algorithmes spectraux de réduction de dimensionnalité en présentant un problème important en apprentissage statistique : le *fléau de la dimensionnalité*. Ce terme réfère au fait qu'un grand nombre d'algorithmes voient leurs performances se dégrader lorsque les exemples sont représentés par un grand nombre de variables. En effectuant une réduction de dimensionnalité sur les données, nous tentons de conserver l'information pertinente dans un espace beaucoup plus compact. Cette représentation des données peut être utilisée pour améliorer les performances de plusieurs algorithmes.

Au chapitre 3, nous introduisons une méthode de réduction linéaire de dimensionnalité : l'*analyse en composantes principales*. À partir de cet algorithme, nous montrons comment obtenir différents algorithmes de réduction non-linéaire de dimensionnalité. Plusieurs de ces algorithmes peuvent être vus comme des méthodes à noyau avec un noyau adaptatif et dépendant des données (Bengio, Delalleau, Le

Roux, Paiement, Vincent et Ouimet 2004) ; les techniques comme LLE (Roweis et Saul 2000), Isomap (Tenenbaum, de Silva et Langford 2000), l'analyse de composantes principales (ACP) à noyau (Schölkopf, Smola et Müller 1998), le Laplacian Eigenmaps (Belkin et Niyogi 2003) et la segmentation spectrale (Weiss 1999; Ng, Jordan et Weiss 2002) nécessitent toutes une décomposition en vecteurs et valeurs propres d'une matrice de taille $n \times n$, n étant le nombre d'exemples dont on dispose, et peuvent toutes être formulées dans un même cadre.

Ces méthodes de réduction non-linéaire de dimensionnalité prennent toutes un temps d'au moins $O(pn^2)$, où p est le nombre de dimensions voulues, et nécessitent un espace mémoire de taille $O(n^2)$. Dans ce mémoire, nous nous intéressons aux techniques pour rendre ces algorithmes moins coûteux. Nous présentons d'abord quelques techniques d'approximation en $O(m^2n)$ (avec $m \ll n$) qui ont été proposées pour les méthodes à noyau (Smola et Schölkopf 2000; Smola et Bartlett 2001; Williams et Seeger 2001; Harmeling, Ziehe, Kawanabe et Müller 2002; Lawrence, Seeger et Herbrich 2003; Engel, Mannor et Meir 2004), puis au chapitre 4, nous proposons une nouvelle technique plus spécifique aux algorithmes spectraux de réduction de dimensionnalité. Comme dans les méthodes précédemment proposées, l'accélération est obtenue en se concentrant sur un sous-ensemble des exemples choisi en utilisant un algorithme vorace. Nous appelons l'ensemble d'exemples choisis le *dictionnaire*. Le critère utilisé pour le choix vorace est la distance dans l'espace des caractéristiques entre un exemple candidat et sa projection sur le sous-espace généré par les exemples choisis. La projection des exemples hors dictionnaire peut être employée pour approximer la matrice de Gram sur tous les exemples. L'algorithme résultant a un temps de calcul de $O(m^2n)$ et nécessite un espace mémoire de taille $O(m^2)$, où m est la taille du dictionnaire. Nous montrons comment, en utilisant un algorithme généralisé de décomposition en vecteurs et valeurs propres, il est possible de tirer profit du fait que la matrice de Gram soit de plus faible rang. Nous montrons également une manière efficace d'obtenir la normalisation du noyau utilisée dans l'ACP à noyau (normalisation additive) et dans la segmentation spectrale (normalisation divisive). L'algorithme proposé se distingue des autres approches par le fait qu'il puisse se formuler comme un algorithme *en-ligne*. Une autre contribution importante de ce mémoire est de fournir une borne sur

l'erreur d'approximation de la matrice de Gram obtenue par l'algorithme proposé.

Nous avons comparé cet algorithme à d'autres algorithmes de la littérature pour vérifier son efficacité. Nous présentons la technique de comparaison utilisée au chapitre 5. Au chapitre 6, nous décrivons les expériences puis rapportons et analysons les résultats obtenus. Nous montrons par plusieurs expériences que l'approximation vorace fonctionne bien et fonctionne mieux que (a) le même algorithme avec un dictionnaire aléatoirement choisi (mais en projetant tous les exemples sur le sous-espace associé au dictionnaire), et (b) en utilisant seulement m points aléatoires pour estimer les vecteurs propres.

1.2 Notation et vocabulaire

Nous donnons ici quelques indications sur la notation mathématique utilisée dans ce mémoire. Nous donnons également la traduction de termes anglais spécifiques au domaine.

- $^\top$: la transposition d'un vecteur ou d'une matrice
- $A_{i\cdot}$: la i^e ligne de la matrice A
- $A_{\cdot j}$: la j^e colonne de la matrice A
- a_i : le i^e élément du vecteur a
- $\mathbf{0}$: un vecteur composé uniquement de zéros
- $x \cdot y$: le produit scalaire entre x et y
- $K(x,y)$, $\phi(x)$, M : noyau, fonction et matrice de Gram indépendants des données
- $K_D(x,y)$, $\tilde{\phi}(x)$, $\tilde{M}$: noyau, fonction et matrice de Gram dépendants des données
- variable : chacune des dimensions d'un exemple
- espace des caractéristiques : *feature space*
- segmentation : *clustering*
- plongement : *embedding*

CHAPITRE 2

LE FLÉAU DE LA DIMENSIONNALITÉ

Dans la plupart des problèmes pour lesquels on recourt aux techniques d'apprentissage on représente les données par un très grand nombre de variables. Par exemple, on représente en général une image par un vecteur dont chaque élément correspond à l'intensité d'un pixel ; une image minuscule de taille 10×10 pixels en tons de gris correspond donc déjà à un vecteur de dimension 100. Malheureusement, plusieurs algorithmes d'apprentissage ne fonctionnent pas bien en haute dimensionnalité. Ce phénomène est connu sous le nom de "fléau de la dimensionnalité". Dans ce chapitre nous nous intéresserons à quelques-unes de ses manifestations : la perte de signification de la distance euclidienne, l'augmentation de la capacité et la réduction de la densité. Nous énumérerons ensuite des solutions à ce fléau, notamment la réduction de dimensionnalité qui fera l'objet du reste de ce document.

2.1 Manifestations du fléau de la dimensionnalité

2.1.1 Distance euclidienne

Plus la dimensionnalité augmente, moins la distance euclidienne a de sens. En effet, en dimension d, la distance entre les exemples x et y est

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

Par conséquent, plus d est grand, plus les dimensions dans lesquelles les deux exemples sont proches deviennent négligeables par rapport au total de celles dans lesquelles ils sont loins. Ce sont seulement les très petites distances qui sont informatives en haute dimension : on peut alors dire que les exemples sont proches dans toutes les dimensions. Ainsi, les méthodes basées sur la distance entre les exemples comme le *k-plus proches voisins* fonctionnent en faible dimension, mais elles verront leurs performances se dégrader rapidement en haute dimension.

2.1.2 Augmentation de la capacité

La tâche de l'apprentissage statistique consiste à chercher parmi un ensemble de fonctions celles qui modélisent bien le problème à résoudre à partir d'un ensemble fini d'exemples. Dans le cas de l'apprentissage supervisé, où à chaque exemple est associée une valeur réelle, on voudra que la fonction donne la bonne valeur lorsqu'on lui présente un exemple qu'elle n'a encore jamais rencontré, c'est-à-dire qu'elle *généralise* bien. De façon générale, lorsque le nombre de dimensions augmente, la taille de l'ensemble de fonctions possibles grandit exponentiellement. Si on veut conserver la même précision dans l'approximation de notre fonction le nombre d'exemples devra aussi croître exponentiellement (Hastie, Tibshirani et Friedman 2003, p. 24). Si la taille de l'ensemble de fonctions dans lequel on cherche est trop grande par rapport au nombre d'exemples disponibles, on risque de sur-apprendre les exemples et donc de mal généraliser.

2.1.3 Distribution des exemples

Supposons qu'on ait n points répartis uniformément dans un hypercube de dimension d ayant un côté de taille 1. On voudrait placer un autre hypercube à l'intérieur de celui-ci de sorte qu'il couvre une fraction r des exemples. Plus on aura de dimensions, plus le côté c de l'hypercube couvrant cette fraction sera grand. En fait, il est facile de voir qu'il faudra un hypercube de côté $c = r^{1/d}$ pour couvrir une fraction r du volume de l'hypercube de côté 1. Les figures 2.1 et 2.2 illustrent ce phénomène. On remarque que lorsque la dimensionnalité augmente, une plus en plus grande proportion d'exemples sera située près des bords. Ceci est problématique car pour ces exemples, nous devrons extrapoler à partir des exemples du centre, or l'extrapolation est beaucoup moins fiable que l'interpolation.

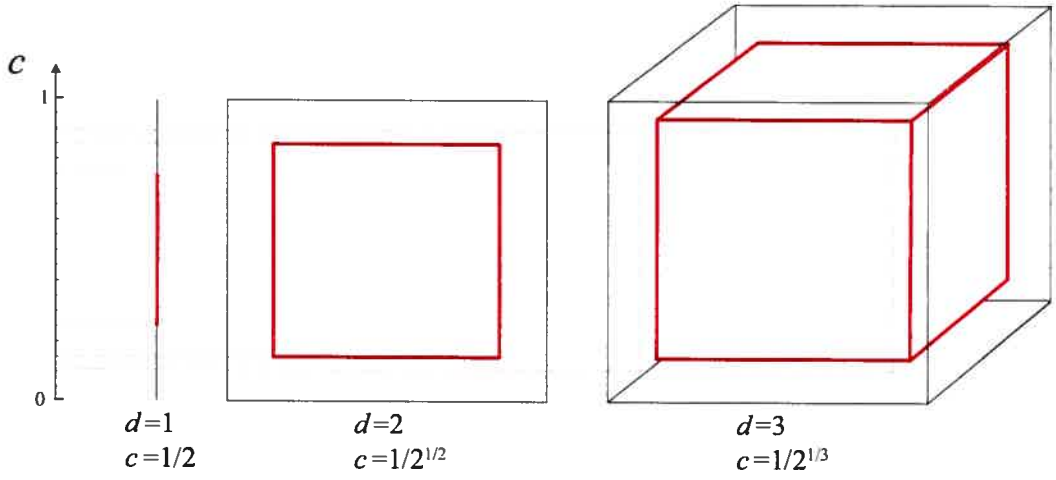

Figure 2.1 – Les trois figures rouges (la ligne, le carré et le cube) ont une longueur, une surface ou un volume correspondant à la moitié des figures noires qui les contiennent. Plus la dimensionnalité augmente, plus le côté de la figure rouge doit grandir pour que la proportion reste 1/2.

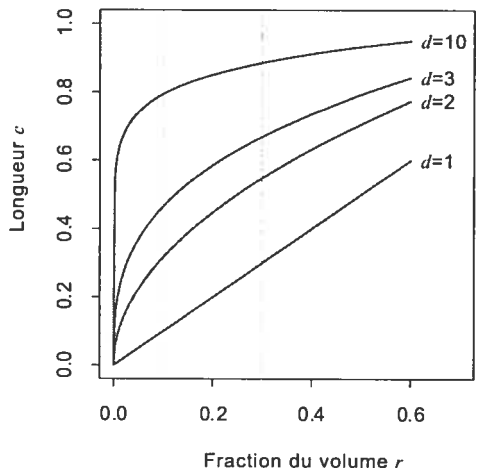

Figure 2.2 – Longueur du côté c du sous-hypercube nécessaire pour occuper une fraction r du volume de l'hypercube unité. Cette longueur augmente rapidement lorsque la dimensionnalité d augmente. *Tiré de (*Hastie, Tibshirani *et* Friedman *2003, p. 23).*

2.2 Vaincre le fléau de la dimensionnalité

Si les données sont réellement réparties uniformément dans l'espace, il n'existe pas de moyen de vaincre le fléau de la dimensionnalité. Cependant, en général ce n'est pas le cas. Les quelques techniques que nous présentons dans cette section vont tirer parti du fait que les données vont se concentrer dans certaines régions de l'espace.

2.2.1 Restreindre la classe de fonctions

À la section 2.1.2 nous avons mentionné qu'une des raisons pour lesquelles les algorithmes d'apprentissage fonctionnent mal en haute dimension est que l'espace des fonctions parmi lesquelles on doit choisir est énorme. Une solution naturelle est de se limiter à certaines classes de fonctions. À titre d'exemple, considérons la classe des surfaces de séparation linéaires dans un problème de classification binaire. En dimension d, pour toute affectation d'étiquettes, il est toujours possible

de séparer linéairement $d + 1$ points non colinéaires. Dans ce cas la taille de notre ensemble d'entraînement ne doit donc croître que linéairement avec le nombre de dimensions. Nous pouvons ainsi vaincre le fléau de la dimensionnalité en imposant des hypothèses fortes quant à la distribution de nos données. Cette façon de procéder est toutefois limitative car dans la plupart des cas nous n'avons pas de connaissances *a priori* sur nos données. De plus, si nous imposons un modèle qui n'est pas le bon, nous introduisons un biais, c'est-à-dire que la véritable fonction qu'on essaie d'approximer ne fait pas partie de la classe dans laquelle nous cherchons, ce qui augmentera notre erreur de généralisation.

2.2.2 La sélection de variables

Une alternative possible est de sélectionner un sous-ensemble de variables (dimensions) et d'ignorer les autres. Nous souhaiterions ne garder que les variables les plus pertinentes à notre problème. Un exemple de telles techniques est le *forward selection*. On débute avec un ensemble de variables vide. On ajoute ensuite une à une les variables qui font le plus diminuer l'erreur de généralisation. On peut trouver un résumé des techniques récentes de sélection de variables dans (Guyon et Elisseeff 2003). Ces approches présentent l'avantage de l'interprétabilité : on découvre quelles variables sont corrélées avec la tâche à apprendre. Toutefois, elles sont en général coûteuses et pas globalement optimales.

2.2.3 La réduction de dimensionnalité

Comme la sélection de variables, cette stratégie vise à exprimer les exemples par un plus petit nombre d'attributs. Toutefois, chacun de ces attributs peut être une fonction non linéaire de toutes les variables des exemples originaux ; par exemple, un produit de variables. De plus, contrairement aux techniques de sélection de variables, les algorithmes de réduction de dimensionnalité sont en général *non supervisés* ; ils se basent simplement sur la distribution des exemples dans l'espace pour déterminer quels attributs sont pertinents. Ils font l'hypothèse que les données se retrouvent sur une variété (pas nécessairement linéaire) de plus faible dimension. L'objectif est de trouver un changement de coordonnées qui fait en sorte qu'on

obtienne les coordonnées le long de la variété. Nous nous pencherons davantage sur ces techniques dans le reste de ce document. Les algorithmes de réduction de dimensionnalité que nous décrirons à la section 3.3 découvrent un tel système de coordonnées en effectuant la décomposition en vecteurs propres et valeurs propres d'une matrice de similarité sur les données. Le résultat ainsi obtenu est globalement optimal au sens où les vecteurs propres sont les vecteurs qui, lorsqu'on projette les données sur le sous-espace généré par ceux-ci, donnent la meilleure approximation possible en terme d'erreur quadratique moyenne (Diamantras et Kung 1996). Notons que ce résultat est optimal pour les données d'entraînement et ne donne aucune garantie sur la généralisation. On sait toutefois qu'asymptotiquement les valeurs propres estimées tendent vers celles du problème continu associé à la véritable densité des données (Shawe-Taylor et Williams 2003).

CHAPITRE 3

MÉTHODES DE RÉDUCTION DE DIMENSIONNALITÉ NON LINÉAIRE

Pour effectuer une réduction de dimensionnalité sur des données, on va supposer que les exemples se trouvent sur une variété de plus faible dimension. On va ensuite chercher à caractériser cette variété. Plus précisément, on souhaiterait trouver un nouveau système de coordonnées dont chacun des axes correspondrait à un déplacement le long d'une des dimensions de la variété. La figure 3.1 illustre cette idée : les données en trois dimensions se trouvent initialement le long d'une spirale. On obtient le nouveau système de cordonnées en deux dimensions en déroulant la spirale.

