# Rapport projet GMM5

**Visualisation de données en grande dimensions et apprentissage non supervisé**

DJIBEEROOU MAHAMADOU Abdoul Jalil

Polytech Clermont-Ferrand

Encadré par BONNET Gaëlle

13 février 2018

# 1 Résumé

Notre projet portait sur la réduction de dimensions et la visualisation de données en grande dimension. Nous avons pu étudier les méthodes ***LLE***, ***LEM***, ***TSNE***, ***PCA***, ***Kernel PCA*** et les ***Auto-encodeurs***. Un premier test, nous a permis de constater l'influence de certains paramètres sur les algorithmes. Une comparaison graphiques sur deux bases de données nous ont amené à conclure en se basant sur le critère de projection des données en clusters ; que les meilleures figures sont obtenues avec les méthodes à noyaux et la TSNE, contrairement à la LLE et aux auto-encodeurs. Par la suite, nous avons calculé sur l'une des bases, les scores associés aux méthodes en utilisant la méthode de classification des plus proches voisins. Cette étude nous a permis de faire un rapprochement avec les résultats graphiques obtenus et d'avoir quasiment les mêmes conclusions.

**Mots clés :**Visualisation de données, réduction de dimensions, validation croisée.

# Table des matières

# 2 Introduction

La réduction des dimensions a une longue histoire en tant que méthode de visualisation des données, et pour l'extraction de caractéristiques clés de faible dimension. Elle est basée sur le processus de conversion d'un ensemble de données ayant de vastes dimensions en données avec des dimensions moindres, garantissant que les informations similaires sont transmises de manière concise. Ces techniques sont généralement utilisées lors de la résolution de problèmes d'apprentissage automatique afin d'obtenir de meilleures caractéristiques pour une tâche de classification ou de régression.

La problématique liée à la dimension (nombre de variables d'entrées en apprentissage automatique) est que lorsqu'elle est plus grande que le nombre d'observations, certains algorithmes ont des difficultés à entrainer des modèles efficaces. Ce problème est plus généralement connu sous le nom de **"fléau de dimentionnalité"** [1].

Les premières techniques de réduction de dimentionnalité existaient depuis les années 1900(analyse en composantes principales ou PCA en anglais inventée en 1901 par Karl Pearson). Au fil des années des nouvelles techniques ont vu le jour notamment le ***LLE* : Localy Linear Embbeding**, ***Laplacian Eignemaps***, ***t-SNE* : student stochastic Neighbors Embeddings**.

Dans un premier nous ferons une étude théorique de ces méthodes, ensuite nous les comparerons sur plusieurs bases de données. En fin nous feront une approche orientée deep-learning en utilisant les auto-encodeurs.

1. terme inventé par Richard Bellman en 1961 pour désigner divers phénomènes qui ont lieu lorsque l'on cherche à analyser ou organiser des données dans des espaces de grande dimension alors qu'ils n'ont pas lieu dans des espaces de dimension moindre, **Wikipédia**

# 3 Descriptions des méthodes

## 3.1 PCA

[4]L'analyse en composantes principales (ACP ou PCA en anglais pour principal component analysis), est une méthode de la famille de l'analyse des données et plus généralement de la statistique multivariée, qui consiste à transformer des variables liées entre elles (dites « corrélées » en statistique) en nouvelles variables dé-corrélées les unes des autres. Ces nouvelles variables sont nommées « composantes principales », ou axes principaux. Elle permet au praticien de réduire le nombre de variables et de rendre l'information moins redondante.

**Échantillon**
On applique généralement une ACP sur un ensemble de N variables aléatoire $X_{1,1}, .....X_N$ connues à partir d'un échantillon de K réalisations conjointes de ces variables.

Cet échantillon de ces N variables aléatoires peut être structuré dans une matrice M, à K lignes et N colonnes.

$$M = \begin{pmatrix} X_{1,1} & \cdots & X_{1,N} \\ \vdots & \ddots & \vdots \\ X_{K,1} & \cdots & X_{K,N} \end{pmatrix}$$

Chaque variable aléatoire $X_n$ dont $X_{1,n}, ...X_{K,n}$ sont des réalisations indépendantes, a une moyenne $\bar{X_n}$ et un écart type $\sigma_{X_n}$.

**Poids**
Si les réalisations (les éléments de la matrice M) sont à probabilités égales alors chaque réalisation (un élément $X_{i,j}$ de la matrice) a la même importance 1/K dans le calcul des caractéristiques de l'échantillon. On peut aussi appliquer un poids $p_i$ différent à chaque réalisation conjointe des variables (cas des échantillons redressés, des données regroupées, ...). Ces poids, qui sont des nombres positifs de somme 1 sont représentés par une matrice diagonale D de taille K :

$$D = \begin{pmatrix} p_1 & & & \\ & p_2 & & \\ & & \ddots & \\ & & & p_K \end{pmatrix}$$

Dans le cas le plus courant de poids égaux, D=$\frac{1}{K}I$ où I est la matrice identité.

**Transformations de l'échantillon**
Le vecteur $\left(\tilde{X_1}, \quad \cdots \quad, \tilde{X_N}\right)$ est le centre de gravité du nuage de points ; on le note souvent g. On a $g = M^T D\tilde{1}$ où $\tilde{1}$ désigne le vecteur de $\Re^K$ dont toutes les composantes sont égales à 1.
La matrice M est généralement centrée sur le centre de gravité :

$$\tilde{M} = \begin{pmatrix} X_{1,1} - \bar{X_1} & \cdots & X_{1,N} - \bar{X_N} \\ \vdots & \ddots & \vdots \\ X_{K,1} - \bar{X_1} & \cdots & X_{K,N} - \bar{X_N} \end{pmatrix}$$

Elle peut être aussi réduite :

$$\tilde{M} = \begin{pmatrix} \frac{X_{1,1}-\bar{X_1}}{\sigma(X_1)} & \dots & \frac{X_{1,N}-\bar{X_N}}{\sigma(X_N)} \\ \vdots & \ddots & \vdots \\ \frac{X_{K,1}-\bar{X_1}}{\sigma(X_1)} & \dots & \frac{X_{K,N}-\bar{X_N}}{\sigma(X_N)} \end{pmatrix}$$

Le choix de réduire ou non le nuage de points (i.e. les K réalisations de la variable aléatoire (X1, ..., XN)) est un choix de modèle :

- si on ne réduit pas le nuage : une variable à forte variance va « tirer » tout l'effet de l'ACP à elle ;
- si on réduit le nuage : une variable qui n'est qu'un bruit va se retrouver avec une variance apparente égale à une variable informative.
- Si les variables aléatoires sont dans des unités différentes, la réduction est obligatoire.

**Calcul de covariances et de corrélations**
Une fois la matrice M transformée en $\bar{M}$ ou $\tilde{M}$, il suffit de la multiplier par sa transposée pour obtenir :

- la matrice de variance-covariance des $X_1, \dots, X_N$ si M n'est pas réduite : $Covariances = 1/K \cdot \bar{M}^T \cdot \bar{M}$
- la matrice de corrélation des $X_1, \dots, X_N$ si M est réduite : $Corrlation = 1/K \cdot \tilde{M}^T \cdot \tilde{M}$

Ces deux matrices sont carrées (de taille N), symétriques, et réelles. Elles sont donc diagonalisables dans une base orthonormée en vertu du théorème spectral.
De façon plus générale, la matrice de variance-covariance s'écrit $V = M^T DM - gg^T = \bar{M}^T \cdot D \cdot \bar{M}$
De plus, si l'on note $D_{1/s}$ la matrice diagonale des inverse des écarts-types :

$$D_{1/s} = \begin{pmatrix} 1_{/s1} & & & \\ & 1_{/s2} & & \\ & & \ddots & \\ & & & 1_{/sN} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sigma(X_1)} & & & \\ & \frac{1}{\sigma(X_2)} & & \\ & & \ddots & \\ & & & \frac{1}{\sigma(X_N)} \end{pmatrix}$$

On a alors : $\tilde{M} = \bar{M} \cdot D_{1/s}$
La matrice des coefficients de corrélation linéaire entre les N variables prises deux à deux, notée R, s'écrit :
$R = \tilde{M}^T \cdot D \cdot \tilde{M} = D_{1/s} V D_{1/s}$

**Critère d'inertie**
Dans la suite de cet article, nous considèrerons que le nuage est transformé (centré et réduit si besoin est). Chaque $X_n$ est donc remplacé par $X_n - \bar{X_n}$ ou $\frac{X_n - \bar{X_n}}{\sigma(X_n)}$

Le principe de l'ACP est de trouver un axe u, issu d'une combinaison linéaire des$X_n$, tel que la variance du nuage autour de cet axe soit maximale.

Pour bien comprendre, imaginons que la variance de u soit égale à la variance du nuage ; on aurait alors trouvé une combinaison des $X_n$ qui contient toute la diversité du nuage original (en

tout cas toute la part de sa diversité captée par la variance).

Un critère couramment utilisé est la variance de l'échantillon (on veut maximiser la variance expliquée par le vecteur u). Pour les physiciens, cela a plutôt le sens de maximiser l'inertie expliquée par u (c'est-à-dire minimiser l'inertie du nuage autour de u).

**Projection**
Finalement, nous cherchons le vecteur u tel que la projection du nuage sur u ait une variance maximale. La projection de l'échantillon des X sur u s'écrit :
$\Pi_u(M) = M \cdot u$
la variance empirique de $\Pi_u(M)$ vaut donc :
$\Pi_u(M)^T \cdot 1/K \cdot \Pi_u(M) = u^T \cdot \underbrace{M^T \cdot 1/K \cdot M}_{C} \cdot u$ où C est la matrice de covariance.
Comme nous avons vu plus haut que C est diagonalisable dans une base orthonormée, notons P le changement de base associé et $\Delta = Diag(\lambda_1, \ldots, \lambda_N)$ la matrice diagonale formée de son spectre :
$\Pi_u(M)^T \cdot 1/K \cdot \Pi_u(M) = u^T \cdot P^T \Delta P u = (Pu)^T \Delta \underbrace{(Pu)}_{v}$
Les valeurs $(\lambda_1, \ldots, \lambda_N)$ de la diagonale $\Delta$ sont rangées en ordre décroissant. Le vecteur unitaire u qui maximise $v^T \Delta v$ est un vecteur propre de C associé à la valeur propre $\lambda_1$ ; on a alors :
$c^T \cdot \Delta \cdot v = \lambda_1$
La valeur propre $\lambda_1$ est la variance empirique sur le premier axe de l'ACP.

Il est aussi possible de démontrer ce résultat en maximisant la variance empirique des données projetées sur u sous la contrainte que u soit de norme 1 (par un multiplicateur de Lagrange $\alpha$) :
$L(u, \alpha) = u^T \cdot C \cdot u - \alpha(u^T u - 1)$
On continue la recherche du deuxième axe de projection w sur le même principe en imposant qu'il soit orthogonal à u.

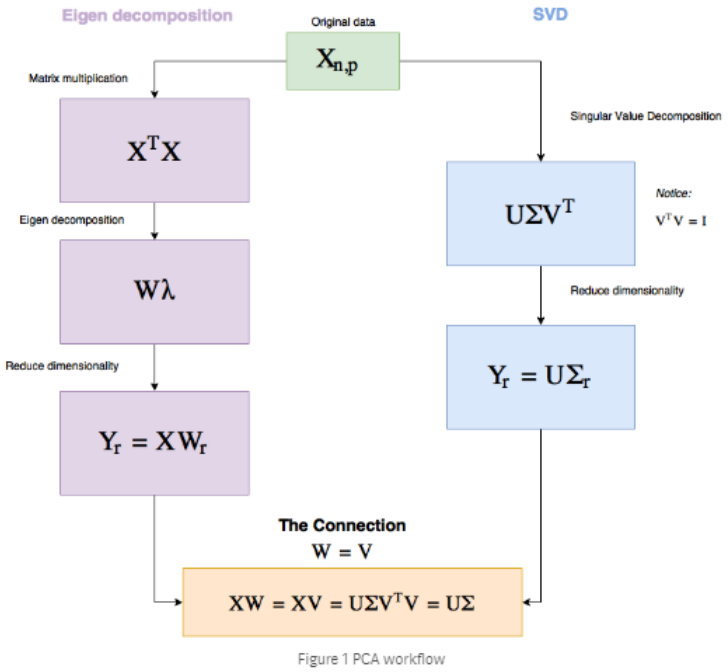


FIGURE 1 – Résumé ACP

### Application

Soit la base de données **vins** de dimension 13, constituée de 3 classes, avec un échantillon total de 178 éléments dont 59, 71, et 48 respectivement pour la première, deuxième et troisième classe. Les attributs sont réels positifs.

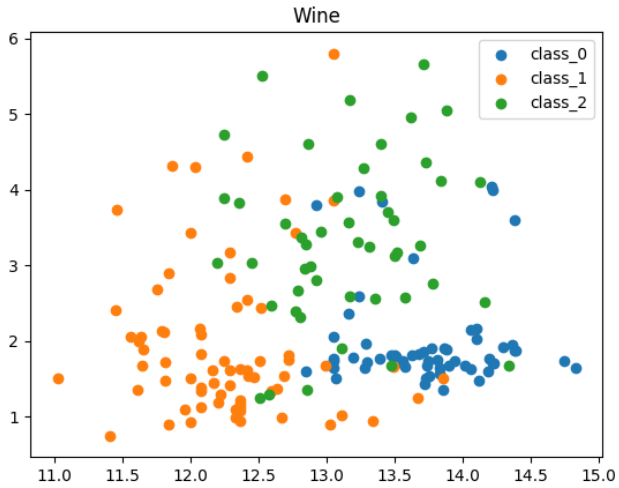


FIGURE 2 – Données

Appliquons sur cette base l'ACP avec deux composantes principales en changeant quelques paramètres après avoir normaliser les données.

