UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES

PAR
Aboubakry Moussa SOW

CLASSIFICATION, RÉDUCTION DE DIMENSIONNALITÉ ET RÉSEAUX DE NEURONES : DONNÉES MASSIVES ET SCIENCE DES DONNÉES

Novembre 2020

Université du Québec à Trois-Rivières

Service de la bibliothèque

Avertissement

L'auteur de ce mémoire ou de cette thèse a autorisé l'Université du Québec à Trois-Rivières à diffuser, à des fins non lucratives, une copie de son mémoire ou de sa thèse.

Cette diffusion n'entraîne pas une renonciation de la part de l'auteur à ses droits de propriété intellectuelle, incluant le droit d'auteur, sur ce mémoire ou cette thèse. Notamment, la reproduction ou la publication de la totalité ou d'une partie importante de ce mémoire ou de cette thèse requiert son autorisation.

Résumé

Le but principal de ce mémoire est d'effectuer le lien entre deux méthodes de réduction de dimensionnalité, en l'occurrence l'ACP, qui est une méthode classique, et l'autoencodeur, qui est une méthode neuronale. La première partie concerne la théorie sur les méthodes de réduction linéaire de la dimensionnalité, suivie d'une illustration avec des jeux de données. Parmi ces méthodes, nous avons la sélection de caractéristiques, l'extraction de caractéristiques et le positionnement multidimensionnel que nous avons illustrés respectivement sur les jeux de données « Swiss », « Iris de Fisher » et les distances en km entre 10 villes du Québec. En particulier, l'ACP s'inscrit dans le cadre des méthodes d'extraction de carctéristiques. La deuxième partie porte sur la théorie sur l'autoencodeur qui est une méthode non linéaire de réduction de dimensionnalité et sa comparaison avec l'ACP. Cette partie a permis de mieux comprendre le fonctionnement de l'autoencodeur, mais aussi le lien théorique qu'il a avec l'ACP. Ce lien stipule qu'un autoencodeur dont la fonction d'activation est linéaire et la fonction objectif est l'erreur quadratique moyenne a des performances similaires à celles de l'ACP. C'est ce que nous avons pu confirmer en analysant l'autoencodeur avec deux bases nommées « Wine » et « Mnist » où l'autoencodeur, moyennant l'ajustement de plusieurs paramètres, donne en général des résultats meilleurs que l'ACP. Par ailleurs, l'ACP peut servir de guide sur la détermination du nombre de neurones dans la couche code de l'autoencodeur.

Abstract

The main goal of this paper is to make the link between two methods of dimensionality reduction, in this case the PCA, which is a classical method, and the autoencoder, which is a neural method. The first part concerns the theory on linear dimensionality reduction methods, followed by an illustration with data sets. Among these methods, we have feature selection, feature extraction and multidimensional positioning which we have illustrated respectively on the data sets « Swiss », « Iris de Fisher » and distances in km between 10 cities in Quebec. In particular, PCA is part of the feature extraction methods. The second part deals with the theory of the autoencoder which is a non-linear method of dimensionality reduction and its comparison with PCA. This part has allowed a better understanding the functioning of the autoencoder, but also the theoretical link it has with PCA. This link stipulates that an autoencoder whose activation function is linear and whose objective function is the root mean square error has performances similar to those of the PCA. This was confirmed by analyzing the autoencoder with two bases named « Wine » and « Mnist » where the autoencoder, by adjusting several parameters, generally gives better results than the PCA. In addition, PCA can be used as a guide to determine the number of neurons in the code layer of the autoencoder.

Avant-propos

Ce document s'intéresse au lien entre l'Analyse en composantes principales qui est une méthode classique et l'autoencodeur qui est une méthode neuronale. La description de chacune des méthodes permettra de bien distinguer les bases de chacune d'elles. Des applications sur des jeux de données variés seront développées afin d'effectuer une étude comparative entre ces deux méthodes.

Je tiens à adresser mes sincères remerciements à ma directrice de recherche Mme Nadia Ghazzali pour avoir accepté de diriger mon projet de recherche. Je souhaiterais également la remercier pour sa disponibilité et son appui tant du côté financier que pédagogique. Merci infiniment pour les efforts consentis dans la lecture et des corrections de fond et de forme apportées, sans lesquelles ce document ne pourrait atteindre ce stade de clarté et de précision. Je remercie aussi le Conseil de Recherches en Sciences Naturelles et en Génie du Canada (CRSNG) et l'Institut des Sciences Mathématiques du Québec (ISM) pour leur appui financier.

Mes remerciements vont également à l'endroit de mes parents qui m'ont toujours orienté, appuyé et motivé. Un grand merci aussi à mes frères et soeurs, mes collègues, mes amis et à tous ceux qui ont contribué de près ou de loin à la rédaction de ce mémoire.

Table des matières

Liste des tableaux

Table des figures

Introduction

La réduction de dimensionnalité consiste à faire correspondre des données d'un espace de dimension élevée (plusieurs variables) dans un espace de dimension inférieure tout en essayant de concerver le mieux possible la structure des données. Elle est principalement utilisée pour lutter contre le fléau de la dimensionnalité. Ce dernier désigne l'ensemble des problèmes rencontrés lorsqu'on veut travailler sur des données de grande dimension. La représentation des données dans un espace de dimension inférieure peut améliorer les performances sur différentes tâches, telles que la classification [6]. L'Analyse en composantes principales (ACP) est souvent utilisée par les statisticiens pour réduire la dimension d'un jeu de données. Elle consiste à transformer des variables corrélées entre elles en nouvelles variables décorrélées les unes des autres appelées composantes principales. L'ACP tire son origine d'un article de Karl Pearson publié en 1901 [20]. Puis en 1930, elle a été formalisée par Harold Hotelling [11]. Cela a incité ce dernier à mettre en œuvre l'analyse canonique des corrélations qui est une généralisation des méthodes d'analyses factorielles et, en particulier, l'ACP. Par ailleurs, la réduction de dimensionnalité a été l'une des principales préoccupations en apprentissage profond. Ce dernier prend sa source dans les travaux de McCulloch et Pitts en 1943 [17], ceux de Hebb en 1949 [4], puis plus tard ceux de Rosenblatt en 1958 [22] sur la conception de modèles de neurones artificiels inspirés du fonctionnement du neurone biologique. Cependant, ces systèmes de réseaux n'étaient pas adaptés pour gérer de grands volumes de données. C'est ainsi que les travaux de McCullotch et Pitts et ceux de Hebb ont été contesté par Minsky et Papert (1969) [18] en raison de leur

inaptitude à modéliser des situations plus complexes en terme de taille des données. Ainsi, l'idée d'étudier les autoencodeurs, réseaux de neurones artificiels permettant de réduire la dimension d'un jeu de données au même titre que l'ACP, est en phase avec l'avènement du « Big Data »(données massives). L'un des articles marquants sur les autoencodeurs est celui de Geoffrey Hinton en 2006 [9] où les résultats de l'autoencodeur étaient plus performants que ceux de l'ACP.

Le but principal de ce mémoire est de contribuer à faire le lien entre l'ACP qui est une méthode statistique et l'autoencodeur qui est une méthode neuronale en matière de réduction de dimensionnalité. Ce document est organisé en deux parties. La première partie décrit les méthodes de réduction linéaire de la dimensionnalité et comporte trois chapitres respectivement sur la sélection de caractéristiques, l'extraction de caractéristiques et le positionnement multidimensionnel. La deuxième partie porte sur le chapitre 4 qui évoque les réseaux de neurones artificiels et l'autoencodeur et le chapitre 5 qui aborde une analyse comparative de l'ACP et de l'autoencodeur.

Première partie

Réduction de dimensionnalité

Pour réduire la dimension d'un ensemble, on peut distinguer trois approches à savoir la sélection de caractéristiques, l'extraction de caractéristiques et le positionnement multidimensionnel [24]. Ces derniers constituent des méthodes de réduction linéaire de la dimension. Néanmoins, on verra dans la deuxième partie du mémoire l'autoencodeur qui fait l'objet de ce travail et permet d'effectuer une réduction non linéaire de la dimension.

Chapitre 1

Sélection de caractéristiques

Soient $X_1, X_2, ..., X_p$ p variables. La sélection de caractéristiques consiste à sélectionner un sous-ensemble de k variables parmi les p variables de telle sorte que les points d'échantillonnage réduits dans k-dimensions reflètent certaines propriétés géométriques des p-dimensionnels d'origine. Nous pouvons citer les méthodes de régression pas à pas. Ces dernières sont appliquées dans le cadre de la régression linéaire et permettent de retenir dans le modèle les variables indépendantes expliquant le mieux possible la variable dépendante. Dans la suite de ce chapitre, seront présentées brièvement les méthodes de régression pas à pas, puis illustrées à travers un exemple pratique.

1.1 Méthodes de régression pas à pas

Soient $X_1, X_2, ..., X_p$ p variables pouvant expliquer une variable dépendante Y. La relation $(*)$: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \varepsilon$, où $\varepsilon \sim N(0, \sigma^2 I_N)$, définit la régression linéaire de Y sur les variables explicatives $X_1, X_2, ..., X_p$. Les méthodes de

type pas à pas consistent à retenir le meilleur modèle avec k variables explicatives, $k \leq p$. Elles s'appuient sur le test de Student pour retenir ou retirer une variable explicative à chaque étape de la recherche du meilleur modèle [27].

Pour ce faire, on peut appliquer les trois méthodes suivantes :

- la méthode ascendante ;
- la méthode descendante ;
- la méthode mixte qui est une combinaison de ces deux méthodes.

Mesure de la significativité d'un coefficient à l'aide du test de Student :[8]

La loi de Student :

Si X_1 suit une loi normale $N(0,1)$ et X_2 suit une loi $\chi^2(H)$ à H degrés de liberté, et si X_1 et X_2 sont indépendants alors $S = \frac{X_1}{\sqrt{\frac{X_2}{H}}}$ suit une loi de Student à H degrés de liberté. On note $S \sim \text{Student}(H)$ ou $S \sim \text{t}(H)$.

Test de Student sur un coefficient :

Soit le modèle linéaire $(*)$, β_0 la constante et $\beta_1, \beta_2, ..., \beta_p$ les paramètres associés aux variables explicatives. Posons N le nombre d'observations, $e = (1, 1, ..., 1)'$ le vecteur constant à N lignes et X la matrice des variables explicatives à laquelle on adjoint le vecteur constant e : $X = (e, X_1, X_2, ..., X_p)$ est donc une matrice dimension $N\text{x}(p+1)$. Alors, le modèle peut s'écrire sous la forme suivante :

$$Y = X\beta + \varepsilon.$$

L'estimateur de β, conditionnellement aux variables explicatives, obtenu par la méthode des moindres carrés ordinaires est :

$$\begin{aligned}\widehat{\beta} &= (X^T X)^{-1} X^T Y \\ &= (X^T X)^{-1} X^T (X\beta + \varepsilon) \\ &= \beta + (X^T X)^{-1} X^T \varepsilon.\end{aligned}$$

La variance de $\widehat{\beta}$ est :

$$\text{Var}(\widehat{\beta}) = \sigma^2(X^TX)^{-1}.$$

L'estimateur de $\text{Var}(\widehat{\beta})$ est :

$$\widehat{\text{Var}(\widehat{\beta})} = \widehat{\sigma^2}(X^TX)^{-1}.$$

où $\widehat{\sigma^2}$ est l'estimateur de σ^2. Sous l'hypothèse de normalité des résidus $\varepsilon \sim N(0, \sigma^2 I_N)$, on a les résultats suivants :

- $\widehat{\beta} \sim N(\beta, \sigma^2(X^TX)^{-1})$
- $[N-(p+1)]\frac{\widehat{\sigma^2}}{\sigma^2} \sim \chi^2(N-(p+1))$
- $\widehat{\beta}$ et $\widehat{\sigma^2}$ sont indépendants.

Les hypothèses du test de Student sont les suivantes pour $i = 0, 1, ..., p$.

$$\begin{cases} H_0 : \beta_i = 0 \\ H_1 : \beta_i \neq 0 \end{cases}$$

La statistique du test est : $S = \frac{\widehat{\beta}_i - \beta_i}{\widehat{\text{Var}(\beta)}} \sim t(N-(p+1))$. Elle suit la loi de Student à $N-(p+1)$ degrés de liberté car les erreurs du modèle suivent une loi normale.

Sous H_0 vraie, on a :

$S = \frac{\widehat{\beta}_i}{\widehat{\text{Var}(\beta)}} \sim t(N-(p+1))$.

La règle de décision est la suivante :

On rejette H_0 si $\mid S \mid > t^*$ où t^* est la valeur critique de la table de Student pour un risque fixé et un nombre de degrés de liberté égal à $N-(p+1)$. Autrement dit, le coefficient β_i est significativement différent de zéro et la variable explicative X_i associée à β_i joue un rôle explicatif dans le modèle.

Remarque 1. *Dans la suite, on dira qu'une variable explicative est significative lorsque le coefficient qui lui est associé est significativement différent de zéro.*

1.1.1 Méthode ascendante

Cette méthode introduit les variables explicatives une par une jusqu'à l'obtention du meilleur modèle. Elle consiste à :

- Effectuer les p régressions possibles avec une seule variable explicative et tester la significativité individuelle dans chaque modèle. Ensuite, on retient le modèle pour lequel la variable explicative est la plus significative.
- Faire les $p-1$ régressions possibles avec deux variables explicatives et effectuer le test de Student avec la deuxième variable introduite. Ensuite, on choisit le modèle pour lequel la nouvelle variable introduite est la plus significative. Dans le cas où aucune variable n'est significative, le processus sera arrêté.
- Poursuivre le même raisonnement en faisant $p-2$ régressions possibles avec trois variables explicatives et choisir celui pour lequel la nouvelle variable introduite est la plus significative. Si aucune des variables introduites n'est significative, on arrête le processus.
- Le processus s'arrête lorsqu'aucune des nouvelles variables introduites n'est significative.

Le problème lié à cette méthode est le fait de ne pas pouvoir supprimer une variable déjà introduite dans le modèle.

1.1.2 Méthode descendante

Contrairement à la méthode ascendante qui introduit des variables à chaque étape, la méthode descendante considère le modèle global avec p variables explicatives, puis procède par élimination. Plus précisément, elle consiste à :

- Effectuer une régression avec les p variables explicatives ;
- Ensuite tester la significativité individuelle de toutes les variables explicatives :
    - Si toutes les variables expicatives sont significatives, on arrête le proces-

sus ;

• Sinon on enlève la variable la moins significative du modèle et on réeffectue la régression avec $p-1$ variables explicatives.

— Effectuer le test de Student sur les $p-1$ variables explicatives :

• Si toutes les variables explicatives sont significatives, on arrête le processus ;

• Sinon on enlève la variable la moins significative du modèle et on réeffectue la régression avec $p-2$ variables explicatives.

— On poursuit le même processus jusqu'à ce que toutes les variables soient significatives.

La limite de cette méthode réside dans le fait de ne plus pouvoir réintroduire une variable une fois qu'elle a été supprimée.

Remarque 2. *Les limites des méthodes ascendantes et descendantes sont une conséquence de la collinéarité entre les variables (dépendance).*

1.1.3 Méthode mixte

La méthode mixte peut être vue comme une combinaison des deux premières méthodes. Elle procède par ajout d'une nouvelle variable, puis par élimination de variables précédemment introduites dans le modèle. Elle permet de prendre en compte le degré de significativité de toutes les variables à chaque étape de l'algorithme. Par exemple, une variable peut être la plus significative à une étape de l'algorithme et devenir non significative après introduction d'autres variables dans le modèle. Après l'introduction d'une nouvelle variable dans le modèle, la procédure mixte effectue un test de Student sur toutes les variables anciennement introduites. Ensuite, s'il y a des variables non singnificatives, alors on retire du modèle la variable la moins significative. Le processus s'arrête lorsque toutes les variables anciennement introduites sont significatives après l'introduction d'une nouvelle variable dans le modèle.

Par conséquent, parmi ces trois méthodes de sélection présentées, la méthode mixte

semble être la meilleure procédure.

1.2 Exemple pratique avec la méthode mixte

Cette section porte sur l'application de la méthode mixte sur le jeu de données nommé « Swiss »(Voir script R en Annexe A). Ce dernier est pris dans le logiciel R. Il comporte 5 variables explicatives et la variable réponse Y (Voir tableau 1.1) :

Y (Fertilité) : la mesure commune normalisée de la fécondité ;

X_1 (Agriculture) : le pourcentage des hommes impliqués dans l'agriculture en tant qu'occupation ;

X_2 (Examination) : le pourcentage de candidats ayant obtenu la meilleure note à l'examen d'armée ;

X_3 (Éducation) : le pourcentage d'éducation au-delà de l'école primaire pour les recrues ;

X_4 (Catholique) : le pourcentage de catholiques (par opposition aux protestants) ;

X_5 (Mortalité infantile) : les naissances qui ont vu le jour, mais n'ayant vécu que moins d'une année.

Ces variables ont été observées sur 47 provinces francophones de la Suisse vers 1888.