Pour introduire les algorithmes de réduction de dimensionnalité non linéaire, nous allons d'abord présenter à la section 3.1 un algorithme de réduction de dimensionnalité linéaire très connu : l'*analyse en composantes principales*. À la section 3.2 nous montrerons comment obtenir une version non linéaire de cet algorithme : l'*analyse en composantes principales à noyau*. À la section 3.3 nous montrerons comment nous pouvons reformuler plusieurs techniques de réduction de dimensionnalité non linéaire dans un même cadre que l'ACP à noyau. Nous insisterons davantage sur une de ces méthodes : la *segmentation spectrale*. À la

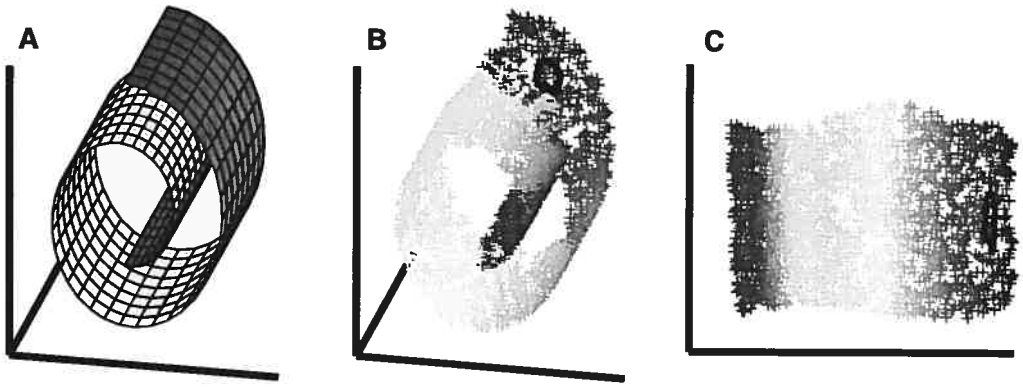

Figure 3.1 – À partir d'une spirale dans un espace en trois dimensions, l'algorithme de réduction de dimensionnalité trouve un système de coordonnées dont la première dimension correspond à un déplacement le long de la spirale du centre vers l'extérieur et la deuxième dimension, à un déplacement de l'avant vers l'arrière. En (A), on représente la variété, en (B), des points tirés de cette variété et en (C), les mêmes points dans le nouveau système de coordonnées. *Tiré de* Roweis *et* Saul *(2000).*

section 3.4 nous attirerons l'attention du lecteur sur le fait que les méthodes de réduction de dimensionnalité non linéaire sont coûteuses en espace mémoire et en temps, les rendant difficilement applicables sur de gros ensembles de données. Nous terminerons ce chapitre en énumérant quelques techniques proposées dans la littérature pour obtenir des approximations moins coûteuses de ces méthodes.

3.1 Réduction linéaire de dimensionnalité

Supposons que les données se retrouvent sur une variété linéaire de plus faible dimension. Pour obtenir le nouveau système de coordonnées, il suffirait de translater la variété de sorte qu'elle passe par l'origine et de prendre comme nouvel espace le sous-espace généré par toutes les données. Cette stratégie ne fonctionne évidemment pas si les points sont bruités, ce qui est le cas pour pratiquement toutes les vraies données. Dans ce cas, l'analyse en composantes principales peut être uti-

lisée pour trouver un système de coordonnées correspondant à la variété linéaire. Ainsi, l'ACP peut être employée non seulement pour faire de la réduction de dimensionnalité, mais pour débruiter les données.

3.1.1 L'analyse en composantes principales

Pour extraire le système de coordonnées correspondant à la variété, on devra être capable de distinguer ce qui représente vraiment une dimension de la variété de ce qui n'est que du bruit. Pour ce faire, on va supposer que les directions où on retrouve seulement une petite variance sont du bruit. Après avoir centré les données autour de l'origine, on va construire la matrice de covariance de ces données. On va ensuite chercher les p plus grandes valeurs propres et les vecteurs propres correspondants. Ces p vecteurs propres seront les axes du nouveau système de coordonnées et nous obtiendrons une représentation des données en plus faible dimension en les projetant sur ces vecteurs. Cette procédure est décrite par l'algorithme 3.1.

Algorithme 3.1 Analyse en composantes principales.

Arguments : ensemble de données $D = \{x_1, \ldots, x_n\}$, nombre de dimensions désirées p

1: $n = |D|$
2: $\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$
3: Construire la matrice $\tilde{X}$ telle que $\tilde{X}_{i\cdot} = x_i - \overline{x},\ i = 1, \ldots, n$
4: $C = \frac{1}{n} \tilde{X}^\top \tilde{X}$
5: Trouver les p plus grandes valeurs propres λ_i de C et les vecteurs propres v_i qui y sont associés
6: Le plongement d'un exemple x est donné par $e_i = (x - \overline{x}) \cdot v_i$ ou par $\frac{e_i}{\sqrt{\lambda_i}}$, $i = 1, \ldots, p$.

Le premier vecteur propre obtenu est la direction dans laquelle les données ont la plus grande variance. La valeur de cette variance est donnée par la valeur propre correspondant à ce vecteur. Le deuxième vecteur est orthogonal au premier et correspond à la deuxième direction principale de variation. En terme d'erreur quadratique, si on projette toutes les données sur les p vecteurs propres principaux,

nous obtenons la meilleure approximation qu'il est possible d'obtenir en projetant sur p vecteurs. La figure 3.2 illustre les deux directions de principale variation de données tirées d'une gaussienne.

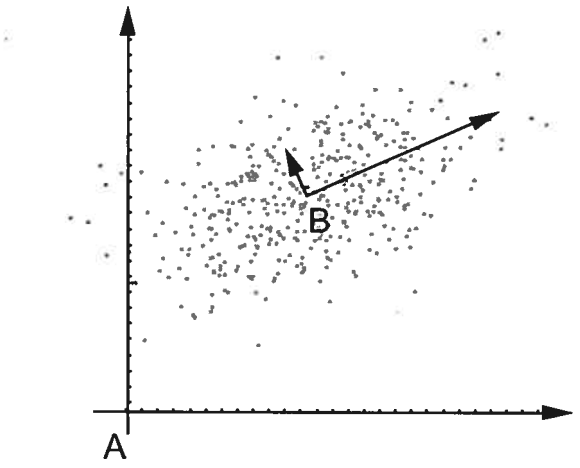

Figure 3.2 – L'ACP appliquée à une gaussienne en deux dimensions. (A) est le système de coordonnées original et (B) correspond aux directions de principale variation des données. Ces directions sont données par les vecteurs propres trouvés par l'ACP et les longueurs des axes, par les valeurs propres et correspondent aux variances dans ces directions.

3.2 L'analyse en composantes principales non linéaire

L'ACP est un algorithme utilisé dans un très grand nombre de domaines. Cependant, en général, les variétés ne sont pas linéaires. Il est alors possible de faire mieux. L'*ACP à noyau* (Schölkopf, Smola et Müller 1999) est une généralisation de l'ACP qui permet de faire de la réduction de dimensionnalité non linéaire. L'idée est de projeter les données dans un espace de beaucoup plus haute dimension de sorte que la variété devienne linéaire et d'effectuer l'ACP dans cet espace. On dénotera la projection par

$$\phi : \mathbb{R}^d \mapsto \mathcal{F}$$

où $\mathcal{F}$ est appelé l'*espace des caractéristiques*. Un exemple de projection est $\phi(x) = (x_1^2, x_1x_2, x_2^2, x_1x_3, x_3^2, x_2x_3)$ qui correspond aux produits de deux variables du vecteur original $x = (x_1, x_2, x_3)$. En général, nous ne connaissons pas le type de projection qui rendrait notre variété linéaire, donc on voudra que $\mathcal{F}$ soit de très haute dimension de sorte à pouvoir contenir plusieurs interactions différentes entre les variables. La figure 3.3 illustre l'idée de l'ACP à noyau.

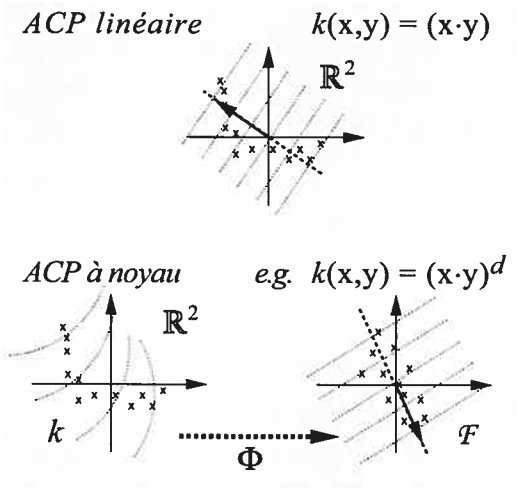

Figure 3.3 – L'ACP classique comparée à l'ACP à noyau. Les lignes pointillées correspondent aux points ayant la même projection sur le premier vecteur propre. Dans le cas de l'ACP à noyau, ces lignes sont des courbes dans $\mathbb{R}^2$. *Tiré de (*Schölkopf, Smola *et* Müller *1998).*

3.2.1 Deuxième version de l'ACP

En étudiant l'algorithme 3.1 on remarque qu'on doit construire une matrice de taille $d \times d$ (étape 4) si on effectue l'ACP dans $\mathbb{R}^d$. Typiquement on veut un $\mathcal{F}$ dont la dimension est beaucoup plus grande que n ; il serait donc plus avantageux d'effectuer les calculs sur une matrice de taille $n \times n$. Il se trouve qu'on peut reformuler l'ACP de sorte qu'on cherche à résoudre le système

$$\frac{1}{n} XX^\top \alpha = \lambda \alpha$$

plutôt que le système

$$\frac{1}{n}X^\top Xv = \lambda v,$$

où X est la matrice dont les n lignes sont les points de notre ensemble de données qu'on suppose centrés pour l'instant, i.e. $\sum_{i=1}^{n} x_i = 0$. Pour reformuler l'ACP de cette façon, on doit d'abord remarquer que v peut être exprimé comme une combinaison linéaire des lignes de X. On a en effet que

$$v = \frac{1}{n\lambda}X^\top Xv = \frac{1}{n\lambda}\sum_{i=1}^{n}(x_i \cdot v)x_i = \sum_{i=1}^{n}\frac{x_i \cdot v}{n\lambda}x_i.$$

Si on nomme α le vecteur tel que $\alpha_i = \frac{x_i \cdot v}{n\lambda}$, $i = 1, \ldots, n$, on a

$$v = \sum_{i=1}^{n}\alpha_i x_i = X^\top \alpha.$$

Par conséquent,

$$\begin{aligned} & \frac{1}{n}X^\top Xv = \lambda v \\ \Rightarrow\ & \frac{1}{n}XX^\top Xv = \lambda Xv \\ \Leftrightarrow\ & \frac{1}{n}XX^\top XX^\top \alpha = \lambda XX^\top \alpha. \end{aligned} \tag{3.1}$$

On peut trouver les solutions de ce problème en résolvant

$$\frac{1}{n}XX^\top \alpha = \lambda \alpha$$

pour $\alpha \neq 0$. Pour obtenir la projection d'un point x sur le vecteur propre v il suffit de faire

$$e = v^\top x = \alpha^\top Xx = \sum_{i=1}^{n}\alpha_i(x_i \cdot x)$$

ou $e/\sqrt{\lambda}$.

3.2.2 L'astuce du noyau

Si nous voulons appliquer cette version de l'ACP dans l'espace des caractéristiques, nous devons remplacer toutes les occurrences de x_i par $\phi(x_i)$ et toutes les occurrences de X par Φ, la matrice telle que $\Phi_{i\cdot} = \phi(x_i)$. La matrice $\Phi\Phi^\top$ est connue sous le nom de *matrice de Gram M* et peut aussi être obtenue en calculant $M_{ij} = \phi(x_i) \cdot \phi(x_j)$ pour $i, j = 1, \ldots, n$. Puisque $\mathcal{F}$ est de très grande dimension, calculer explicitement le produit scalaire $\phi(x_i) \cdot \phi(x_j)$ serait très coûteux. L'astuce du noyau nous permet d'éviter ce problème. Elle consiste à exprimer le produit scalaire par une expression qui se calcule en $O(d)$ au lieu d'en $O(|\mathcal{F}|)$. Pour ce faire, nous introduisons un noyau K tel que $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$. En utilisant un tel noyau, l'ACP à noyau pourra se calculer en $O(pn^2)$. L'ordre d'exécution ne dépend pas de $|\mathcal{F}|$ car les $\phi(x_i)$ ne se retrouvent qu'à l'intérieur de produits scalaires dans cet algorithme. La projection $\phi(x_i)$ n'aura jamais à être calculée ni définie explicitement ; on définira plutôt le noyau K qui implicitement définira une projection ϕ. On appelle noyaux de Mercer la classe de tels noyaux pour lesquels il existe un espace dans lequel ils correspondent à un produit scalaire (Cristianini et Shawe-Taylor 2000; Schölkopf et Smola 2002).

Cette astuce a été utilisée dans de nombreuses méthodes linéaires pour les rendre non linéaires efficacement. Un exemple notoire est la *machine à vecteur de support* (Vapnik 1995) qui trouve une surface de séparation linéaire dans l'espace des caractéristiques. Parmi les noyaux fréquemment rencontrés dans la littérature, nous trouvons :

- le noyau gaussien :

$$K(x, y) = \exp\left(-\frac{\| x - y \|^2}{\sigma^2}\right)$$

- le noyau polynomial :

$$K(x, y) = (x \cdot y + b)^c$$

- le noyau sigmoïdal :

$$K(x, y) = \tanh\left(b(x \cdot y) + c\right).$$

À la figure 3.4, on illustre les différents effets obtenus en utilisant ces noyaux dans

l'ACP à noyau.

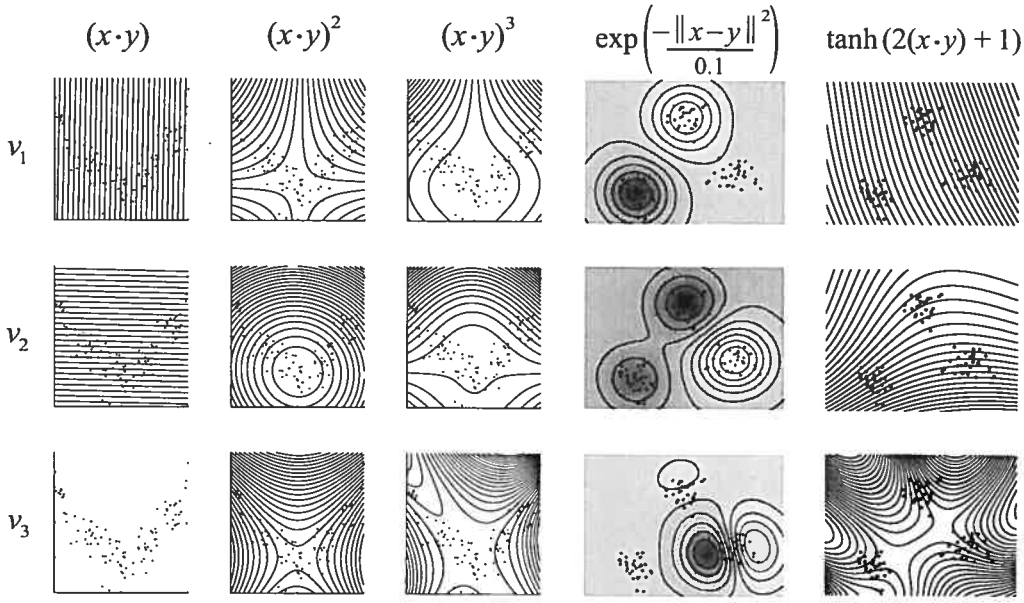

Figure 3.4 – L'effet de différents noyaux sur l'ACP à noyau. On affiche les lignes de projection constante sur les 3 premiers vecteurs propres. Les trois premières colonnes correspondent au noyau $K(x,y) = (x \cdot y)^c$, pour $c = 1, 2, 3$. Dans la quatrième colonne on a utilisé le noyau gaussien et dans la dernière colonne, le noyau sigmoïdal. L'ACP classique, dans la première colonne, ne parvient pas à une bonne réduction de dimensionnalité, tandis qu'avec tous les autres noyaux on parvient à extraire de l'information pertinente. *Images tirées de (*Schölkopf, Smola *et* Müller *1998).*

3.2.3 Centrer les points dans l'espace des caractéristiques

Soit $\phi(x_i)$, les points avant d'avoir été centrés. Nous voulons obtenir les points centrés $\tilde{\phi}(x_i) = \phi(x_i) - E_x[\phi(x)]$, où $E_x[\phi(x)]$ correspond à $\frac{1}{n}\sum_{i=1}^{n}\phi(x_i)$, la moyenne sur l'ensemble de données D. Le noyau centré obtenu à partir d'une telle fonction

$\tilde{\phi}$ sera

$$\begin{aligned} K_D(x_i, x_j) &= \tilde{\phi}(x_i) \cdot \tilde{\phi}(x_j) \\ &= (\phi(x_i) - E_x[\phi(x)]) \cdot (\phi(x_j) - E_x[\phi(x)]) \\ &= K(x_i, x_j) - E_x[K(x_i, x)] - E_y[K(y, x_j)] + E_x[E_y[K(x, y)]]. \end{aligned} \tag{3.2}$$

Montrons que si $K_D(x_i, x_j)$ est donné par cette équation, alors pour tout $\tilde{\phi}(\cdot)$ tel que $K_D(x_i, x_j) = \tilde{\phi}(x_i) \cdot \tilde{\phi}(x_j)$ on a $E_x[\tilde{\phi}(x)] = \mathbf{0}$ sur les données D, où $\mathbf{0}$ est un vecteur ne contenant que des 0.