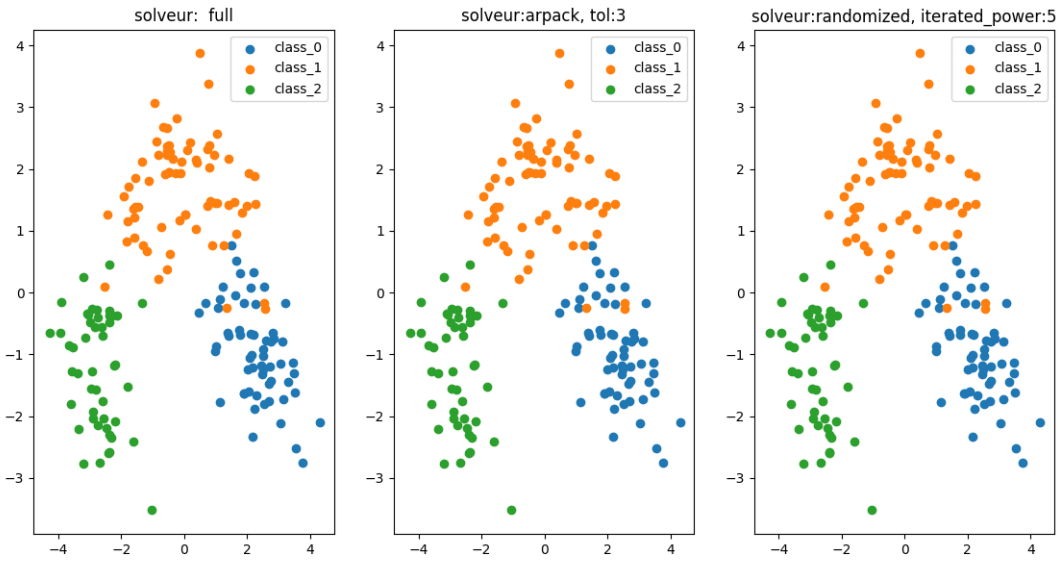


FIGURE 3 – ACP

Comme on peut le constater sur la figure 3, la réduction à 2 dimensions nous a permis d'avoir 3 clusters légèrement séparés correspondant aux 3 classes. On remarque qu'il n'y a pas d'effet du solveur pour la décomposition en valeurs singulières, du paramètre tolérance et la puissance d'itération sur cette base.

Le pourcentage d'inertie des deux composantes principales par rapport à l'inertie totale est 55.4% avec respectivement 36.2% et 19.2% pour la première et la deuxième composante.

## 3.2 Kernel PCA

[4] Dans le domaine des statistiques multivariées, l'analyse des composantes principales du noyau est une extension de l'analyse en composantes principales (PCA) utilisant des techniques de noyau. A l'aide d'un noyau, les opérations linéaires originales du PCA sont effectuées dans un espace de Hilbert de noyau de reproduction.

Pour comprendre l'utilité du PCA à noyau, en particulier pour le clustering, remarquez que si les N points ne peuvent généralement pas être linéairement séparés dans les dimensions d $<$ N, ils peuvent presque toujours être séparés linéairement dans d $\geq$ N dimensions. C'est-à-dire, étant donné N points, $x_i$, si nous les mappons à un espace N-dimensionnel avec

$$\Phi(x_i), \Phi : \Re^d \to \Re^N$$

Il est facile de construire un hyperplan qui divise les points en clusters arbitraires. Ce $\Phi$ crée des vecteurs linéairement indépendants, donc il n'y a pas de covariance sur laquelle exécuter explicitement la décomposition en vecteurs comme on le ferait en ACP linéaire.
Au lieu de cela, dans l'ACP à noyau, une fonction $\Phi$ arbitraire et non triviale qui n'est jamais calculée explicitement est 'choisie', ce qui permet d'utiliser des fonctions $\Phi$ 'de très grande dimension' si nous n'avons jamais à évaluer réellement les données dans cet espace. Puisque nous essayons généralement d'éviter de travailler dans le $\Phi$-espace, que nous appellerons "l'espace des caractéristiques", nous pouvons créer le noyau N-par-N

$$K = k(x, y) = (\Phi(x), \Phi(y)) = \Phi(x)^T \Phi(y)$$

qui représente l'espace de produit interne de l'espace caractéristique autrement intraitable. La forme double qui apparaît dans la création d'un noyau nous permet de formuler mathématiquement une version de l'ACP dans laquelle nous ne résolvons jamais réellement les vecteurs propres et les valeurs propres de la matrice de covariance dans le $\Phi$-espace. Les N-éléments dans chaque colonne de K représentent le produit scalaire d'un point des données transformées par rapport à tous les points transformés (N points).

Parce que nous ne travaillons jamais directement dans l'espace des entités, la formulation de l'ACP à noyau est limitée en ce qu'elle ne calcule pas les composants principaux eux-mêmes, mais les projections de nos données sur ces composants. Pour évaluer la projection d'un point dans l'espace des fonctions $\Phi$ sur le kième composant principal $V^k$ (où l'exposant k signifie la composante k)

$$V^{k^T} \Phi(x) = (\sum_{i=1}^{N} a_i^k \Phi(x_i))^T \Phi(x)$$

Notons que $\Phi(x_i)^T \Phi(x)$ est le produit scalaire et qui représente simplement les éléments du noyau K. Tout ce qui reste est pour calculer et normaliser les $a_i$,ce qui peut être fait en résolvant le problème :

$$Ka = N\lambda a$$

où : N est le nombre de points, $\lambda$ et a sont les valeurs et vecteurs propres et pour normaliser les $a^k$ qui nécessitent :

$$(V^k)^T(V^k) = 1$$

La normalisation des données étant nécessaire, le K est normaliser en K' comme suit :

$$K' = K - 1_N K - K 1_N + 1_N K 1_N$$

Où $1_N$ est la matrice N-par-N dont les valeurs sont $\frac{1}{N}$

En ACP linéaire, nous pouvons utiliser les valeurs propres pour classer les vecteurs propres en fonction de la quantité de variation des données capturées par chaque composante principale. Ceci est utile pour la réduction de la dimensionnalité des données et pourrait également être appliqué à KPCA. Cependant, dans la pratique, il y a des cas où toutes les variations des données sont identiques. Ceci est généralement causé par un mauvais choix de l'échelle du noyau.

**Application**

Pour l'application de cette méthode nous considérerons la base de données**iris** de dimensions 4 qui est constituée de 3 classes dont 50 échantillons par classe. Les données sont réelles positives.

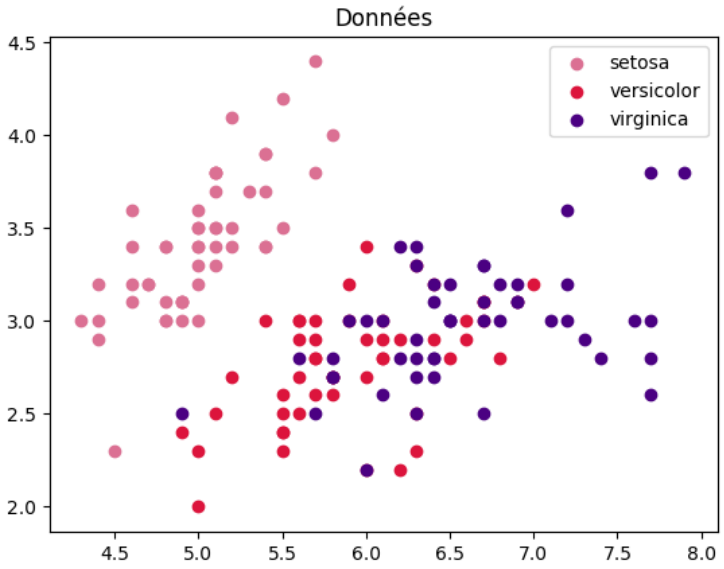


FIGURE 4 – Iris

Comme dans la méthode précédente nous ferons varier quelques paramètres de la méthode et observerons s'il y a des effets ou pas.

Les paramètres que nous changerons sont le noyau qui prendra les valeurs "rbf","sigmoid","poly" et, "cosine" ; et le gamma qui vaudra 0.000001, 0.00001, 0.0001, 0.001, 0.01, 1, 10, et 30.
On obtient les figures suivantes :

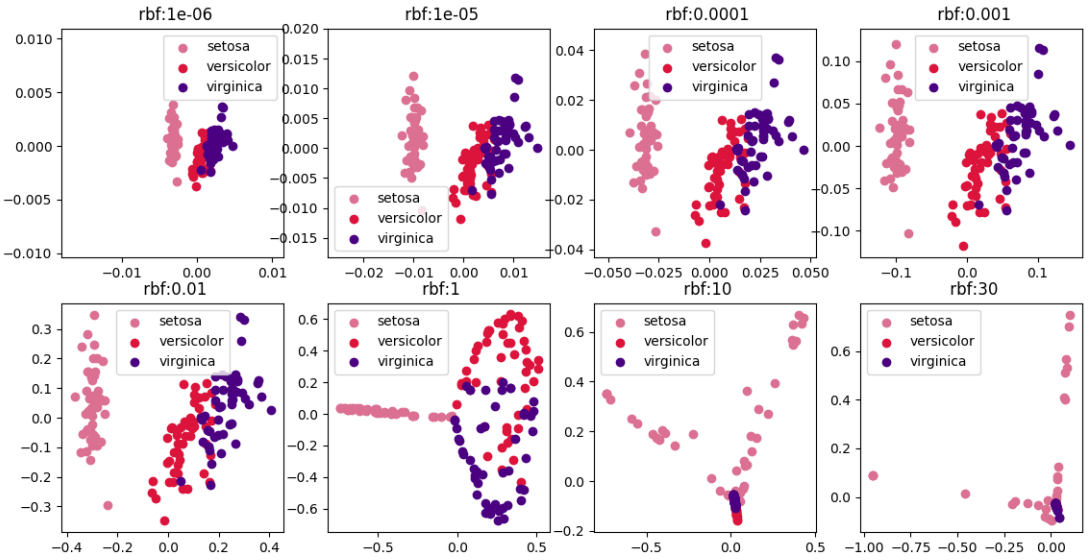


FIGURE 5 – Noyau rbf avec différentes valeurs de gamma

Avec le noyau rbf, on remarque jusqu'à certaines valeurs du gamma (0.01), les données sont plutôt regrouper en classe. A partir de la valeur 1, la qualité de regroupement se détériore et avec la valeur 30 du gamma la classe versicolor devient même invisible.

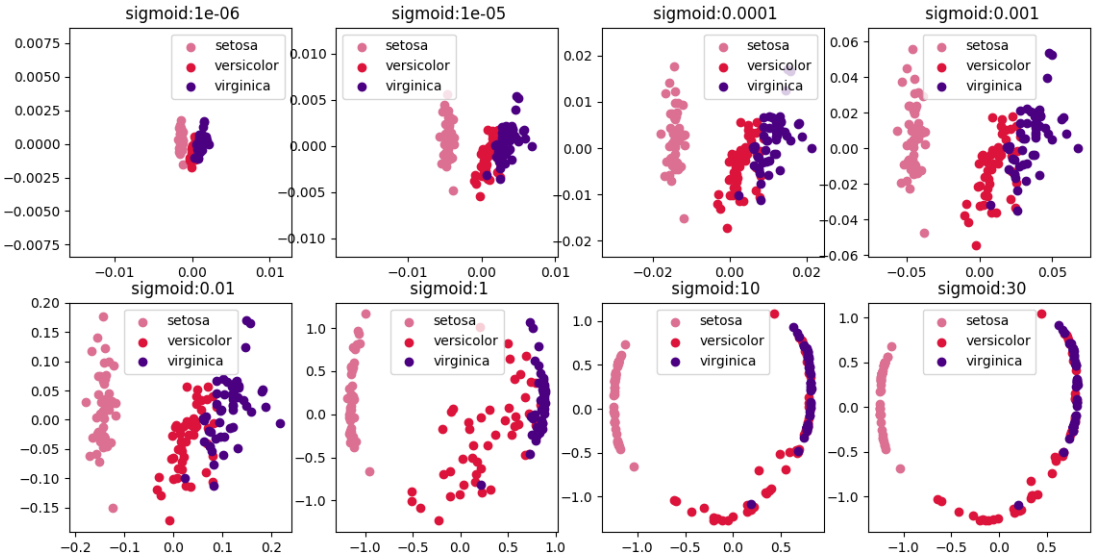


FIGURE 6 – Noyau sigmoid avec différentes valeurs de gamma

Quant à cet noyau, on remarque qu'on a quasiment les mêmes figures avec le noyau précédent pour les valeurs de gamma jusqu'à 0.01. Contrairement à la figure pour gamma = 1 du noyau rbf, pour ce noyau les données restent regrouper et on peut distinguer les 3 clusters. Et pour les autres valeurs de gamma, les classes restent visibles mais pas séparées sauf la classe setosa. On remarque aussi la différence de silhouette entre les deux noyaux à partir de la valeur 1 du gamma. Nous nous intéresserons plus en détaille à cette dernière dans la suite.

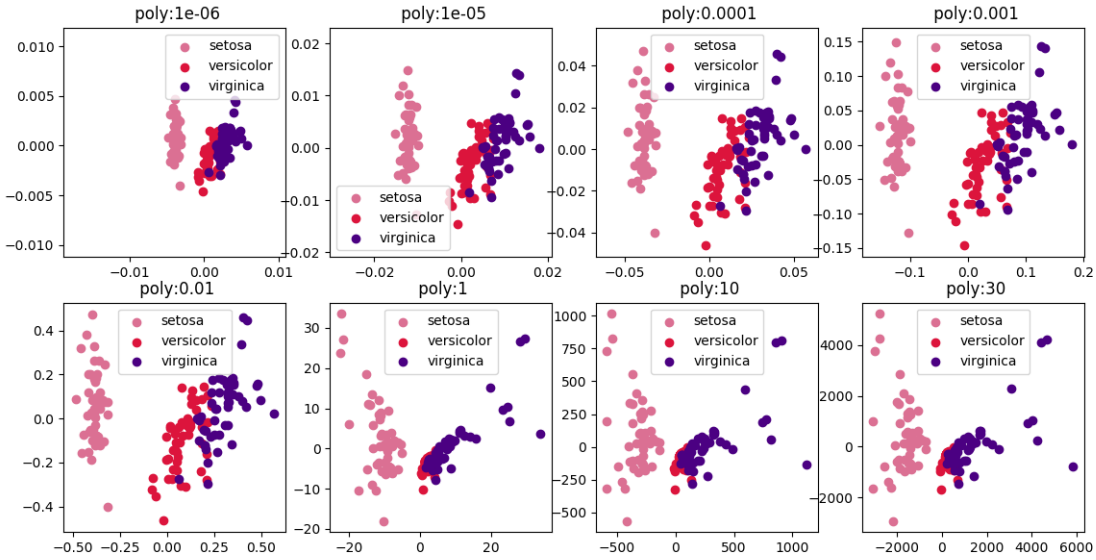


FIGURE 7 – Noyau poly avec différentes valeurs de gamma

Pour ce noyau, mêmes remarques concernant la similarité des figures jusqu'à la valeur 0.01. Pour les autres valeurs de gamma, les classes setosa et virginica sont séparées mais la classe versicolor est presque invisible.