	Fertility	Agriculture	Examination	Education	Catholic	Infant.Mortality
Courtelary	80.2	17.0	15	12	9.96	22.2
Delemont	83.1	45.1	6	9	84.84	22.2
Franches-Mnt	92.5	39.7	5	5	93.40	20.2
Moutier	85.8	36.5	12	7	33.77	20.3
Neuveville	76.9	43.5	17	15	5.16	20.6
Porrentruy	76.1	35.3	9	7	90.57	26.6
Broye	83.8	70.2	16	7	92.85	23.6
Glane	92.4	67.8	14	8	97.16	24.9
Gruyere	82.4	53.3	12	7	97.67	21.0
Sarine	82.9	45.2	16	13	91.38	24.4
Veveyse	87.1	64.5	14	6	98.61	24.5

TABLE 1.1 – Swiss - Affichage de 11 observations (provinces) du jeu de données Swiss

Le modèle global s'écrit sous la forme suivante :

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \varepsilon$$

où $\varepsilon \sim N(0, \sigma^2 I_N)$.

Le but est de trouver le meilleur modèle qui permet de faire la régression de la fertilité sur les autres variables de la base.

Les résultats du meilleur modèle retenu après application de la méthode mixte sont présentés dans le tableau 1.2 :

	Coefficients estimés	Erreur standard	S	$Pr(\geq \mid S \mid)$	P-valeur
constante	62,10131	9,60489	6,466	8,49e-08	***
X_1	-0,15462	0,06819	-2,267	0,02857	*
X_3	-0,98026	0,14814	-6,617	5,14e-08	***
X_4	0,12467	0,02889	4,315	9,50e-05	***
X_5	1,07844	0,38187	2,824	0,00722	**

TABLE 1.2 – Swiss - Résultats du modèle final retenu

Codes pour la significativité : '***' $0,001$, '**' $0,01$, '*' $0,05$, '.' $0,1$

$R^2 = 0,6993$; $R^2_{Ajuste} = 0,6707$; $P - value$ du modèle global $= 0,717e - 10(***)$.

L'erreur standard est $\sqrt{\widehat{\text{Var}(\widehat{\beta})}}$.

Le modèle est globalement significatif au seuil de $0,001$ et la variable Examination (X_2) est la seule qui n'est pas retenue. Il s'écrit donc sous la forme suivante :

Fertilité $= 62,10131 - 0,15462$(Agriculture)$- 0,98026$(Education)$+ 0,12467$(Catholique)$+$ $1,07844$(Mortalite infantile).

Chapitre 2

Extraction de caractéristiques

L'extraction des caractéristiques consiste à considérer chaque observation p-dimensionnelle par k combinaisons linéaires des variables, où k est beaucoup plus petit que p. Elle peut être réalisée grâce aux méthodes d'analyse factorielle comme l'analyse en composantes principales (ACP). Ainsi, nous allons effectuer un bref rappel de l'ACP puis l'illustrer à l'aide d'un jeu de données pertinent.

2.1 Analyse en composantes principales

L'Analyse en composantes principales (ACP) est une méthode d'extraction de caractéristiques appliquée sur des jeux de données décrits par des unités statistiques en lignes et des variables quantitatives en colonnes [5]. Par ses résultats, elle cherche à :

- repérer les similitudes entre les individus vis à vis de l'ensemble des variables,
- relever les différences entre les individus et mettre en évidence ceux dont le comportement est atypique par rapport à l'ensemble des variables,

— savoir si l'information brute ne pourrait être obtenue à partir d'un nombre restreint de variables,

— réécrire simultanément les liaisons entre les variables.

Pour ce faire, on diminue la dimension du jeu de données en déterminant successivement les axes factoriels. C'est une méthode de réduction de la dimensionnalité des données avec perte d'information.

2.1.1 Recherche des axes factoriels

Considérons n individus sur lesquels on a mesuré p variables quantitatives.

Posons $X = (X_1, X_2, ..., X_n)^T$ et $\Sigma = \text{Var}(X)$: la matrice de variances-covariances.

La première composante principale est donnée par :

$Y_1 = u_1^T X = \Sigma_{i=1}^{p} u_{1i} X_i$, où u_1 de norme 1 ($u_1^T u_1 = 1$) est choisi pour maximiser $\text{Var}(Y_1) = u_1^T \Sigma u_1$.

Le problème revient donc à maximiser la fonction suivante :

$$F(u_1, \lambda) = u_1^T \Sigma u_1 - \lambda(u_1^T u_1 - 1),$$

où λ est le multiplicateur de Lagrange. Sa solution s'obtient en dérivant par rapport à $u_{11}, u_{12}, ..., u_{1p}$ et λ.

On a $\|u_1\| = 1$ car $\frac{\partial F(u_1,\lambda)}{\partial \lambda} = -(u_1^T u_1 - 1) = 1 - u_1^T u_1 = 0$.

$$\frac{\partial F(u_1, \lambda)}{\partial u_1} = (\frac{\partial F(u_1, \lambda)}{\partial u_{11}}, \frac{\partial F(u_1, \lambda)}{\partial u_{12}}, ..., \frac{\partial F(u_1, \lambda)}{\partial u_{1p}}).$$

Après avoir dérivé par rapport à u_1, on aura : $\frac{\partial F(u_1,\lambda)}{\partial u_1} = 2\Sigma u_1 - 2\lambda u_1 = 0$.

Donc $\Sigma u_1 = \lambda u_1$.

Donc u_1 est un vecteur propre normé de Σ et λ est la valeur propre correspondante.

De plus $\text{Var}(Y_1) = u_1^T \Sigma u_1 = u_1^T(\lambda u_1) = \lambda u_1^T u_1 = \lambda$ puisque $u_1^T u_1 = 1$.

Ainsi $\lambda = \lambda_1$ la plus grande valeur propre de Σ est celle qui maximise $\text{Var}(Y_1)$.

Le vecteur propre associé à la valeur propre λ_1 est u_1.

La deuxième composante principale est donnée par $Y_2 = u_2^T X$ définie telle que :

$\text{Var}(Y_2) = u_2^T \Sigma u_2$ est maximale,

$u_2^T u_2 = 1$ et $\text{Cov}(Y_1, Y_2) = u_1^T \Sigma u_2 = 0$. On détermine donc le vecteur u_2 qui maximise la fonction suivante :

$F(u_2, \lambda, \nu) = u_2^T \Sigma u_2 - \lambda(u_2^T u_1 - 1) - \nu(u_2^T u_2 - 0)$.

En effet, $\text{Cov}(Y_1, Y_2) = \text{Cov}(u_1^T X, u_2^T X)$

Donc $\text{Cov}(Y_1, Y_2) = u_1^T \Sigma u_2 = u_2^T \Sigma u_1 = \lambda_1 u_2^T u_1$.

u_2 est normé car $\frac{\partial F(u_2, \lambda, \nu)}{\partial \lambda} = 1 - u_2^T u_2 = 0$.

u_1 et u_2 sont linéairement indépendants car

$$\frac{\partial F(u_2, \lambda, \nu)}{\partial \nu} = -u_2^T u_1 = -u_1^T u_2 = 0.$$

On a :

$$\frac{\partial F(u_2, \lambda, \nu)}{\partial u_2} = 2\Sigma u_2 - 2\lambda u_2 - \nu u_1.$$

On a :

$$u_1^T(2\Sigma u_2 - 2\lambda u_2 - \nu u_1) = 2u_1^T \Sigma u_2 - 2u_1^T \lambda u_2 - \nu u_1^T u_1.$$

Or

$$u_1^T \Sigma = \lambda_1 u_1^T, u_1^T u_2 = 0 \text{ et } u_1^T u_1 = 1.$$

Puisque $\text{Var}(Y_2) = u_2^T \Sigma u_2 = u_2^T \lambda u_2 = \lambda$, donc $\text{Var}(Y_2)$ est maximale si $\lambda = \lambda_2$ qui représente la deuxième plus grande valeur propre de Σ.

Ainsi, u_2 est le vecteur propre normé correspondant. En maximisant successivement $\text{Var}(Y)$, on aura donc la $k^{\text{ème}}$ composante principale $Y_k = u_k^T X$, où u_k est le vecteur propre normé associé à λ_k. De façon générale, les composantes principales sont : $Y = A^T X$, où

$$A = (u_1, ..., u_p) = \begin{pmatrix} u_{11} & u_{21} & ... & u_{p1} \\ u_{12} & u_{22} & ... & u_{p2} \\ . & . & ... & . \\ . & . & ... & . \\ . & . & ... & . \\ u_{1p} & u_{2p} & ... & u_{pp} \end{pmatrix}.$$

La matrice A a pour colonnes les vecteurs propres de Σ avec

$A^T A = AA^T = I_p$, $A^T = A^{-1}$.

$\Sigma A = \Lambda A$, où $\Lambda = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_p)$.

On a : $\text{Var}(Y) = A^T \Sigma A = \Lambda$ de telle sorte que $\text{Cov}(Y_i, Y_j) = 0$ si $i \neq j$ et $\text{Var}(Y_i) = \lambda_i \geq \text{Var}(Y_j) = \lambda_j$ ssi $i \leq j$.

En résumé, la recherche des axes factoriels consiste à diagonaliser la matrice de variances-covariances Σ. Le $k^{\text{ème}}$ axe factoriel correspond au $k^{\text{ème}}$ vecteur propre associé à la $k^{\text{ème}}$ plus grande valeur propre de Σ, avec $k = 1, 2, ..., p$.

Les données de départ sont reconstruites par la formule suivante :

$\widehat{X} = \Sigma_{i=1}^{k} \sqrt{\lambda_i} v_i {u_i}^T$ avec $v_i = \frac{1}{\sqrt{\lambda_i}} X u_i$, $k < p$ tel que $(p - k)$ valeurs propres sont très petites.

2.1.2 Aides à l'interprétation

Les deux aides à l'interprétation suivantes sont en général utilisées en ACP : le cosinus carré Cos^2 et la contribution CTR.

— Cosinus carré (Cos^2)

Le cosinus carré d'une variable constitue un critère important qui permet de mesurer la qualité de représentation d'une variable par rapport à sa projection sur l'axe orthonormé. Une variable est bien représentée par rapport à un axe si elle est proche du cercle de corrélation et que son Cos^2 par rapport à cet axe

est proche de 1.

$$\text{Cos}^2(\theta) = \frac{G_\alpha^2(j)}{d^2(O, Q^j)}$$

où O est l'origine du repère, Q^j est le point associé à la variable X_j sur le cercle de corrélation, $G_\alpha(j)$ est la coordonnée du point Q^j sur l'axe α et $d(O, Q^j)$ est la distance euclidienne entre O et Q^j. θ est l'angle entre l'axe factoriel α et la droite (O, Q^j).

— Contribution (CTR)

La contribution d'une variable X_k à l'inertie expliquée par l'axe u (λ_k) est la part de la variable dans l'inertie. La contribution permet donc de classer les variables selon le rôle qu'elle joue dans la détermination des axes. Ainsi, les variables les plus contributives à la formation d'un axe sont celles qui ont une forte coordonnée sur cet axe.

$$\text{CTR}_\alpha(j) = \frac{G_\alpha^2(j)}{\lambda_\alpha}$$

où $G_\alpha(j)$ est la coordonnée du point Q^j sur l'axe α et λ_α est valeur propre associée à l'axe α.

— Cercle de corrélation

Le cercle de corrélation permet de visualiser graphiquement les deux critères susmentionnés (le cosinus carré et la contribution).

2.1.3 Mise en éléments supplémentaires de variables

Il est possible de représenter dans le cercle de corrélations des variables n'ayant pas participés à la formation des axes. Ces variables sont dites illustratives ou supplémentaires. On peut faire appel à la mise en éléments supplémentaires lorsque dans nos données figurent plusieurs paquets de variables de statuts différents. Par exemple, supposons que notre jeu de données décrit des individus à l'aide des variables économiques et démographiques. Dans ce cas, on peut prendre pour variables actives les

variables économiques et chercher les rapports entre les structures économiques et les variables démographiques illustratives. Par ailleurs, les variables nominales peuvent également être utilisées comme supplémentaires en ce sens qu'elles définissent une partition des individus en autant de groupes que la variable a de modalités.

Remarque 3. *Il n'y a pas lieu de calculer la corrélation des variables supplémentaires car ces variables ne sont pas intervenues dans la formation des axes.*

2.1.4 Règles pour retenir les axes

Généralement, quatre règles sont utilisées pour retenir les axes.

Règle 1 : critère du taux d'inertie

Le critère du taux d'inertie consiste à garder les premiers axes expliquant une part importante de la variation. On peut choisir de façon arbitraire un seuil de 70% ou de 80%. Cependant, il faudrait tenir compte de la valeur décroissante des valeurs propres.

Règle 2 : Critère de Kaiser

Kaiser propose de choisir les axes pour lesquels les valeurs propres correspondantes sont supérieures ou égales à la moyenne des valeurs propres :

$\lambda_k \geq \bar{\lambda} = \frac{\lambda_1+\lambda_2+...+\lambda_p}{p}$. Dans le cas où on travaille sur la matrice de corrélations, on retiendra les axes pour lesquels leurs valeurs propres sont supérieures ou égales à 1 ($\lambda_k \geq 1$) car $\lambda_1 + \lambda_2 + ... + \lambda_p = p$ [12].

Règle 3 : Règle du pieds de l'éboulis

Cette règle consiste d'abord à tracer l'histogramme des valeurs propres qui est décroissant. Ensuite, les axes à retenir sont ceux précédant le « pieds de l'éboulis ». Ce dernier est le niveau à partir duquel on observe une forte diminution de l'inertie [3].

Règle 4 : Règle de Horn

Cette règle consiste à suivre les étapes suivantes pour choisir les axes factoriels [10] :

— Simuler une matrice de corrélations de n objets de la loi normale centrée réduite

$\mathcal{N}(0, I_p)$,

— Extraire les valeurs propres de cette matrice $m_{k1}, m_{k2}, ..., m_{kp}$,

— Répéter l'expérience K fois de suite,

— Calculer la moyenne $\bar{m}_i = \frac{1}{K}\Sigma_{i=1}^{K} m_{ik}$,

— Retenir les axes pour lesquels $\lambda_k \geq \bar{m}_k$, $k = 1, 2, ..., p$.

2.2 Exemple pratique en Analyse en Composantes Principales

Dans cette section, on appliquera l'ACP sur le jeu de données Iris de Fisher (Voir script R en Annexe B). Ce dernier comporte 5 variables et 150 individus (Voir tableau 2.1). Les individus sont des fleurs connues sous le nom d'Iris. Les quatre premières variables sont quantitatives et désignent respectivement la longueur des sépales, la largeur des sépales, la longueur des pétales et la largeur des pétales. Elles sont toutes mesurées en cm. La dernière variable porte sur le type d'Iris pouvant être Setosa, Versicolor ou Virginica ; chaque type est au nombre de 50 dans ce jeu de données. Cette variable, étant qualitative nominale, sera mise en éléments supplémentaires afin de rendre pertinente l'interprétation du nuage des individus.

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa
9	4.4	2.9	1.4	0.2	setosa
10	4.9	3.1	1.5	0.1	setosa

TABLE 2.1 – Iris de Fisher - Affichage de 10 Setosa

2.2.1 Interprétation des résultats de l'ACP sur les Iris de Fisher à l'aide du logiciel R

Puisque nous avons quatre variables actives, donc le nombre maximal d'axes factoriels possible sera quatre. Nous avons calculé les valeurs propres et leurs pourcentages d'inertie décrits dans le tableau 2.2.

Axes	Valeurs propres	% de variance	Cumul des % de variance
Axe 1	2,92	72,96	72,96
Axe 2	0,91	22,85	95,81
Axe 3	0,15	3,67	99,48
Axe 4	0,02	0,52	100,00

TABLE 2.2 – Iris de Fisher - Valeurs propres et pourcentages de variations des axes factoriels

Les deux premiers axes factoriels contribuent respectivement à 72,96% et 22,85% de l'inertie, soit 95,81% au total. Autrement dit, ils restituent plus de 80% de l'information contenue dans le jeu de données. Ainsi, d'après le critère du taux d'inertie, nous pouvons retenir ces deux axes pour visualiser nos données.

Le tableau 2.3 présente le cos^2 des variables longueur du sépale, largeur du sépale, longueur du pétale et largeur du pétale.

Variables	Axe 1	Axe 2	Axe 3	Axe 4
Longueur du sépale	0,79	0,13	0,08	0,00
Largeur du sépale	0,21	0,78	0,01	0,00
Longueur du pétale	0,98	0,00	0,00	0,01
Largeur du pétale	0,93	0,00	0,06	0,01

TABLE 2.3 – Iris de Fisher - Qualité de représentation (Cos^2)

Les variables longueur du pétale, largeur du pétale et longueur du sépale sont respectivememt bien représentées sur l'axe 1. En effet, elles ont de fortes valeurs sur cet axe. Par contre sur l'axe 2, c'est la variable largeur du sépale qui est bien représentée.
Le tableau 2.4 présente la contribution des variables longueur du sépale, largeur du sépale, longueur du pétale et largeur du pétale.

Variables	Axe 1	Axe 2	Axe 3	Axe 4
Longueur du sépale	27,15	14,24	51,78	6,83
Largeur du sépale	7,25	85,25	5,97	1,53
Longueur du pétale	33,69	0,06	2,02	64,23
Largeur du pétale	31,91	0,00	0,06	0,01

TABLE 2.4 – Iris de Fisher - Contribution (CTR) (en %)

Si on se place sur les deux premiers axes factoriels, on constate que les variables longueur du pétale, largeur du pétale et longueur du sépale sont respectivememt les plus contributives à la formation du premier axe. En ce qui concerne le deuxième axe, c'est la variable largeur du sépale qui contribue le plus à sa formation, soit 85,25%. Par ailleurs, seule la longueur du sépale apporte une plus grande contribution à la formation du troisième axe, soit 51,78%. On note aussi que c'est la variable longueur du pétale qui contribue le plus à la formation de l'axe 4 (64,23%).
La figure 2.1 illustre le cercle de corrélation de l'ACP appliquée sur les Iris de Fisher.