Preuve Procédons par contradiction. Supposons qu'il existe c tel que $\frac{1}{n}\sum_{i=1}^{n} \tilde{\phi}(x_i)_c \neq 0$. En utilisant cette hypothèse, nous trouvons que

$$\begin{aligned} &\sum_{i=1}^{n}\sum_{j=1}^{n} K_D(x_i, x_j) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} \tilde{\phi}(x_i) \cdot \tilde{\phi}(x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{|\mathcal{F}|} \tilde{\phi}(x_i)_k \tilde{\phi}(x_j)_k \\ &= \sum_{k=1}^{|\mathcal{F}|}\left(\sum_{i=1}^{n} \tilde{\phi}(x_i)_k\right)^2 = \left(\sum_{i=1}^{n} \tilde{\phi}(x_i)_c\right)^2 + \sum_{k=1, k\neq c}^{|\mathcal{F}|}\left(\sum_{i=1}^{n} \tilde{\phi}(x_i)_k\right)^2 \\ &> 0. \end{aligned}$$

Or,

$$\begin{aligned} &\sum_{i=1}^{n}\sum_{j=1}^{n} K_D(x_i, x_j) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\left(K(x_i, x_j) - \frac{1}{n}\sum_{v=1}^{n} K(x_i, x_v) - \frac{1}{n}\sum_{u=1}^{n} K(x_u, x_j) + \frac{1}{n^2}\sum_{u=1}^{n}\sum_{v=1}^{n} K(x_u, x_v)\right) \\ &= \left(\sum_{i=1}^{n}\sum_{j=1}^{n} K(x_i, x_j)\right) - \left(\sum_{i=1}^{n}\sum_{v=1}^{n} K(x_i, x_v)\sum_{j=1}^{n}\frac{1}{n}\right) - \left(\sum_{j=1}^{n}\sum_{u=1}^{n} K(x_u, x_j)\sum_{i=1}^{n}\frac{1}{n}\right) \\ &\quad + \left(\sum_{u=1}^{n}\sum_{v=1}^{n} K(x_u, x_v)\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{n^2}\right) \\ &= 0. \end{aligned}$$

■

Nous appelons K_D le noyau de l'équation 3.2 pour mettre l'emphase sur le fait

qu'il dépend de l'ensemble de données D, contrairement à K. Avec ce noyau nous avons tout ce qu'il faut pour effectuer l'ACP dans l'espace des caractéristiques. L'algorithme 3.2 en résume les étapes. La matrice $\tilde{M}$ de cet algorithme (étape 5) est la matrice de Gram associée au noyau K_D :

$$\tilde{M}_{ij} = K_D(x_i, x_j) = M_{ij} - E_x[M_{ix}] - E_y[M_{yj}] + E_x[E_y[M_{xy}]] \tag{3.3}$$

où $E_x[M_{ix}]$ correspond à $\frac{1}{n}\sum_{j=1}^{n} M_{ij}$. Dans le reste de ce document, nous référerons à ces opérations de centrage (éq. 3.2 et éq. 3.3) par le terme *normalisation additive*.

Algorithme 3.2 Analyse en composantes principales à noyau.

Arguments : ensemble de données $D = \{x_1, \ldots, x_n\}$, nombre de dimensions désirées p, noyau indépendant des données K

1: $n = |D|$
2: Construire la matrice M telle que $M_{ij} = K(x_i, x_j),\ i, j = 1, \ldots, n$
3: Construire le vecteur S tel que $S_i = \frac{1}{n}\sum_{j=1}^{n} M_{ij},\ i = 1, \ldots, n$
4: $t = \frac{1}{n}\sum_{i=1}^{n} S_i$
5: Construire la matrice $\tilde{M}$ telle que $\tilde{M}_{ij} = M_{ij} - S_i - S_j + t$
6: Trouver les p plus grands vecteurs propres v_i et valeurs propres λ_i de $\tilde{M}$.
7: Le plongement d'un exemple x est donné par $\frac{\sqrt{n}}{\sqrt{\lambda_i}}\sum_{j=1}^{n} v_{ij} K_D(x_j, x)$, $i = 1, \ldots, p$, où K_D est donné par l'éq. 3.2.

3.3 Quelques méthodes à noyaux

En définissant différents noyaux nous pouvons exprimer divers algorithmes de réduction de dimensionnalité tels la segmentation spectrale (Weiss 1999; Ng, Jordan et Weiss 2002), le *laplacian eigenmaps* (Belkin et Niyogi 2003), le *multi-dimensional scaling* (Cox et Cox 1994), *isomap* (Tenenbaum, de Silva et Langford 2000) et *LLE* (Roweis et Saul 2000) dans le même cadre que l'ACP à noyau. On peut tous les formuler ainsi :

À partir de notre ensemble de données D et d'un noyau K, on construit

une matrice de Gram M telle que $M_{i,j} = K(x_i, x_j)$. On normalise ensuite M pour obtenir $\tilde{M}$ et on calcule les p plus grandes (ou plus petites) valeurs propres λ_i et les vecteurs propres v_i associés. Le plongement (la projection dans l'espace de plus faible dimension) de $x_j \in D$ est donné par le $j^{\text{ième}}$ élément de chacun des vecteurs propres v_i multiplié par $\sqrt{n}$ (et possiblement multiplié par $\sqrt{\lambda_i}$).

La plupart de ces méthodes ne fournissaient à l'origine qu'un plongement pour les points d'entraînement. En les formulant dans un même cadre comme l'apprentissage des fonctions propres d'un noyau, Bengio, Delalleau, Le Roux, Paiement, Vincent et Ouimet (2004) ont pu donner une formule pour obtenir le plongement d'un point de test. Chacune des p composantes de la projection d'un nouveau point x peut être obtenue par la formule de Nyström :

$$f_p(x) = \frac{\sqrt{n}}{\lambda_p} \sum_{i=1}^{n} v_{ip} K_D(x, x_i) \tag{3.4}$$

ou bien par $\sqrt{\lambda_p} f_p(x)$. Comme dans le cas de l'ACP à noyau, K_D est un noyau dépendant des données obtenu à partir de K qui est défini pour que $K_D(x_i, x_j) = \tilde{M}_{ij}$. Il est facile de montrer que dans le cas des exemples d'entraînement, la formule de Nyström se réduit à $f_p(x_i) = \sqrt{n}\, v_{ip}$.

Nous avons vu qu'en appliquant une normalisation additive sur M, nous obtenions l'ACP à noyau. Si nous utilisons plutôt la normalisation divisive, nous obtenons la segmentation spectrale. À la section suivante nous décrivons plus en détail cet algorithme.

3.3.1 La segmentation spectrale

Même s'il effectue un type de réduction de dimensionnalité, la motivation à l'origine de la segmentation spectrale est plutôt, comme son nom l'indique, la segmentation. Il existe quelques variantes de cet algorithme (Weiss 1999). L'idée qui les a motivées origine de la théorie des graphes : il s'agit de former un graphe en reliant les voisins de chaque point. On voudrait ensuite trouver la coupe à coût minimum de ce graphe nous permettant de séparer le graphe en sous-graphes tel

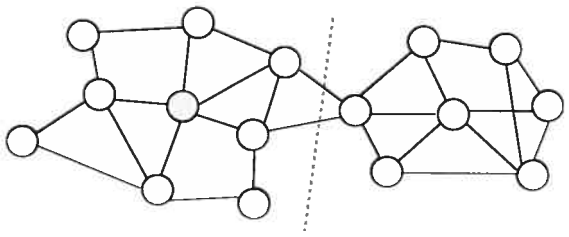

Figure 3.5 – Avec la décomposition en vecteurs propres du Laplacien d'un graphe on peut trouver une coupe qui permet de bien segmenter le graphe.

qu'illustré à la figure 3.5. En prenant les vecteurs propres principaux du Laplacien du graphe, on obtient la solution d'une version relaxée de ce problème (Spielman et Teng 1996; Chung 1997). Pour obtenir la segmentation, on applique l'algorithme des *k-moyennes* sur le plongement donné par ces vecteurs propres (Ng, Jordan et Weiss 2002).

L'algorithme 3.3 décrit les étapes d'une version de la segmentation spectrale. La matrice $\tilde{M}$ qu'on retrouve à l'étape 4 est obtenue en appliquant la *normalisation divisive* (Weiss 1999; Ng, Jordan et Weiss 2002) sur la matrice de Gram M associée au noyau gaussien :

$$\tilde{M}_{ij} = \frac{M_{ij}}{\sqrt{E_x[M_{ix}]E_y[M_{yj}]}}. \tag{3.5}$$

Nous pouvons également définir directement le noyau normalisé K_D qui aurait pu générer $\tilde{M}$:

$$K_D(x_i, x_j) = \frac{K(x_i, x_j)}{\sqrt{E_x[K(x_i, x)]E_x[K(x, x_j)]}}. \tag{3.6}$$

3.4 Limitations de ces méthodes

Nous avons vu que l'ACP à noyau revient à l'ACP ordinaire si on prend le noyau $K(x, y) = x \cdot y$. Toutefois, au niveau algorithmique, les deux techniques sont différentes : alors qu'avec l'ACP à noyau on est contraint de construire une matrice

Algorithme 3.3 La segmentation spectrale telle que présentée dans (Bengio, Delalleau, Le Roux, Paiement, Vincent et Ouimet 2004).

Arguments : ensemble de données $D = \{x_1, \ldots, x_n\}$, nombre de dimensions désirées p, noyau K (souvent le noyau gaussien)

1: $n = |D|$
2: Construire la matrice M telle que $M_{ij} = K(x_i, x_j),\ i, j = 1, \ldots, n$
3: Construire le vecteur S tel que $S_i = \frac{1}{n} \sum_{j=1}^{n} M_{ij},\ i = 1, \ldots, n$
4: Construire la matrice $\tilde{M}$ telle que $\tilde{M}_{ij} = \frac{M_{ij}}{\sqrt{S_i S_j}}$
5: Trouver les p plus grands vecteurs propres v_i et valeurs propres λ_i de $\tilde{M}$.
6: Le plongement d'un exemple x est donné par $\frac{\sqrt{n}}{\lambda_i} \sum_{j=1}^{m} v_{ji} K_D(x, x_j),\ i = 1, \ldots, n$, où K_D est donné par l'éq. 3.6.
7: Pour obtenir la segmentation, appliquer l'algorithme des *k-moyennes* sur le plongement.

$n \times n$ dont on doit calculer la décomposition en vecteurs et valeurs propres, dans l'ACP classique on effectue en général les calculs sur une matrice de taille $d \times d$ puisque la plupart du temps d est plus petit que n. Sachant qu'il en coûte $O(pn^2)$ en temps de calcul pour trouver les p valeurs et vecteurs propres principaux d'une matrice de taille $n \times n$, si n est très grand par rapport à d, l'ACP à noyau sera donc beaucoup plus coûteuse que l'ACP classique. De plus, cette matrice doit être gardée en mémoire. Ces considérations constituent des limitations importantes aux méthodes énumérées à la section 3.3.

3.5 Solutions proposées

Supposons qu'on dispose d'un très grand nombre d'exemples et qu'on veuille obtenir un plongement pour chacun des exemples, que peut-on faire si la matrice de Gram est trop grande pour être gardée en mémoire ou s'il est trop long de calculer sa décomposition ? Avant de décrire la technique que nous proposons pour remédier à ce problème, nous décrivons les techniques proposées dans la littérature. Leurs temps d'exécution sont tous de $O(m^2 n)$, où $m \ll n$.

Dans (Smola et Schölkopf 2000), le cadre est très général. On veut approximer une matrice K par $\tilde{K}$, une matrice de plus faible rang, en minimisant la norme de

Frobenius du résidu $K - \tilde{K}$. On itère sur un sous-ensemble aléatoire des n colonnes du résidu courant en choisissant chaque fois la colonne qui arrive le mieux à approximer les autres colonnes jusqu'à ce qu'on atteigne une erreur inférieure à ϵ sur la norme du résidu, ϵ étant une constante fixée d'avance. Dans la même veine, Smola et Bartlett (2001) et Lawrence, Seeger et Herbrich (2003) proposent des algorithmes pour les processus gaussiens. Encore ici, l'idée est de parcourir m fois un sous-ensemble aléatoire des données en choisissant à chaque fois l'exemple qui réduit le plus un critère de coût. Alors que dans (Smola et Bartlett 2001) on détermine un seuil d'erreur ϵ à atteindre, dans (Lawrence, Seeger et Herbrich 2003), on spécifie directement la valeur de m désirée.

Une approche alternative est d'appliquer l'algorithme de réduction de dimensionnalité sur un sous-ensemble de m exemples et d'utiliser la formule de Nyström pour obtenir le plongement des $n - m$ autres points, suivant les travaux de Bengio, Paiement, Vincent, Delalleau, Le Roux et Ouimet (2004). Cette approche a été employée par Fowlkes, Belongie, Chung et Malik (2004) et Williams et Seeger (2001) dans le but d'accélérer les méthodes à noyau. Williams et Seeger (2001) affirment que bien que leur méthode soit moins précise que celle de Smola et Schölkopf (2000), elle est beaucoup plus rapide. Au chapitre 6 nous comparerons notre technique à celle de Williams et Seeger (2001), montrant que pour le même temps d'exécution, l'erreur de généralisation obtenue avec notre algorithme est moins grande.

Dans l'article de Harmeling, Ziehe, Kawanabe et Müller (2002), on cherche le plus grand sous-ensemble aléatoire d'exemples pour lequel la matrice de Gram est de plein rang et on projette tous les exemples sur le sous-espace généré par ce sous-ensemble dans l'espace des caractéristiques. Cette technique est plus précise que celle de Williams et Seeger (2001) car elle utilise l'information de chaque exemple disponible lors de la décomposition en vecteurs et valeurs propres. Toutefois, bien qu'elle soit aussi en $O(m^2 n)$, elle est assez coûteuse car on doit effectuer un grand nombre de décompositions de matrices afin de calculer leur rang.

La technique que nous proposons utilise aussi la projection des exemples sur un sous-espace généré par un sous-ensemble d'exemples dans l'espace des caractéristiques, mais la principale différence avec les méthodes proposées est qu'on

peut le formuler comme un algorithme en-ligne. Le sous-ensemble de m points formant une matrice de Gram de plein rang est choisi en un seul parcours des données. Le critère utilisé pour ce faire est celui présenté dans (ENGEL, MANNOR et MEIR 2004), alors utilisé dans un contexte supervisé pour faire du moindre carré récursif à noyau. On obtient un algorithme très rapide qui n'occupe qu'un espace mémoire de $O(m^2)$; il est donc possible de l'utiliser pour de très grands ensembles de données.

CHAPITRE 4

MÉTHODE VORACE POUR LA REDUCTION NON LINÉAIRE DE DIMENSIONNALITÉ

Les méthodes spectrales pour la réduction de dimensionnalité étant coûteuses en temps et en mémoire, nous avons développé une version vorace de l'ACP à noyau et de la segmentation spectrale prenant un temps de $O(m^2n)$ et un espace mémoire de taille $O(m^2)$. À la différence des techniques précédemment proposées, elle peut être formulée comme un algorithme en-ligne. Elle convient donc bien pour de très grands ensembles de données et, en permettant d'ajouter des données sans avoir à refaire tous les calculs, elle s'avère utile dans les cas où de nouvelles données sont régulièrement disponibles. À la section 4.1, après avoir décrit en détail la technique, nous montrons qu'on peut borner l'erreur faite sur la matrice de Gram. Nous présentons ensuite une justification intuitive de la méthode à la section 4.3. À la section 4.4, nous décrivons comment obtenir, dans le cadre de cet algorithme, les normalisations additives et divisives de l'ACP à noyau et de la segmentation spectrale présentées au chapitre précédent. Finalement, à la section 4.5 nous analysons la complexité de cet algorithme. Nous donnons également le pseudo-code de différentes variantes de la technique.