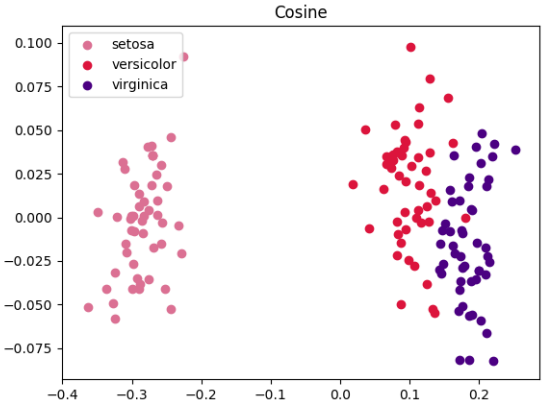


FIGURE 8 – Noyau cosine avec différentes valeurs de gamma

Pour le noyau cosine on remarque que toutes les classes sont séparées. Tandis que les classes

vercicolor et virginica sont rapprochées, la classe setosa est loin de ces deux dernières.

Ainsi pour la méthode de l'ACP à noyau on remarque que sur cette base, pour certaines valeurs de gamma nous obtenons les mêmes figures quelque soit le noyau.

### 3.3 LLE

L'algorithme standard du LLE[1] : repose sur le principe suivant : Trouver un ensemble des voisins les plus proches de chaque point(en utilisant par exemple l'algorithme KNN). Il calcule ensuite un ensemble de poids pour chaque point qui décrit le mieux le point comme une combinaison linéaire de ses voisins. Enfin, il utilise une technique d'optimisation basée sur le vecteur propre pour trouver l'encastrement de points de faible dimension, de sorte que chaque point est toujours décrit avec la même combinaison linéaire de ses voisins.

Le LLE calcule les coordonnées barycentriques d'un point $X_i$ en fonction de ses voisins $X_j$. Le point original est reconstruit par une combinaison linéaire, donnée par la matrice de poids $W_{ij}$, de ses voisins. L'erreur de reconstruction est donnée par la fonction de coût E (W).

$$E(W) = \sum_i |X_i - \sum_j W_{ij} X_j|^2 \tag{1}$$

Les poids $W_{ij}$ se réfèrent à la quantité de contribution que le point $X_j$ a en reconstruisant le point $X_i$. La fonction de coût est minimisée sous deux contraintes :

— Chaque point de données $X_i$ est reconstruit uniquement à partir de ses voisins, forçant ainsi $W_{ij}$ à être nul si le point $X_j$ n'est pas voisin du point $X_i$

— La somme de chaque ligne de la matrice de poids est égale à 1.

$$\sum_j W_{ij} = 1$$

Les points de données originaux sont collectés dans un espace dimensionnel D et le but de l'algorithme est de réduire la dimensionalité à d telle que D » d. Les mêmes poids $W_{ij}$ qui reconstruisent le ième point de données dans l'espace dimensionnel D seront utilisés pour reconstruire le même point dans l'espace dimensionnel inférieur. Chaque point $X_i$ dans l'espace dimensionnel D est mappé sur un point $Y_i$ dans l'espace dimensionnel d en minimisant la fonction de coût :

$$E(W) = \sum_i |Y_i - \sum_j W_{ij} Y_j|^2 \tag{2}$$

Dans cette fonction de coût, contrairement au précédent, les poids $W_{ij}$ sont maintenus fixes et la minimisation est faite sur les points $Y_i$ pour optimiser les coordonnées. Ce problème de minimisation peut être résolu en résolvant un problème de valeur propre N*N(N étant le nombre de points de données), dont les vecteurs propres inférieurs d non nuls fournissent un ensemble orthogonal de coordonnées. Généralement, les points de données sont reconstruits à partir de K voisins les plus proches, tels que mesurés par la distance euclidienne. Pour une telle implémentation, l'algorithme n'a qu'un seul paramètre libre **K**, qui peut être choisi par validation croisée.

#### Application

Nous conserverons la même base de données iris pour l'application de cette méthode. Les paramètres testés sont : la **méthode de résolution du LLE** dont les valeurs sont : **'standard'**, **'hessian'**, **'modified'**,**'ltsa'**(lignes sur la figure 9) et l'**algorithme du plus proche voisin** :

**'brute'**, **'kd tree'**, **'ball tree'**(colonnes sur la figure 9).Pour les autres paramètres nous avons pris 6 pour le nombre de voisins et 'auto' pour pour le solveur de valeurs propres.

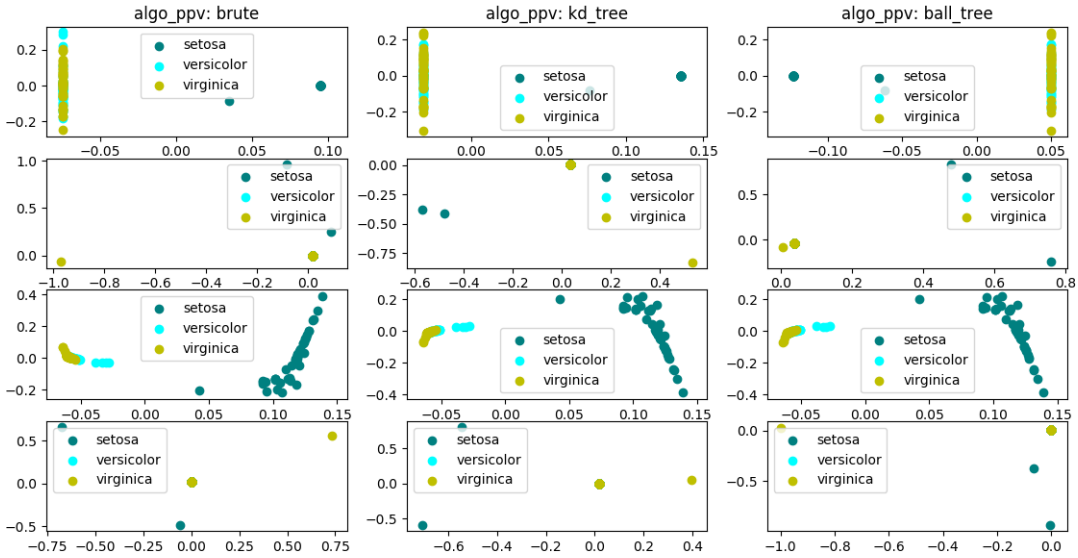


FIGURE 9 – LLE

Comme nous pouvons le remarquer seule la méthode 'modified' parmi les 4 semble marcher sur les données. Pour cette méthode, nous pouvons distinguer 3 clusters différents. La classe setosa, est bien représentée et séparée des autres classes. On remarque aussi une représentation dense des données pour les classes virginica et versicolor. Enfin, remarquons que la figure obtenue pour l'algorithme 'brute' semble être la transposée des deux autres figures.

Ainsi, il y'a bien effets des paramètres sur les résultats obtenus.

## 3.4 t-SNE

L'algorithme t-SNE [2] se base sur une interprétation probabiliste des proximités. Une distribution de probabilité est définie sur les paires de points de l'espace d'origine de telle sorte que des points proches l'un de l'autre ont une forte probabilité d'être choisi tandis que des points éloignés ont une faible probabilité d'être sélectionnés. Une distribution de probabilité est également définie de la même manière pour l'espace de visualisation. L'algorithme t-SNE consiste à faire concorder les deux densités de probabilité, en minimisant la divergence de **Kullback-Leibler** entre les deux distributions par rapport à l'emplacement des points sur la carte.

**Démarche de l'algorithme**
Étant donné un ensemble de N objets de grande dimension $x_1, \ldots, x_N$ calculer la probabilité de similarité des objets $x_i$ et $x_j$ comme suit :

$$p_{j|i} = \frac{exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-\|x_i - x_k\|^2/2\sigma_i^2)} \tag{3}$$

Calculer :

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

En effet, les auteurs définissent la similitude entre les points $x_j$ et $x_i$ comme étant la probabilité conditionnelle $p_{j|i}$ que $x_i$ choisisse comme voisin $x_j$ si les voisins étaient choisis proportionnellement à leur densité de probabilité sous une loi Gaussienne centré sur $x_i$. L'algorithme a pour but d'apprendre une carte à **d** dimension $y_1, \ldots, y_N$ (avec $y_i \in \Re^d$) qui reflète au mieux les similitudes $q_{ij}$ entre points de la carte $y_i$ et $y_j$ comme suit :

$$q_{ji} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_i - y_k\|^2)^{-1}} \tag{4}$$

Ici, une distribution de Student est utilisée pour mesurer les similitudes entre les points de dimension réduite afin de permettre à des objets dissemblables d'être modélisés sur la carte.

Les emplacements des points dans la carte sont déterminés en minimisant la divergence **Kullback-Leibler** (non symétrique) de la distribution Q à partir de la distribution P, c'est-à-dire :

$KL(P||Q) = \sum_{i \neq j} p_{ij} \log(\frac{p_{ij}}{q_{ij}})$

La minimisation de la divergence de Kullback-Leibler par rapport aux points $y_i$ est effectuée en utilisant la descente du gradient.

**Algorithme**

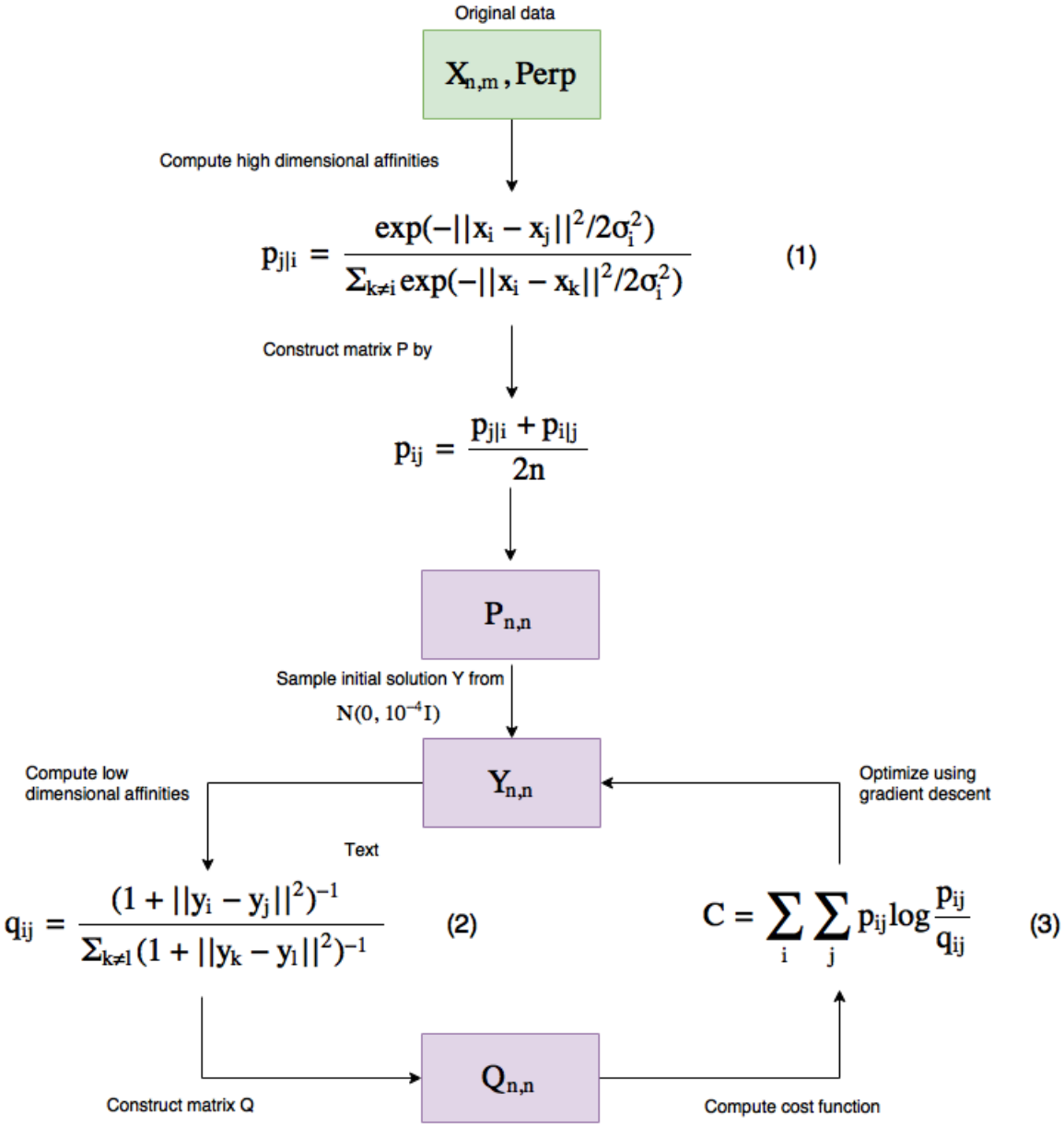


FIGURE 10 – Résumé t-SNE

### Application

Appliquons cette méthode à la base de données cancer de seins de dimension 30 constituée de deux classes : **malignant** et **benign**. Le nombre total d'échantillons est de 569 et les données sont réelles positives.

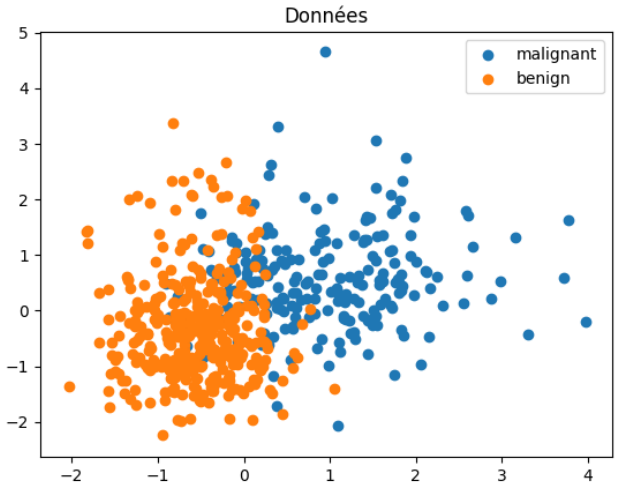


FIGURE 11 – Données

Nous testerons les paramètres **init** qui prendra les valeurs **pca** et **random** pour l'initialisation de la méthode et **perplexity** qui correspond au nombre de voisins. Nous considérerons les valeurs **15**, **20**, **30**, **35**, **45** pour ce paramètre (choix arbitraire). La valeur par défaut est 30.