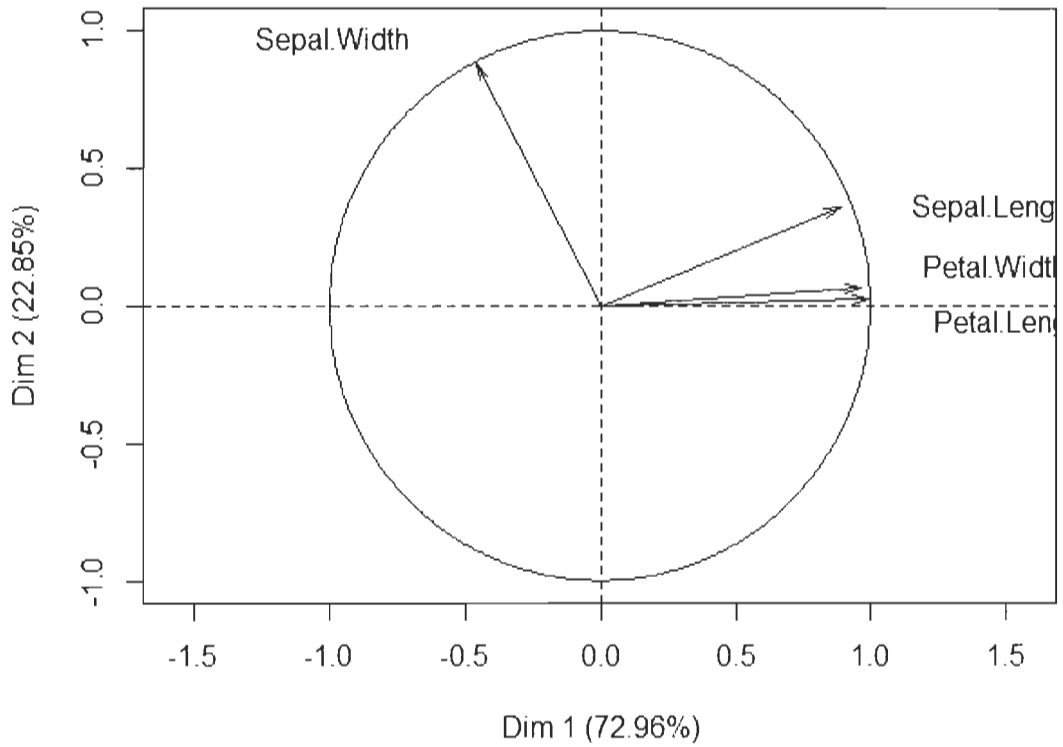

FIGURE 2.1 – Iris de Fisher - Cercle de corrélation

L'analyse du cercle de corrélation montre que le premier axe factoriel est caractérisé par la longueur du pétale, la largeur du pétale et la longueur du sépale, tandis que le deuxième axe factoriel est caractérisé par la largeur du sépale. Ce qui confirme les constats précédents.

2.2.2 Mise en éléments supplémentaires

Le figure 2.2 présente le nuage des 150 individus dans lequel le type d'iris a été mis en éléments supplémentaires.

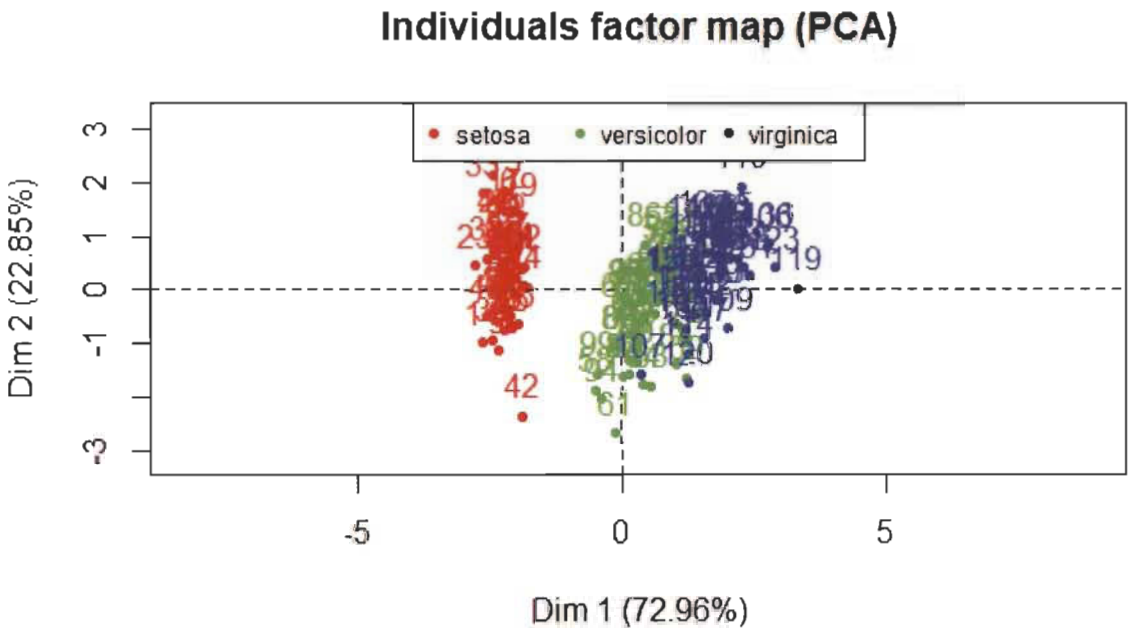

FIGURE 2.2 – Iris de Fisher - Représentation des individus (Iris) dans le plan factoriel

La mise en éléments suplémentaires du type d'iris dans le nuage des individus nous a permis de savoir que les virginica sont les iris ayant de fortes valeurs pour les longueur et largeur du pétale et la longueur du sépale alors que les setosa sont ceux ayant de faibles valeurs pour ces dernières. Les versicolor sont les iris qui semblent avoir une valeur moyenne pour toutes ces variables et il y a un chevauchement entre les versicolor et les virginica.

Chapitre 3

Positionnement multidimensionnel

Le positionnement multidimensionnel est une technique de réduction de la dimension qui consiste à approcher une configuration p-dimensionnelle X_1,X_2,X_3,...,X_n par une nouvelle configuration Y_1,Y_2,Y_3,...,Y_n de dimension k, avec k plus petit que p, de telle sorte que la seconde configuration soit résonnablement proche de la première et les distances entre les points soient préservées autant que possible. Il existe deux méthodes de positionnement multidimensionnel à savoir la méthode classique et la méthode non métrique. Très souvent en pratique, les X_i ne sont pas donnés, mais plutôt la mesure de proximité entre les paires d'individus r et s, avec r, $s = 1, 2, 3, ..., n$ [24].

3.1 Méthode classique

Considérons la configuration p-dimensionnelle X_1,X_2,...,X_n (c'est à dire n individus décrits par p variables $X^1, X^2, ..., X^p$) et la proximité $\sigma_{rs} = \|X_r - X_s\|$,
r, $s = 1, 2, 3, ..., n$.

Supposons que σ_{rs} est une dissimilarité c'est à dire $\sigma_{rr} = 0$, $\sigma_{rs} \geq 0$, et $\sigma_{rs} = \sigma_{sr}$ $\forall\, r,\, s = 1, 2, 3, ..., n$. $D = [(\sigma_{rs})]$ est appelée matrice de dissimilarité. D est euclidienne s'il existe une configuration $Y_1, Y_2, Y_3, ..., Y_n$ de dimension p tel que $\forall\, r,\, s = 1, 2, 3, ..., n$; $d_{rs} = \sigma_{rs}$ avec $d_{rs} = \|Y_r - Y_s\|$.

Gower(1966) a proposé la méthode classique sous le nom d'analyse de coordonnées principales et a montré qu'elle est étroitement liée à l'ACP. Le positionnement multidimensionnel classique découle du théorème suivant :

— **Théorème de (Gower,1966) [7], (Mardia et al.,1978) [16]**

Enoncé :

Considérons $A = [(a_{rs})]$, où $a_{rs} = -\frac{1}{2}\sigma_{rs}^2$, $\bar{a}_{r.} = \frac{1}{n}\sum_{s=1}^{n} a_{rs}$ avec $r \neq s$, $\bar{a}_{.s} = \frac{1}{n}\sum_{r=1}^{n} a_{rs}$ avec $s \neq r$ et $\bar{a}_{..} = \frac{1}{n}\sum_{r=1}^{n}\sum_{s=1}^{n} a_{rs}$, $r,\, s = 1, 2, 3, ..., n$.

Posons $B = [(b_{rs})]$, où $b_{rs} = a_{rs} - \bar{a}_{r.} - \bar{a}_{.s} - \bar{a}_{..}$. A et B sont des matrices carrées d'ordre n.

D est euclidienne si et seulement si B est semi-définie positive. Une matrice est dite semi-définie positive si toutes ses valeurs propres sont non-négatives.

Démonstration :

a) Supposons que D est euclidienne et montrons que B est semi-définie positive.

On a : $-2a_{rs} = \sigma_{rs}^2 = \|X_r - X_s\|^2 = X_r^T X_r + X_s^T X_s - 2X_r^T X_s$.

Donc

$$-2\bar{a}_{r.} = X_r^T X_r + \frac{1}{n}\sum_i X_i^T X_i - 2X_r^T \bar{X},$$

De même,

$$-2\bar{a}_{.s} = X_s^T X_s + \frac{1}{n}\sum_i X_i^T X_i - 2\bar{X}^T X_s,$$

$$2\bar{a}_{..} = 2\frac{1}{n}\sum_i X_i^T X_i - 2\bar{X}^T \bar{X}.$$

Donc

$$b_{rs} = X_r^T X_s - X_r^T \bar{X} - \bar{X}^T X_s + \bar{X}^T \bar{X} = (X_r - \bar{X})^T (X_s - \bar{X})$$

et

$B = \widetilde{X}\widetilde{X}^T \geq 0$ où $\widetilde{X}^T = (X_1 - \bar{X}, X_2 - \bar{X}, ..., X_n - \bar{X})$.

Ainsi B est semi-définie positive.

b) Supposons que B est semi-définie positive de rang p et montrons que D est euclidienne. Il existe une matrice orthogonale $V = (v_1, v_2, ..., v_n) \in \mathcal{M}_{n \times n}$ telque

$$V^T B V = \begin{pmatrix} \Gamma & 0 \\ 0 & 0 \end{pmatrix} \quad ,$$

où $\Gamma = \text{diag}(\gamma_1, \gamma_2, ..., \gamma_p)$, $\Gamma \in \mathcal{M}_{n \times p}$, et $\gamma_1 \geq \gamma_2 \geq ... \geq \gamma_p$ sont les valeurs propres positives de B.

Considérons $V_1 = (v_1, v_2, ..., v_p) \in \mathcal{M}_{n \times p}$. La matrice V_1 est une extraction des p premières colonnes de V.

$$B = V \begin{pmatrix} \Gamma & 0 \\ 0 & 0 \end{pmatrix} V^T$$

$$= V_1 \Gamma V_1^T$$

$$= (V_1 \Gamma^{1/2})(V_1 \Gamma^{1/2})^T$$

$$= YY^T,$$

où $Y = (\sqrt{\gamma_1}v_1, \sqrt{\gamma_2}v_2,...,\sqrt{\gamma_p}v_p) = (y^{(1)}, y^{(2)},...,y^{(p)}) \in \mathcal{M}_{n \text{ x } p}$.

$$= \begin{pmatrix} y_1^T \\ y_2^T \\ . \\ . \\ . \\ y_n^T \end{pmatrix} .$$

On a $\|y^{(j)}\|^2 = \gamma_j \|v_j\|^2 = \gamma_j$. Puisque $b_{rs} = y_r^T y_s$,

$\|y_r - y_s\|^2 = y_r^T y_r + y_s^T y_s - 2y_r^T y_s$

$\|y_r - y_s\|^2 = b_{rr} + b_{ss} - b_{rs} - b_{sr}$

$\|y_r - y_s\|^2 = a_{rr} + a_{ss} - 2a_{rs}$

$\|y_r - y_s\|^2 = -2a_{rs}$ (puisque $a_{rr} = a_{rr} = 0$)

$\|y_r - y_s\|^2 = \sigma_{rs}^2$, et les y_i donnent la configuration requise. Donc D est euclidienne.

Conclusion : D est euclidienne *ssi* B est semi-définie positive.

Ainsi, ce théorème donne une méthode de construction de la configuration des y_i plus connue sous le nom de positionnement multidimensionnel classique. En résumé, l'algorithme du positionnement multidimensionnel prend d'abord comme entrée la matrice de distance D obtenue à partir de la configuration de départ X. Ensuite, il détermine la matrice B à partir de D. Enfin, la sortie de cet algorithme sera la configuration Y obtenue à partir de la matrice B.

Si D est euclidienne, la configuration des y_i est appelée la solution classique. Cette dernière n'est pas unique en ce sens qu'une rotation ou un décalage d'origine ne modifie pas la distance entre les points :

Soit L une matrice orthogonale (p x p) :

$$\|L(y_r - y_s)\|^2 = (y_r - y_s)^T L^T L(y_r - y_s) = \|y_r - y_s\|^2.$$

Par ailleurs, la moyenne $\bar{y}$ est nulle.

En effet, $B = [(b_{rs})] = (I_n - n^{-1}1_n 1_n^T)A(I_n - n^{-1}1_n 1_n^T)$.
On a $(I_n - n^{-1}1_n 1_n^T) = 0 \Rightarrow B1_n = 0$ et $n^2\bar{y}^T\bar{y} = (Y^T 1_n)'(Y^T 1_n) = 1_n^T B 1_n = 0$, donc $\bar{y} = 0$.
Généralement, D n'est pas euclidienne. Dans ce cas, on ne pourra pas écrire $\Gamma = \Gamma^{1/2}\Gamma^{1/2}$. Si seules les k premières valeurs propres sont positives, $Y_k = (y^{(1)}, y^{(2)}, ..., y^{(k)})$ donnera une configuration raisonnable dans k dimensions. L'avantage de cette méthode est que son algorithme fournit rapidement une solution réalisable. Cette dernière est surtout optimale lorsque la matrice B a des valeurs propres non négatives. Néanmoins, il peut arriver que B soit semi-définie négative. Le cas échéant, le positionnement multidimensionnel clasique ne sera pas la méthode appropriée pour réduire la dimension de la configuration X.

3.2 Méthode non métrique

Dans la section précédente, nous avons considéré le problème en cherchant un p-dimensionnel de vecteurs y_i pour lesquels les d_{rs} soient aussi proches que possible des dissimilarités σ_{rs}. Toutefois, il est possible d'élargir la recherche de configuration et en chercher une telle que $d_{rs} \approx \mathrm{f}(\sigma_{rs})$ pour certaines fonctions monotones inconnues f satisfaisant :
$(\sigma_{r_1 s_1} < \sigma_{r_2 s_2} \Longleftrightarrow \mathrm{f}(\sigma_{r_1 s_1}) \leq \mathrm{f}(\sigma_{r_2 s_2}))$.
L'objectif serait donc de trouver une configuration en p dimensions de sorte que la représentation des d_{rs} par rapport à σ_{rs} augmente (ou approximativement) de façon monotone. Par exemple, si n = 3 et $\sigma_{23} < \sigma_{13} < \sigma_{12}$, nous requiérons dans ce cas 3 points y_i tels que $d_{23} \leq d_{13} \leq d_{12}$. Pour ce faire, Kruskal (1964)[13] a suggéré d'ajuster les distances d_{rs} par $\hat{d}_{rs}$ qui est une fonction croissante des σ_{rs}. Cette dernière est obtenue en minimisant la fonction STRESS notée S^2 et définie par :

$$S^2 = \frac{\sum\sum(d_{rs} - \hat{d}_{rs})^2}{\sum\sum d_{rs}^2}, r < s,$$

avec r, s =$1, 2, 3, ..., n$. Et on aurra :

$$\hat{d}_{r_1 s_1} \leq \hat{d}_{r_2 s_2} \leq ... \leq \hat{d}_{r_m s_m}.$$

où r_i et s_i, i =$1, 2, 3, ..., m$, sont respectivement des éléments fixes de r et s.
Le positionnement multidimensionnel non métrique est applicable sur des distances et sur des indices de similarité en général. Par contre, contrairement à la méthode classique, il est moins rapide en temps et la solution est un optimum local et non global. De plus, la méthode non métrique privilégie l'ordre des distances plutôt que leurs valeurs.

3.3 Exemple pratique sur le positionnement multidimensionnel

Cette sous-section concerne l'application des 2 méthodes de positionnement multidimensionnel sur les distances en km entre 10 villes de la province du Québec (Montréal, Québec, Laval, Gatineau, Longueil, Sherbrooke, Saguenay, Lévis, Trois-Rivières et Terrebonne). Plus précisément, le jeu de données porte sur les distances en voiture entre chaque paire de villes. Ces distances sont prises sur Google Maps (Voir tableau 3.1). Elles définissent donc la proximité σ_{rs} entre deux villes r et s. Nous allons approché la configuration des X_i, $i = 1, 2, ..., 10$, non connue par la configuration des Y_i, $i = 1, 2, ..., 10$, dans un espace à deux dimensions. Le fichier script R associé à l'application du positionnement multidimensionnel se trouve dans l'annexe C.