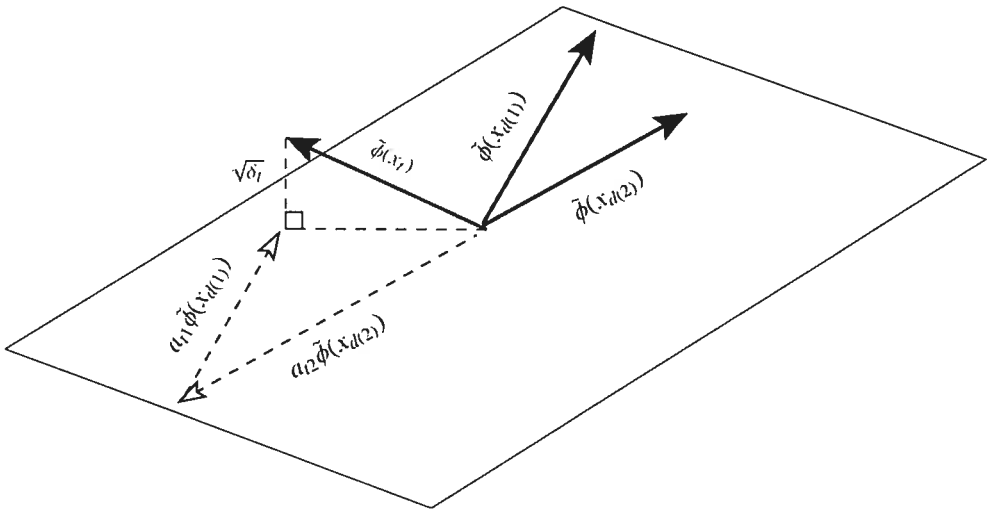

Figure 4.1 – Dans l'espace des caractéristiques, l'exemple hors dictionnaire $\tilde{\phi}(x_t)$ est approximé par une combinaison linéaire des exemples du dictionnaire $\tilde{\phi}(x_{d(i)})$, $i = 1, \ldots, m_{t-1}$ si $\sqrt{\delta_t}$, sa distance à sa projection sur le sous-espace généré par les exemples du dictionnaire, n'est pas trop grande.

4.1 La technique du dictionnaire

Pour développer un algorithme moins coûteux, nous allons utiliser un sous-ensemble des exemples comme prototypes : sur tous les points de notre ensemble de données $D = \{x_1, \ldots, x_n\}$, nous en gardons $m \ll n$ pour former ce que nous appelons un *dictionnaire*. Soit $\mathcal{P}$, le sous-espace généré par les exemples du dictionnaire dans l'espace des caractéristiques. Tous les points de notre ensemble de données seront alors approximés par leur projection sur $\mathcal{P}$. Le dictionnaire est construit de façon itérative, suivant une idée présentée dans (Engel, Mannor et Meir 2004) : on y ajoute un nouveau point si, dans l'espace des caractéristiques, il est loin de sa projection sur $\mathcal{P}$; autrement dit, s'il est mal approximé par sa projection. Cette technique nous permet de trouver une approximation de moindre rang de la matrice de Gram dont nous pourrons trouver les vecteurs et les valeurs propres efficacement.

4.1.1 Mise à jour du dictionnaire

Supposons que nous ayons déjà considéré $t-1$ exemples et que nous disposions d'un dictionnaire $\mathcal{D}_{t-1} = \{x_{d(1)}, \ldots, x_{d(m_{t-1})}\}$. Nous dirons que $\tilde{\phi}(x_t)$ est approximativement linéairement dépendant de $\tilde{\phi}(x_{d(1)}), \ldots, \tilde{\phi}(x_{d(m_{t-1})})$ (donc bien approximé par sa projection sur $\mathcal{P}$) si

$$\delta_t = \min_{a_t} \| \sum_{i=1}^{m_{t-1}} a_{it} \tilde{\phi}(x_{d(i)}) - \tilde{\phi}(x_t) \|^2 \leq \epsilon \tag{4.1}$$

où $a_t = (a_{1t}, \ldots, a_{m_{t-1}t})$ et ϵ est un paramètre contrôlant l'exactitude de la dépendance linéaire.

Soit $\tilde{M}_{t-1}$ la matrice de Gram des m_{t-1} exemples du dictionnaire et soit $k_{t-1}(x)$ le vecteur $(K_D(x, x_{d(1)}), \ldots, K_D(x, x_{d(m_{t-1})}))$. En utilisant l'astuce du noyau, nous trouvons que

$$\begin{aligned}
\| \sum_{i=1}^{m_{t-1}} a_{it} \tilde{\phi}(x_{d(i)}) - \tilde{\phi}(x_t) \|^2 &= \left(\sum_{i=1}^{m_{t-1}} a_{it} \tilde{\phi}(x_{d(i)}) - \tilde{\phi}(x_t) \right) \cdot \left(\sum_{i=1}^{m_{t-1}} a_{it} \tilde{\phi}(x_{d(i)}) - \tilde{\phi}(x_t) \right) \\
&= \sum_{i=1}^{m_{t-1}} \sum_{j=1}^{m_{t-1}} a_{it} a_{jt} \tilde{\phi}(x_{d(i)}) \cdot \tilde{\phi}(x_{d(j)}) - 2 \sum_{i=1}^{m_{t-1}} a_{it} \tilde{\phi}(x_{d(i)}) \cdot \tilde{\phi}(x_t) + \tilde{\phi}(x_t) \cdot \tilde{\phi}(x_t) \\
&= \sum_{i=1}^{m_{t-1}} \sum_{j=1}^{m_{t-1}} a_{it} a_{jt} K_D(x_{d(i)}, x_{d(j)}) - 2 \sum_{i=1}^{m_{t-1}} a_{it} K_D(x_{d(i)}, x_t) + K_D(x_t, x_t) \\
&= a_t^\top \tilde{M}_{t-1} a_t - 2 k_{t-1}(x_t)^\top a_t + K_D(x_t, x_t),
\end{aligned} \tag{4.2}$$

donc

$$\delta_t = \min_{a_t} \left(a_t^\top \tilde{M}_{t-1} a_t - 2 k_{t-1}(x_t)^\top a_t + K_D(x_t, x_t) \right). \tag{4.3}$$

Le vecteur a_t qui minimise 4.3 est trouvé en dérivant :

$$\begin{aligned}
&\frac{d\left(a_t^\top \tilde{M}_{t-1} a_t - 2 k_{t-1}(x_t)^\top a_t + K_D(x_t, x_t) \right)}{da} = 2 a_t^\top \tilde{M}_{t-1} - 2 k_{t-1}(x_t)^\top = \mathbf{0}^\top \\
&\Leftrightarrow a_t = \tilde{M}_{t-1}^{-1} k_{t-1}(x_t)
\end{aligned} \tag{4.4}$$

Avec ce a_t, l'expression de δ_t donnée à l'équation 4.3 se simplifie à

$$\delta_t = K_D(x_t, x_t) - k_{t-1}(x_t)^\top a_t \,. \tag{4.5}$$

Si $\delta_t \leq \epsilon$, l'approximation est bonne ; $\tilde{\phi}(x_t)$ n'est pas trop loin de sa projection sur le sous-espace généré par les exemples du dictionnaire. Le dictionnaire reste donc le même : $\mathcal{D}_t = \mathcal{D}_{t-1}$, $m_t = m_{t-1}$ et $\tilde{M}_t = \tilde{M}_{t-1}$. Par contre, si $\delta_t > \epsilon$, on ajoute x_t au dictionnaire : $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{x_t\}$, $m_t = m_{t-1} + 1$ et

$$\tilde{M}_t = \begin{pmatrix} \tilde{M}_{t-1} & k_{t-1}(x_t) \\ k_{t-1}(x_t)^\top & K_D(x_t, x_t) \end{pmatrix}. \tag{4.6}$$

Pour l'équation 4.4, nous avons aussi besoin de $\tilde{M}_t^{-1}$. Afin d'obtenir un algorithme en $O(m^2 n)$, nous devons pouvoir faire la mise à jour de $\tilde{M}_t^{-1}$ en $O(m^2)$, il n'est donc pas question de recalculer l'inverse à chaque fois. Nous utilisons plutôt l'approche présentée dans (Engel, Mannor et Meir 2004), ce qui nous donne :

$$\tilde{M}_t^{-1} = \frac{1}{\delta_t} \begin{pmatrix} \delta_t \tilde{M}_{t-1}^{-1} + a_t a_t^\top & -a_t \\ -a_t^\top & 1 \end{pmatrix}. \tag{4.7}$$

Pour l'initialisation, il suffit de prendre $\mathcal{D}_1 = \{x_1\}$, $\tilde{M}_1 = (K_D(x_1, x_1))$ et $\tilde{M}_1^{-1} = (1/K_D(x_1, x_1))$.

Preuve Il est facile de prouver ce résultat si on voit la matrice $\tilde{M}_t$ comme une matrice en blocs :

$$\tilde{M}_t = \begin{pmatrix} A & B \\ B^\top & C \end{pmatrix}$$

où $A = \tilde{M}_{t-1}$, $B = k_{t-1}(x_t)$ et $C = K_D(x_t, x_t)$.

Décomposons tout d'abord cette matrice en 3 matrices bloc-triangulaires et bloc-diagonales de la façon suivante :

$$\begin{pmatrix} A & B \\ B^\top & C \end{pmatrix} = \begin{pmatrix} I & 0 \\ B^\top A^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & S_A \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix}$$

où $S_A = (C - B^\top A^{-1} B)$. On peut donc inverser la matrice de gauche en inversant

chacune des trois matrices bloc-triangulaires :

$$\begin{aligned}\begin{pmatrix} A & B \\ B^\top & C \end{pmatrix}^{-1} &= \begin{pmatrix} I & -A^{-1}B \\ 0 & I \end{pmatrix}\begin{pmatrix} A^{-1} & 0 \\ 0 & S_A^{-1} \end{pmatrix}\begin{pmatrix} I & 0 \\ -B^\top A^{-1} & I \end{pmatrix} \\ &= \begin{pmatrix} A^{-1} + A^{-1}BS_A^{-1}B^\top A^{-1} & -A^{-1}BS_A^{-1} \\ -S_A^{-1}B^\top A^{-1} & S_A^{-1} \end{pmatrix}.\end{aligned}$$

Étant donné que $a_t = \tilde{M}_{t-1}^{-1}k_{t-1}(x_t) = A^{-1}B$ et que $\delta_t = K_D(x_t, x_t) - k_{t-1}(x_t)^\top a_t = C - B^\top A^{-1}B = S_A$, nous obtenons

$$\begin{aligned}\tilde{M}_t^{-1} &= \begin{pmatrix} A^{-1} + A^{-1}BS_A^{-1}B^\top A^{-1} & -A^{-1}BS_A^{-1} \\ -S_A^{-1}B^\top A^{-1} & S_A^{-1} \end{pmatrix} \\ &= \begin{pmatrix} \tilde{M}_{t-1}^{-1} + a_t\delta_t^{-1}a_t^\top & -a_t\delta_t^{-1} \\ -\delta_t^{-1}a_t^\top & \delta_t^{-1} \end{pmatrix} \\ &= \delta_t^{-1}\begin{pmatrix} \delta_t\tilde{M}_{t-1}^{-1} + a_t a_t^\top & -a_t \\ -a_t^\top & 1 \end{pmatrix}.\end{aligned}$$

■

4.1.2 Mise à jour des coefficients a_t

Une fois tous les exemples parcourus, nous avons un dictionnaire de taille m_n. Pour alléger la notation, nous utiliserons également m pour désigner m_n. Nous pouvons mettre à jour les coefficients de la combinaison linéaire associés aux exemples qui ne sont pas dans le dictionnaire. Nous construisons la matrice A de taille $n \times m$ dont les lignes sont

$$A_{t\cdot} = a_t^\top = k_n(x_t)^\top \tilde{M}_n^{-1}. \tag{4.8}$$

Ceci donnera des coefficients différents de ceux précédemment calculés à l'équation 4.4 car on utilise la matrice $\tilde{M}_n^{-1}$ correspondant au dictionnaire final et non $\tilde{M}_{t-1}^{-1}$. Autrement dit, l'approximation de $\tilde{\phi}(x_t)$ sera meilleure car au lieu d'être approximé par $\sum_{i=1}^{m_{t-1}} a_{it}\tilde{\phi}(x_{d(i)})$, il sera approximé par $\sum_{i=1}^{m} a_{it}\tilde{\phi}(x_{d(i)})$ (par abus de notation on utilise le symbole a_{it} dans les deux cas même si ces coefficients ne sont pas les mêmes). Cette étape n'est toutefois pas obligatoire ; si nous désirons un al-

gorithme purement en-ligne, nous pouvons nous contenter des a_t déjà calculés pour former la matrice A. Ceci équivaut à obliger les coefficients a_{t+1} à a_m à être 0 pour l'exemple x_t.

4.1.3 L'approximation de la matrice de Gram

En utilisant le fait que $\tilde{\phi}(x_t) \approx \sum_{i=1}^{m} a_{it}\tilde{\phi}(x_{d(i)})$, nous trouvons que la matrice de Gram $\tilde{M}$ sur tous les exemples est approximée par $A\tilde{M}_n A^\top$. En effet,

$$\begin{aligned}
(\tilde{M})_{uv} &= \tilde{\phi}(x_u) \cdot \tilde{\phi}(x_v) \\
&\approx \sum_{i=1}^{m} a_{iu}\tilde{\phi}(x_{d(i)}) \cdot \sum_{j=1}^{m} a_{jv}\tilde{\phi}(x_{d(j)}) \\
&= a_u^\top \tilde{M}_n a_v \\
&= A_{u\cdot}\, \tilde{M}_n\, (A^\top)_{\cdot v}\,.
\end{aligned}$$

Nous souhaiterions donc trouver les vecteurs propres et les valeurs propres de la matrice $A\tilde{M}_n A^\top$, et ce en $O(m^2 n)$. Si nous trouvons les vecteurs propres du problème $\tilde{M}_n A^\top A v = \lambda v$, nous avons ceux de $A\tilde{M}_n A^\top$. En effet,

$$\begin{aligned}
&\tilde{M}_n A^\top A v = \lambda v \\
\Rightarrow\ &A\tilde{M}_n A^\top A v = \lambda A v
\end{aligned}$$

Donc les valeurs propres sont les mêmes pour les deux problèmes et les vecteurs propres de $A\tilde{M}_n A^\top$ sont donnés par Av. Puisque la matrice $\tilde{M}_n A^\top A$ n'est pas symétrique, nous utilisons une décomposition en vecteurs propres et valeurs propres généralisée (voir la section 4.1.4) sur la matrice semi-définie positive $\tilde{M}_n$ et la matrice symétrique $A^\top A$, toutes deux étant de taille $m \times m$. L'algorithme 4.1 résume les étapes de la méthode du dictionnaire.

4.1.4 Décomposition en vecteurs et valeurs propres généralisée

Soit la matrice semi-définie positive P et la matrice réelle et symétrique Q. Soit le problème de décomposition en vecteurs et valeurs propres $PQz = \lambda z$. Étant donné

que le produit des matrices P et Q n'est en général pas symétrique, il s'agit d'un problème de vecteurs et valeurs propres *généralisé* (Gourlay et Watson 1973). Pour résoudre un tel problème, on va le transformer de sorte qu'on puisse le résoudre par une décomposition en vecteurs et valeurs propres ordinaire de forme $Ry = \lambda y$, où R est une matrice semi-définie positive. Si on décompose P à l'aide d'une factorisation de Cholesky, on obtient $P = LL^\top$, où L est une matrice triangulaire inférieure. On a donc

$$\begin{aligned} & PQz = \lambda z \\ \Leftrightarrow\ & LL^\top Qz = \lambda z \\ \Rightarrow\ & L^{-1}LL^\top Q(LL^{-1})z = \lambda L^{-1}z \\ \Leftrightarrow\ & (L^\top QL)(L^{-1}z) = \lambda(L^{-1}z) \end{aligned}$$

Il suffit de résoudre le problème $(L^\top QL)y = \lambda y$ et les vecteurs propres désirés seront donnés par Ly, les valeurs propres restant les mêmes.