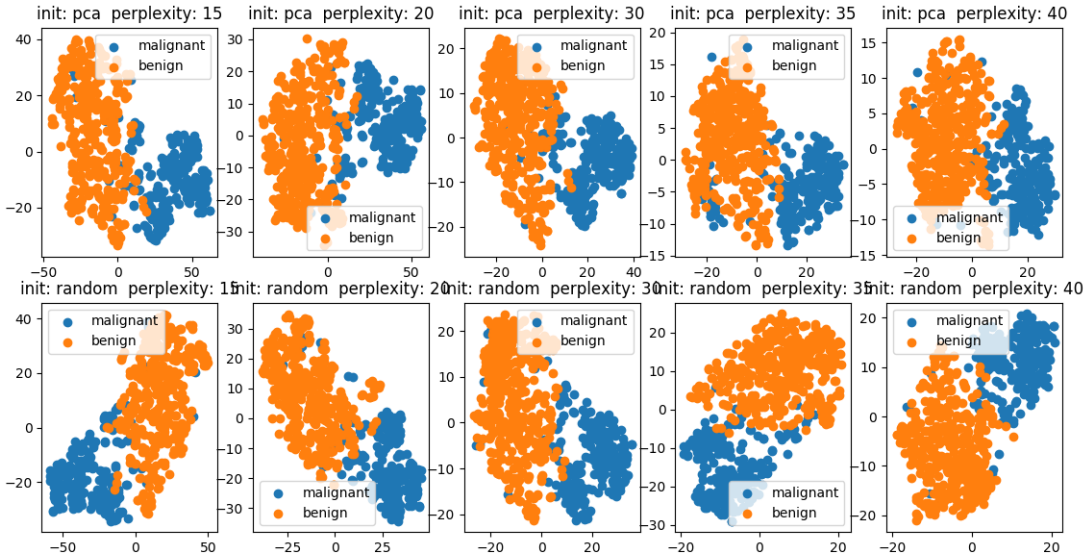


FIGURE 12 – Résultats du TSNE avec différents paramétrages

Comme nous pouvons le constater il n'y a pas de différence majeur entre les 10 figures obtenues. La meilleures décomposition est obtenue pour **init** = random et **perplexity** = 35. En effet pour ces paramètres les deux classes sont moins mélangées.

## 3.5 LEM

L'algorithme LEM [3] construit un graphe à partir des informations de voisinage de l'ensemble de données. Chaque point de données sert de nœud sur le graphe et la connectivité entre les nœuds est régie par la proximité des points voisins (en utilisant par exemple l'algorithme k-plus proche voisin). Le graphe ainsi généré peut être considéré comme une approximation discrète du collecteur de faible dimension dans l'espace de haute dimension. La minimisation d'une fonction de coût basée sur le graphe garantit que les points proches les uns des autres sur le collecteur sont mappés les uns près des autres dans l'espace de faible dimension, en préservant les distances locales.

### Algorithme

soit k points $x_1, \ldots, x_k \in \Re^l$, construire un graphe pondéré avec k nœuds pour chaque point et un ensemble d'arc connectant les points voisins. La carte de projection est donnée en calculant les vecteurs propres du graphe Laplacien.

1. **Étape 1** Construction du graphe adjacents
   mettre un arc entre les nœuds i et j si $x_i$ et $x_j$ sont "proche". On distingue deux cas :

   (a) $\epsilon$ - voisins. [ paramètre $\epsilon \in \Re$ ]
   les nœuds i et j sont connectés par un arc si $\|x_i - x_j\|^2 < \epsilon$, ou la norme est la norme euclidienne dans $\Re^l$.
   **Avantages** : la relation est symétrique.
   **Désavantages** : relie souvent à des graphes avec plusieurs composants connectés entrainant une difficulté du choix de $\epsilon$.

   (b) n plus proches voisins. [paramètre n $\in N$]
   les nœuds i et j sont connectés par un arc si i est parmi les voisins de j ou inversement. Cette relation est aussi symétrique.
   **Avantages** : Facilité de choix, ne tend pas lier à des graphes déconnectés.
   **Désavantages** : mois géométriquement intuitive.

2. **Étape 2** : Choix des poids.
   Ici il y a deux variations de pondérer les arcs :
   (a) paramètre t $\in \Re$
   Si i et j sont connectés mettre :

   $$W_{ij} = exp(-\frac{\|x_i - x_j\|^2}{t})$$

   sinon mettre

   $$W_{ij} = 0$$

   (b) Pas de paramètres
   $W_{ij} = 1$ si i et j sont connectés par un arc et $W_{ij} = 0$ sinon.

3. **Étape 3** Eigenmap
   Supposons G le graphe construit ci-dessus est connecté, sinon procéder à l'étape 3 pour chaque composant connecté.
   Calculer les vecteurs propres et valeurs propres pour le problème de vecteurs propres généralisé :

$$Lf = \lambda Df(*)$$

   Où D est la matrice diagonale des poids. $D_{ii} = \sum_j Wij$. $L = D - W$ est la matrice laplacienne, matrice symétrique, semi-définie positive qui peut être un opérateur définie sur les nœuds de G.
   Soient $f_0, \ldots, f_{k-1}$ les solutions de l'équation (*), ordonnées suivant leurs valeurs propres

$$Lf_0 = \lambda_0 Df_0$$
$$Lf_1 = \lambda_1 Df_1$$
$$\ldots$$
$$Lf_{k-1} = \lambda_{k-1} Df_{k-1}$$
$$0 = \lambda_0 <= \lambda_1 <= \ldots <= \lambda_{k-1}$$

   Le vecteur propre $f_0$ correspondant à la valeur propre 0, ainsi il faut prendre les vecteurs restant pour la projection dans l'espace Euclidien de dimension m.
   $x \longrightarrow (f_1(i), \ldots, f_m(i))$

**Application**

Nous garderons la même base précédente pour l'application de cette méthode et testerons les paramètres **affinity** pour la construction de la matrice d'affinité qui prendra les valeurs **nearest neighbors** pour la construction de la matrice avec le graph knn, et **rbf** avec le noyau rbf . Nous testerons les valeurs $10^{-5}$ $10^{-4}$, $10^{-3}$, $10^{-2}$, $10^{-1}$, **1**, **10**, **20** pour rbf.

Pour le knn, nous obtenons la figure suivante :

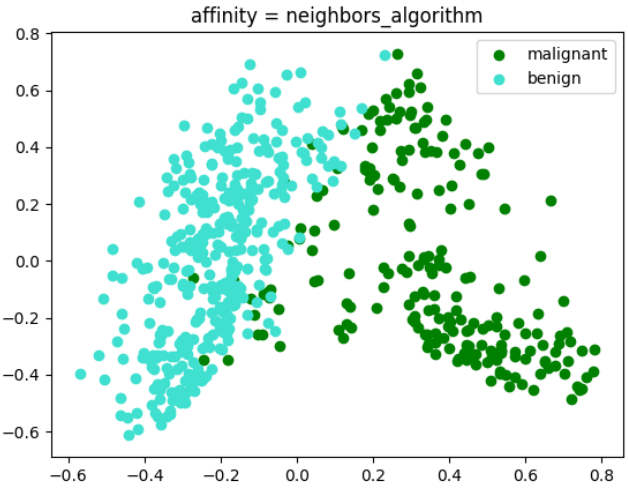


FIGURE 13 – Résultats pour knn

Et pour la rbf nous obtenons :

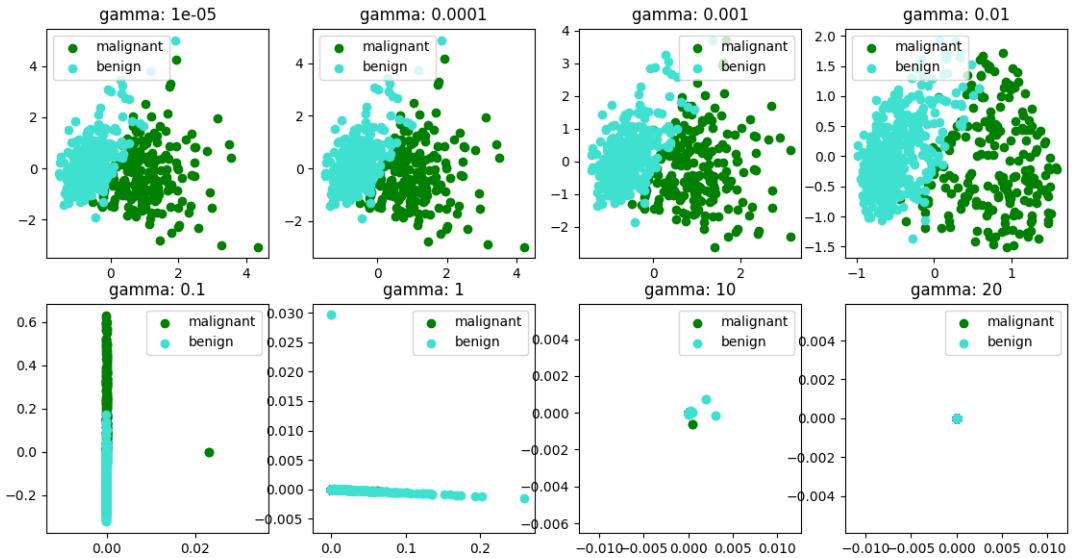


FIGURE 14 – Résultats pour le noyau rbf

Pour le knn, bien qu'on peut voir les 2 clusters, ces derniers ne sont pas totalement séparés. Pour la rbf, pour certaines valeurs de gamma, nous obtenons des figures de moins bonne qualité.

## 3.6 Auto-encodeurs

Un auto-encodeur, est un réseau de neurones artificiels utilisé pour l'apprentissage non supervisé de caractéristiques discriminantes. L'objectif d'un auto-encodeur est d'apprendre une représentation (encodage) d'un ensemble de données, généralement dans le but de réduire la dimension de cet ensemble. Récemment, le concept d'auto-encodeur est devenu plus largement utilisé pour l'apprentissage de modèles génératifs.

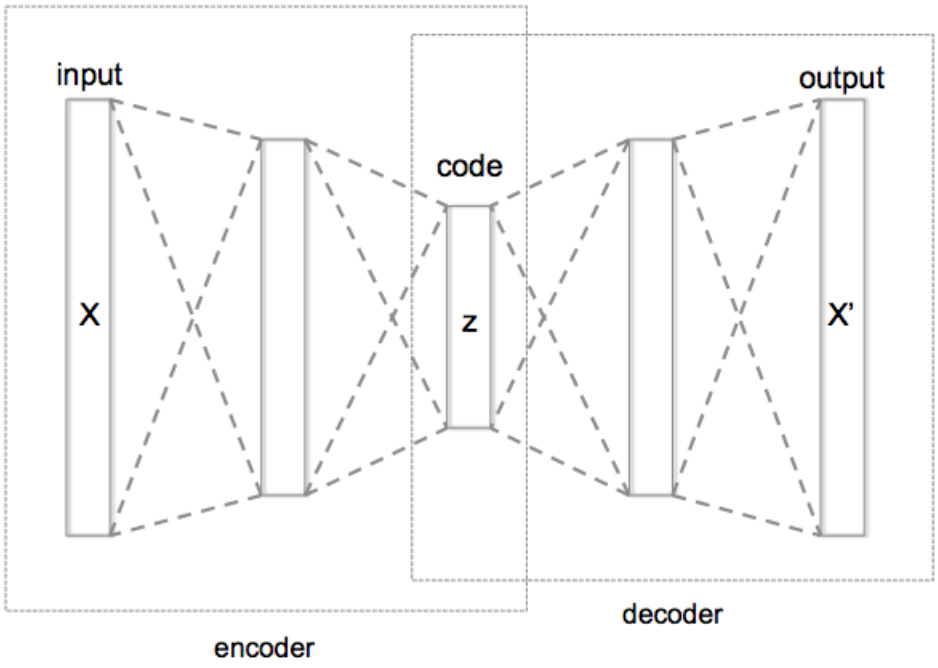


FIGURE 15 – Architecture d'auto-encodeur

Pour plus de détaille sur les auto-encodeurs voir polycopié cours de BONNET Gaëlle.

### Application

Générons une base de données avec la méthode **make blobs** de **scikit learn**. Nous paramétrons cette méthode comme suit :
n_samples=200, centers=5, n_features=2.

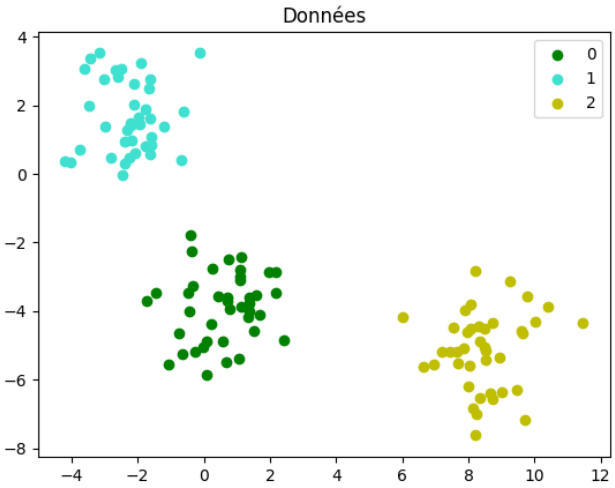


FIGURE 16 – Données

Soit les paramètres suivants :
learning_rate = 0.01
num_iter = 20
batch_size = 50
n_hidden_1 = 6
n_hidden_2 = 4
n_hidden_3 = 2

Ce paramétrage sera gardé pour toute la suite.

Nous testerons uniquement deux algorithmes d'optimisation : **GradientDescentOptimizer**, et **AdagradOptimizer**.