Villes	Mtl	Qc	Lav	Gtu	Lgl	Shbe	Sgy	Lvs	TR	Tbe
Montréal	0	250	33	198	10	156	461	242	141	31
Québec	250	0	264	438	246	235	211	19	129	251
Laval	33	264	0	184	35	188	472	253	140	16
Gatineau	198	438	184	0	214	353	643	433	310	193
Longueil	10	246	35	214	0	157	454	235	142	31
Sher-brooke	156	235	188	353	157	0	440	221	149	184
Saguenay	461	211	472	643	454	440	0	233	333	455
Lévis	242	19	253	433	235	221	233	0	129	248
Trois-Rivières	141	129	140	310	142	149	333	129	0	124
Terre-bonne	31	251	16	193	31	184	455	248	124	0

TABLE 3.1 – Distance entre les villes - Matrice de distances (en km) en voiture entre 10 paires de villes du Québec

3.3.1 Application de la méthode classique

A la suite de l'application du positionnement multidimensionnel classique, nous avons obtenu la configuration des Y_i, $i = 1, 2, ..., 10$, représentée dans la figure 3.1 en deux dimensions (dim 1, dim 2) :

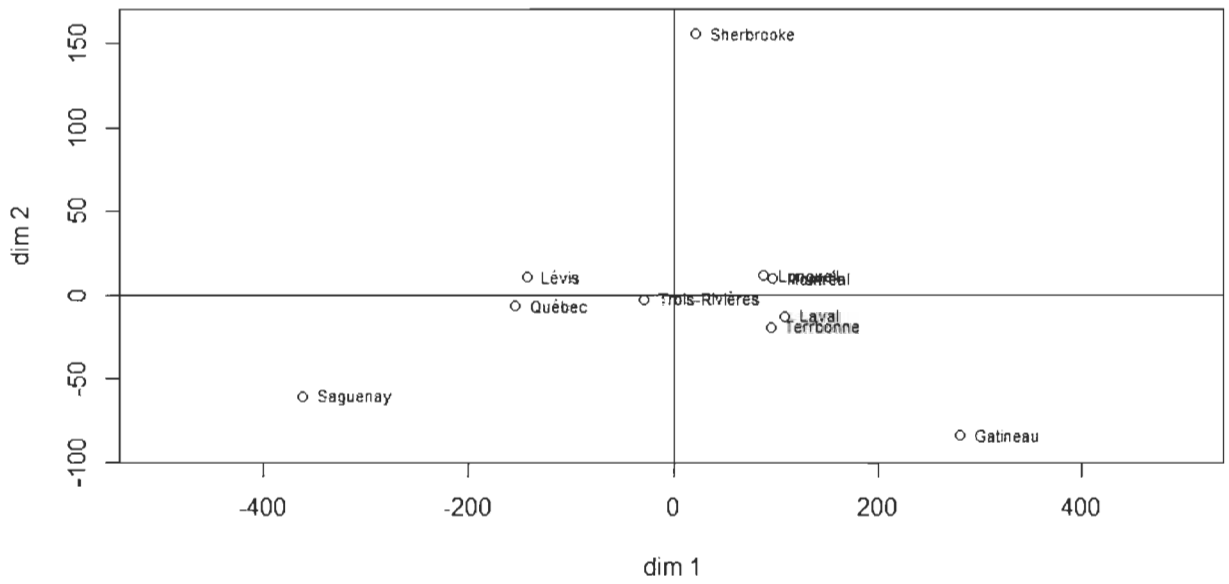

FIGURE 3.1 – Distance entre les villes - Représentation de la configuration de sortie en deux dimensions par la méthode classique

L'analyse de la figure 3.1 montre que Montréal et Longueuil sont proches. Nous constatons la même chose pour les villes de Lévis et Québec et celles de Laval et de Terrebonne.

3.3.2 Application de la méthode non métrique

L'application du positionnement multidimensionnel non métrique nous a permis d'obtenir la configuration des Y_i, $i = 1, 2, ..., 10$, représentée dans la figure 3.2 en deux dimensions (dim 1, dim 2) :

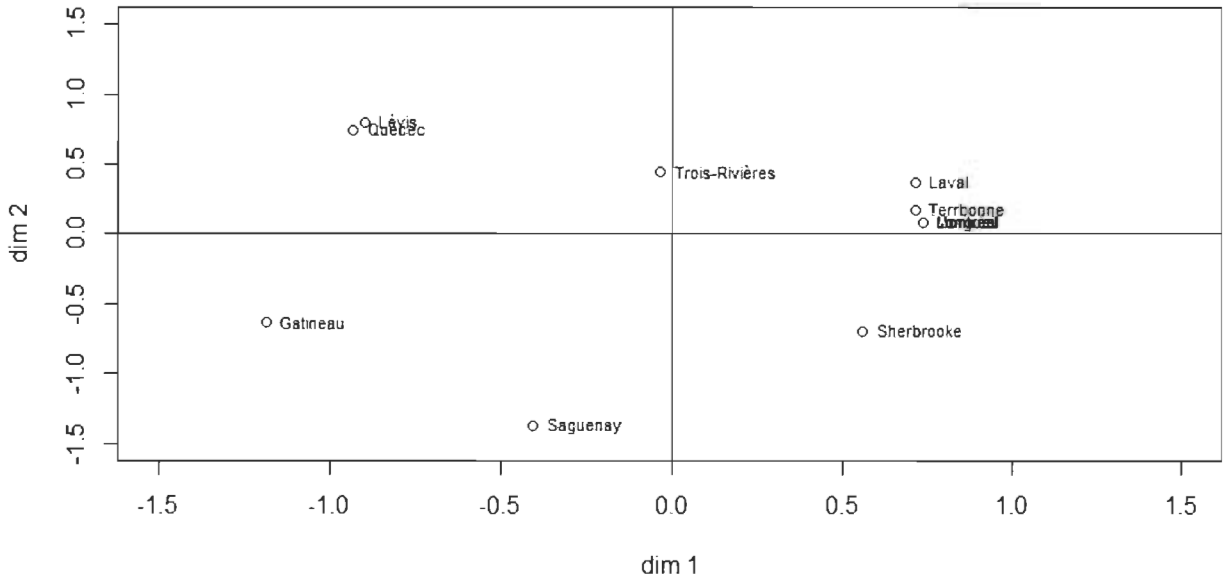

FIGURE 3.2 – Distance entre les villes - Représentation de la configuration de sortie en deux dimensions par la méthode non métrique

Il ressort de l'analyse de cette figure que :

- Lévis est plus proche de Québec par rapport aux autres villes,
- Laval, Montréal, Terrebone et Longueil sont proches.

Deuxième partie

Réseaux de neurones artificiels et Autoencodeur. Etude comparative entre l'ACP et l'autoencodeur

Chapitre 4

Réseaux de neurones artificiels et autoencodeur

Dans ce chapitre, il s'agit de présenter la théorie nécessaire pour la compréhension d'un réseau de neurones et celle d'autoencodeur en particulier.

4.1 Généralités sur les réseaux de neurones artificiels

Dans cette section, nous nous référons principalement à [19].

4.1.1 Définition

Un réseau de neurones artificiel peut être défini comme étant un modèle dont la conception est inspirée du fonctionnement du neurone biologique. Ce dernier contient trois composantes essentielles à savoir (Voir figure 4.1) :

- les dendrites : elles constituent les entrées du neurone et servent à acheminer vers le corps du neurone des signaux électriques en provenance d'autres neurones ;
- le corps cellulaire accumule des charges électriques. Par la suite, le neurone peut être suffisamment excité (i.e la charge électrique dépasse un certain seuil) à tel autoencoder qu'il engendre un potentiel électrique se propageant à travers son axone afin exciter d'autres neurones ;
- l'axone : elle représente la sortie du neurone vers d'autres neurones. Le autoencoder de contact entre l'axone d'un neurone et le dendrite d'un autre neurone s'appelle le synapse.

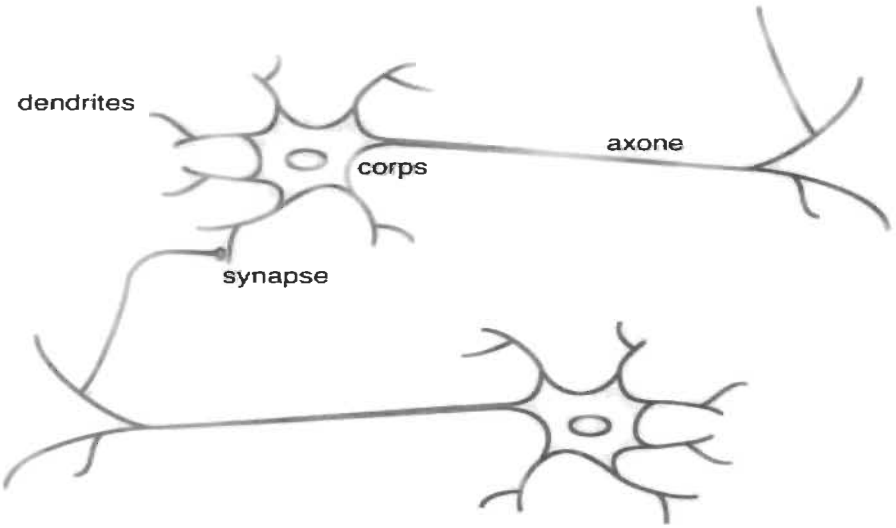

FIGURE 4.1 – Neurone biologique [19]

Parallèlement au neurone biologique, tel qu'illustré par la figure 4.2, un neurone artifi-

ciel reçoit sous forme vectorielle des entrées x_1,x_2,...,x_p, puis effectue une combinaison affine de ses entrées moins le seuil d'activation du neurone ou biais b ($w_{i,1}x_1 + w_{i,2}x_2 + ... + w_{i,p}x_p$ - b). Ensuite une fonction d'activation f est appliquée sur cette sortie : $f(w_{i,1}x_1 + w_{i,2}x_2 + ... + w_{i,p}x_p - b) = f(\sum_{k=1}^{p} w_{i,p}x_i - b)$ afin de créer une sortie y. Les paramètres $w_{i,1}, w_{i,2}, ..., w_{i,p}$ et b représentent respectivement les poids et le biais du neurone.

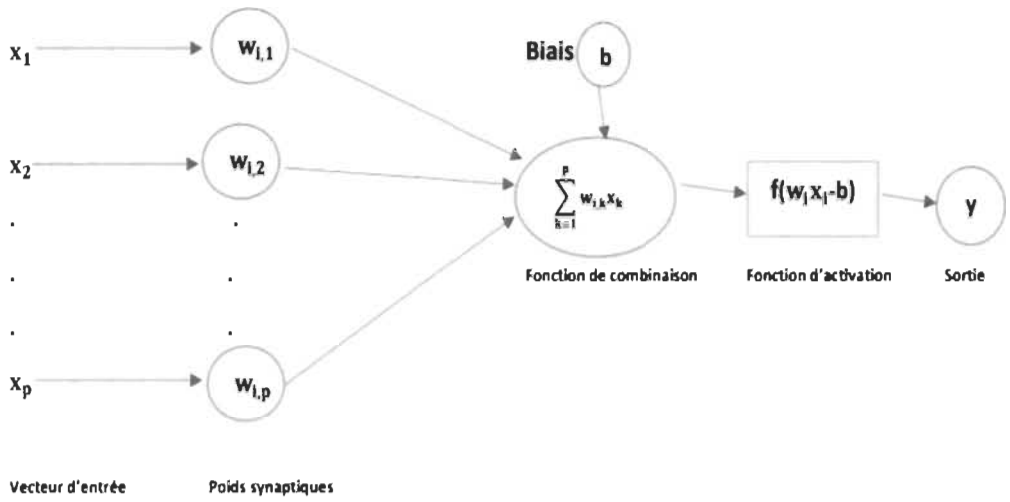

FIGURE 4.2 - Neurone artificiel

Un ensemble de neurones reliés en réseau forme ce qu'on appelle un réseau de neurones. Les poids et le biais sont estimés lors de la phase d'apprentissage du réseau de neurones.

Il existe deux types d'apprentissage des réseaux de neurones : l'apprentissage supervisé et l'apprentissage non supervisé. L'apprentissage supervisé est caractérisé par la présence d'un professeur ayant des connaissances sur l'environnement dans lequel évolue le réseau. Cependant l'apprentissage non supervisé s'effectue sur des données dont on ne connait pas les étiquettes des observations. Il est caractérisé par l'absence de professeur. Il s'agira donc de regrouper les individus statistiques présentant des

caractéristiques communes.

4.1.2 Fonction d'activation

Plusieurs fonctions d'activation sont utilisées dans les réseaux de neurones artificiels. Parmi elles, nous pouvons citer entre autres :

— la fonction seuil $f(x) = \mathbf{1}_{[0,+\infty]}$ (Voir figure 4.3) ;

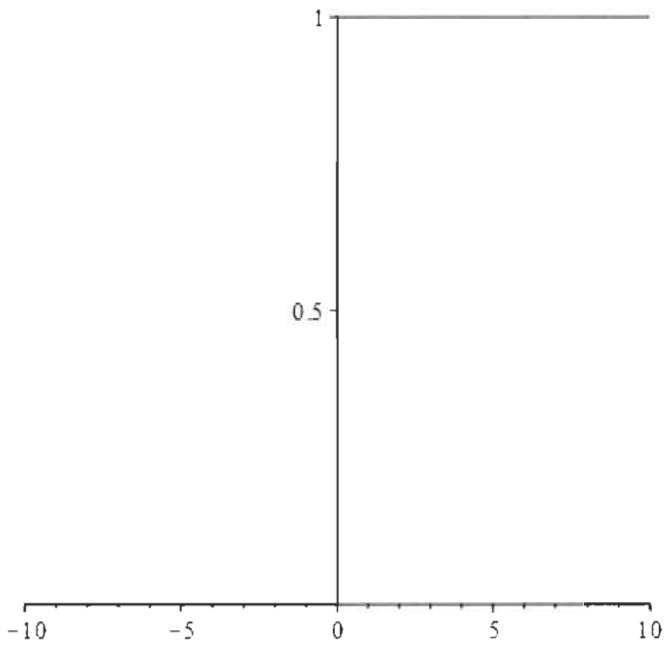

FIGURE 4.3 – Fonction seuil

— la fonction linéaire $f(x) = ax$, $a \in \mathbb{R}^*$, $x \in \mathbb{R}$ (Voir figure 4.4) ;

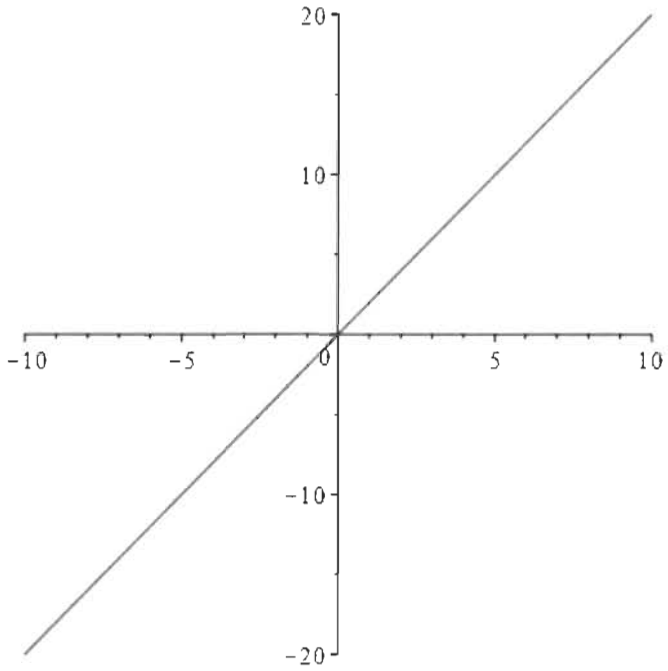

FIGURE 4.4 – Fonction linéaire

— la fonction sigmoïde $f(x) = \frac{1}{1+e^{-x}}$, $x \in \mathbb{R}$ (Voir figure 4.5) ;

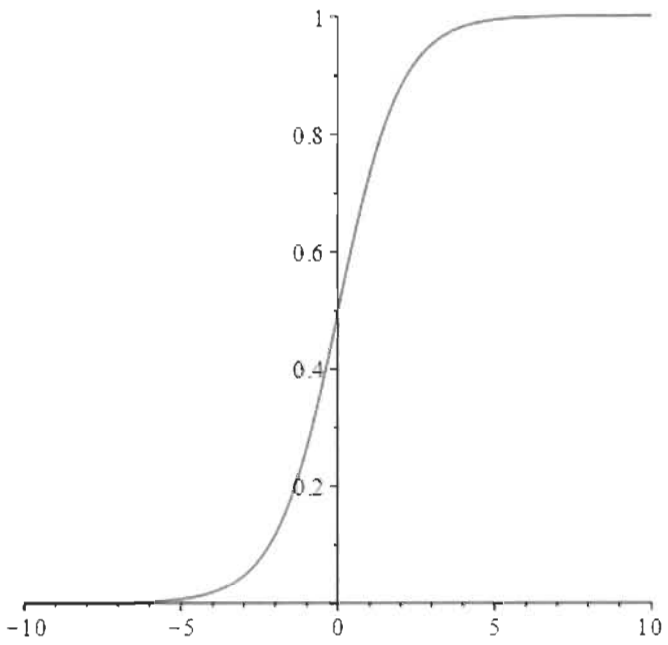

FIGURE 4.5 – Fonction sigmoïde

— la fonction ReLU (Unité de Rectification Linéaire) $f(x) = max(0, x)$, $x \in \mathbb{R}$ (Voir figure 4.6).