À cause de la factorisation de Cholesky, cette méthode prend un temps de $O(m^3)$, m étant le nombre de lignes de la matrice P.

4.2 Qualité de l'approximation

À l'aide du paramètre ϵ contrôlant l'exactitude de la dépendance linéaire, il est aussi possible de borner l'erreur faite sur chaque entrée de la matrice de Gram en l'approximant par $A\tilde{M}_nA^\top$:

Proposition 4.1 *Pour tout* $x_t, x_u \in D$, $|K_D(x_t, x_u) - (A\tilde{M}_nA^\top)_{tu}| \leq \epsilon$.

Preuve Soit $r_t = \tilde{\phi}(x_t) - \sum_{i=1}^m a_{it}\tilde{\phi}(x_{d(i)})$. On remarque que si $x_t \in \mathcal{D}$ alors $\|r_t\| = 0$, sinon, $\|r_t\| = \sqrt{\delta_t}$ et r_t est orthogonal au sous-espace généré par $\tilde{\phi}(x_{d(i)})$, $i = 1, ..., m$. On a donc

$$\begin{aligned} & K_D(x_t, x_u) = \tilde{\phi}(x_t) \cdot \tilde{\phi}(x_u) \\ & = (r_t + \textstyle\sum_{i=1}^m a_{it}\tilde{\phi}(x_{d(i)})) \cdot (r_u + \sum_{i=1}^m a_{iu}\tilde{\phi}(x_{d(i)})) \\ & = r_t \cdot r_u + r_t \cdot \textstyle\sum_{i=1}^m a_{iu}\tilde{\phi}(x_{d(i)}) + r_u \cdot \sum_{i=1}^m a_{it}\tilde{\phi}(x_{d(i)}) + (\sum_{i=1}^m a_{it}\tilde{\phi}(x_{d(i)})) \cdot (\sum_{i=1}^m a_{iu}\tilde{\phi}(x_{d(i)})) \end{aligned}$$

$$= r_t \cdot r_u + (\sum_{i=1}^{m} a_{it}\tilde{\phi}(x_{d(i)})) \cdot (\sum_{i=1}^{m} a_{iu}\tilde{\phi}(x_{d(i)}))$$
$$= r_t \cdot r_u + (A\tilde{M}_n A^\top)_{tu}.$$

Par conséquent, $|K_D(x_t, x_u) - (A\tilde{M}_n A^\top)_{tu}| = |r_t \cdot r_u| \leq \sqrt{\delta_t}\sqrt{\delta_u} \leq \epsilon$. ■

Algorithme 4.1 Algorithme vorace de plongement spectral.

Arguments : ensemble de données ordonnées aléatoirement D, tolérance ϵ, nombre de dimensions désirées p, noyau normalisé K_D donné par l'éq. 3.2 ou l'éq 3.6

1: $n = |D|$
2: initialiser $\mathcal{D} = \{x_1\}$, $\tilde{M}_1 = (K_D(x_1, x_1))$, $\tilde{M}_1^{-1} = \left(K_D(x_1, x_1)^{-1}\right)$, $m_1 = 1$.
3: **for** $t = 2$ **to** n **do**
4: **for** $i = 1$ **to** m_{t-1} **do**
5: calculer $(k_t(x_t))_i = K_D(x_t, x_{d(i)})$
6: **end for**
7: calculer les poids de projection $a_t = \tilde{M}_{t-1}^{-1} k_t(x_t)$
8: calculer l'erreur de projection $\delta_t = K_D(x_t, x_t) - k_{t-1}(x_t)^\top a_t$.
9: **if** $\delta_t > \epsilon$ **then**
10: $m_t = m_{t-1} + 1$, $\mathcal{D} = \mathcal{D} \cup \{x_t\}$
11: $\tilde{M}_t^{-1}$ est calculé suivant l'éq. 4.7.
12: **else**
13: $m_t = m_{t-1}$, $\tilde{M}_t^{-1} = \tilde{M}_{t-1}^{-1}$
14: **end if**
15: **end for**
16: construire la matrice A dont chaque ligne $A_{t\cdot}$ est donnée par l'éq. 4.8, $t = 1, \ldots, n$
17: calculer $B = A^\top A$
18: Trouver les p vecteurs propres principaux u_k et les valeurs propres λ_k du problème généralisé avec la matrice B à gauche et la matrice $\tilde{M}_n^{-1}$ à droite.
19: Les k-ièmes coordonnées des plongements des exemples d'entraînement sont données par $\sqrt{n}v_k = \sqrt{n}Au_k$ ou $\sqrt{\lambda_k n}Au_k$.
20: Le plongement d'un exemple de test x est donné par $f_k(x)$ ou $\sqrt{\lambda_k}f_k(x)$, où $f_k(x)$ est calculé avec l'éq. 3.4 en utilisant $v_k = Au_k$.

4.3 Intuition de la méthode

Pourquoi ne pas simplement appliquer les méthodes spectrales sur un sous-ensemble de points et généraliser avec la formule de Nyström ? La figure 4.2 nous donne une intuition géométrique de la raison pour laquelle notre technique devrait donner de meilleurs résultats. En a), on estime les directions de principale variation de nos données dans l'espace des caractéristiques sur seulement 3 points avec une ACP. On remarque qu'on sur-apprend les données. En b), les gros points représentent les points de l'ensemble de données : les points noirs constituent le dictionnaire et les rouges sont situés à l'extérieur du sous-espace généré par les exemples du dictionnaire. Les petits points noirs représentent la projection des points rouges sur le sous-espace. Dans la technique du dictionnaire, pour obtenir de meilleures directions de principale variation, on tire parti de l'information donnée par tous les points en faisant l'ACP sur la projection des points en plus des 3 points du dictionnaire. C'est ce qu'on voit en c). Même si ces directions ne sont pas exactement celles qu'on aurait trouvées en considérant tous les vrais points, elles s'en rapprochent certainement plus que celles trouvées en a).

4.4 Normalisation du noyau

Comme nous l'avons vu à la section 3.3, nous obtenons différents algorithmes de réduction de dimensionnalité en appliquant une normalisation à la matrice de Gram correspondant à utiliser un noyau dépendant des données. En particulier, la normalisation additive (éq. 3.3) est utilisée pour obtenir l'ACP à noyau et la normalisation divisive (éq. 3.5), pour la segmentation spectrale. Dans cette section nous décrivons comment ces normalisations peuvent être faites dans la cadre de notre algorithme sans que le coût ne dépasse $O(m^2 n)$.

4.4.1 Normaliser avant ou après l'approximation ?

Dans l'ACP à noyau et la segmentation spectrale ordinaires, nous obtenons le même résultat si nous construisons directement la matrice de Gram $\tilde{M}$ telle que

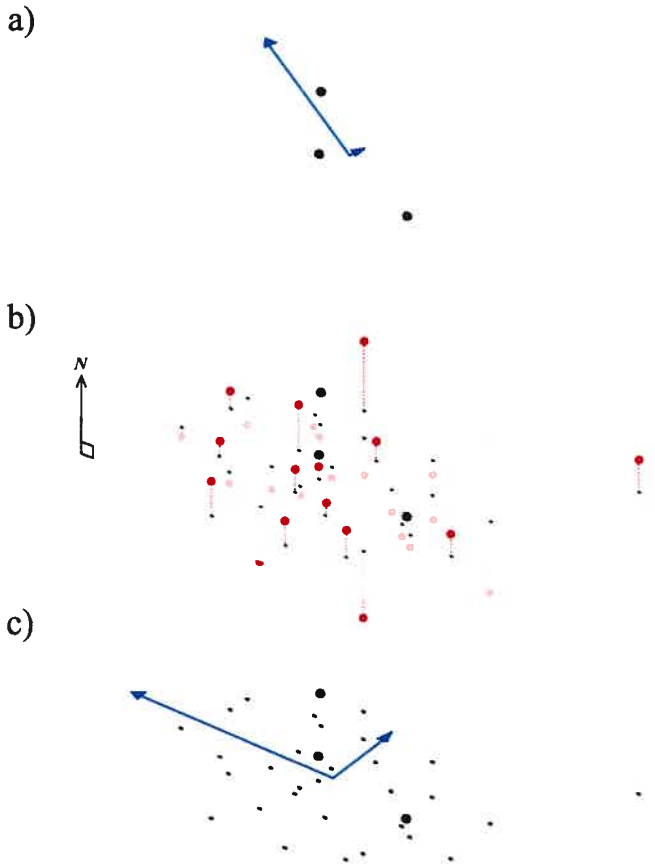

Figure 4.2 – En gris on représente le sous-espace généré par les 3 gros points noirs dans l'espace des caractéristiques (on suppose que ce sous-espace passe par l'origine). Les flèches bleues représentent les directions de principale variation estimées par l'ACP. Dans notre technique, plutôt que d'estimer les directions de principale variation sur un sous-ensemble de points (a), on les estime sur la projection (b) de tous les points sur le sous-espace généré par le sous-ensemble de points (c).

$\tilde{M}_{ij} = K_D(x_i, x_j)$, où $K_D(x_i, x_j)$ est un noyau dépendant des données, ou si nous construisons d'abord la matrice de Gram M à partir d'un noyau indépendant des données pour ensuite obtenir $\tilde{M}$ en normalisant M par rapport à la moyenne de ses lignes. Toutefois, d'un point de vue algorithmique, il est plus avantageux de normaliser la matrice pour ne pas avoir à effectuer de calculs redondants.

Avec la technique du dictionnaire, nous ne calculons jamais explicitement la matrice de Gram M, mais il est quand même possible de faire une transformation qui reviendrait à normaliser $AM_nA^\top$. Toutefois, cette approche n'est pas équivalente à utiliser un noyau déjà normalisé dans l'algorithme 4.1. La différence tient au fait que l'approximation (le calcul des coefficients a_t) n'est pas faite dans le même espace des caractéristiques. En effet, si nous voulons que les coefficients a_t soient optimaux dans l'espace de caractéristique correspondant au noyau de l'ACP à noyau ou à celui de la segmentation spectrale, nous devons utiliser un noyau déjà normalisé (éq. 3.2 ou 3.6) dans l'algorithme 4.1. Toutefois, si dans ces équations nous effectuons les moyennes $E_x[K(x_i, x)] = \frac{1}{n}\sum_{j=1}^{n} K(x_i, x_j)$ sur tous les n exemples, nous nous retrouvons avec un algorithme en $O(n^2m)$, car l'étape 5 de l'algorithme 4.1 est exécutée $O(nm)$ fois. Pour garder un algorithme en $O(m^2n)$ on peut effectuer la normalisation en choisissant un sous-ensemble aléatoire de taille inférieure à m^2 sur lequel on va effectuer les moyennes.

Pour l'ACP à noyau, il existe aussi une façon valable d'effectuer la normalisation après l'approximation, sur la matrice $AM_nA^\top$. Il s'agit d'imposer une contrainte aux coefficients a_t qui rendent la combinaison linéaire invariante à une translation dans l'espace des caractéristiques. Comme la normalisation additive de l'ACP à noyau correspond justement à une translation dans l'espace des caractéristiques, elle peut être effectuée après l'approximation avec la technique décrite à la section suivante.

4.4.2 Normaliser après l'approximation

Les normalisations additive (équation 3.3) et divisive (équation 3.5) peuvent être appliquées sur la matrice implicite $AM_nA^\top$ de façon peu coûteuse, en effectuant une transformation sur la matrice A.

Proposition 4.2 *La normalisation additive de $AM_nA^\top$ peut être obtenue en utili-*

sant $\tilde{A}M_n\tilde{A}^\top$ *au lieu de* $AM_nA^\top$ *où*

$$\tilde{A} = A - \bar{A}, \tag{4.9}$$

$\bar{A}$ *étant une matrice dont les lignes sont toutes égales à* $E_t[A_{t\cdot}]$, *la moyenne des lignes de A. De façon similaire, la normalisation divisive peut être obtenue par* $\tilde{A}M_n\tilde{A}^\top$, *où*

$$\tilde{A}_{i\cdot} = \frac{A_{i\cdot}}{\sqrt{A_{i\cdot}M_nE_t[(A_{t\cdot})^\top]}}. \tag{4.10}$$

Preuve

- Pour la normalisation additive :

$$\begin{aligned}
\tilde{M}_{ij} &\approx (AM_nA^\top)_{ij} - E_x[(AM_nA^\top)_{ix}] - E_y[(AM_nA^\top)_{yj}] \\
&\quad + E_x[E_y[(AM_nA^\top)_{xy}]] \\
&= a_i^\top M_n a_j - a_i^\top M_n E_x[a_x] - E_y[a_y^\top]M_n a_j + E_y[a_y^\top]M_nE_x[a_x] \\
&= A_{i\cdot}M_n(A^\top)_{\cdot j} - A_{i\cdot}M_n(\bar{A}^\top)_{\cdot j} - \bar{A}_{i\cdot}M_n(A^\top)_{\cdot j} + \bar{A}_{i\cdot}M_n(\bar{A}^\top)_{\cdot j} \\
&= \left(AM_nA^\top - AM_n\bar{A}^\top - \bar{A}M_nA^\top + \bar{A}M_n\bar{A}^\top\right)_{ij} \\
&= \left((A-\bar{A})M_n(A^\top - \bar{A}^\top)\right)_{ij} \\
&= \left((A-\bar{A})M_n(A-\bar{A})^\top\right)_{ij}.
\end{aligned}$$

- Pour la normalisation divisive :

$$\begin{aligned}
\tilde{M}_{ij} &\approx \frac{(AM_nA^\top)_{ij}}{\sqrt{E_x[(AM_nA^\top)_{ix}]E_y[(AM_nA^\top)_{yj}]}} \\
&= \frac{a_i^\top M_n a_j}{\sqrt{E_x[(AM_nA^\top)_{ix}]}\sqrt{E_y[(AM_nA^\top)_{yj}]}} \\
&= \frac{a_i^\top}{\sqrt{E_x[(AM_nA^\top)_{ix}]}} M_n \frac{a_j}{\sqrt{E_y[(AM_nA^\top)_{yj}]}} \\
&= \frac{a_i^\top}{\sqrt{a_i^\top M_nE_x[a_x]}} M_n \frac{a_j}{\sqrt{E_y[a_y^\top]M_na_j}} \\
&= \frac{a_i^\top}{\sqrt{a_i^\top M_nE_x[a_x]}} M_n \frac{a_j}{\sqrt{\left(a_j^\top M_nE_y[a_y]\right)^\top}}.
\end{aligned}$$

■

4.4.3 Invariance à la translation dans l'espace des caractéristiques

Le fait d'effectuer la normalisation en modifiant la matrice A permet d'effectuer les moyennes sur tous les points de façon peu coûteuse. Toutefois, comme nous l'avons mentionné, les coefficients a_t ne sont alors pas optimaux dans l'espace des caractéristiques correspondant au noyau de l'ACP à noyau ou de la segmentation spectrale, mais dans un autre espace correspondant au noyau indépendant des données $K(\cdot,\cdot)$. Dans le cas de l'ACP à noyau, pour que les coefficients a_t demeurent optimaux même s'ils sont calculés avant la normalisation, nous introduisons la contrainte $\sum_{i=1}^{m} a_{it} = 1$. Cette contrainte permet de rendre l'approximation invariante à une translation des points dans l'espace de caractéristiques. En effet, $(\phi(x_t) - b) - \sum_{i=1}^{m} a_{it}(\phi(x_{d(i)}) - b) = \phi(x_t) - \sum_{i=1}^{m} a_{it}\phi(x_{d(i)})$.

Avec cette contrainte, nous obtenons une approximation un peu plus restrictive qu'une combinaison linéaire, mais moins restrictive qu'une combinaison convexe, car on ne requiert pas que les coefficients a_t soient non négatifs. En pratique, pour un ϵ donné, l'ajout de cette contrainte n'a pas fait augmenter la taille des dictionnaires de façon significative dans nos expériences.