Le graph associé à l'auto-encodeur est le suivant :

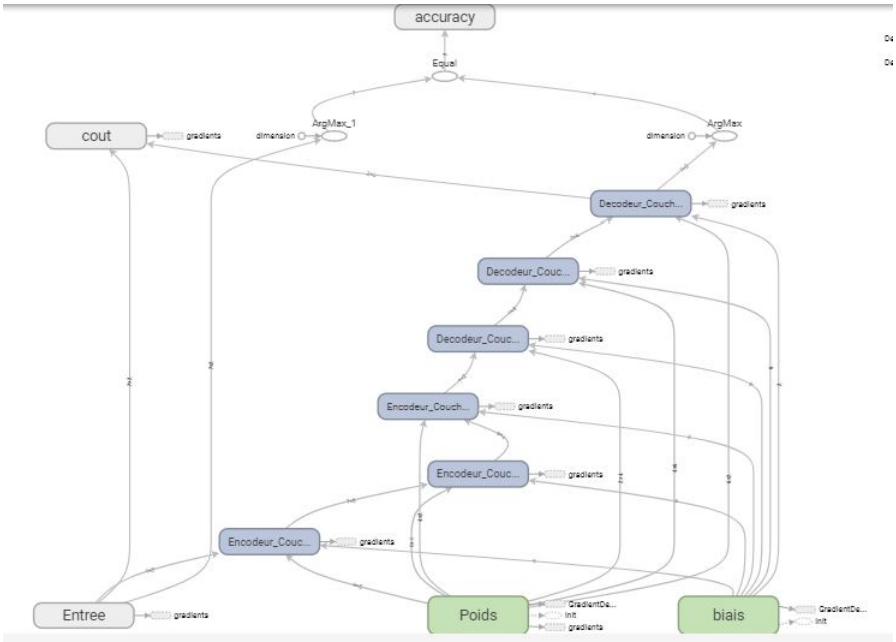


FIGURE 17 – Graph sous tensorboard

Pour **GradientDescentOptimizer**, nous obtenons la figure suivante :

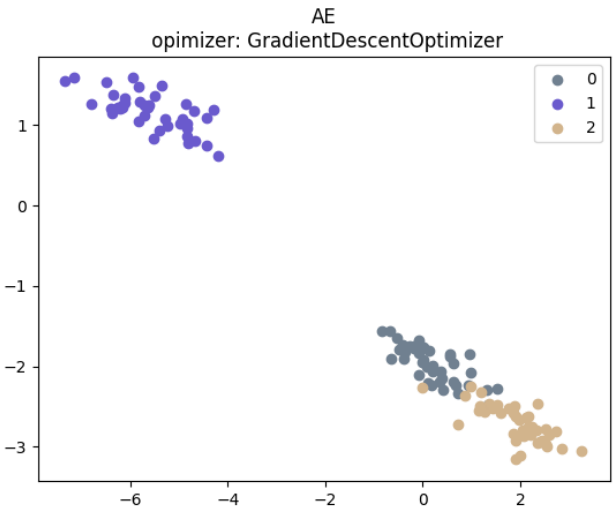


FIGURE 18 – Résultat avec GradientDescentOptimizer

Pour **AdagradOptimizer** nous obtenons :

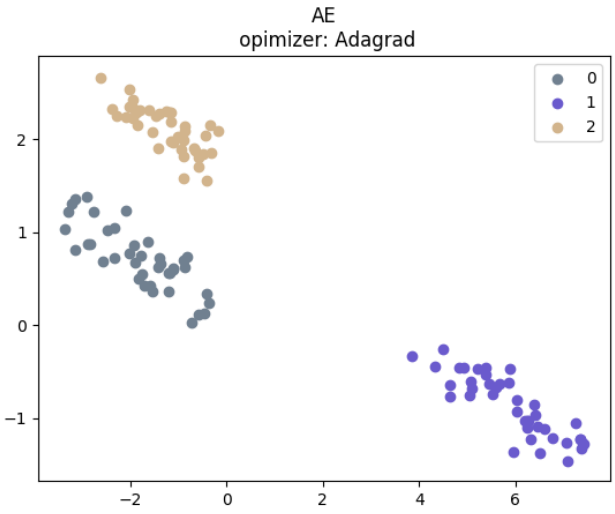


FIGURE 19 – Résultat avec AdagradOptimizer

Comme nous pouvons le constater, nous obtenons le meilleur résultat avec l'algorithme d'optimisation : **AdagradOptimizer**. Cependant, ces résultats sont à prendre avec précaution. En effet, l'initialisation aléatoire des variables peut affecter ces derniers. Surtout que pour ce test, nous avons compiler séparément le programme pour les deux méthodes.

Dans cette section, nous avons décrit et mis en œuvre les méthodes. Nous avons remarquer que pour certaines méthodes le choix des paramètres influe grandement sur les résultats alors que pour d'autres, cela n'est pas le cas. A présent, nous passerons à leur expérimentation.

# 4 Expérimentations

Dans cette partie, nous comparerons les différentes méthodes présentées ci-haut sur les mêmes bases de données. Pour chaque base, et pour chaque méthode, nous donnerons la meilleure figure obtenue, un score de clustering et les résultats chiffrés avec la méthode de classification des k plus proches voisins. Les bases de données pour la comparaison seront les suivantes :

1. **digit** de dimension 64, composée de 10 classes, avec approximativement 180 échantillons par classe. Les attributs sont entiers et ont des valeurs entre 0 et 16.
2. **Classement mondial des universités** (variable "broad impact" supprimée) de dimension 12, et un total de 2200 observations. Les attributs sont réels.
   Les dimensions sont :
   world_rank

institution
national_rank
quality_of_education
alumni_employment
quality_of_faculty
publications
influence
citations
patents
score
year

Nous considérerons les pays comme étiquettes.

Pour ces deux bases, suite à plusieurs plantages de l'ordinateur lors du tracé des figures du knn pour les méthodes **TSNE** et les auto-encodeurs, nous ne les aurons donc pas.

Le choix de la "meilleure figure" reposera sur le critère suivant :
La meilleure des figures sera celle sur laquelle on peut distinguer clairement les clusters. Pour la qualité de la figure, nous calculerons le coefficient de silhouette associé.

**coefficient de silhouette**
Le coefficient de la silhouette (**Coef** sur les figures) est calculé en utilisant la distance intra-cluster moyenne (a) et la distance moyenne du cluster le plus proche (b) pour chaque échantillon. Le coefficient de la silhouette pour un échantillon est (b - a) / max (a, b). La meilleure valeur est 1 et la pire valeur est -1. Les valeurs proches de 0 indiquent des groupes chevauchant. Les valeurs négatives indiquent généralement qu'un échantillon a été attribué au mauvais cluster, car un cluster différent est plus similaire.

Les figures obtenues avec la méthode knn sont paramétrées comme suit :
poids : **distance**
n_neighbors : 15

1. **base de données digits**

   **Méthode : PCA**

   La meilleure figure est obtenue avec le solveur **full** :


FIGURE 20 – PCA

Comme on peut le voir sur la figure ci-dessus, toutes les classes sont mélangées, ce qui donne une représentation de mauvaise qualité dont le coefficient de silhouette obtenue confirme. Avec un knn sur les données réduit, nous obtenons la figure suivante :

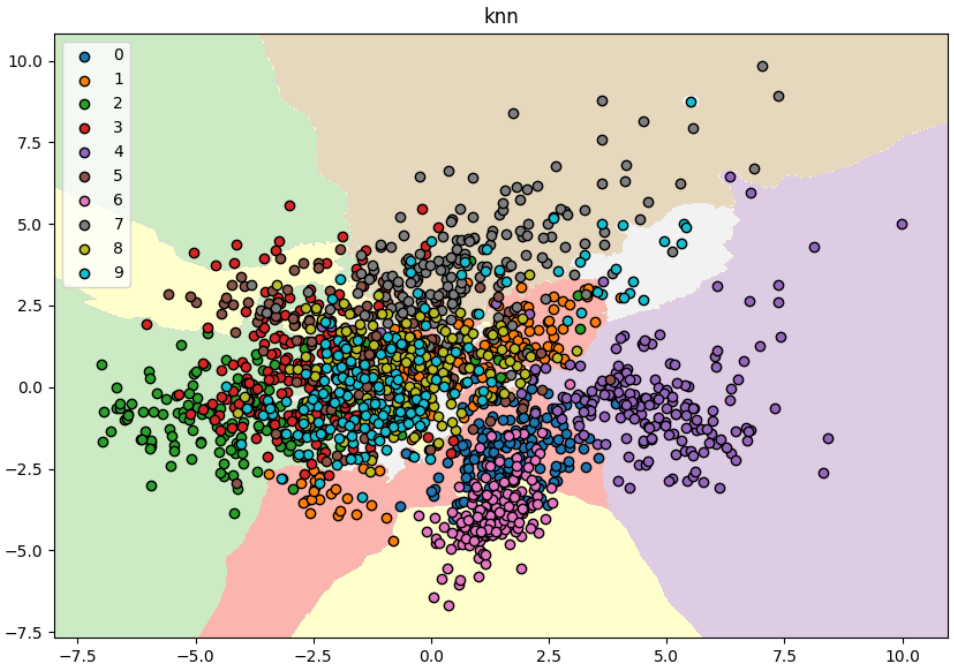


FIGURE 21 – knn sur la PCA

Sur cette figure, on remarque que certaines classes ont été plus ou moins retrouvées comme la classe 6 et 4, tandis que certaines sont totalement mélangées avec les autres comme la 9, et la 8.

**Méthode : KernelPCA**

La meilleure figure est obtenue avec le noyau rbf et gamma = 0.01.

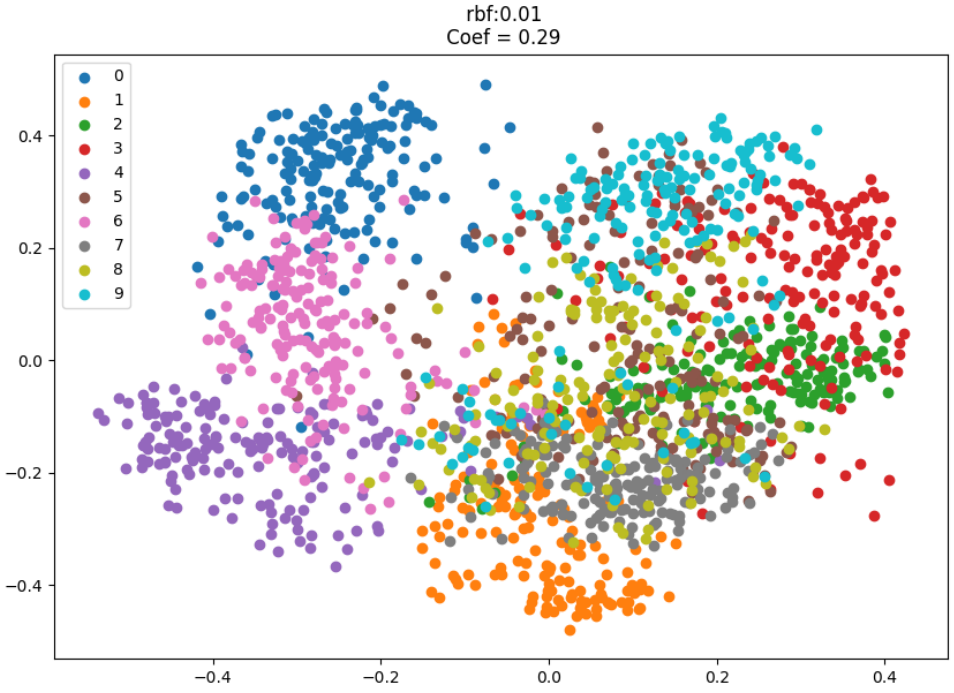


FIGURE 22 – Meilleure figure KPCA

Comme on peut le voir sur la figure, les classes 0, 6 et 4 on été bien retrouvée bien qu'ils soient un peu mélangées sur certaines parties de la figure. Les autres classes sont totalement mélangées et la classe la moins bien représentée est la 8. Par rapport à la méthode précédente, on a une meilleure qualité de représentation avec un coefficient 0.29.

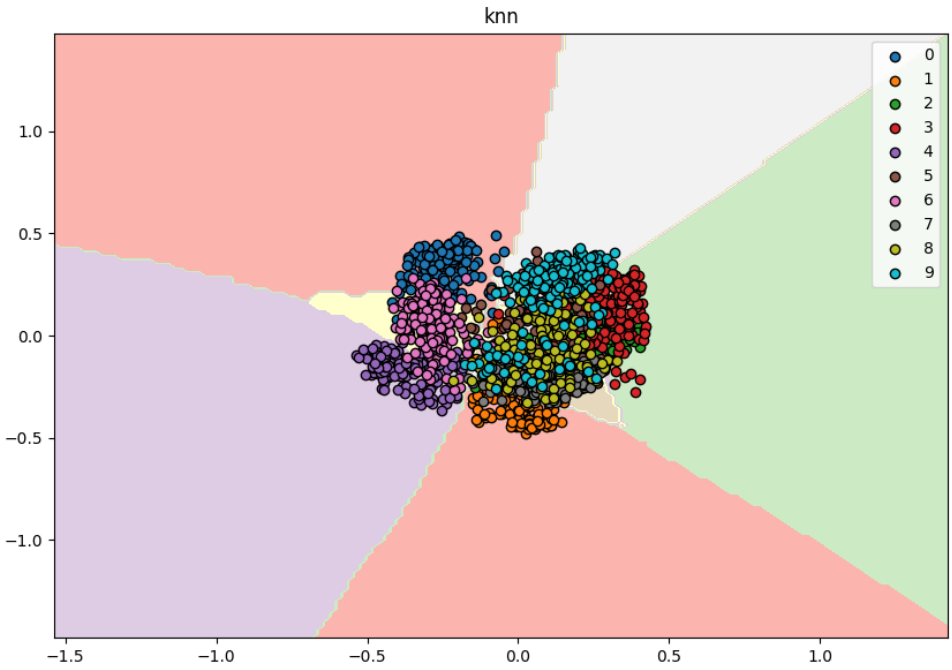


FIGURE 23 – knn sur la KPCA

Sur la figure du knn obtenue, on peut distinguer 6 classes : 0, 1, 3, 9, 4, 6. La classe 2 est quasi-invisible.