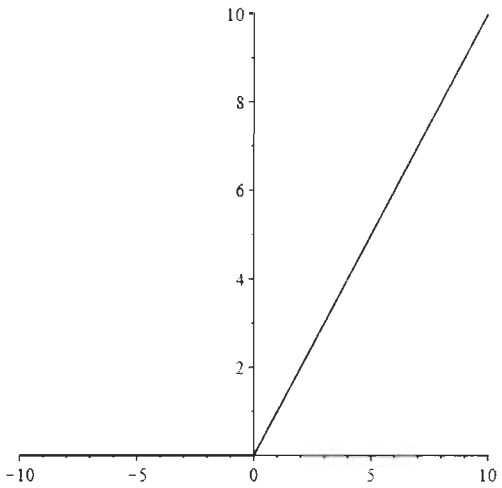

FIGURE 4.6 – Fonction RELU

4.1.3 Descente du gradient

L'entrainement d'un réseau de neurones est en général effectué par la méthode de la descente du gradient. Tel qu'illustré par la figure 4.7, la descente du gradient cherche à perfectionner le réseau en répétant une mise à jour des poids, visant à minimiser la fonction objectif. La fonction objectif est en général l'erreur quadratique moyenne :

$$E = \sum_{i=1}^{n} \mathcal{L}(x_i, y_i) = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2$$

$\mathcal{L}$ étant une fonction de coût permettant de mesurer la divergence entre l'échantillon d'apprentissage en entrée et les sorties.

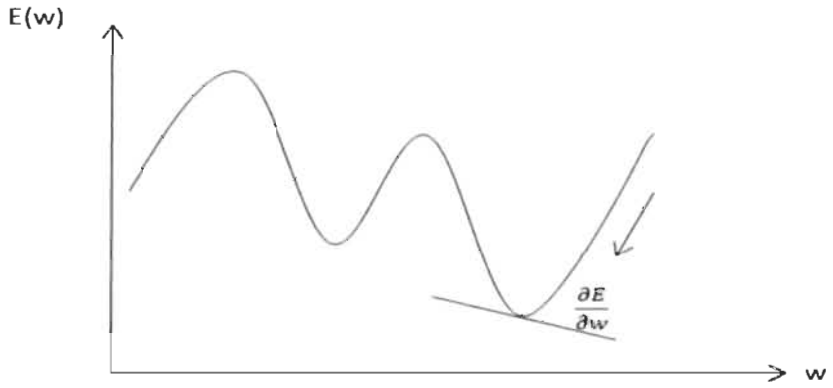

FIGURE 4.7 – La descente du gradient suivant W

Plus précisément, les étapes de l'algorithme de la descente du gradient sont les suivantes [23] :

1. Phase d'initialisation : on choisit aléatoirement la matrice des poids w ;
2. Avant qu'un processus d'apprentissage est effectué, on présente les données d'entrainement x_i au réseau par propagation pour obtenir les sorties y_i ;
3. On calcule la fonction objectif entre les x_i et y_i puis on la minimise par rapport aux poids w en résolvant l'équation $\frac{\partial E}{\partial w} = 0$
4. On effectue la rétro-propagation du gradient de la fonction objectif. Cette dernière consiste à la mise à jour des poids des neurones à partir de la couche de sortie vers la couche d'entrée afin de minimiser la fonction objectif.
5. On fait une itération de la mise à jour des poids jusqu'à ce que la fonction objectif convergence vers un minimum global.

Par abus de langage, le terme rétro-propagation du gradient est aussi utilisé pour faire référence à la fois à l'algorithme de descente du gradient et à la rétro-propagation du gradient.

4.2 Autoencodeur

4.2.1 Définition

Un autoencodeur est un réseau de neurones artificiels dans lequel la couche d'entrée a le même nombre de neurones que celle de la sortie. Il est utilisé pour l'apprentissage non supervisé et vise à reconstruire l'entrée avec le minimum d'erreur possible. En d'autres termes, un auto-encodeur apprend une approximation de la fonction identité. Comme pour l'ACP, une des principales applications des autoencodeurs est la réduction de dimensionnalité [15].

4.2.2 Architecture

On peut décomposer un autoencodeur en deux parties, à savoir un encodeur, f_θ, suivi d'un décodeur, g_ψ. L'encodeur permet de calculer le code $z_j = f_\theta(x_i)$ pour chaque échantillon d'apprentissage en entrée (x_{ij}), avec i allant de 1 jusqu'à n, n étant le nombre de lignes et j allant de 1 jusqu'à p, p étant le nombre de colonnes. Le décodeur vise à reconstituer l'entrée à partir du code z_i : $\widehat{x_i} = g_\psi(z_i)$. Les paramètres de l'encodeur et du décodeur sont appris simultanément pendant la tâche de reconstruction, tout en minimisant la fonction objectif :

$$\mathcal{J}_{AE}(\theta, \psi) = \sum_{i=1}^{n} \mathcal{L}(x_i, g_\psi(f_\theta(x_i))) = \sum_{i=1}^{n} \mathcal{L}(x_i, g_\psi(z_i)) = \sum_{i=1}^{n} \mathcal{L}(x_i, \widehat{x_i}).$$

$\mathcal{L}$ étant une fonction de coût permettant de mesurer la divergence entre l'échantillon d'apprentissage en entrée et les données reconstruites.

f_θ et g_ψ sont des fonctions de transition :

$f_\theta : \mathcal{X} \longrightarrow \mathcal{F}$

$g_\psi : \mathcal{F} \longrightarrow \mathcal{X}$ où $\mathcal{X}$ et $\mathcal{F}$ sont respectivement les ensembles d'entrée et de sortie de

l'encodeur ;

$f_\theta, g_\psi = \arg\min_{\theta,\psi} \|x - (f_\theta o g_\psi)(x)\|^2$ où $(f_\theta o g_\psi)(x) = f_\theta[g_\phi(x)]$ pour tout $x \in \mathcal{X}$;
Dans le cas où il n'y a qu'une seule couche cachée, l'étape d'encodage prend l'entrée $x \in \mathbb{R}^p = \mathcal{X}$ et l'associe à $z \in \mathbb{R}^k = \mathcal{F}$, $p \geq k$:

$$z = f_\theta(Wx + b)$$

où z est généralement appelé code, variable latente ou représentation latente, θ est une fonction d'activation (e.g., sigmoïde, ReLU...), W est une matrice de poids du réseau de neurones et b un vecteur de biais.
Ensuite, l'étape de décodage associe z à la reconstruction $\widehat{x}$ de forme identique à x :

$$\widehat{x} = g_\psi(W'z + b')$$

où W' et b' pouvant être différents ou non de W et b de l'encodeur, selon la conception de l'autoencodeur. La figure 4.8 illustre l'architecture d'un autoencodeur.

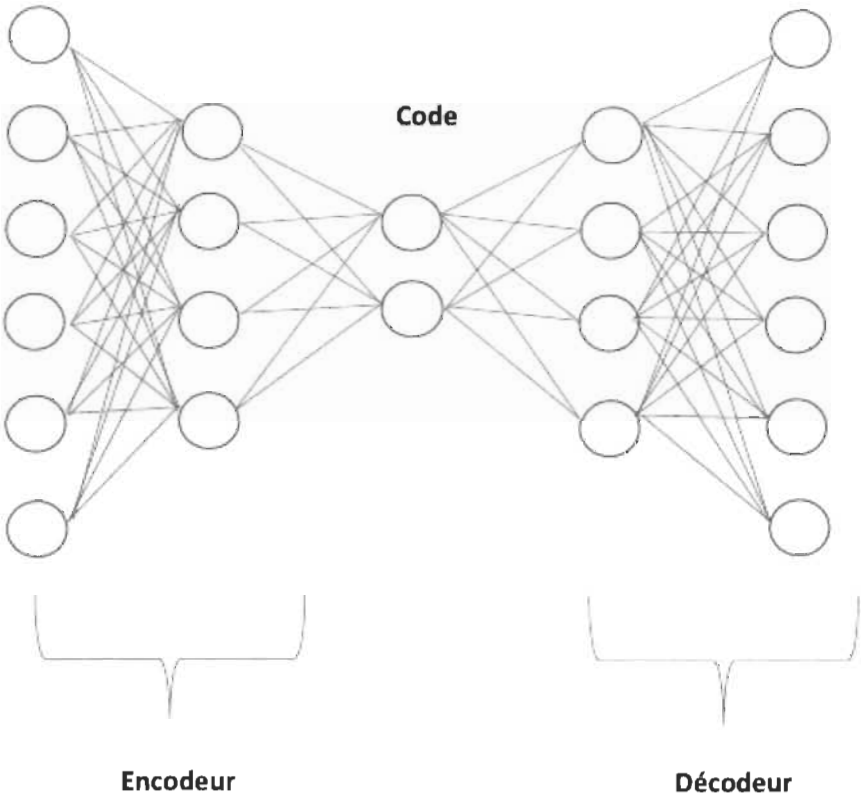

FIGURE 4.8 – Architecture d'un autoencodeur avec 2 couches d'encodeur et de décodeur et une couche cachée dite code

L'encodeur représente la partie du réseau qui compresse l'entrée dans un espace latent représentant la couche code.

Le code est la couche cachée qui est généralement représentée sous forme compressée dans une dimension réduite. Il constitue aussi l'entrée alimentée du décodeur.

Le décodeur est la partie du réseau qui tente de reconstruire l'entrée à partir de l'espace latent.

Exemple : Soit une entrée X sous forme d'une image en noir et blanc d'un chiffre manuscrit de dimension 28 par 28 soit 784 pixels. Chaque pixel est représenté par 0(noir) ou 1(blanc). L'encodeur procède au codage des données qui sont en 784 dimensions dans un espace de représentation latent Z de dimension plus petite. C'est ce qu'on appelle goulot d'étranglement. En effet, l'encodeur doit apprendre une compression efficace des données dans ce petit espace. Le décodeur prend pour entrée la représentation Z et sort les paramètres de la probabilité de distribution des données et le

biais b en cours d'exécution. La probabilité de la distribution d'un seul pixel peut être représentée par exemple à l'aide d'une loi de Bernoulli. La distribution du décodeur obtient en entrée une représentation latente du chiffre et en sortie 784 paramètres de Bernoulli pour chacun des 784 pixels dans l'image représentée dans Z. Une partie de l'information est perdue en ce sens qu'on part d'un espace de petite dimension vers un espace de grande dimension.

4.2.3 Lien avec l'ACP

L'ACP peut être vue comme un autoencodeur linéaire avec $W \in \mathcal{R}^{p \times k}$ où $k \leq p$. En prenant $f_\theta(X) = XW$ et $g_\psi \circ f_\theta(X) = XWW^T$, l'ACP cherche à optimiser la fonction objectif $\|X - XWW^T\|^2$. L'ACP est donc un autoencodeur dont la fonction d'activation est linéaire et la fonction de coût est l'erreur quadratique moyenne. Cependant, un autoencodeur utilisant des activations non linéaires a une capacité beaucoup plus grande et peut apprendre des représentations plus complexes. L'autoencodeur est donc une généralisation de l'ACP. Il est une forme d'ACP non linéaire [1]. Cette forme d'ACP ne sera pas développée dans ce mémoire mais plutôt comme travail futur.

4.2.4 Paramètres à définir pour entraîner un autoencodeur

Comme pour le réseau de neurones en général, l'algorithme de descente du gradiant est souvent utilisé pour entrainer un autoencodeur. Quatre paramètres sont à prendre en compte pour entrainer un autoencodeur :

- La taille du code (couche cachée) qui représente le nombre de neurones dans la couche code.
- Le nombre de couches : un autoencodeur peut avoir une ou plusieurs couches

— Le nombre de neurones par couches qui, en général, diminue dans l'encodeur et augmente dans le décodeur (symétrie par rapport à la couche code de l'encodeur et du décodeur).

— La fonction objectif : Un autoencodeur est aussi entrainé pour minimiser la fonction objectif. Cette dernière est en général l'erreur quadratique. Dans le cas où les données d'entrée sont sous forme de probabilités (compris entre 0 et 1), la fonction de perte la plus adaptée est l'entropie croisée.

— Le nombre d'époques :C'est le nombre de fois que l'on entraine le réseau de neurones avec toutes les données.

4.2.5 Types d'autoencodeurs

Nous pouvons distinguer deux types d'autoencodeurs selon la contrainte supplémentaire imposée pour limiter la capacité de représentation de l'autoencodeur lors de l'optimisation de la fonction objectif. Nous avons, d'une part, les autoencodeurs « undercomplete » qui limitent la dimension du code latent et, d'autre part, les autoencodeurs « overcomplete ». Ces derniers ajoutent un terme de régulation dans la fonction objectif.

Autoencodeurs « undercomplete »

Les autoencodeurs « undercomplete » sont des autoencodeurs dans lesquels on impose une contrainte au niveau de la couche code afin d'avoir des représentations utiles. Cette contrainte consiste à limiter la taille du code. Dans ce cas, comme l'ACP, l'autocodeur extrait la principale caractéristique des données en réduisant la dimension du jeu de données. L'ACP est donc un autoencodeur « undercomplete » utilisant une fonction d'activation linéaire [9].

Autoencodeurs épars

Les autoencodeurs épars utilisent une méthode alternative pour introduire le goulot d'étranglement sans nécessiter une réduction du nombre de nœuds au niveau des couches cachées. Plus précisément, il s'agit de construire une fonction objectif en pénalisant les activations dans une couche compte tenu de l'observation en question. Donc le réseau de neurones effectue l'encodage et le décodage en se fondant sur l'activation d'un certain nombre de neurones. Une des stratégies consiste à ajouter un terme supplémentaire dans la fonction objectif pendant l'entraînement afin de pénaliser la divergence de Kullback-Leibler entre les marginaux des unités cachées $\widehat{\rho_j}$ et un taux de parcimonie souhaité ρ [2] :

$$\mathcal{J}_{AE}(\theta, \psi) = \sum_{i=1}^{n} \mathcal{L}(x_i, \widehat{x_i})) + \lambda \sum_{j=1}^{m} KL(\widehat{\rho_j}||\rho)$$

où $\mathcal{L}$ étant une fonction de coût permettant de mesurer la divergence entre l'échantillon d'apprentissage en entrée et les données reconstruites, $\widehat{\rho_j} = \sum_{i=1}^{n} z_j(x_i)$ représente également l'activation moyenne de l'unité cachée j pour l'entrée x_i sur la distribution de données et z_j est l'activation de l'unité cachée j.
$KL(\widehat{\rho_j}||\rho) = \rho[log\frac{\rho}{\rho_j}] + (1-\rho)[log\frac{1-\rho}{1-\rho_j}]$ la divergence de Kullback-Leibler entre un paramètre de rareté ρ et son approximation $\widehat{\rho_j}$.
Une autre approche consiste à ajouter dans la fonction objectif un terme de régularisation qui pénalise la norme L_1 du code latent pour toute entrée x donnée. Dans ce cas, la fonction objectif sera :

$$\mathcal{J}_{AE}(\theta, \psi) = \sum_{i=1}^{n} \mathcal{L}(x_i, \widehat{x_i}) + \lambda \|z\|_1$$

$\mathcal{L}$ étant une fonction de coût permettant de mesurer la divergence entre l'échantillon d'apprentissage en entrée et les données reconstruites, L_1 est la norme des fonctions à valeurs dans $\mathbb{R}$ dont la valeur absolue est intégrable au sens de Lebesgue.

Autoencodeurs débruiteurs

Une autre façon de limiter la capacité de représentation d'un autoencodeur consiste à ajouter du bruit aux données d'entrée pendant l'entraînement afin d'avoir le plus possible des sorties proches des entrées. L'autoencodeur est entraîné pour reconstruire son entrée x à partir de sa version corrompue $\tilde{x}$ [26, 25].

$$\mathcal{J}_{AE}(\theta, \psi) = \sum_{i=1}^{n} \mathcal{L}(x_i, g_\psi(f_\theta(\tilde{x}_i))$$

$\mathcal{L}$ étant une fonction de coût permettant de mesurer la divergence entre l'échantillon d'apprentissage en entrée et les données reconstruites.

Autoencodeurs contractifs

L'auto-encodeur contractif est un autoencodeur dans lequel on pénalise la norme de Frobenius de la matrice Jacobienne des activations de l'encodeur par rapport à l'entrée en ajoutant un terme supplémentaire dans la fonction objectif afin d'apprendre des représentations utiles : $\mathcal{J}_{AE}(\theta, \psi) = \sum_{i=1}^{n} \mathcal{L}(x_i, \widehat{x_i}) + \lambda \sum_{i=1}^{m} \|\nabla_x z_i\|_F^2$, $\mathcal{L}$ étant une fonction de coût permettant de mesurer la divergence entre l'échantillon d'apprentissage en entrée et les données reconstruites, z_i l'activation de l'unité cachée i et m est le nombre d'unités cachées. La norme de Frobenius est donc définie comme suit : $\|A\|_F = tr(A^*A)$ avec $A \in \mathcal{M}_{m,n}$, A^* la matrice adjointe de A [21].

4.2.6 Application de l'autoencodeur aux Iris de Fisher

Nous avons appliqué l'autoencodeur undercomplete sur le jeu de données Iris pratiqué sur l'ACP dans le chapitre 2. Nous l'avons effectué sur Python (Voir script Python en annexe D). L'autoencodeur est entrainé avec 90% de l'échantillon initial. Les 10%

restants ont servi d'échantillon test afin de mesurer la performance du modèle avec la précision que nous désignerons par la suite « Accuracy »(ACC).