Nous dénoterons $\hat{a}_t$ les coefficients optimaux qui somment à 1. Sous cette contrainte ils sont donnés par :

$$\hat{a}_t = a_t + \frac{\alpha}{\beta} M_{t-1}^{-1}(1,\ldots,1)^\top \tag{4.11}$$

où a_t désigne les coefficients optimaux sans la contrainte tels que donnés par l'équation 4.4, $\alpha = 1 - \sum_i a_{it}$ et $\beta = \sum_{i,j}(M_{t-1}^{-1})_{ij}$.

Preuve Pour minimiser l'expression 4.2 sous la contrainte $\sum_{i=1}^{m} a_{it} = 1$, nous utilisons la technique du multiplicateur de Lagrange :

$$\mathcal{L}(\hat{a}_t, \ell) = (\hat{a}_t^\top M_{t-1}\hat{a}_t - 2k_{t-1}(x_t)^\top \hat{a}_t + K(x_t, x_t)) + \ell((1,\ldots,1)\hat{a}_t - 1).$$

On pose ensuite les dérivées partielles égales à 0 :

$$\frac{\partial \mathcal{L}}{\partial \hat{a}_t} = 2\hat{a}_t^\top M_{t-1} - 2k_{t-1}(x_t)^\top + \ell(1,\dots,1) = \mathbf{0}^\top$$
$$\frac{\partial \mathcal{L}}{\partial \ell} = (1,\dots,1)\hat{a}_t - 1 = 0.$$

On obtient donc $(1,\dots,1)\hat{a}_t = 1$ et

$$\begin{aligned}
& M_{t-1}\hat{a}_t = k_{t-1}(x_t) - \frac{\ell}{2}(1,\dots,1)^\top \\
\Leftrightarrow\ & \hat{a}_t = M_{t-1}^{-1}k_{t-1}(x_t) - \frac{\ell}{2}M_{t-1}^{-1}(1,\dots,1)^\top \\
\Rightarrow\ & (1,\dots,1)\hat{a}_t = (1,\dots,1)M_{t-1}^{-1}k_{t-1}(x_t) - \frac{\ell}{2}(1,\dots,1)M_{t-1}^{-1}(1,\dots,1)^\top \\
\Leftrightarrow\ & 1 = \sum_{i=1}^{m_{t-1}} a_{it} - \frac{\ell}{2}\sum_{i=1}^{m_{t-1}}\sum_{j=1}^{m_{t-1}}\left(M_{t-1}^{-1}\right)_{ij} \\
\Leftrightarrow\ & -\frac{\ell}{2} = \frac{1 - \sum_{i=1}^{m_{t-1}} a_{it}}{\sum_{i=1}^{m_{t-1}}\sum_{j=1}^{m_{t-1}}\left(M_{t-1}^{-1}\right)_{ij}} \\
\Rightarrow\ & \hat{a}_t = a_t + \frac{\alpha}{\beta}M_{t-1}^{-1}(1,\dots,1)^\top.
\end{aligned}$$

■

La distance à la projection est alors donnée par

$$\begin{aligned}
\hat{\delta}_t &= \hat{a}_t^\top M_{t-1}\hat{a}_t - 2k_{t-1}(x_t)^\top\hat{a}_t + K(x_t,x_t) \\
&= a_t^\top M_{t-1}a_t + \frac{\alpha^2}{\beta^2}(1,\dots,1)M_{t-1}^{-1}M_{t-1}M_{t-1}^{-1}(1,\dots,1)^\top + \frac{\alpha}{\beta}a_t^\top M_{t-1}M_{t-1}^{-1}(1,\dots,1)^\top \\
&\quad + \frac{\alpha}{\beta}(1,\dots,1)M_{t-1}^{-1}M_{t-1}a_t - 2k_{t-1}(x_t)^\top a_t - 2\frac{\alpha}{\beta}a_t^\top(1,\dots,1)^\top + K(x_t,x_t) \\
&= k_{t-1}(x_t)^\top M_{t-1}^{-1}M_{t-1}a_t + \frac{\alpha^2}{\beta^2}\beta + \frac{\alpha}{\beta}a_t^\top(1,\dots,1)^\top \\
&\quad + \frac{\alpha}{\beta}(1,\dots,1)a_t - 2k_{t-1}(x_t)^\top a_t - 2\frac{\alpha}{\beta}a_t^\top(1,\dots,1)^\top + K(x_t,x_t) \\
&= K(x_t,x_t) - k_{t-1}(x_t)^\top a_t + \frac{\alpha^2}{\beta}.
\end{aligned} \tag{4.12}$$

Comme avant, la mise à jour de M_t^{-1} se fait à l'aide des coefficients non contraints

a_t et δ_t car la matrice M_t est restée la même qu'à l'équation 4.6 :

$$M_t^{-1} = \frac{1}{\delta_t}\begin{pmatrix} \delta_t M_{t-1}^{-1} + a_t a_t^\top & -a_t \\ -a_t^\top & 1 \end{pmatrix}. \tag{4.13}$$

Finalement, les lignes de la matrice A sont données par $\hat{a}_t$. On trouve le pseudo-code de cette variante dans l'algorithme 4.2. Le noyau qui y est utilisé est un noyau indépendant des données (par exemple le noyau gaussien). Nous insistons sur le fait que l'approximation n'est optimale que pour l'ACP à noyau, elle ne l'est pas pour la segmentation spectrale. Il est préférable d'utiliser l'algorithme 4.1 pour la segmentation spectrale, où on applique la normalisation avant l'approximation.

4.5 Analyse de la complexité

L'algorithme 4.1 a un temps d'exécution de $O(m^2n)$. La boucle de l'étape 3 prend un temps $O(m^2n)$ en autant que les moyennes pour la normalisation du noyau se fassent sur $O(m)$ points. Les étapes 7 et 11 coûtent chacune $O(m_{t-1})$ à chaque tour de boucle. L'étape 16 coûte $O(m^2n)$. Étant donné que la multiplication $A^\top A$ (étape 17) peut se faire en $O(m^2n)$, le calcul des vecteurs et valeurs propres généralisé (étape 18) se fait en temps $O(m^3)$ (à cause de la factorisation de Cholesky sur M_n) et la multiplication Av_k (étape 19) en temps $O(pmn)$, nous avons donc bien un algorithme en $O(m^2n)$.

L'algorithme 4.2 est très semblable à l'algorithme 4.1 et prend aussi $O(m^2n)$. Il diffère en ce que l'étape 5 prend $O(1)$, l'étape 8, $O(m^2)$ et l'étape 19, $O(mn)$.

Dans les deux cas, l'espace mémoire occupé est $O(m^2)$ pour stocker M_t (qui devient M_n) et B si on ne garde pas A en mémoire, ce qui implique que les a_t soient recalculés pour obtenir les vecteurs propres à l'avant-dernière étape de l'algorithme. Si A est gardée en mémoire, on occupera alors $O(mn)$ d'espace mémoire et on gagnera en temps d'exécution par un facteur constant. Dépendant de la grandeur de n et de l'espace mémoire disponible, l'une ou l'autre version peut s'avérer plus avantageuse.

Même si l'objectif était d'obtenir un algorithme en-ligne, il est aussi possible de

Algorithme 4.2 Algorithme vorace de plongement spectral invariant à la translation dans l'espace des caractéristiques.

Arguments : ensemble de données ordonnées aléatoirement D, tolérance ϵ, nombre de dimensions désirées p, noyau non normalisé K

1: $n = |D|$
2: initialiser $\mathcal{D} = \{x_1\}$, $M_1 = (K(x_1, x_1))$, $M_1^{-1} = \left(K(x_1, x_1)^{-1}\right)$, $m_1 = 1$.
3: **for** $t = 2$ **to** n **do**
4: **for** $i = 1$ **to** m_{t-1} **do**
5: calculer $(k_t(x_t))_i = K(x_t, x_{d(i)})$
6: **end for**
7: calculer le produit matrice-vecteur $a_t = M_{t-1}^{-1} k_t(x_t)$
8: calculer $\alpha = 1 - \sum_i a_{ti}$ et $\beta = \sum_{i,j}(M_{t-1}^{-1})_{ij}$.
9: calculer $\delta_t = K(x_t, x_t) - k_{t-1}(x_t)^\top a_t$.
10: calculer l'erreur de projection $\hat{\delta}_t = \delta_t + \frac{\alpha^2}{\beta}$.
11: **if** $\hat{\delta}_t > \epsilon$ **then**
12: $m_t = m_{t-1} + 1$, $\mathcal{D} = \mathcal{D} \cup \{x_t\}$
13: M_t^{-1} est calculé suivant l'éq. 4.13.
14: **else**
15: $m_t = m_{t-1}$, $M_t^{-1} = M_{t-1}^{-1}$
16: **end if**
17: **end for**
18: Construire la matrice A dont chaque ligne $A_{t\cdot}$ est donnée par l'éq. 4.11 transposée, $t = 1, \ldots, n$
19: Obtenir $\tilde{A}$ à partir de A en effectuant la transformation 4.9 pour l'ACP à noyau ou 4.10 pour la segmentation spectrale
20: Calculer $B = \tilde{A}^\top \tilde{A}$
21: Trouver les p vecteurs propres principaux u_k et les valeurs propres λ_k du problème généralisé avec la matrice B à gauche et la matrice M_n^{-1} à droite.
22: Les k-ièmes coordonnées des plongements des exemples d'entraînement sont données par $\sqrt{n}v_k = \sqrt{n}\tilde{A}u_k$ ou $\sqrt{\lambda_k n}\tilde{A}u_k$.
23: Le plongement d'un exemple de test x est donné par $f_k(x)$ ou $\sqrt{\lambda_k} f_k(x)$, où $f_k(x)$ est calculé avec l'éq. 3.4 en utilisant $v_k = \tilde{A}u_k$.

formuler la méthode dans un cadre similaire à celui des techniques précédemment proposées (voir section 3.5) où au lieu de spécifier le paramètre d'exactitude ϵ, on spécifie m, la taille du sous-ensemble voulue. Pour ce faire, avant d'appliquer l'algorithme 4.1, on doit chercher le ϵ qui donnera le bon m. Ceci peut également être fait en $O(m^2n)$ opérations et en utilisant $O(m^2)$ d'espace mémoire. L'algorithme 4.3 construit un dictionnaire de la taille voulue avec un niveau d'erreur ϵ presque minimal. On commence avec un ϵ donnant un trop petit dictionnaire et un autre en donnant un trop gros. On fait ensuite une recherche binaire pour le bon ϵ en gardant toujours les trop petits dictionnaires obtenus et en y ajoutant des points. De cette manière, on ajoute d'abord les points les plus loins de $\mathcal{P}$ et graduellement on y ajoute des points de plus en plus proches. Ceci a comme effet que l'erreur faite par l'approximation sera près de l'erreur minimale possible avec un dictionnaire de m points. En utilisant ce dictionnaire et les matrices qui lui sont associées, on obtient la réduction de dimensionnalité voulue en exécutant les étapes 16 à 20 de l'algorithme 4.1.

Algorithme 4.3 Construit un dictionnaire de la taille spécifiée m_n avec un niveau d'erreur approximativement minimal.

Arguments : Ensemble de données ordonnés aléatoirement D, taille de dictionnaire désirée m_n, nombre maximal d'itérations `MAX_ITER`.

1: Initialiser $\mathcal{D} = \{x_1\}$, $\tilde{M}_t = (K_D(x_1, x_1))$, $\tilde{M}_t^{-1} = \left(K_D(x_1, x_1)^{-1}\right)$.
2: $\epsilon^+ = 0$ ou une autre valeur telle que $m_n^+ > m_n$
3: $\epsilon^- = 1$ ou une autre valeur telle que $m_n^- < m_n$
4: Vérifier que $m_n^- < m_n$ en exécutant les étapes 3 à 15 de l'algo. 4.1 en utilisant $\epsilon^-, \mathcal{D}, \tilde{M}_t, \tilde{M}_t^{-1}$ et l'ensemble de données $D - \mathcal{D}$, en arrêtant dès que $m_t^- > m_n$. Stocker le dictionnaire obtenu et les matrices correspondantes dans $\mathcal{D}^-, \tilde{M}_t^-, (\tilde{M}_t^-)^{-1}$.
5: **if** $m_t^- = m_n$ **then**
6: **retourner** $\mathcal{D}^-, \tilde{M}_t^-, (\tilde{M}_t^-)^{-1}$
7: **else if** $m_t^- > m_n$ **then**
8: $\epsilon^+ = \epsilon^-$
9: $\epsilon^- = 10\epsilon^-$
10: Aller à l'étape 4
11: **else**
12: $\mathcal{D} = \mathcal{D}^-, \tilde{M}_t = \tilde{M}_t^-, \tilde{M}_t^{-1} = (\tilde{M}_t^-)^{-1}$
13: **end if**
14: **for** $i = 1$ **à** `MAX_ITER` **do**
15: $\epsilon' = \epsilon^+ + (\epsilon^- - \epsilon^+)/2$
16: Exécuter les étapes 3 à 15 de l'algo. 4.1 en utilisant $\epsilon', \mathcal{D}, \tilde{M}_t, \tilde{M}_t^{-1}$ et l'ensemble de données $D - \mathcal{D}$, en arrêtant dès que $m_t' > m_n$. Stocker le dictionnaire obtenu et les matrices correspondantes dans $\mathcal{D}', \tilde{M}_t', (\tilde{M}_t')^{-1}$.
17: **if** $m_t' = m_n$ **then**
18: **retourner** $\mathcal{D}', \tilde{M}_t', (\tilde{M}_t')^{-1}$
19: **else if** $m_t' > m_n$ **then**
20: $\epsilon^+ = \epsilon'$
21: **else**
22: $\epsilon^- = \epsilon', \mathcal{D} = \mathcal{D}', \tilde{M}_t = \tilde{M}_t', \tilde{M}_t^{-1} = (\tilde{M}_t')^{-1}$
23: **end if**
24: **end for**
25: Exécuter les étapes 3 à 15 de l'algo. 4.1 en utilisant $\epsilon^+, \mathcal{D}, \tilde{M}_t, \tilde{M}_t^{-1}$ et l'ensemble de données $D - \mathcal{D}$, en arrêtant dès que $m_t^+ = m_n$. Stocker le dictionnaire obtenu et les matrices correspondantes dans $\mathcal{D}^+, \tilde{M}_t^+, (\tilde{M}_t^+)^{-1}$.
26: **retourner** $\mathcal{D}^+, \tilde{M}_t^+, (\tilde{M}_t^+)^{-1}$

CHAPITRE 5

CADRE EXPÉRIMENTAL

Nous décrivons dans ce chapitre la méthode employée pour évaluer les performances de notre algorithme en comparaison avec d'autres algorithmes. Nous présentons également les données, la version de l'algorithme et les hyper-paramètres que nous avons utilisés.

5.1 Comparer les algorithmes de réduction de dimensionnalité

Étant donné que les algorithmes de réduction de dimensionnalité sont non-supervisés, leur comparaison n'est pas évidente. Quel critère pouvons-nous utiliser pour affirmer qu'un algorithme donne de meilleurs résultats qu'un autre ?

Une possibilité est d'utiliser la représentation obtenue par les algorithmes de réduction de dimensionnalité comme données pour un algorithme supervisé. Ce sont alors les performances de l'algorithme supervisé qui servent à évaluer les algorithmes de réduction de dimensionnalité. Cette façon de procéder peut être justifiée si on s'intéresse à un problème pratique particulier, mais autrement, il serait souhaitable d'avoir une méthode d'évaluation plus directe.

Dans la littérature, les algorithmes de réduction de dimensionnalité sont parfois comparés de façon qualitative en observant les deux ou trois premières dimensions des plongements obtenus. Si cette technique convient lorsqu'on veut illustrer qu'un certain algorithme parvient mieux qu'un autre à capturer certains types de variations, elle convient moins bien lorsqu'on veut comparer des algorithmes qui donnent des résultats qualitativement similaires, comme dans le cas qui nous intéresse.