**Méthode : LLE**

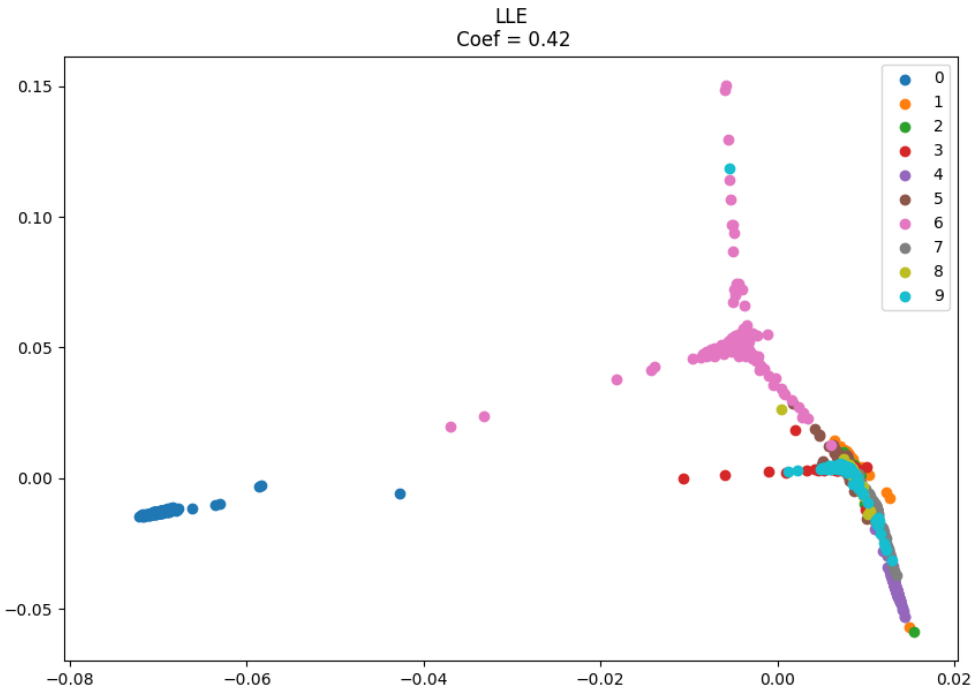


FIGURE 24 – LLE

La qualité de la figure est plutôt bonne avec un coefficient de **0.42**, plus élevés que par rapport aux méthodes précédentes. En effet, on peut distinguer en particulier trois classes : la 6, la 0 et la 9. Néanmoins les autres classes restent quasi-invisibles. Une particularité de cette méthode est que dans une classe, on a une région dense et une où les points sont éparpillés.

En appliquant le knn on obtient la figure ci-dessous :

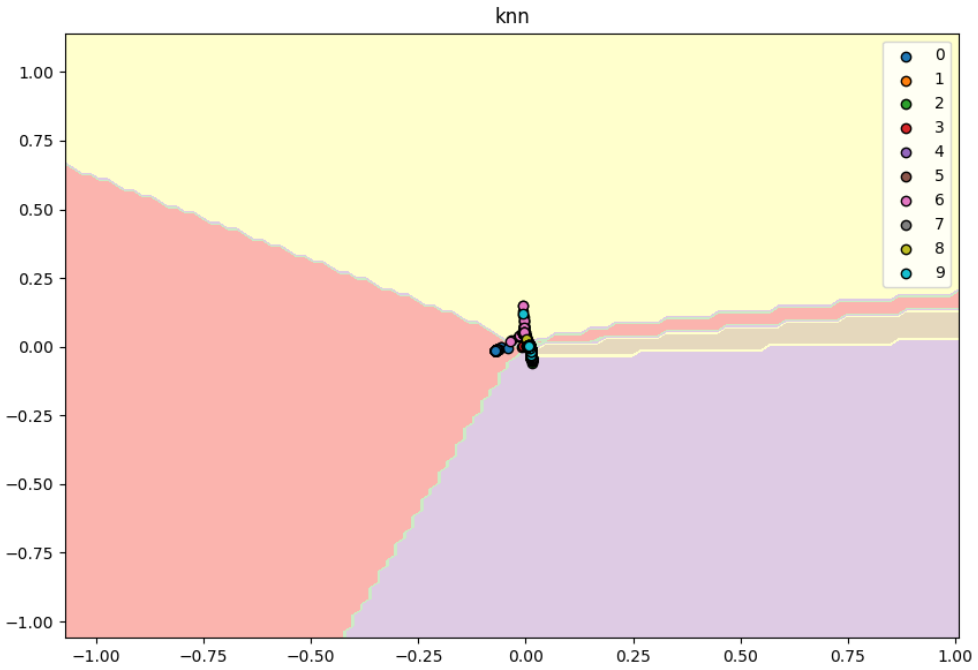


FIGURE 25 – knn sur la LLE

Comme on peut le remarquer, contrairement à la figure précédente, sur celle-ci on peut à peine voir une classe.

**Méthode : TSNE**

Pour cette méthode on obtient la figure suivante :

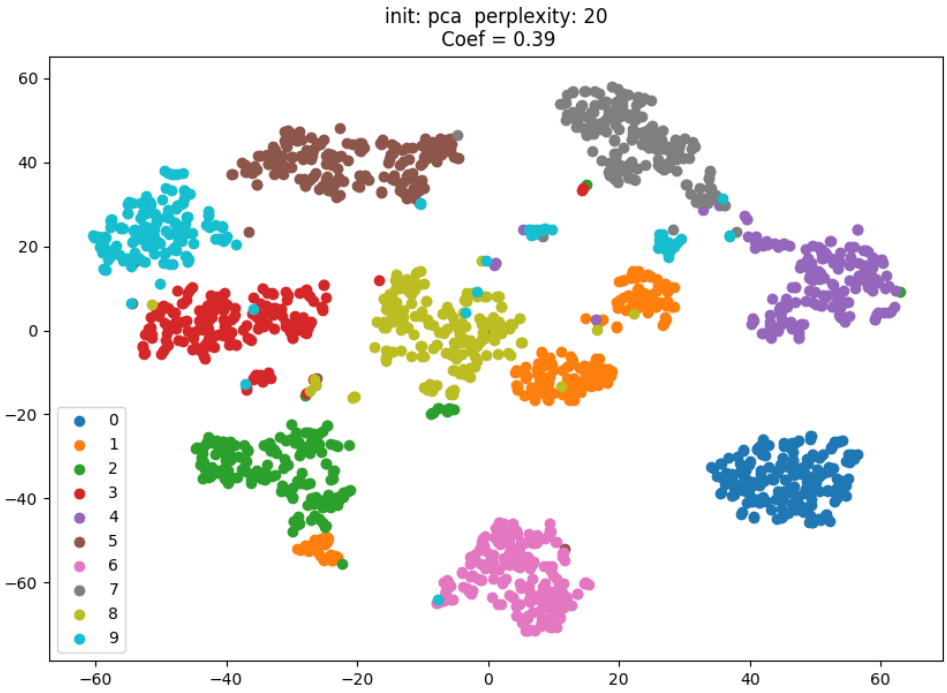


FIGURE 26 – TSNE

Toutes les classes ont été retrouvées, avec un coefficient de 0.39, on ne se doute pas de la bonne qualité de la silhouette. Néanmoins, les données de certaines classes, ne sont pas rattachées à celles-ci et se trouvent donc mélanger avec les autres classes. C'est le cas de la classe 9 et la 1.

**Méthode : LEM**

La meilleure figure est obtenue avec le paramétrage par défaut :

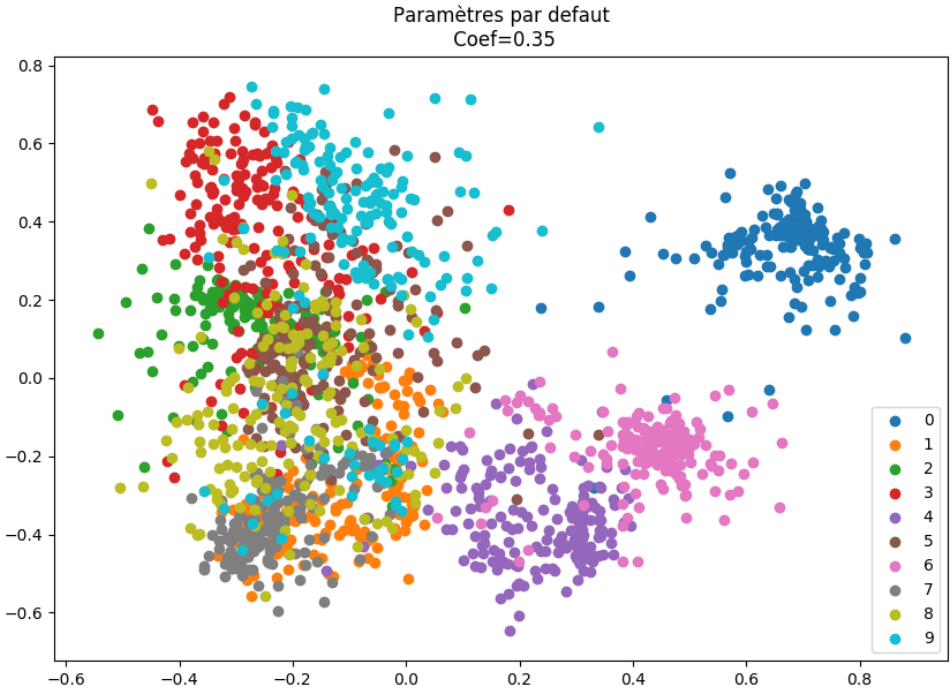


FIGURE 27 – LEM

Remarquons qu'une classe a été bien retrouvée et séparée des autres, la classe 0. Les autres classes sont mélangées. La qualité de la silhouette est plutôt bonne avec un coefficient de 0.35, comparée à certaines méthodes.

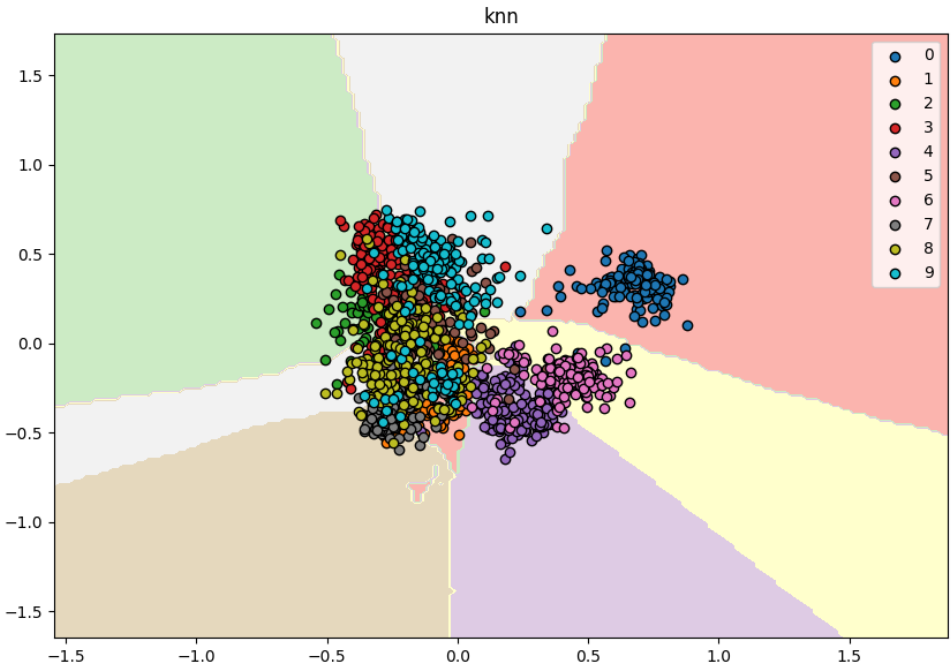


FIGURE 28 – knn sur la LEM

Sur la figure du knn, seule 3 classes se distinguent des autres.

**Méthode : AE**

La meilleure figure est obtenue avec l'algorithme d'optimisation **Adagrad** :

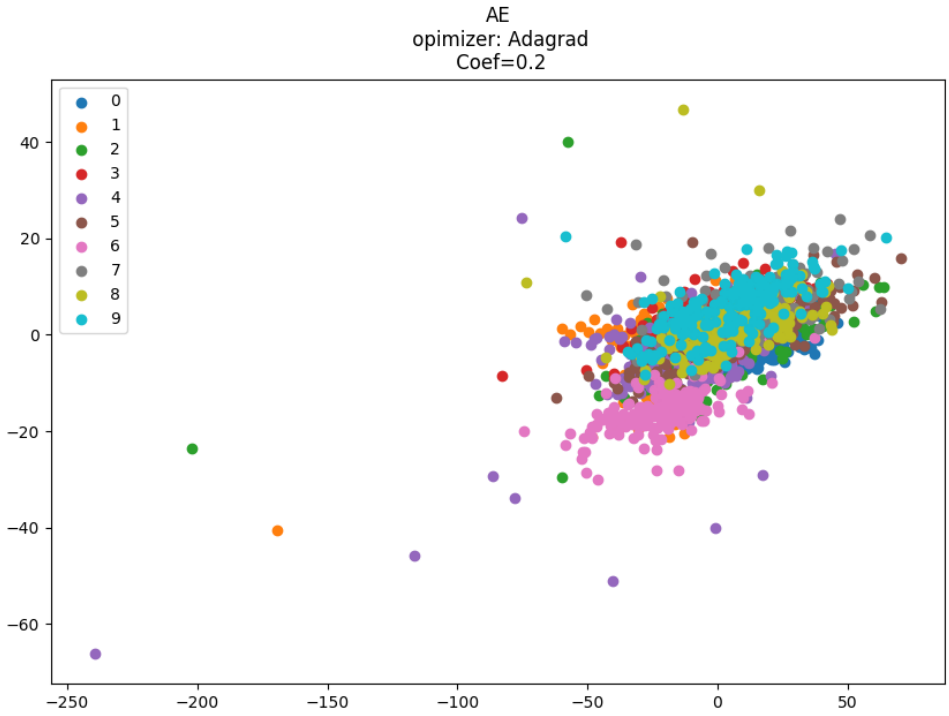


FIGURE 29 – Auto-encodeur

Toutes classes sont mélangées bien qu'on a un coefficient de silhouette de 0.2.

Pour cette base, la meilleure figure est obtenue avec la méthode TSNE et la moins bonne avec l'AE. On constate que pour toutes les méthodes, les classes sont partiellement ou totalement mélangées.