L'« accuracy » est le ratio entre le nombre d'observations bien prédites et le nombre total d'observations. Elle est donnée par la formule suivante :

$$ACC = max\frac{\sum_{i=1}^{n} 1(r_i = m(c_i))}{n}$$

où $1(.)$ est une fonction indicatrice, r_i l'étiquette de l'observation i (i allant de 1 à n), c_i la classe contenant i et $m(c_i)$ l'ensemble des possibilités d'affectation des observations dans la classe c_i.

Nous avons choisi 4 neurones dans la couche d'encodeur, 2 ou 3 neurones dans la couche code et 4 neurones dans la couche de décodeur. Après une série de simulation faisant varier les fonctions d'activation, le tableau 4.1 montre que la meilleure précision obtenue est de 0,760 avec la fonction d'activation Relu avec 3 neurones dans la couche code. Cela signifie que pour 1000 observations 760 sont bien classées.

	3 neurones	2 neurones
Autoencodeur linéaire	0,400	0,233
Autoencodeur Relu	0,760	0,400

TABLE 4.1 – Iris de Fisher - Précision avec 2 ou 3 neurones dans la couche code

La figure 4.9 indique l'évolution de l'erreur quadratique moyenne en fonction du nombre d'époques. Il en ressort que 125 époques suffisent pour entrainer l'autoencodeur.

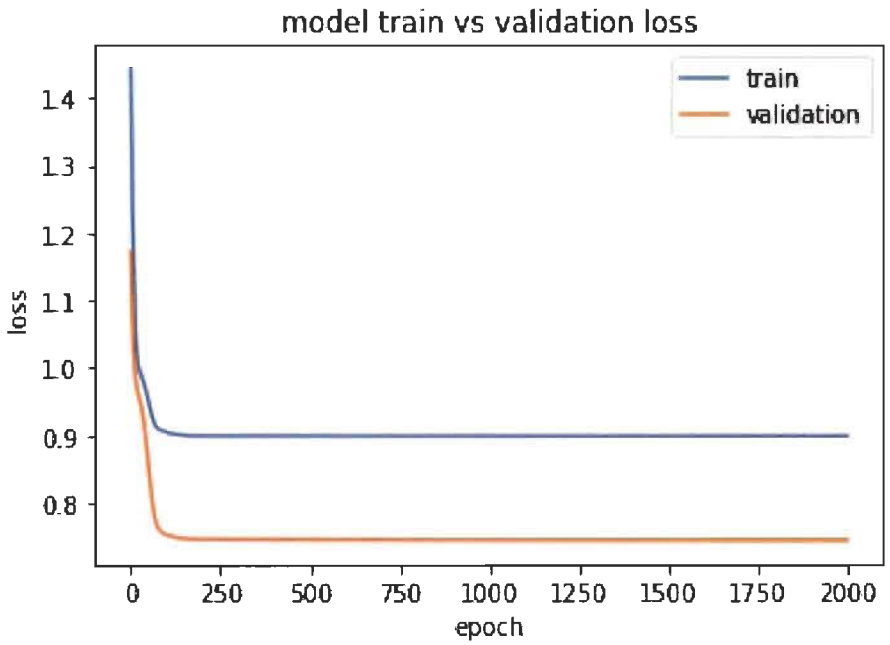

FIGURE 4.9 – Iris de Fisher - Erreur quadratique moyenne en fonction du nombre d'époques

Chapitre 5

Etude comparative de l'ACP et de l'Autoencodeur

Ce chapitre est consacré à une analyse comparative de l'analyse en composantes principales et de l'autoencodeur à l'aide de deux jeux de données. Il comporte deux sections portant sur deux applications respectivement sur les bases de données « Wine » qui est quantitative et « Mnist » qui est un ensemble d'images. Pour chacune de ces sections, nous présenterons d'abord le jeu de données puis décrirons la démarche utilisée. Ensuite, nous comparerons les deux méthodes à l'aide des résultats obtenus après leurs applications sur les données.

5.1 Mise en œuvre à l'aide des données « Wine »

5.1.1 Présentation des données et de la démarche utilisée

Nous avons eu cet ensemble de données sur le vin rouge et le vin blanc sur le site kaggle (www.kaggle.com). Ce dernier était utilisé par Digitas Advanced Analytics dans le but d'appréhender les relations entre certaines propriétés physiochimiques et la qualité perçue des vins afin de prendre des décisions plus éclairées pendant la production. Le jeu de données comporte 6497 observations (1599 vins rouges et 4898 vins blancs) et 13 variables à savoir l'acidité fixe (quantitative), l'acidité volatile (quantitative), l'acide citrique (quantitative), le sucre résiduel (quantitative), la chlorure (quantitative), le dioxyde de soufre libre (quantitative), le dioxyde de soufre total (quantitative), la densité (quantitative), le pH (quantitative), le sulfate (quantitative), l'alcool (quantitative), la qualité (qualitative ordinale) et le style (qualitative nominale) avec deux modalités possibles : rouge (red) ou blanc (white). Toutes ces dernières sont d'unités différentes. Le tableau 5.1 présente les paramètres des variables quantitatives à savoir la moyenne, le minimum et le maximum.

Variables	moyenne	minimum	maximum
acidité fixe	3,800	7,220	15,900
acidité volatile	0,080	0,340	1,580
acide citrique	0,000	0,319	1,660
sucre résiduel	0,600	5,440	65,800
chlorures	0,009	0,056	0,611
dioxyde de soufre libre	1	30,53	289
dioxyde de soufre total	6	115,700	440
densité	0,987	0,995	1,039
pH	2,720	3,219	4,010
sulfates	0,22	0,531	2
alcool	8	10,49	14,90

TABLE 5.1 – Wine - Moyenne, minimum et maximum des variables quantitatives

La qualité du vin varie sur une échelle de 3 à 9 (voir Tableau 5.2).

Modalités	3	4	5	6	7	8	9
Nombre	30	216	2138	2836	1079	198	5

TABLE 5.2 – Wine - Qualité du vin

Dans cette application, nous avons choisi d'enlever les variables suivantes : dioxyde de soufre libre et qualité. En effet, le dioxyde de soufre libre n'est pas retenu car il est une composante du dioxyde de soufre total. Donc ces deux variables sont fortement corrolées. C'est ce qui nous oblige à retenir l'une d'entre elles. Par ailleurs, la qualité du vin est une variable qualitative ordinale. Or l'ACP ne s'applique que sur des variables quantitatives. Ainsi, pour ne pas fausser la comparabilité de l'ACP et de l'autoencodeur, il convient de ne pas tenir en compte la qualité du vin. Finalement, on se retrouve avec 10 variables quantitatives et la variable qualitative type de vin qu'on utilise juste pour la mise en élèments supplémentaires.
La démarche utilisée permet d'évaluer la performance des deux méthodes en matière de réduction de dimensionnalité. Elle consiste à réduire la dimension du jeu de données avant d'appliquer un k-means sur les composantes principales retenues s'il s'agit d'une ACP ou sur la couche code lorsque la méthode utilisée est l'autoencodeur. Une normalisation des variables est effectuée afin de supprimer l'effet des unités sur l'ensemble des variables. Elle s'est effectuée en retranchant chaque variable par sa moyenne puis en divisant cela par l'écart-type de ladite variable, soit $norm(x) = \frac{x-\bar{x}}{s_x}$.

5.1.2 Présentation des résultats

Les résultats obtenus permettront de comparer les performances de l'ACP et de l'autoencodeur (Voir script R en annexe E). En se fondant sur le critère de Kaiser, l'ACP montre que les trois premières composantes principales peuvent être retenues. En effet, ces composantes principales ont chacune des valeurs propres supérieures à 1 (Voir Tableau 5.3). Donc nous allons retenir ces trois composantes principales puis évaluer, à l'aide d'un kmeans, la capacité de réduction de dimension de l'ACP.

Axes	Valeurs propres	% de variance	Cumul des % de variance
Axe 1	2,61	26,10	26,10
Axe 2	2,45	24,50	50,60
Axe 3	1,53	15,31	65,90
Axe 4	0,89	8,88	74,78
Axe 5	0,70	6,98	81,76
Axe 6	0,57	5,70	87,46
Axe 7	0,51	5,08	92,54
Axe 8	0,43	4,27	96,81
Axe 9	0,29	2,86	99,67
Axe 10	0,03	0,33	100,00

TABLE 5.3 – Wine - Valeurs propres et pourcentages de variation des composantes principales

La figure 5.1 illustre l'analyse par ACP où les données sont représentées par un nuage de points sur les deux premiers axes principaux (PC1, PC2).

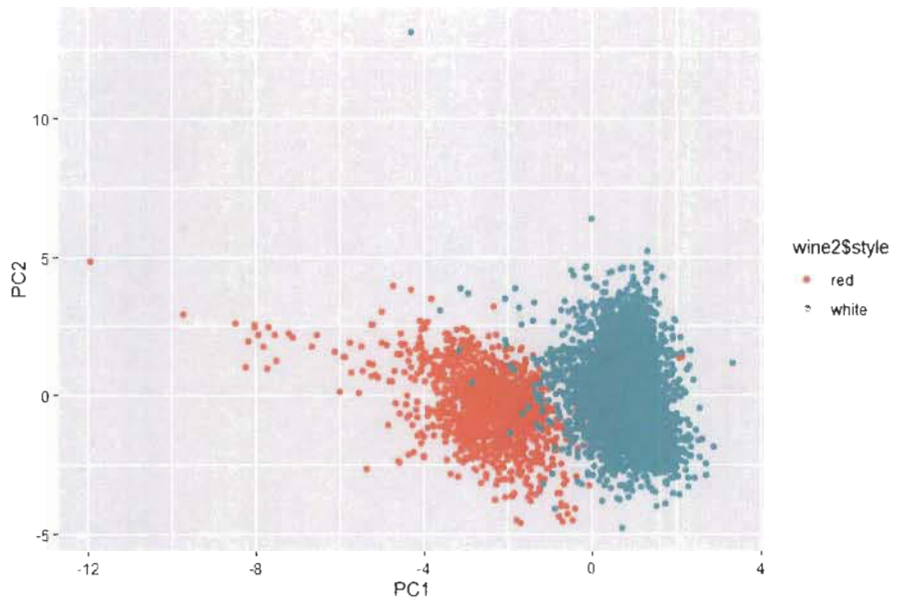

FIGURE 5.1 – Wine - L'Analyse par ACP : Nuage des points sur les 2 premiers axes principaux avec le style du vin en élèments supplémentaires

L'erreur quadratique moyenne est de 0,3409.

Le kmeans appliqué sur les 3 premières composantes principales montre un pourcentage de variance inter-classe de 32,1%.

Pour des fins de comparaisons, le nombre de composantes principales retenues va également correspondre au nombre de neurones sur la couche code de l'autoencodeur.

-Analyse par l'autoencodeur

L'autoencodeur constitué est composé de 3 couches dont une couche d'encodeur de 10 neurones correspondant aux 10 variables quantitatives, une couche code de 3 neurones et une couche de décodeur de 10 neurones afin de pouvoir reconstituer les données de départ. Puisque les variables sont quantitatives positives, il convient de choisir l'erreur quadratique moyenne comme fonction objectif. En ce qui concerne la fonction d'activation, il est préférable d'utiliser une fonction qui renvoie à une valeur numérique positive et non une probabilité. En effet, les sorties d'un autoencodeur doivent être proches des valeurs d'entrée. Par conséquent, nous allons entraîner l'autoencodeur avec les fonctions d'activation linéaire et Relu. C'est ce qui nous pousse à considérer les quatre types d'autoencodeurs suivants :

-L'autoencodeur Relu-Relu : les deux fonctions d'activation sont des fonctions Relu.

-L'autoencodeur Relu-linéaire : la première fonction d'activation est Relu alors que la seconde est linéaire.

-L'autoencodeur linéaire-Relu : la première fonction d'activation est linéaire alors que la seconde est Relu.

-L'autoencodeur linéaire-linéaire : les deux fonctions d'activation sont linéaires.

L'estimation des paramètres s'est effectuée par le biais de la descente du gradient. Le choix du nombre d'époques a été fait de façon empirique en visualisant l'évolution de l'erreur quadratique moyenne en fonction du nombre d'époques (Voir Tableau 5.4).

	Relu-Relu	Relu-linéaire	linéaire-Relu	linéaire-linéaire
EQM avec 500 époques	0,6553045	0,3416634	0,6001859	0,3409629
EQM avec 1000 époques	0,6550702	0,3412146	0,600049	0,3409684
EQM avec 1500 époques	0,6551351	0,3410933	0,6000681	0,3409539
EQM avec 2000 époques	0,6551502	0,3410155	0,6001366	0,3409944
EQM avec 2500 époques	0,655207	0,3410549	0,6000824	0,3409493

TABLE 5.4 – Wine - Erreur quadratique moyenne selon différentes époques simulées et par type d'autoencodeur

Nous avons entraîné chaque autoencodeur pour différentes valeurs du nombre d'époques. Ensuite, nous avons retenu ceux qui fournissent les erreurs quadratiques moyennes les plus petites. Après simulation, nous avons maintenu :

-l'autoencodeur Relu-Relu avec 1000

-l'autoencodeur Relu-linéaire avec 2000

-l'autoencodeur linéaire-Relu avec 1000

-l'autoencodeur linéaire-linéaire avec 1500

Le tableau 5.5 indique la précision obtenue après entrainement et le pourcentage de variance inter-classe après application du k-means sur la couche code de chaque autoencodeur.

	ACC	% de variance inter-classe
Relu-Relu	0,4637525	39,9
Relu-linéaire	0,527936	41,2
linéaire-Relu	0,5424042	32,6
linéaire-linéaire	0,5305526	33,2

TABLE 5.5 – Wine - Précision et pourcentage de variance inter-classe selon le type d'autoencodeur

On s'intéresse à l'autoencodeur Relu-linéaire car il donne le plus grand pourcentage de variance inter-classe (41,2%) avec une bonne précision (0,528). La figure 5.2 illustre l'analyse par l'autoencodeur Relu-linéaire où les données sont représentées par un nuage de points sur l'espace latent, défini par la couche code, en 2 dimensions (Neurone 1, Neurone 2).

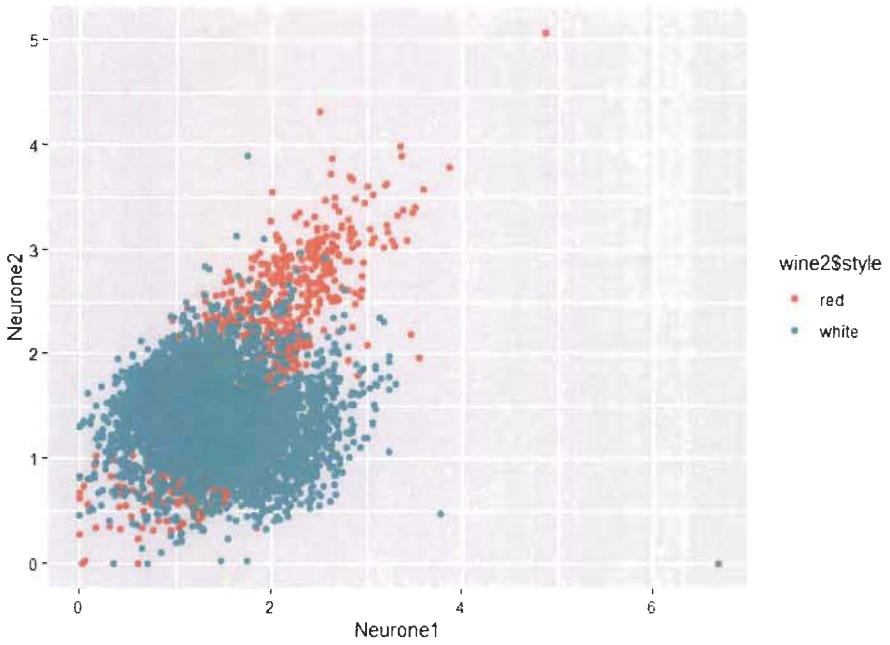

FIGURE 5.2 – Wine - L'Analyse par autoencodeur Relu-linéaire : Nuage de points sur l'espace latent en deux dimensions avec le style du vin en éléments supplémentaires

L'analyse des résultats montre que l'autoencodeur Relu-linéaire et Relu-Relu différencient mieux les vins rouges des vins blancs par rapport aux autoencodeurs linéaire-linéaire et linéaire-Relu et l'ACP. En effet, les pourcentages de variance inter classe les plus élevés sont notés avec l'autoencodeur Relu-linéaire (41,2%) et l'autoencodeur Relu-Relu (39,9%). Par ailleurs, en ce qui concerne l'autoencodeur linéaire-linéaire et l'autoencodeur linéaire-Relu, nous observons des pourcentages de variance inter-classe proches et légérement supérieurs à celui de l'ACP (32,1% pour l'ACP, 33,2% pour l'autoencodeur linéaire-linéaire et 32,6% pour l'autoencodeur linéaire-Relu). Cette remarque permet de confirmer que l'ACP est proche de l'autoencodeur dont la fonction d'actvation dans la couche d'encodeur est linéaire.

Avec l'ACP, les k premières composantes principales peuvent être choisis en maximisant la variance du jeu de données. Néanmoins il n'existe pas de directives pour choisir la taille de la couche code de l'autoencodeur. Nous pouvons donc utiliser l'ACP comme guide pour connaître le nombre de neurones à retenir dans la couche code de l'autoencodeur. La structure de l'ACP réside aussi sur le fait que les autres composantes principales non retenues ne soient pas visibles puisqu'il y a perte d'information. Donc

nous ne captons pas toute l'information contenue dans le jeu de données. Par conte, l'autoencodeur garde la structure des données en compressant toute l'information sur la couche code.