Nous allons plutôt comparer les plongements de façon quantitative. Nous procédons de la façon suivante : soit $\mathcal{A}$, un algorithme de réduction de dimensionnalité et soit $\mathcal{A}^1$ et $\mathcal{A}^2$, deux algorithmes qui tentent d'approximer $\mathcal{A}$ de façon moins coûteuse. Nous souhaitons déterminer lequel des algorithmes $\mathcal{A}^1$ et $\mathcal{A}^2$ donne les résultats les plus similaires à ceux de $\mathcal{A}$. Pour ce faire, nous allons diviser notre ensemble de données D en trois parties : D_1, D_2, D_3, où D_2 et D_3 sont grands. D_1 est un ensemble d'entraînement, D_2 un ensemble de test et D_3 est un ensemble de données utilisé uniquement à titre de référence. Nous savons que plus on a de données, plus le plongement calculé par l'algorithme de réduction de dimensionnalité sera fiable. Pour obtenir un plongement de référence, nous entraînons donc l'algorithme $\mathcal{A}$ sur un très grand ensemble de données, soit $D_2 \cup D_3$, et gardons la partie correspondant à D_2. Nous entraînons alors les méthodes $\mathcal{A}^1$ et $\mathcal{A}^2$ sur D_1. À partir des vecteurs propres obtenus, nous calculons le plongement hors échantillon pour chaque élément de l'ensemble de test D_2 avec la formule de Nyström. Nous comparons ensuite ces plongements de D_2 au plongement de référence de D_2 en les alignant à l'aide d'une régression linéaire simple (qui fait correspondre les coordonnées de chaque exemple d'un des plongements à celles de l'autre plongement). L'alignement est nécessaire car même de très petites perturbations dans les données peuvent mener à un changement de signe ou à une rotation des vecteurs propres. Nous utilisons comme mesure d'erreur la moyenne des distances au carré entre les points correspondants de ces plongements alignés.

5.1.1 Intervalles de confiance

À chacun des résultats donnés au chapitre suivant, nous associons un intervalle de confiance à 95%. Soit $\{e_{11}, \ldots, e_{1r}\}$, les distances au carré pour chacun des

points du plongement obtenu par l'algorithme $\mathcal{A}^1$ pour les données D_2. Soit μ_{e_1}, la moyenne de ces mesures et soit σ_{e_1}, leur écart-type empirique. En supposant que ces mesures sont indépendantes et identiquement distribuées, l'écart-type de leur moyenne est donné par

$$\overline{\sigma}_{e_1} = \frac{\sigma_{e_1}}{\sqrt{r}}$$

où $r = |D_2|$. Sous l'hypothèse de la normalité de la moyenne (ce qui est raisonnable aussitôt que $|D_2|$ devient assez grand), nous obtenons un intervalle de confiance à 95% avec

$$\mu_{e_1} \pm (1.96 \times \overline{\sigma}_{e_1}).$$

5.2 Données utilisées

Nous avons d'abord testé l'algorithme sur des données artificielles très simples. Le but était de voir si la matrice de Gram correspondant à ces données pouvait être bien approximée par notre algorithme avec un très petit dictionnaire. Nous avons généré des points en 2 dimensions correspondant à une spirale à partir de la distribution suivante :

$$x = 0.04\, t \sin(t) + \varepsilon_x$$
$$y = 0.04\, t \cos(t) + \varepsilon_y$$

où $t \sim U(3, 15)$, $\varepsilon_x \sim N(0, 0.01)$, $\varepsilon_y \sim N(0, 0.01)$, $U(a, b)$ étant la loi uniforme sur l'intervalle (a, b) et $N(\mu, \sigma)$, la loi normale. À la figure 5.1 nous illustrons les données ainsi obtenues.

Nous avons aussi testé l'algorithme sur des données réelles. Nous avons utilisé les classes 0 et 1 de l'ensemble de données MNIST (chiffres manuscrits). Ces images initialement de taille 28×28 pixels ont été redimensionnées à 14×14 pixels. À la figure 5.2, nous montrons quelques-unes de ces images. Ces données ont été choisies car les représentations en plus faible dimension obtenues par l'ACP à noyau et la segmentation spectrale étaient intéressantes : avec 2 composantes principales les 0 et les 1 étaient bien distribués en deux groupements.

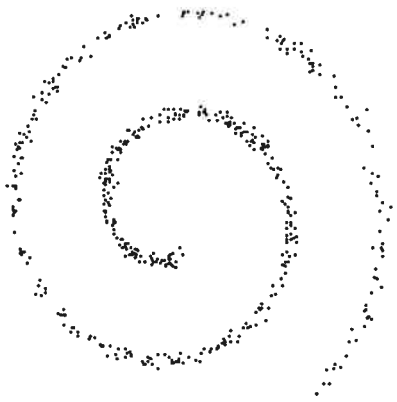

Figure 5.1 – Cinq cents points tirés d'une spirale avec un bruit gaussien.

Pour la spirale en deux dimensions, nous avons employé des ensembles de tailles $|D_1| \leq 3333$, $|D_2| = 3330$ et $|D_3| = 3332$. Pour les données de MNIST, $|D_1| \leq 3365$, $|D_2| = 3267$ et $|D_3| = 3332$.

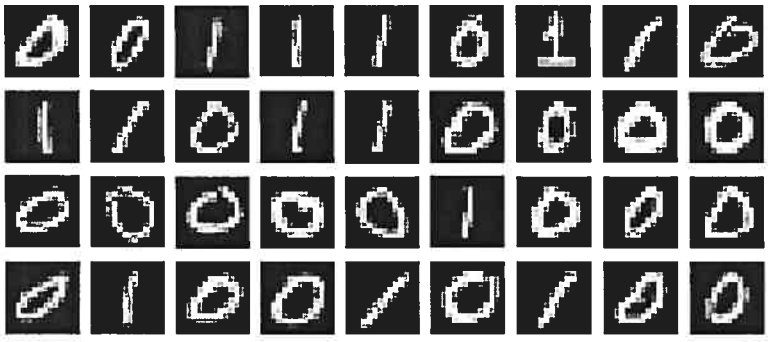

Figure 5.2 – Quelques images tirées des classes 0 et 1 des données MNIST.

5.3 Version de l'algorithme utilisée

Dans nos expériences, nous avons employé l'algorithme 4.2 dans le cas de l'ACP à noyau et l'algorithme 4.1 dans le cas de la segmentation spectrale et la contrainte $\sum_{i=1}^{m} a_{it} = 1$ a été utilisée dans les deux cas. Bien qu'elle ne soit pas requise dans le cas de la segmentation spectrale, elle n'a pas affecté la qualité des résultats, donnant pour cet algorithme des performances similaires à lorsqu'on l'omettait. La matrice A a été gardée en mémoire. Finalement nous avons employé le noyau gaussien comme noyau indépendant des données.

5.3.1 Choix des hyper-paramètres

Les valeurs d'écarts-types du noyau gaussien et le nombre de composantes principales ont été choisies de telle sorte que les plongements obtenus soient raisonnables. Nous voulions que la spirale soit bien déroulée et que les 0 et les 1 soient bien séparés pour les données MNIST. Les valeurs d'écart-type qui ont été retenues sont $\sigma = 1$ pour la spirale et $\sigma = 31.6$ pour MNIST. On a gardé deux composantes principales dans le premier cas et trois dans le second.

Pour ce qui est du paramètre d'exactitude ϵ, plusieurs valeurs ont été utilisées pour montrer l'effet obtenu avec différentes grosseurs de dictionnaire.

CHAPITRE 6

EXPÉRIENCES

Ce chapitre présente les expériences effectuées afin d'évaluer la technique du dictionnaire. Après avoir décrit les objectifs de ces expériences, on rapporte les résultats obtenus et on analyse leur signification. Nous montrons que le fait de projeter les points sur le sous-espace généré par le dictionnaire dans l'espace des caractéristiques donne de bien meilleures performances que de simplement utiliser une matrice de Gram sur un dictionnaire aléatoire de même taille. Même pour des temps d'exécution égaux, la technique du dictionnaire bat la technique sans projection. Nous montrons aussi que la technique de sélection vorace pour le dictionnaire donne de meilleurs résultats qu'une simple sélection aléatoire.

6.1 Description des expériences

La technique du dictionnaire a été évaluée en utilisant la procédure décrite à la section 5.1 pour deux méthodes de réduction de dimensionnalité : la segmentation spectrale et l'ACP à noyau. Les buts des expériences étaient :

(1) de vérifier que l'algorithme fonctionne (au sens de donner un plongement qui ressemble à celui obtenu quand on utilise la matrice de Gram complète) ;

(2) d'évaluer son efficacité au niveau du temps de calcul ;

(3) de vérifier qu'il fonctionne mieux que (a) le même algorithme avec un dictionnaire aléatoirement choisi (technique similaire à celle de HARMELING, ZIEHE, KAWANABE et MÜLLER (2002)) et que (b) si on entraîne sur un petit sous-ensemble et qu'on généralise au reste des exemples avec la formule de Nyström (WILLIAMS et SEEGER 2001; BENGIO, DELALLEAU, LE ROUX, PAIEMENT, VINCENT et OUIMET 2004).

6.2 Résultats

6.2.1 Évaluation de la technique du dictionnaire en comparaison avec lorsqu'on utilise la matrice de Gram complète

Nous souhaitons comparer la méthode du dictionnaire aux techniques d'ACP à noyau et de segmentation spectrale lorsqu'on utilise la matrice de Gram complète. Des exemples de plongements obtenus avec la méthode du dictionnaire appliquée à l'ACP à noyau sur les classes 0 et 1 de l'ensemble de données MNIST sont données à la figure 6.1. Avec peu de points dans le dictionnaire, on semble obtenir des plongements très similaires à ceux de l'ACP ordinaire. Les résultats des expériences rapportés dans cette section confirment cette hypothèse.

À la figure 6.2, nous vérifions que l'algorithme vorace fonctionne approximativement aussi bien que quand nous employons la matrice de Gram complète. Sur l'axe horizontal, nous varions la taille de l'ensemble d'entraînement D_1 et sur l'axe vertical, nous indiquons les erreurs hors échantillon obtenues avec la formule de Nyström. La méthode du dictionnaire (courbes étiquetées avec ϵ, m, σ) est entraînée avec des valeurs fixes de ϵ, donnant différentes tailles de dictionnaire à mesure que n augmente. Dans l'expression "$m \leq X$", X désigne la plus grande taille de dictionnaire obtenue pour cette courbe. Les erreurs de généralisation pour l'ACP à noyau et la segmentation spectrale ordinaires (avec matrice de Gram sur tous les exemples de D_1) sont rapportées par les courbes étiquetées avec σ seulement (courbes noires pleines).

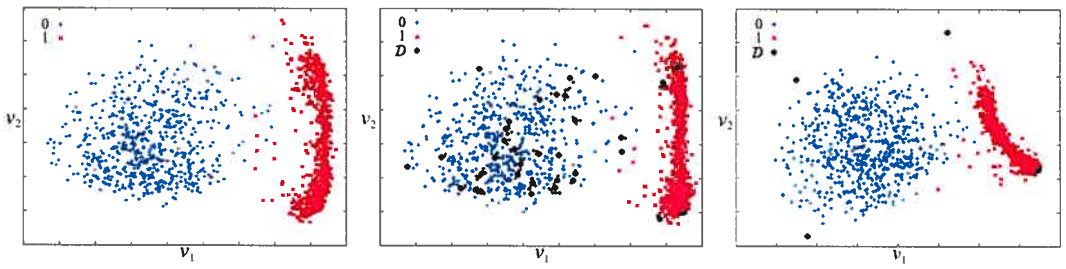

Figure 6.1 – Les deux premières dimensions de la représentation obtenue avec l'ACP à noyau sur les classes 0 (+) et 1 (×) des données MNIST ($\sigma = 31.6$, $n = 1300$). L'image à gauche a été obtenue avec la matrice de Gram complète et les deux autres, avec la technique du dictionnaire. Les points du dictionnaire sont désignés par le symbole ◆. On obtient une représentation presque identique avec 34 points dans le dictionnaire. Même avec seulement 4 points, le résultat est raisonnable et similaire.

La première chose à remarquer dans cette figure est que pour un ensemble d'entraînement de taille n, les résultats avec un dictionnaire de taille $m \ll n$ sont très près des résultats obtenus en utilisant un entraînement ordinaire avec tous les exemples. En d'autres mots, l'algorithme vorace généralise pratiquement aussi bien que l'entraînement complet. L'autre conclusion importante à tirer de cette figure est que la méthode du dictionnaire fonctionne bien mieux que la méthode simple de Nyström (Williams et Seeger 2001; Bengio, Delalleau, Le Roux, Paiement, Vincent et Ouimet 2004) lorsque les vecteurs propres sont estimés seulement sur un dictionnaire aléatoire (comparez par exemple l'erreur obtenue pour la matrice de Gram complète entraînée sur un ensemble de données de taille $n = 125$ avec la méthode du dictionnaire pour $n = 1400$ et $m = 125$ dans le graphique de l'ACP à noyau sur MNIST). Ce résultat est confirmé par les expériences de la section 6.2.3.

6.2.2 Temps d'exécution

Pour être plus juste, au lieu comparer la méthode de Nyström à la méthode du dictionnaire pour des tailles égales de sous-ensembles on peut les comparer pour des temps d'exécution égaux. Le tableau 6.1 montre que pour le même temps d'exécution, la méthode du dictionnaire donne une erreur de généralisation beaucoup plus petite.

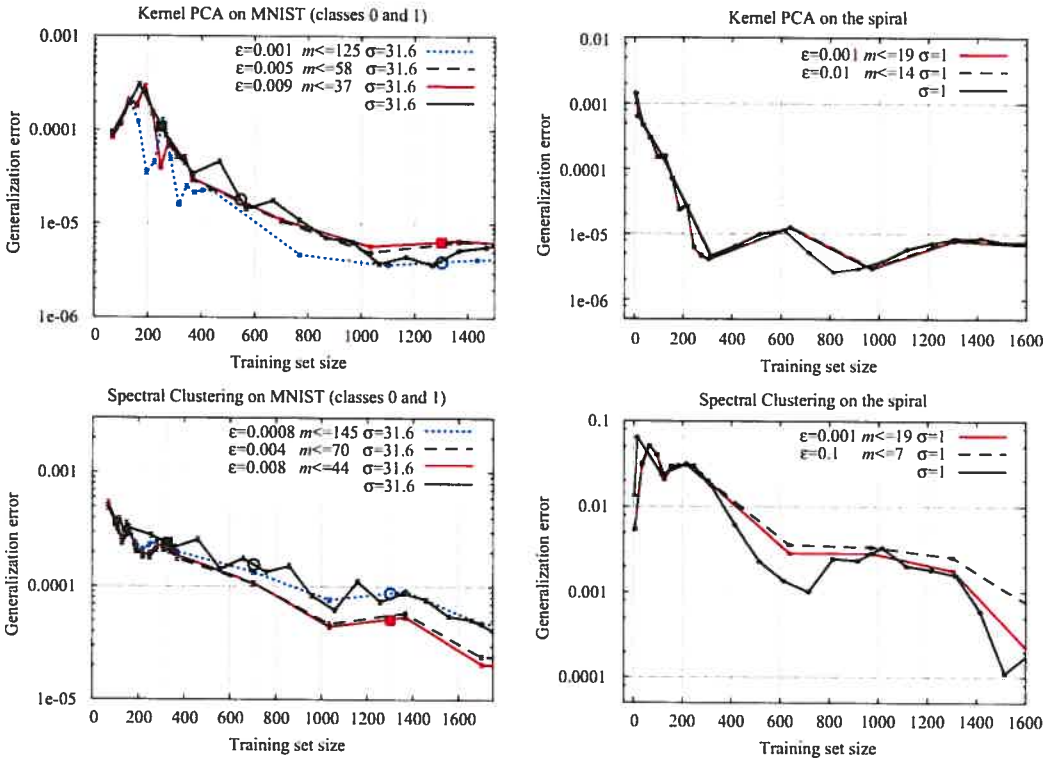

Figure 6.2 – Même avec de très petits dictionnaires, l'erreur de généralisation est comparable à celle obtenue avec la matrice de Gram correspondant à l'ensemble d'entraînement au complet. Les courbes correspondant à la méthode du dictionnaire sont étiquetées avec σ, l'écart-type du noyau gaussien, ϵ, le paramètre d'exactitude et m, le nombre maximal de points dans le dictionnaire pour cette courbe (obtenu pour le n maximal). Les courbes correspondant aux méthodes sans dictionnaire sont étiquetées avec σ seulement. On a gardé trois composantes principales pour MNIST et deux pour la spirale. Les expériences individuelles correspondant aux symboles ■ et ■ ont le même temps d'exécution, mais l'erreur de généralisation est bien plus faible pour la méthode du dictionnaire. Nous observons le même phénomène pour des symboles o et o correspondant à un plus long temps d'exécution. Plus de détails sur la relation entre le temps d'exécution et l'erreur de généralisation sont donnés au Tableau 6.1.