2. **Base de données classement mondial des universités**

**Méthode : PCA**

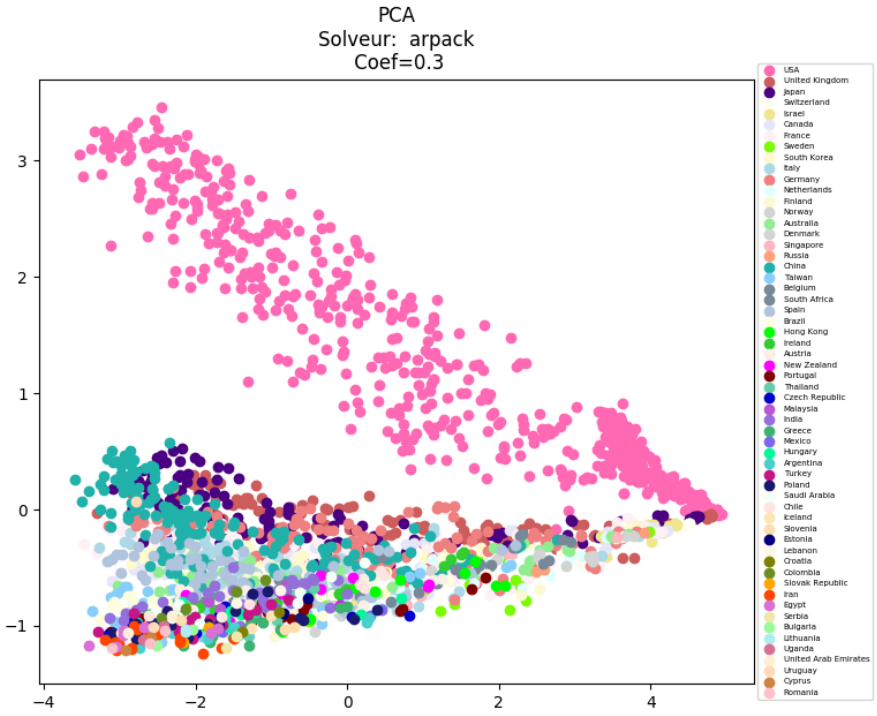


FIGURE 30 – PCA

La meilleure figure est obtenue avec le solveur **arparck** avec un coefficient de silhouette de **0.3**. On peut remarquer qu'on a principalement deux régions sur la figure : la première où sont représentées les universités des USA et la seconde pour les autres. Ainsi La classe la mieux retrouvée est **USA**, les autres classes restent très mélangées.

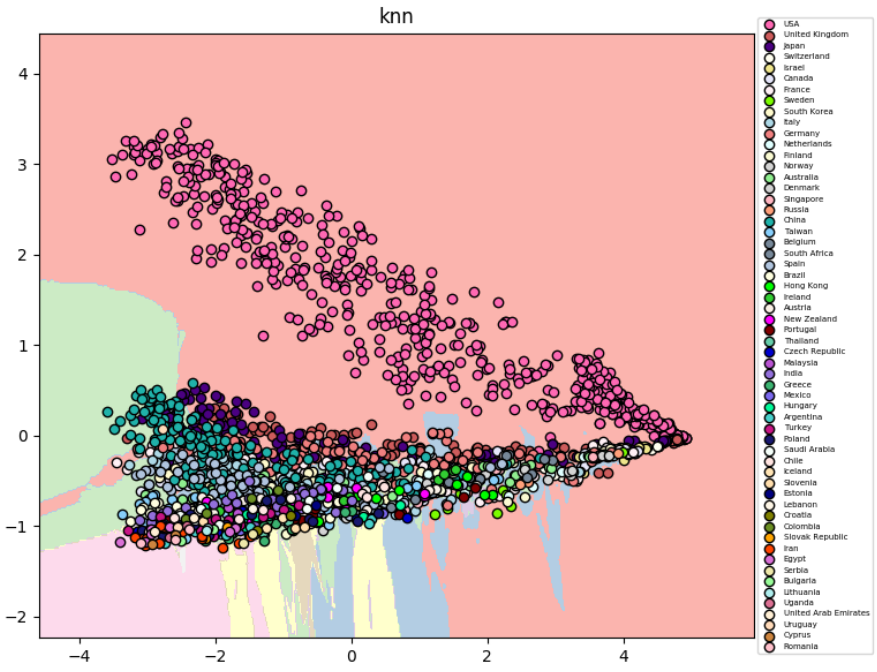


FIGURE 31 – knn sur la PCA

Sur cette figure, on peut voir que la zone de décision pour l'appartenance à la classe **USA** est très largement supérieure aux autres classes.

**Méthode : KPCA**

Les résultats pour cette méthode ressemblent beaucoup à ceux obtenus avec le PCA. La classe la mieux représentée est la même et est obtenue avec le noyau **sigmoid** avec gamma = 0.01.

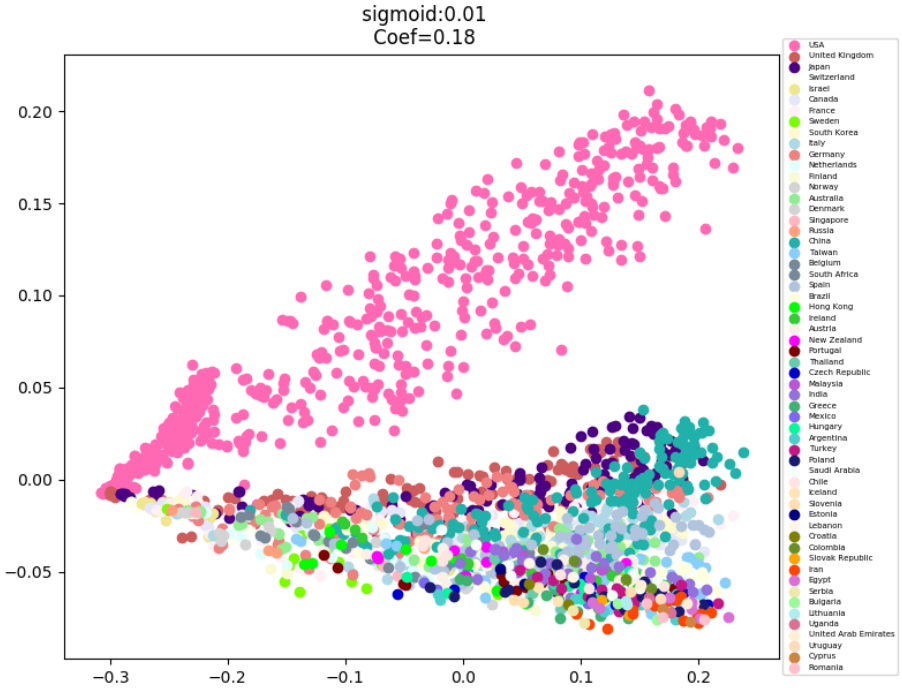


FIGURE 32 – KPCA

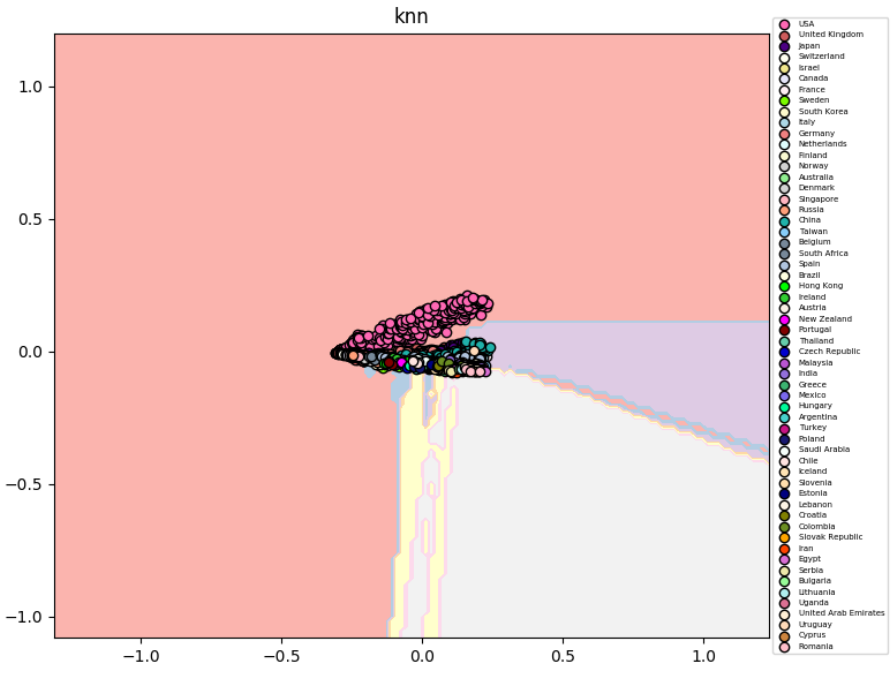


FIGURE 33 – knn sur la KPCA

Néanmoins pour le knn, on remarque sur la figure que les données sont très compactifiées. A part la classe **USA**, les autres sont quasi-invisibles.

**Méthode : LLE**

Une fois de plus avec cette méthode, la classe **USA** est la mieux représentée. Cependant, la qualité de silhouette obtenue est mauvaise. Les classes sont très mélangées ressemblant à la **tour Eiffel**.

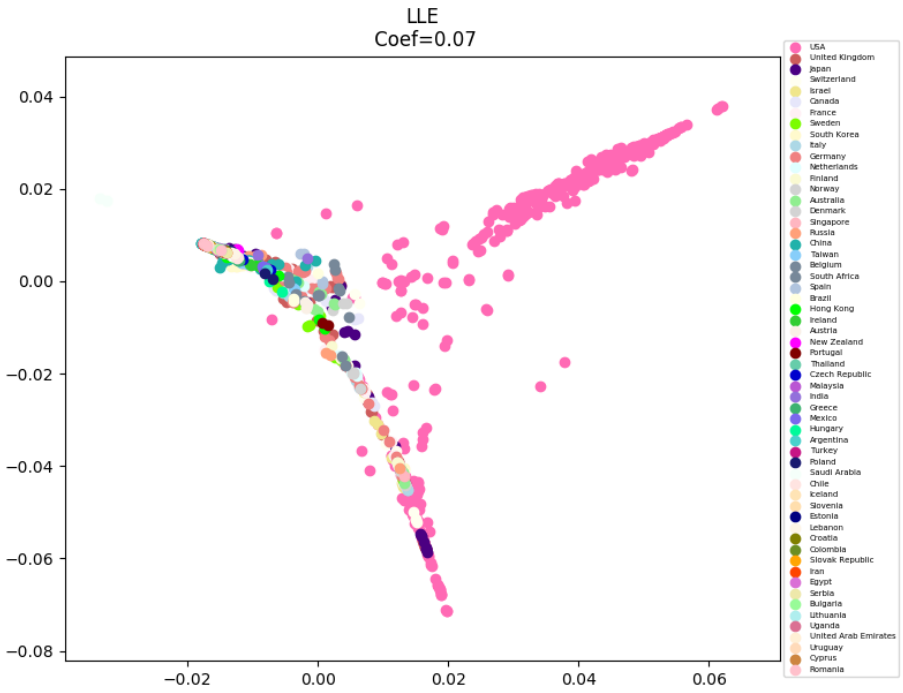


FIGURE 34 – LLE

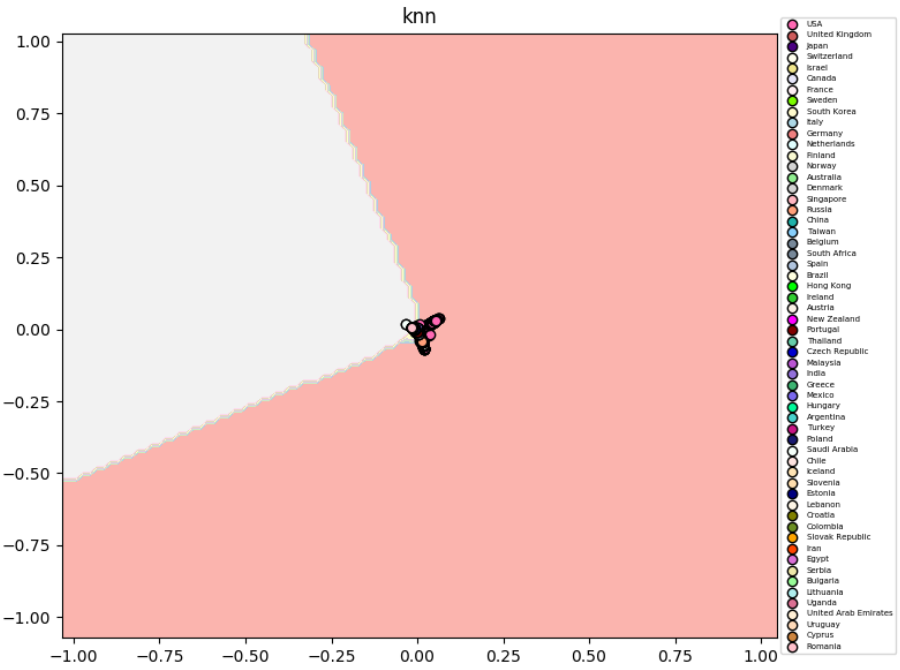


FIGURE 35 – knn sur la LLE

Comme on pouvait s'attendre suite aux résultats graphiques obtenus ci-haut et le coefficient de silhouette, la figure obtenue avec le knn est parmi les moins bonnes des méthodes testées.

**Méthode : TSNE**

Sur la base digits, les résultats obtenus avec la TSNE étaient plutôt bon. Néanmoins, sur cette base, l'algorithme a vraisemblablement eu des difficultés à retrouver toutes les classes. La meilleure figure est obtenue avec une initialisation = pca et une perplexité de 35. Comme pour les autres méthodes, la classe la mieux représentée reste la classe **USA**.