5.2 Mise en œuvre à l'aide des données « Mnist »

5.2.1 Présentation des données et de la démarche utilisée

Tel qu'illustré par la figure 5.3, le jeu de données « Mnist »porte sur des images de chiffres manuscrits (http ://yann.lecun.com/exdb/mnist/).

FIGURE 5.3 – Mnist - Quelques images de chiffres de la base « Mnist » [14]

Il est souvent utilisé pour illustrer des méthodes d'apprentissage automatique. Il est composé de 70 000 images (en noir et blanc) dont 60 000 formant l'ensemble d'entraînement et 10 000 constituant l'ensemble de test. Chaque image est représentée par 785 variables. La première variable correspond au libellé de la variable allant de 0 à 9. Elle sera mise en éléments supplémentaires afin d'évaluer la performation de l'ACP et de l'autoencodeur en termes de séparabilité des images. Les 784 autres variables sont les valeurs des pixels de l'image du chiffre représentée en niveaux de gris allant de 0 à 255, de taille 28 x 28. Pour ce qui est de cette application, nous nous sommes focalisés uniquement sur les données d'entrainement de « Mnist ». Le tableau 5.6 présente la répartion des images des 10 chiffres sur l'ensemble d'entrainement.

Modalités	0	1	2	3	4	5	6	7	8	9
Nombre	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949

TABLE 5.6 – Mnist - Chiffres de 0 à 9

A la différence de l'application sur le jeu de données « Wine », nous avons normalisé les données à l'aide de la fonction minmax qui permet de ramener les variables à l'intervalle [0,1]. La fonction minmax est égale au rapport entre la valeur du pixel moins le minimum et le maximum moins le minimun, soit $norm(x) = \frac{x-min(x)}{max(x)-min(x)}$. Pour cette application, elle sera : $norm(x) = \frac{x}{255}$ car $min(x) = 0$ et $max(x) = 255$ où x est la valeur du pixel en niveaux de gris allant de 0 à 255. Comme pour la première application, nous avons appliqué un k-means sur les composantes principales retenues pour l'ACP et sur la couche code de l'autoencodeur avant d'évaluer la performance de ces méthodes en matière de réduction de dimensionnalité.

5.2.2 Présentation des résultats

Les résultats sont obtenus à l'aide du script R qui se trouve dans l'annexe F.

-Analyse par l'ACP

En appliquant l'ACP, nous constatons que les 44 premières composantes principales expliquent 80,33% de l'information contenue dans le jeu de données (Figure 5.4). Donc nous allons effectuer un k-means sur ces 44 composantes principales puis évaluer la capacité de réduction de l'ACP.

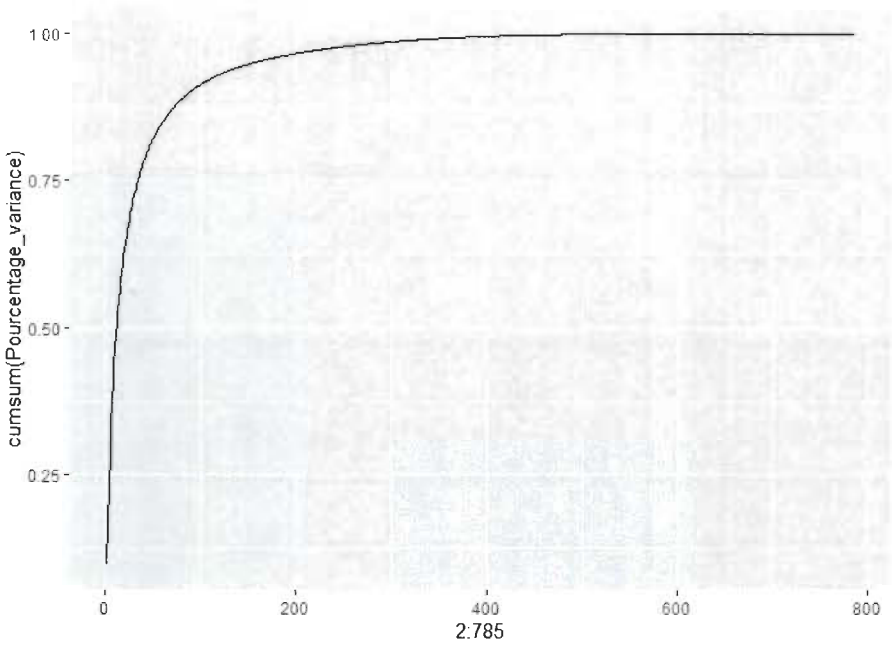

FIGURE 5.4 – Mnist - Evolution du cumul des pourcentages de variation sur les composantes principales

Le k-means appliqué sur les 44 premières composantes principales indique un pourcentage de variance inter-classe de 31,8%. La figure 5.5 illustre l'analyse par ACP où les données sont représentées par un nuage de points sur les deux premiers axes principaux (PC1, PC2).

Sa légende est une palette de couleurs de 0 à 9 contenant les couleurs associées aux 10 chiffres. Sur cette figure, nous arrivons à distinguer un peu les différents chiffres. Par exemple les 0 (couleur rouge foncée) sont plus concentrés à droite et les 6 (couleur bleue foncée) sont plus en bas à gauche.

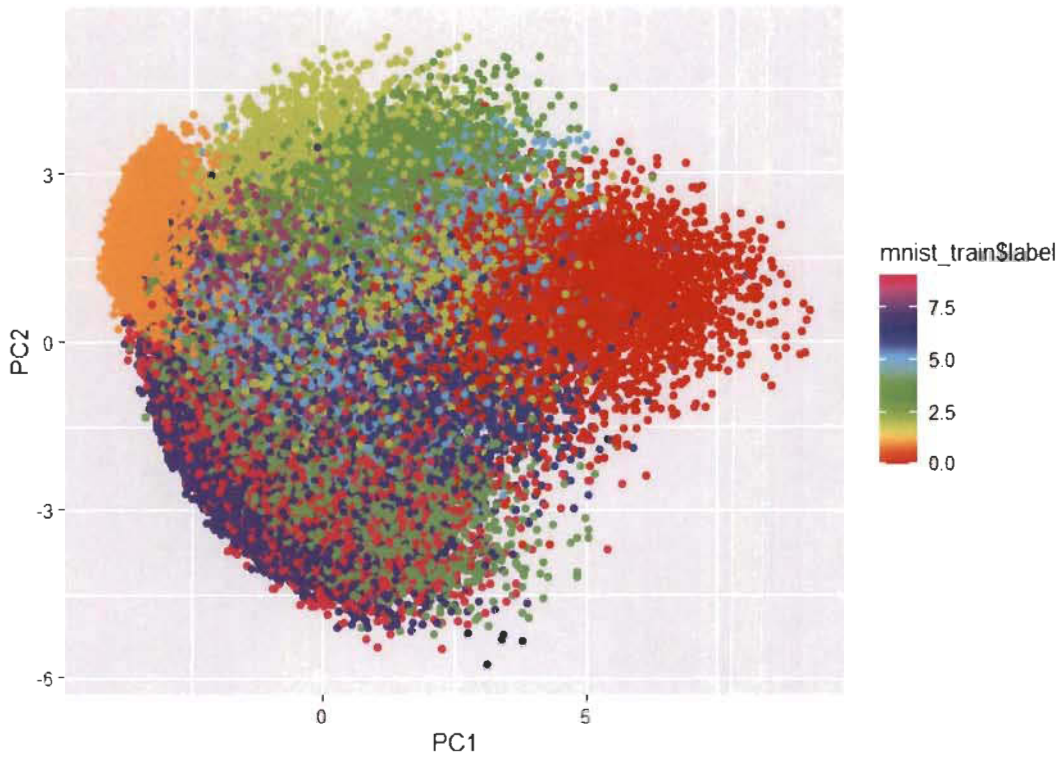

FIGURE 5.5 – Mnist - L'Analyse par ACP : Nuage des points sur les 2 premiers axes principaux avec le libellé du chiffre en élèments supplémentaires

-Analyse par l'autoencodeur

Comme pour la précédente application, nous avons retenu comme nombre de neurones dans la couche code de l'autoencodeur le nombre de composantes principales de l'ACP utilisées pour réduire la dimension du jeu de données, soit 44 neurones. L'autoencodeur à construire aura donc 3 couches dont une couche d'encodeur de 784 neurones, une couche code de 44 neurones et une couche décodeur de 784 neurones. Les données d'entrée étant sur l'intervalle [0,1], elles peuvent donc être vues comme des probabilités. Nous allons donc choisir la fonction d'activation sigmoide pour la couche de sortie afin que les sorties soient des probabilités ou la fonction d'activation Relu pour concerver les données sur [0,1]. Concernant la fonction objectif, il convient de choisir l'entropie croisée qui est la plus adaptée lorsque les sorties sont des probabilités. Nous avons deux possibilités pour les deux fonctions d'activation : la fonction

Relu et la fonction sigmoide. Pour pallier à cela, nous avons défini selon les fonctions d'activation utilisées les quatre autoencodeurs suivants :

-L'autoencodeur Relu-Relu : les deux fonctions d'activation sont des fonctions Relu.

-L'autoencodeur Sigmoide-Relu : la première fonction d'activation est sigmoide alors que la seconde est Relu.

-L'autoencodeur Sigmoide-Sigmoide : les deux fonctions d'activation sont sigmoides.

-L'autoencodeur Relu-Sigmoide :la première fonction d'activation est Relu alors que la seconde est sigmoide.

Le choix du nombre d'époques s'est fait de facon empirique. Nous avons simulé l'entrainement de chaque autoencodeur pour différentes époques, ensuite nous avons retenu l'autoencodeur qui fournit l'erreur quadratique moyenne la plus faible (Voir Tableau 5.7).

	Relu-Relu	Sigmoide-Relu	Sigmoide-Sigmoide	Relu-Sigmoide
EQM avec 50 époques	649,5458	676,9072	511,637	511,1364
EQM avec 100 époques	648,7354	670,8135	511,1794	510,9997
EQM avec 150 époques	669,6369	689,9105	510,8862	510,959
EQM avec 200 époques	691,218	652,4606	510,6389	510,8303
EQM avec 250 époques	692,8315	649,5458	510,639	510,8645

TABLE 5.7 – Mnist - Erreur quadratique moyenne selon différentes époques simulées et par type d'autoencodeur

Après quelques simulations, nous avons choisi les autoencodeurs suivants ayant les erreurs quadratiques moyennes les plus basses :

-l'autoencodeur Relu-Relu avec 100 époques,

-l'autoencodeur Sigmoide-Relu avec 250 époques,

-l'autoencodeur Relu-Relu avec 200 époques,

-l'autoencodeur Relu-Sigmoide avec 200 époques.

La précision est environ 1% (Voir tableau 5.8) pour chaque type d'entoencodeur.

	ACC	% de variance inter-classe
Relu-Relu	0.01073333	51.3
Sigmoide-Relu	0.009816667	27.5
Sigmoide-Sigmoide	0.01201667	25.4
Relu-Sigmoide	0.01185	29.0

TABLE 5.8 – Mnist - Précision et pourcentage de variance inter-classe selon le type d'autoencodeur

La figure 5.6 représente le nuage de points sur l'espace latent en 2 dimensions (Neurone 1, Neurone 2) de l'autoencodeur Relu-Relu. Sa légende représente une palette de couleurs allant de 0 à 9 et contient les couleurs associées aux 10 chiffres. Cependant, sur cette figure, il n'est pas facile et aisé de distinguer les chiffres car ces derniers semblent mêlés sur l'espace latent en 2 dimensions.

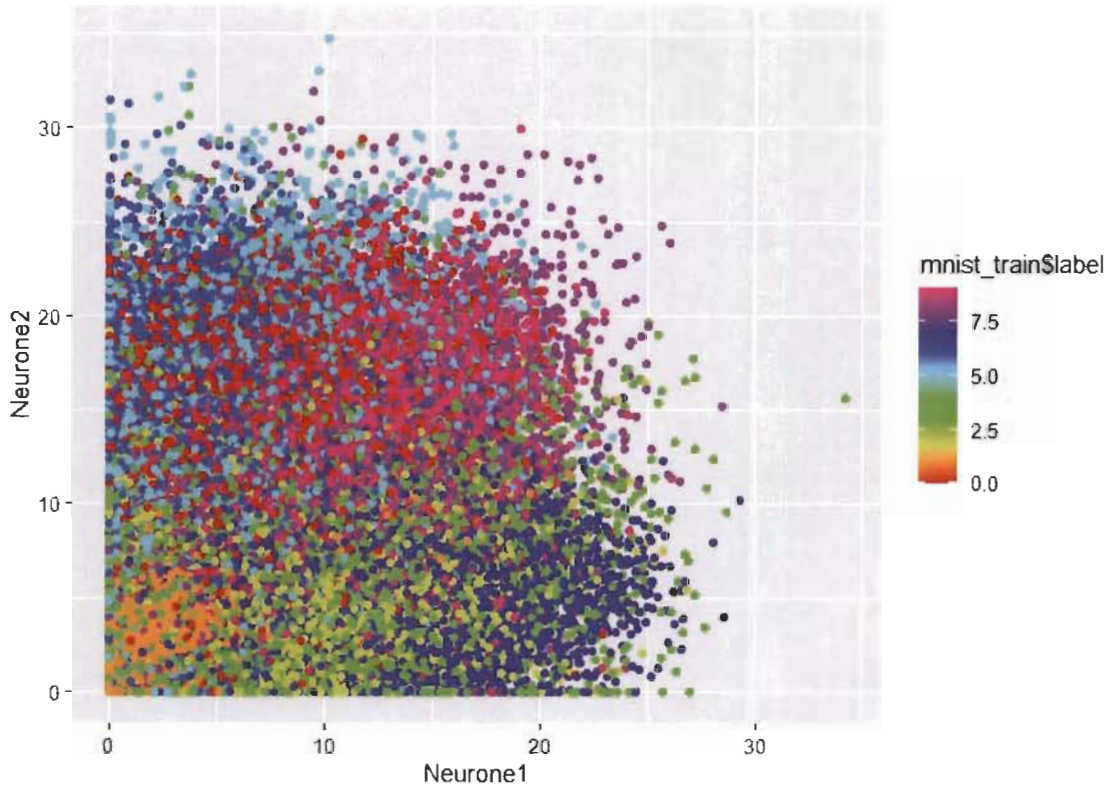

FIGURE 5.6 – Mnist - L'Analyse par autoencodeur Relu-Relu : Nuage de points sur l'espace latent en deux dimensions avec le chiffre en éléments supplémentaires

En se basant sur les simualations effectuées, nous observons que l'autoencodeur Relu-Relu est meilleur que l'ACP et les autres types d'autoencodeur en matière de dimen-

sionnalité. Cependant la dispersion est plus notée sur l'ACP que sur les autoencodeurs Relu-Sigmoide, Sigmoide-Relu et Sigmoide-Sigmoide. Donc l'ACP peut dans certains cas donner des résultats meilleurs que ceux fournis par l'autoencodeur. Il semble donc nécessaire de bien spécifier les paramètres de l'autoencodeur avant d'évaluer sa performance et de le comparer avec d'autres méthodes de réduction de dimensionnalité comme l'ACP.

Conclusion et perspectives

L'objectif de ce mémoire est d'établir un cadre de comparaison des méthodes statistiques et neuronales en matière de réduction de dimmensionnalité. Plus précisément, il s'agit d'effectuer le lien entre deux méthodes de réduction de dimensionnalité en l'occurence l'ACP qui est une méthode classique et l'autoencodeur qui est une méthode neuronale.

Nous avons comparé les deux méthodes, en appliquant un k-means sur la couche code de l'autoencodeur et les composantes principales retenues pour l'ACP, avec les bases de données « Wine » et « Mnist ». L'analyse des résultats révèle que l'autoencodeur, moyennant l'ajustement de plusieurs paramètres, est en général meilleur que l'ACP en matière de réduction de dimensionnalité. En particulier, lorsque la fonction d'activation dans la couche d'encodeur est linéaire, nous avons constaté que l'autoencodeur est proche de l'ACP. Cela confirme le lien théorique qui existe entre ces deux méthodes. En effet, ce lien stipule qu'un autoencodeur dont la fonction d'activation est linéaire et la fonction objectif est l'erreur quadratique moyenne a des performances similaires à celles de l'ACP. L'ACP est une méthode de réduction de dimensionnalité avec perte d'information alors que l'autoencodeur garde la structure des données en compressant toute l'information sur la couche code. L'autoencodeur nécessite l'ajustement de plusieurs paramètres que nous avons eus empiriquement, en particulier le nombre de neurones sur la couche code que l'ACP permet de déterminer. Ainsi, l'ACP peut servir de guide pour la détermination d'un tel nombre de neurones. Pour

ce qui est de la taille des données, il est préférable d'opter pour l'ACP pour les petits ensembles de données et l'autoencodeur pour les ensembles plus grands.

Comme perspectives, il serait intéressant d'élargir notre étude comparative à d'autres jeux de données. Il serait également envisageable de mener une étude comparative entre l'autoencodeur et l'ACP non linéaire qui est moins connue et prend en compte le caractère non linéaire des données.

Bibliographie

[1] S. Affeldt, L. Labiod, and M. Nadif. Spectral clustering via ensemble deep autoencoder learning (sc-edae). *Pattern Recognition*, 2014.