À la figure 6.2 et au tableau 6.1 on peut remarquer une légère augmentation de l'erreur lorsque la taille du dictionnaire augmente pour la segmentation spectrale sur les données MNIST, mais comme prévu, la courbe pour le plus grand dictionnaire est plus près de celle pour l'ensemble de données entier. Cette augmentation de l'erreur est probablement due à du sur-apprentissage ; employer un petit dictionnaire est une manière de régulariser.

Tableau 6.1 – Comparaison entre les erreurs de généralisation pour la technique du dictionnaire et la technique classique pour un temps d'exécution fixe. Les erreurs et les n correspondent à ceux donnés à la figure 6.2 pour MNIST. Certains de ces résultats y sont rapportés par des cercles et des carrés sur les courbes. Des résultats semblables ont été obtenus pour un espace mémoire fixe.

	Sans dictionnaire		Avec dictionnaire	
Temps KPCA (s)	Taille max.	Erreur ±95% I.C.	Taille max.	Erreur ±95% I.C.
5.71	$n = 250$	$1.48e^{-4} \pm 8.9e^{-6}$	$n = 1300,\ m = 34$	$6.64e^{-6} \pm 2.3e^{-7}$
5.72	$n = 250$	$1.48e^{-4} \pm 8.9e^{-6}$	$n = 1300,\ m = 55$	$6.40e^{-6} \pm 2.1e^{-7}$
6.46	$n = 550$	$8.27e^{-6} \pm 4.0e^{-7}$	$n = 1300,\ m = 126$	$3.99e^{-6} \pm 1.5e^{-7}$
14.02	$n = 1300$	$3.47e^{-6} \pm 1.4e^{-7}$		
Temps SC (s)	Taille max.	Erreur ±95% I.C.	Taille max.	Erreur ±95% I.C.
5.80	$n = 325$	$2.41e^{-4} \pm 1.2e^{-5}$	$n = 1300,\ m = 41$	$5.16e^{-5} \pm 2.3e^{-6}$
5.95	$n = 400$	$2.37e^{-4} \pm 1.2e^{-5}$	$n = 1300,\ m = 65$	$5.61e^{-5} \pm 2.5e^{-6}$
7.18	$n = 700$	$1.58e^{-4} \pm 7.3e^{-6}$	$n = 1300,\ m = 138$	$8.74e^{-5} \pm 3.8e^{-6}$
14.06	$n = 1300$	$7.93e^{-5} \pm 3.4e^{-6}$		

6.2.3 Comparaison avec d'autres techniques d'approximation

À la figure 6.3 nous comparons la méthode vorace pour la sélection du dictionnaire à une simple sélection aléatoire. Dans les deux cas nous projetons les autres points sur le sous-espace généré par ce sous-ensemble de points dans l'espace des caractéristiques. Nous montrons également les performances de la technique de Nyström lorsque la décomposition est faite seulement sur la matrice de Gram correspondant à un sous-ensemble aléatoire de points, ignorant les autres points de l'ensemble d'entraînement. Dans ces expériences nous avons employé un

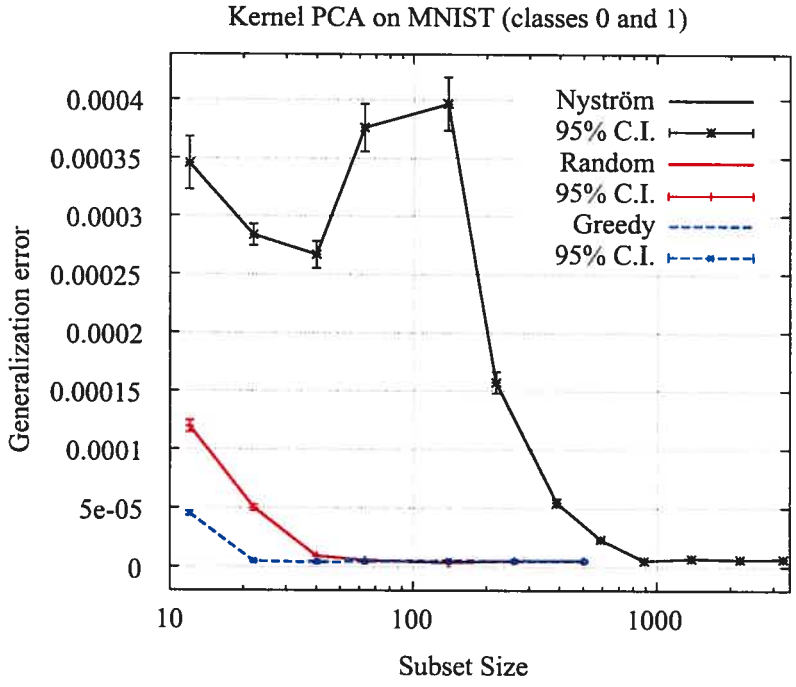

Figure 6.3 – Comparaison de la méthode du dictionnaire avec la méthode de Nyström où la décomposition est faite seulement sur la matrice de Gram correspondant à un sous-ensemble aléatoire de l'ensemble d'entraînement en ignorant les autres exemples de cet ensemble. Nous montrons deux stratégies de choix de sous-ensemble pour la méthode du dictionnaire : le sélection vorace proposée et un choix aléatoire simple. Ces expériences ont été exécutées sur les données MNIST avec un ensemble d'entraînement de la taille 3365 en gardant 3 composantes principales.

ensemble d'entraînement de taille 3365 et montrons les erreurs de généralisation pour différentes tailles de sous-ensembles m. La première chose à remarquer est que le fait de projeter les points qui ne sont pas dans le sous-ensemble améliore grandement les performances. Pour les données de MNIST, nous avons besoin d'un sous-ensemble de taille environ 900 pour que la technique de Nyström atteigne les performances que nous obtenons avec seulement 100 points dans le sous-ensemble si nous projetons les 3265 autres points. Bien que la sélection vorace et la sélection aléatoire se comportent de la même façon pour des sous-ensembles de la taille 65 et plus, ce n'est pas le cas pour de plus petites tailles. Comme nous nous y attendions, plus la taille du sous-ensemble est petite plus la sélection vorace se démarque de la sélection aléatoire, et ce de manière significative. Notez que pour

ce problème, un dictionnaire vorace de la taille 44 a donné des résultats similaires à quand la matrice de Gram complète est employée, tel qu'illustré à la figure 6.2, mais la figure 6.3 suggère qu'un dictionnaire de taille environ 22 aurait fait aussi bien. Pour ces tailles de dictionnaire, les performances de la méthode avec sélection aléatoire du dictionnaire ne sont pas aussi bonnes et par conséquent, une approche comme celle de HARMELING, ZIEHE, KAWANABE et MÜLLER (2002) devra sélectionner un sous-ensemble de la taille au moins 65 pour atteindre un niveau d'erreur à peu près identique.

CHAPITRE 7

CONCLUSION

Les méthodes spectrales sont utiles pour l'apprentissage de variétés (réduction non linéaire de dimensionnalité) et la segmentation (segmentation spectrale). Toutefois, elles ont un temps de calcul de $O(pn^2)$ et requièrent $O(n^2)$ en espace mémoire, n étant le nombre d'exemples dont on dispose et p, le nombre de dimensions désirées. Dans le but de rendre ces méthodes applicables à de grands ensembles de données, nous avons proposé un algorithme efficace qui donne des résultats similaires aux méthodes exactes (du moins pour l'ACP à noyau et la segmentation spectrale). Cette technique réduit le temps de calcul à $O(m^2n)$ et l'espace mémoire occupé à $O(m^2)$, où m est la taille d'un petit sous-ensemble d'exemples (le dictionnaire). L'algorithme choisit des exemples de façon vorace et séquentielle en se basant sur leur distance au sous-espace généré par les exemples déjà choisis dans l'espace des caractéristiques induit par un noyau. On utilise alors la projection de tous les n exemples sur ce sous-espace pour estimer les vecteurs propres.

Cet algorithme peut être formulé de différentes façons, dépendant qu'on soit plus soucieux de l'espace mémoire utilisé ou du temps de calcul. Contrairement aux techniques d'approximation précédemment proposées dans la littérature, on peut le formuler comme un algorithme en-ligne. Cet algorithme prend en paramètre un niveau de tolérance qui sert à déterminer si un point est suffisamment bien ap-

proximé par sa projection sur le sous-espace généré par le dictionnaire. Ce paramètre détermine indirectement la taille de sous-ensemble qu'on va obtenir. À l'aide de ce même niveau de tolérance, il est possible de borner l'erreur faite sur l'approximation de la matrice de Gram par une matrice de plus faible rang.

Si la stratégie vorace, la mise à jour itérative et l'utilisation du plongement hors échantillon ont été utilisés ailleurs dans la littérature, en combinant ces techniques nous obtenons un nouvel algorithme qui fait mieux que les techniques précédemment proposées. En effet, les expériences montrent que la méthode de sélection est significativement meilleure qu'un choix aléatoire, et meilleure que de simplement utiliser la méthode de Nyström pour généraliser à partir d'un sous-ensemble d'exemples. En fait, les résultats obtenus sont presque aussi bons que lorsqu'on entraîne avec tous les exemples.

7.1 Pistes de recherche

Il pourrait être intéressant d'essayer de trouver automatiquement une valeur pour l'hyper-paramètre ϵ. On pourrait, par exemple, trouver un lien entre le spectre des valeurs propres et la valeur d'ϵ pour laquelle l'approximation est bonne. On peut montrer que pour l'ACP, l'erreur de reconstruction des exemples lorsqu'on garde p composantes est donnée par la somme des $n-p$ plus petites valeurs propres. On pourrait peut-être choisir un ϵ du même ordre que cette somme. En effectuant la décomposition d'une matrice de Gram correspondant à un petit sous-ensemble aléatoire de points on obtiendrait un spectre qui approxime le spectre obtenu avec une plus grosse matrice de Gram et on pourrait ainsi obtenir une idée de la bonne valeur de ϵ.

Dans le même ordre d'idées, il serait intéressant de voir s'il y a une relation entre la taille du dictionnaire pour laquelle on obtient une bonne approximation et la dimensionnalité intrinsèque de la variété qu'on pourrait estimer avec une technique comme celle présentée dans (Kegl 2003).

Deux versions de l'algorithme ont été proposées, une correspondant à l'ACP à noyau et l'autre à la segmentation spectrale. Nous souhaiterions développer des ver-

sions pour d'autres méthodes à noyau comme LLE ou Isomap. Comme dans le cas de l'ACP à noyau et de la segmentation spectrale, nous pouvons écrire une forme fonctionnelle dépendante des données pour les noyaux de LLE et d'Isomap (Bengio, Delalleau, Le Roux, Paiement, Vincent et Ouimet 2004). Cependant, la mise à jour des matrices (éq. 4.7) devient inefficace dans ces cas car plusieurs entrées de la matrice doivent être modifiées lorsqu'on ajoute un exemple au dictionnaire. Il faudrait trouver une façon de contourner ce problème.

Dans l'analyse des expériences nous avons émis l'hypothèse que réduire le nombre d'exemples dans le dictionnaire pouvait être un moyen de régulariser. De plus amples expériences seraient nécessaires pour confirmer ce phénomène, mais de prime abord ça semblerait logique. Si tel est le cas, notre technique pourrait s'avérer utile autant pour les grands ensembles de données que pour les petits, pour lesquels du sur-apprentissage peut se produire.

Références

Belkin, M. et P. Niyogi (2003), « Laplacian Eigenmaps for Dimensionality Reduction and Data Representation », *Neural Computation 15*(6), p. 1373–1396.

Bengio, Y., O. Delalleau, N. Le Roux, J.-F. Paiement, P. Vincent et M. Ouimet (2004), « Learning eigenfunctions links spectral embedding and kernel PCA », *Neural Computation 16*(10), p. 2197–2219.

Bengio, Y., J. Paiement, P. Vincent, O. Delalleau, N. Le Roux et M. Ouimet (2004), « Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering », *Advances in Neural Information Processing Systems 16*. MIT Press.

Chung, F. (1997), « Spectral graph theory », *CBMS Regional Conference Series*, Volume 92. American Mathematical Society.

Cox, T. et M. Cox (1994), *Multidimensional Scaling*, London : Chapman & Hall.

Cristianini, N. et J. Shawe-Taylor (2000), *An Introduction to Support Vector Machines and other kernel-base learning methods*. Cambridge University Press.

Diamantras, K. et S. Kung (1996), *Principal Component Neural Networks : theory and applications*. Wiley.

Duda, R. O., P. E. Hart et D. G. Stork (2000), *Pattern Classification*. Wiley-Interscience Publication.

Engel, Y., S. Mannor et R. Meir (2004), « The kernel recursive least squares algorithm », *IEEE Trans. Sig. Proc. 52*(8), p. 2275–2285.

FOWLKES, C., S. BELONGIE, F. CHUNG et J. MALIK (2004), « Spectral grouping using the Nyström method », *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*(2), p. 214–225.

GOURLAY, A. R. et G. A. WATSON (1973), *Computational Methods for Matrix Eigenproblems*, New York, NY : John Wiley & Sons.

GUYON, I. et A. ELISSEEFF (2003), « An introduction to variable and feature selection », *J. Mach. Learn. Res. 3*, p. 1157–1182.

HARMELING, S., A. ZIEHE, M. KAWANABE et K.-R. MÜLLER (2002), « Kernel Feature Spaces and Nonlinear Blind Souce Separation », *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.

HASTIE, T., R. TIBSHIRANI et J. FRIEDMAN (2003), *The elements of statistical learning : data mining, inference and prediction*, New York, NY : Springer.

HINTON, G. et T. J. SEJNOWSKI (1999), *Unsupervised Learning : Foundation of Neural Computation*, Cambridge, MA : The MIT Press.

KEGL, B. (2003), « Intrinsic dimension estimation using packing numbers », *Advances in Neural Information Processing Systems 15*. The MIT Press.

LAWRENCE, N., M. SEEGER et R. HERBRICH (2003), « Fast Sparse Gaussian Process Methods : The Informative Vector Machine », *Advances in Neural Information Processing Systems 15*, p. 609–616. MIT Press.

NG, A. Y., M. I. JORDAN et Y. WEISS (2002), « On Spectral Clustering : analysis and an algorithm », *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.

ROWEIS, S. et L. SAUL (2000, Dec.), « Nonlinear dimensionality reduction by locally linear embedding », *Science 290*(5500), p. 2323–2326.

SCHÖLKOPF, B., A. SMOLA et K.-R. MÜLLER (1998), « Nonlinear component analysis as a kernel eigenvalue problem », *Neural Computation 10*, p. 1299–1319.

SCHÖLKOPF, B. et A. J. SMOLA (2002), *Learning with Kernels*, Cambridge, MA : The MIT Press.

SCHÖLKOPF, B., A. J. SMOLA et K.-R. MÜLLER (1999), « Kernel principal component analysis », *Advances in kernel methods : support vector learning*, p. 327–352. MIT Press.

SHAWE-TAYLOR, J. et C. WILLIAMS (2003), « The Stability of Kernel Principal Components Analysis and its Relation to the Process Eigenspectrum », *Advances in Neural Information Processing Systems 15*. MIT Press.

SMOLA, A. et B. SCHÖLKOPF (2000), « Sparse greedy matrix approximation for machine learning », *International Conference on Machine Learning*, San Francisco, p. 911–918. Morgan Kaufmann.

SMOLA, A. J. et P. BARTLETT (2001), « Sparse Greedy Gaussian Process Regression », *Advances in Neural Information Processing Systems 13*.

SPIELMAN, D. A. et S.-H. TENG (1996), « Spectral Partitioning Works : Planar Graphs and Finite Element Meshes », Rapport technique UCB CSD-96-898, U.C. Berkeley.

TENENBAUM, J., V. DE SILVA et J. LANGFORD (2000, Dec.), « A Global Geometric Framework for Nonlinear Dimensionality Reduction », *Science 290*(5500), p. 2319–2323.

VAPNIK, V. (1995), *The Nature of Statistical Learning Theory*, New York : Springer.

WEISS, Y. (1999), « Segmentation using eigenvectors : a unifying view », *Proceedings IEEE International Conference on Computer Vision*, p. 975–982.

WILLIAMS, C. K. I. et M. SEEGER (2001), « Using the Nyström Method to Speed Up Kernel Machines », *Advances in Neural Information Processing Systems 13*, Cambridge, MA, p. 682–688. MIT Press.