[2] A. Ng Andrew and al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011) :1-19, 2011.

[3] R. B. Cattell. The scree test for the number of factors. *Multivariate Behavioral Research*, pages 1,245–276, 1966.

[4] O. Donald D.O Hebb. The organization of behavior : A neuropsychological theory. 1949.

[5] N. Ghazzali. Notes de cours sur les méthodes d'analyse des données. *Université du Québec à Trois-Rivières*, 2018.

[6] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. 2016.

[7] J. C Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3-4) :325–338, 1966.

[8] H. Hamisultane. Econométrie. *France*, pages 18–21, 2002.

[9] G.E Hinton and R.R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786) :504–507, 2006.

[10] J. L. Horn. A rationale and test for the number of factors infactor analysis. *Psychometrika*, pages 30,179–186, 1965.

[11] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6) :417, 1933.

[12] H. F. Kaiser. The application of electronic computers to factor analysis. *Educational and Psychological Measurement*, pages 20,141–151, 1960.

[13] J. B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1) :1–27, 1964.

[14] Y. LeCun, B.E Boser, J. S Denker, D. Henderson, R. E Howard, W. E Hubbard, and L.D Jackel. Handwritten digit recognition with a back-propagation network. pages 396–404, 1990.

[15] M. Leyli-Abadi, L. Labiod, and M. Nadif. Denoising autoencoder as an effective dimensionality reduction and clustering of text data. pages 801–813, 2017.

[16] K. V. Mardia. Some properties of clasical multi-dimesional scaling. *Communications in Statistics-Theory and Methods*, 7(13) :1233–1241, 1978.

[17] W.S McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4) :115–133, 1943.

[18] M.L Minsky and S. Papert. Perceptrons : An introduction to computational geometry. 1969.

[19] M. Parizeau. Notes de cours sur les réseaux de neurones. *Université Laval*, 2009.

[20] K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11) :559–572, 1901.

[21] S. Rifai, G. Mesnil, P. Vincent, X. Muller, Y. Bengio, Y. Dauphin, and X. Glorot. Higher order contractive auto-encoder. *Université de Montréal*, pages 645–660, 2011.

[22] F. Rosenblatt. The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65 :386–408, 1958.

[23] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, pages vol. 323, no. 6088, 533–536, 1986.

[24] GAF Seber. Multivariate observations. *University of Auckland, New Zealand*, pages 175–176, 235–252, 1984.

[25] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. pages 1096-1103, 2008.

[26] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P. Manzagol, and L. Bottou. Stacked denoising autoencoders : Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.

[27] D. Yadolah and V. Rousson. Analyse de régression appliquée. *Dunod*, 1999.

Annexe A

Régression pas à pas sur le jeu de données Swiss avec Rstudio

```
# Importation de librairies nécessaires

library(tidyverse)

# Importation de librairies nécessaires

library(tidyverse)

library(caret)

library(leaps)


library(MASS)
```

```
# Application du modele complet

full.model <- lm(Fertility ~., data = swiss)

# Application du modele stepwise

step.model <- stepAIC(full.model, direction = "both",
trace = FALSE)

summary(step.model)
```

Annexe B

Application de l'ACP sur les Iris de Fisher avec Rstudio

```
# Installation de la librairie FactoMineR de la fonction PCA

library(FactoMineR)

# change le répertoire courant

setwd('C:/Users/aboub/OneDrive/Bureau/ADD-avec-Nadia')

# Affiche le répertoire de travail actuel

getwd()


# Importer de la base de donnees iris
```

```
iris

as.data.frame(iris)

# EXplication détaillée de la fonction PCA sur l'aide

?PCA

# Fonction PCA sur les 2 variables quantitatives de
la fonction PCA

iris.pca=PCA(iris[,1:5], ncp=4, quali.sup=5, graph=T)

# Valeurs propres, Pourcentage de variance,
 cumul du pourcentage de variance

iris.pca$eig

# Cosinus carré des variables

iris.pca$var$cos2

# Contribution des variables

iris.pca$var$contrib

# Correlations variables - dimensions

iris.pca$var$cor
```

```
# Nuage des variables

plot(iris.pca, choix = "var")


# Nuage des individus

plot(iris.pca, choix = "ind")

plot(exemple.pca, choix = "ind", habillage = "ind",
col.hab=rainbow(3)[iris$Species])
legend("top", legend = levels(iris$Species),
col = rainbow(3), pch = 19, cex = 0.75,
ncol = 3)
```

Annexe C

Positionnement multidimmensionnel avec RStudio sur 10 villes du Qc

```
# Importation de librairies

library(MASS)

library(vegan)

# N lignes x p colonnes (variables)

# Importation de la base de données

library(readxl)

Distance_villes_QC <- read_excel("C:/Users/aboub/OneDrive/Bureau/
```

```
These_UQTR_Original/Distance_villes_QC.xlsx",
col_types = c("text", "numeric", "numeric",
"numeric", "numeric", "numeric",
"numeric", "numeric", "numeric",
"numeric", "numeric"))

View(Distance_villes_QC)


d <- Distance_villes_QC[,-1]

d

# Transformation de la base de données en dataframe

as.data.frame(d)

View(d)

# Positionnement multidimensionnel classique avec la fonction cmdscale

fit <- cmdscale(d,eig=TRUE, k=2) # k est la dimension

fit

# Représentation graphique

x <- fit$points[,1]
y <- fit$points[,2]
plot(x, y,xlim=c(-500,500), ylim=c(-90,160), xlab="Y1", ylab="Y2") # ,
```

```
main="Metric MDS")
text(x, y, labels = colnames(d), cex=.7, pch=2, pos=4)
abline(h=0, v=0);


# Positionnement multidimensionnel non métrique avec la fonction
monoMDS

fit <- monoMDS(d, k=2)

fit

# Représentation graphique

x <- fit$points[,1]
y <- fit$points[,2]
plot(x, y, xlim=c(-1.5,1.5), ylim=c(-1.5,2), xlab="Y1",
ylab="Y2")#  ,main="Nonmetric MDS")
text(x, y, labels = colnames(d), cex=.7, pch=2, pos=4)
abline(h=0, v=0);
```

Annexe D

Application de l'autoencodeur sur les Iris de Fisher avec Python

```
# Importation des package et library

import sklearn.metrics as metrics
from tensorflow import keras
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
import sklearn
from sklearn.datasets import load_iris
from keras.models import Sequential, Model
from keras.layers import Dense, Input
import numpy as np
import matplotlib.pyplot as plt
from keras import regularizers
```

```
import tensorflow as tf
from sklearn.metrics import accuracy_score

# Importation de la base de données iris dans sklearn

iris = sklearn.datasets.load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names

# Normalisation des variables

X_scaled=(X-np.mean(X))/np.std(X)

X_scaled


# Creation de la fonction d'affichage


def plot3clusters(X, title, vtitle):
plt.figure()
colors = ['navy', 'turquoise', 'darkorange']
lw = 2

for color, i, target_name in zip(colors, [0, 1, 2], target_names):
plt.scatter(X[y == i, 0], X[y == i, 1], color=color, alpha=1., lw=lw,
label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title(title)
```

```
plt.xlabel(vtitle + "1")
plt.ylabel(vtitle + "2")
plt.show()


# Application de l'ACP


pca = sklearn.decomposition.PCA()
pca_transformed = pca.fit_transform(X_scaled)
plot3clusters(pca_transformed[:,:2], 'PCA', 'PC')

# Autoencodeur avec 3 couches et des fonctions lineaires

input_dim = X_scaled.shape[1]
encoding_dim = 2
input_img = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='linear')(input_img)
decoded = Dense(input_dim, activation='linear')(encoded)
autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mse')
print(autoencoder.summary())

history = autoencoder.fit(X_scaled, X_scaled,
epochs=1000,
batch_size=16,
shuffle=True,
validation_split=0.1,
verbose = 0)
```

```
# Representation de l'Erreur quadratique moyenne en fonction
du nombre d'époques

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model train vs validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()


encoder = Model(input_img, encoded)
encoded_input = Input(shape=(encoding_dim,))
decoder_layer = autoencoder.layers[-1]
decoder = Model(encoded_input, decoder_layer(encoded_input))
encoded_data = encoder.predict(X_scaled)

plot3clusters(encoded_data[:,:2], 'Linear AE', 'AE')

# Autoencodeur avec 3 couches et des fonctions Relu

input_dim2 = X_scaled.shape[1]
encoding_dim2 = 3
input_img2 = Input(shape=(input_dim2,))
encoded2 = Dense(encoding_dim2, activation='relu')(input_img2)
decoded2 = Dense(input_dim2, activation='relu')(encoded2)
autoencoder2 = Model(input_img2, decoded2)
autoencoder2.compile(optimizer='adam', loss='mse')
print(autoencoder2.summary())
```

```
history2 = autoencoder2.fit(X_scaled, X_scaled,
epochs=2000,
batch_size=16,
shuffle=True,
validation_split=0.1,
verbose = 0)

# Representation de l'Erreur quadratique moyenne en fonction
du nombre d'époques

plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('model train vs validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()


encoder2 = Model(input_img2, encoded2)
encoded_input2 = Input(shape=(encoding_dim2,))
decoder_layer2 = autoencoder2.layers[-1]
decoder2 = Model(encoded_input2, decoder_layer2(encoded_input2))
encoded_data2 = encoder2.predict(X_scaled)

plot3clusters(encoded_data2[:,:2], 'Non-Linear relu-based AE', 'AE')


#  Calcul de l'accuracy
```

```
labels_true = y
titles = ['PCA', 'Linear AE', 'Relu AE']
for n_clusters_ in [2,3]:
estimators = [('PCA'      , KMeans(n_clusters=n_clusters_),
pca_transformed),
('AE linear' , KMeans(n_clusters=n_clusters_), encoded_data),
('AE relu', KMeans(n_clusters=n_clusters_), encoded_data3)]

print(type(y))
print('Number of clusters: %d' % n_clusters_)
for name, est, data in estimators:
X = data
est.fit(X)
labels = est.labels_
print(name,':')
print(labels[:])
print("Accuracy: %0.3f" % metrics.accuracy_score(labels_true, labels))
print()
print()
print('----------------------------------------------------------------------------')
print()
```

Annexe E

L'ACP et l'Autoencodeur avec Rstudio : Données Wine

```
# Installation de packages et librairies

suppressPackageStartupMessages(library(DAAG))

library(ggplot2)

library(readr)

install_keras()

install_tensorflow()

library(tensorflow)

suppressPackageStartupMessages(library(keras))
```

```
# Importation de la base de données

datasets_4292_6628_wine_dataset <- read_csv("C:/Users
/aboub/Downloads/datasets_4292_6628_wine_dataset.csv")

View(datasets_4292_6628_wine_dataset)

wine  <- datasets_4292_6628_wine_dataset[1:11]


wine1  <- datasets_4292_6628_wine_dataset[,-6]

wine2  <- wine1[,-11]

# Transformation de la base en dataframe

wine2 <- as.data.frame(wine2)

wine2

wine2.class<- wine2[,"style"]


# Normalisation des variables

norme <- function(x) (x - mean(x))/sd(x)

x_train <- apply(wine2[,1:10], 2, norme)
```

```
x_train

# Application de l'ACP avec la fonction prcomp

pca <- prcomp(x_train)

# valeurs propres

pca$sdev^2

# Pourcentage de variance inter classe

a=pca$sdev^2/sum(pca$sdev^2)

a

# Cumul du Pourcentage de variance inter classe

cumsum(a)


# Graphique du Cumul du Pourcentage de variance inter classe

qplot(x = 1:10, y = cumsum(pca$sdev^2/sum(pca$sdev^2)), geom = "line")

# Mise en éléments supplémentaires du style de vin

ggplot(as.data.frame(pca$x), aes(x = PC1, y = PC2,
col = wine2$style)) + geom_point()
```

```
# Calcul de la reconstitution des variables avec l'ACP

mu <- matrix(rep(pca$center, nrow(pca$x)), nrow = nrow(pca$x),
byrow = T)

recon <- pca$x[,1:3] %*% t(pca$rotation[,1:3]) + mu

dim(recon)

recon

# Erreur quadratique moyenne de l'ACP

mse <- mean((recon - x_train)^2)

mse


# K-means sur les composantes principales de l'ACP


cl1=kmeans(pca$x[,1:3], 2, nstart=1000)

cl1

table(cl1$cluster,wine2.class)

# Autoencodeur avec la fonction keras_model_sequential()
```

```
x_train <- as.matrix(x_train)


modellin <- keras_model_sequential()
modellin %>%
layer_dense(units = 3, activation = "linear", name = "bottleneck",
input_shape = ncol(x_train)) %>%
layer_dense(units = 10, activation = "relu") %>%

# Résumé du modèle appliqué

summary(modellin)


# compilation du modèle

modellin %>% compile(
loss = "mean_squared_error", metrics="accuracy",
optimizer = "adam"
)

# Entrainement du modèle

modellin %>% fit(
x = x_train,
y = x_train,
epochs = 1000,
verbose = 0
)
```

```
# Mesures de performance du modèle

metric_binary_accuracy(y_true, y_pred)

mse.ae2 <- evaluate(modellin, x_train, x_train)

mse.ae2

# Extraction da couche code (goulot d'étranglement) et de la sortie

intermediate_layer_model <- keras_model(inputs = modellin$input,
outputs = get_layer(modellin, "bottleneck")$output)

intermediate_output <- predict(intermediate_layer_model, x_train)

# Mise en éléments supplémentaires du style de vin

ggplot(data.frame(Neurone1 = intermediate_output[,1],
Neurone2 = intermediate_output[,2]), aes(x = Neurone1,
y = Neurone2, col = wine2$style)) + geom_point()

# K-means sur la couche code de l'autoencodeur

cl2=kmeans(intermediate_output[,1:3], 2, nstart=1000)

cl2

cl2$cluster

table(cl2$cluster,wine2.class)
```

Annexe F

L'ACP et l'Autoencodeur avec Rstudio : Données Mnist

```
# Installation de packages et librairies

suppressPackageStartupMessages(library(DAAG))

library(ggplot2)

library(readr)

install_keras()

install_tensorflow()

library(tensorflow)
```

```
suppressPackageStartupMessages(library(keras))

# Importation de la base de données

mnist_train <- read_csv("C:/Users/aboub/OneDrive
/Bureau/Application mnist/mnist_train.csv")

View(mnist_train)

mnist_train1  <- mnist_train[2:785]

# Transformation de la base en dataframe

mnist_train = as.data.frame(mnist_train)

mnist_train.class <- mnist_train[,"label"]

# Normalisation des variables

norme <- function(x) (x)/255

x_train <- apply(mnist_train1, 2, norme)

x_train

# Application de l'ACP avec la fonction prcomp

pca <- prcomp(x_train)
```

```
# valeurs propres

pca$sdev^2

# Pourcentage de variance inter classe

a=pca$sdev^2/sum(pca$sdev^2)

a

# Cumul du Pourcentage de variance inter classe

cumsum(a)


# Graphique du Cumul du Pourcentage de variance inter classe

qplot(x = 2:785, y = cumsum(pca$sdev^2/sum(pca$sdev^2)), geom = "line")

# Mise en éléments supplémentaires du chiffre

ggplot(as.data.frame(pca$x), aes(x = PC1, y = PC2,
col = mnist_train.class$label)) + geom_point()

# Calcul de la reconstitution des variables avec l'ACP

mu <- matrix(rep(pca$center, nrow(pca$x)),
nrow = nrow(pca$x), byrow = T)

recon <- pca$x[,1:44] %*% t(pca$rotation[,1:44]) + mu
```

```
dim(recon)

recon

# Erreur quadratique moyenne de l'ACP

mse <- mean((recon - x_train)^2)

mse


# K-means sur les composantes principales de l'ACP


cl1=kmeans(pca$x[,1:44], 10, nstart=1000)

cl1

table(cl1$cluster,mnist_train.class$label)

# Autoencodeur avec la fonction keras_model_sequential()


x_train <- as.matrix(x_train)


modelsigrelu <- keras_model_sequential()
modelsigrelu %>%
layer_dense(units = 3, activation = "relu",
```

```
name = "bottleneck", input_shape = ncol(x_train)) %>%
layer_dense(units = 10, activation = "sigmoid") %>%

# Résumé du modèle appliqué

summary(modelsigrelu)


# compilation du modèle

modelsigrelu %>% compile(
loss = "mean_squared_error", metrics="accuracy",
optimizer = "adam"
)

# Entrainement du modèle

modelsigrelu %>% fit(
x = x_train,
y = x_train,
epochs = 1000,
verbose = 0
)

# Mesures de performance du modèle

metric_binary_accuracy(y_true, y_pred)

mse.ae2 <- evaluate(modelsigrelu, x_train, x_train)
```

```
mse.ae2

# Extraction da couche code (goulot d'étranglement) et de la sortie

intermediate_layer_model <- keras_model(inputs = modelsigrelu$input,
outputs = get_layer(modelsigrelu, "bottleneck")$output)

intermediate_output <- predict(intermediate_layer_model, x_train)

# Mise en éléments supplémentaires du chiffre

ggplot(data.frame(Neurone1 = intermediate_output[,1],
Neurone2 = intermediate_output[,2]), aes(x = Neurone1, y = Neurone2,
col = mnist_train.class$label)) + geom_point()

# K-means sur la couche code de l'autoencodeur

cl2=kmeans(intermediate_output[,1:44], 10, nstart=1000)

cl2

cl2$cluster

table(cl2$cluster,mnist_train.class$label)
```