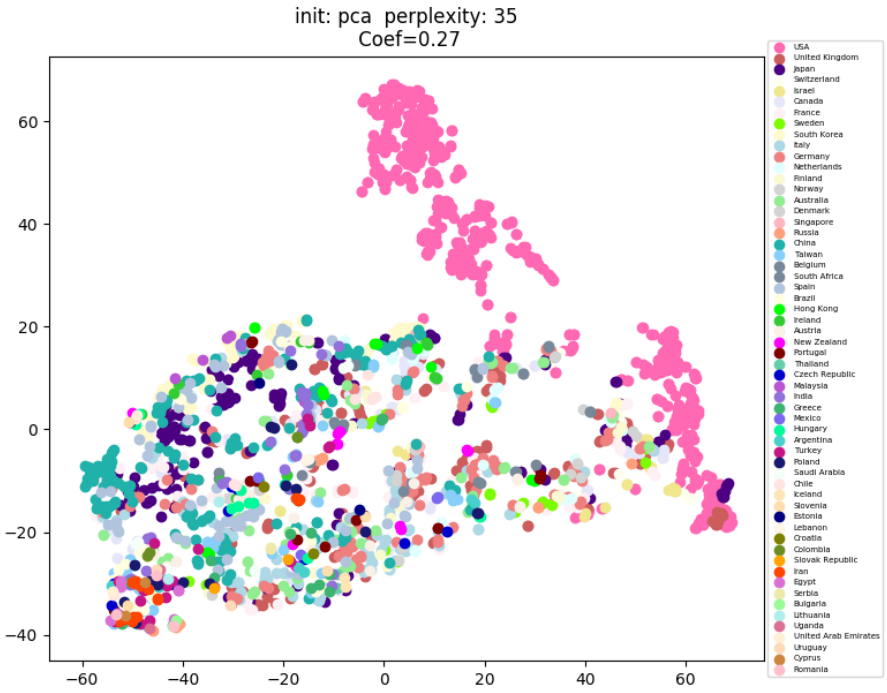


FIGURE 36 – LLE

**Méthode : LEM**

La meilleure figure pour cette méthode est obtenue avec **rbf** comme affinité, 0.01 pour gamma et eigen_solveur = **None**. Le coefficient de silhouette obtenue est 0.2, ce qui peut expliquer le fait que la majorité des classes soient mélangées.

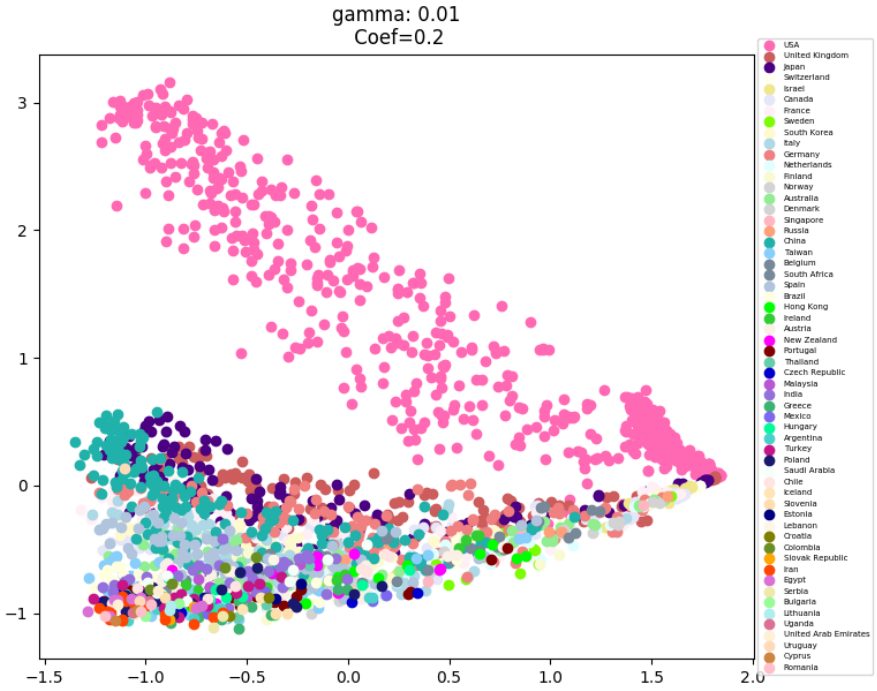


FIGURE 37 – LEM, affinity = rbf

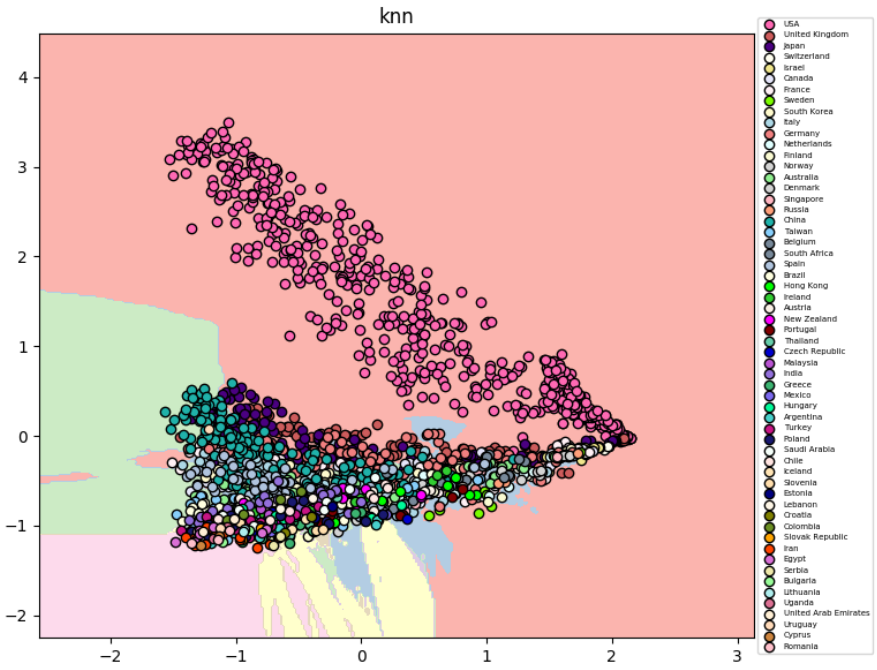


FIGURE 38 – knn sur la LEM

Pour cette figure, on peut faire les mêmes remarques avec les résultats obtenus avec le pca.

**Méthode : AE**

Pour cette méthode, aucune classe n'a été retrouvée. Ces dernières sont très mélangées. On remarque qu'il y a des points extrêmes comme ceux de la classe **Russia**.

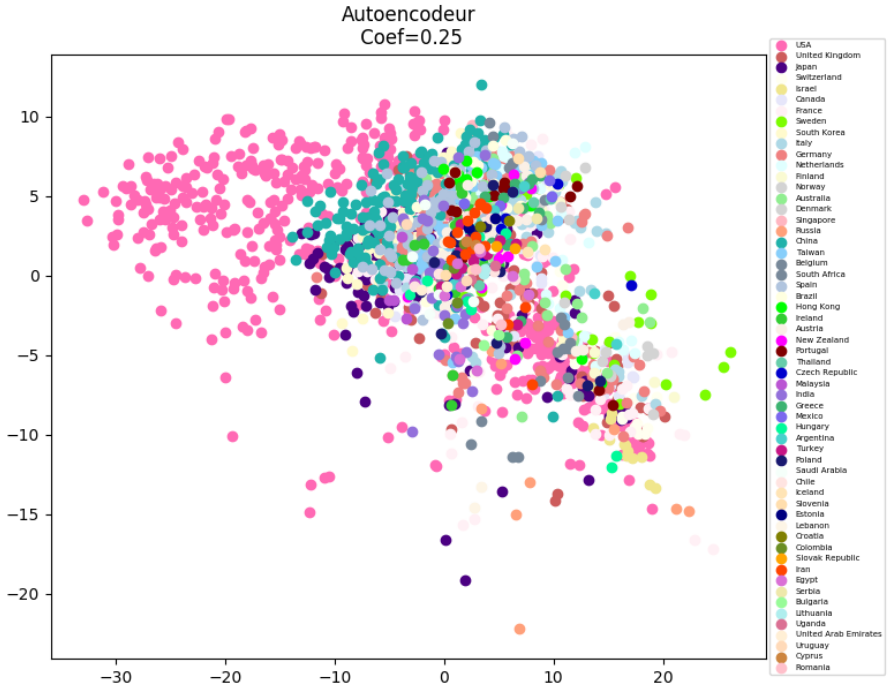


FIGURE 39 – AE

Ainsi pour cette base, la classe **USA** est majoritairement retrouvée. Pour toutes les méthodes, les données sont très mélangées bien que certains coefficient de silhouette sont relativement bon.

De manière générale, la méthode la plus efficace pour la décomposition de ces deux bases de données est la méthode à noyau PCA. Les moins bonnes sont la LLE et l'auto-encodeur. La méthode ayant donnée la meilleure décomposition est la TSNE avec la base de données digits.

# 5 Comparaisons des méthodes

Dans cette partie, nous comparerons les différentes méthodes en utilisant la méthode de classification du plus proche voisin sur les sorties. Pour ce fait, nous changerons quelques paramètres des méthodes et la dimension et calculerons les scores associés afin de déterminer la méthode la plus performante. Nous utiliserons la base de données **digits**.

Une particularité de la méthode TSNE est que pour la méthode de résolution "barnes hut", le nombre de composants retourner doit être inférieur à 4, et quant à la méthode "exact" sa complexité est $O(N^2)$, de ce fait nous ne comparerons cette méthode aux autres qu'en dimension 2.

Pour la LLE, le nombre de voisins doit être supérieur au nombre de composants. Nous prendrons le nombre de voisins égale au nombre de composants + 1, sauf pour la dimension où nous prendrons 10 afin de faire une comparaisons avec les résultats graphiques.

Divisons la base de données en deux(random state : 20) : une base d'apprentissage(80%) et une base de test. Dans un premier temps, pour les méthodes à noyau pour chaque noyau nous chercherons le gamma, le nombre de voisins, le poids pour le KNN, le CV pour la validation croisée optimaux, et pour les autres méthodes tous la paramètres précédents sauf le gamma, ensuite nous déterminerons parmi ces méthodes la plus performante.

Soit les paramètres suivant :
Poids = ['uniform','distance'] pour le knn, les valeurs correspondent aux valeurs possibles pour le poids de la méthode knn.
NbVoisin = [5,10,15,20] pour le KNN, ces valeurs sont choisies arbitrairement.
CV =[3,5,7] pour la validation croisée, choix arbitrairement également.

Calculons dans le score sur les données initiales. Rappelons que la dimension des données est 64.
On obtient les résultats suivants :

```
Poids   NbVoisin        CV       Scores
uniform  5              3  0.964498754234
uniform  5              5  0.971453221076
uniform  5              7  0.971451280421
uniform  10             3  0.9754661993
uniform  10             5  0.964522995294
uniform  10             7  0.966599583645
uniform  15             3  0.943616089222
uniform  15             5  0.958928212062
uniform  15             7  0.957546583846
uniform  20             3  0.93945792331
uniform  20             5  0.949190288932
uniform  20             7  0.949898208767
distance  5             3  0.968683018449
distance  5             5  0.972854379301
distance  5             7  0.972848825662
distance  10            3  0.965187459468
distance  10            5  0.966582431297
distance  10            7  0.967973303865
distance  15            3  0.954076995456
distance  15            5  0.963806702625
distance  15            7  0.961003800268
distance  20            3  0.949944764051
distance  20            5  0.954071469943
distance  20            7  0.9561651694

le meilleur parametrage est:
Poids: distance NbVoisin: 5     CV: 5   Score:0.972854379301
```

FIGURE 40 – Scores sur les données initiales

Ainsi le meilleur paramétrage sur les données initiales est :
Poids : distance
NbVoisin : 5
CV : 5

Score :97%
Et le score sur les données test est :**0.93**.

**Par la suite nous itérons les étapes de la figure 40 afin d'obtenir les paramètres optimaux** (i.e : ceux donnant le meilleur score) pour chaque méthode.

**Calculs des scores pour une réduction de dimension à 60** :

On obtient les résultats optimaux suivants :

| Kernels | Gamma | Poids | Nb Voisins | CV | Scores |
|---|---|---|---|---|---|
| Poly | $10^{-2}$ | Uniform | 5 | 5 | 0,97 |
| Rbf | $10^{-6}$ | Distance | 5 | 5 | 0,97 |
| Sigmoïde | $10^{-2}$ | Distance | 5 | 5 | 0,98 |
| Cosine | | Distance | 10 | 5 | 0,96 |

| Méthodes | Poids | Nb Voisins | CV | Scores |
|---|---|---|---|---|
| PCA | Distance | 5 | 5 | 0,97 |
| LEM | Distance | 5 | 7 | 0,94 |
| LLE | Uniform | 20 | 7 | 0,10 |
| Autoencodeur | Distance | 15 | 5 | 0,40 |

FIGURE 41 – Scores réduction à 60 dimensions

Comme on peut le constater, à l'exception de l'auto-encodeur et la LLE toutes les méthodes on un score supérieur à 90%. Notons que pour la LLE, la méthode requiert le nombre de voisin supérieur à la dimension (ici 61), ce qui peut expliquer la mauvaise performance. Pour l'auto-encodeur, ceci peut s'expliquer par l'initialisation aléatoire des variables.
On remarque également que la meilleur valeur du nombre de voisin pour le KNN et le CV pour le calcul du score est **5**. Quand au poids, la valeur "distance" semble être la mieux adaptée.
Enfin, la méthode la plus performante est la **sigmoïde** pour un gamma $10^{-2}$ et un score de 98%.

**Réduction de la dimension à 30 :**

| Kernels | Gamma | Poids | NbVoisins | CV | Scores |
|---|---|---|---|---|---|
| Poly | $10^{-3}$ | Distance | 5 | 7 | 0,97 |
| Rbf | $10^{-3}$ | Uniform | 5 | 7 | 0,97 |
| Sigmoïde | $10^{-2}$ | Distance | 5 | 5 | 0,98 |
| Cosine | | Distance | 5 | 5 | 0,96 |

| Méthodes | Poids | NbVoisins | CV | Scores |
|---|---|---|---|---|
| PCA | Distance | 5 | 7 | 0,97 |
| LEM | Distance | 5 | 5 | 0,94 |
| LLE | Distance | 5 | 7 | 0,53 |
| Autoencodeur | Distance | 20 | 7 | 0,49 |

FIGURE 42 – Scores réduction à 30 dimensions

Pour cette réduction, la méthode sigmoïde reste la plus performante avec le même paramétrage que ci-haut. Comme précédemment, sauf l'auto-encodeur et la LLE on un score inférieur à 90%, cependant ce dernier s'est un peu amélioré. On obtient ainsi 0.49 % et 0.53 % respectivement pour l'auto-encodeur et la LLE contre 0.40 % et 0.10 % précédemment. Remarquons pour toutes les méthodes, on obtient quasiment les mêmes scores qu'à la réduction à 60. Le paramètre ayant le plus changer est le CV. On passe par exemple de 5 à 7 pour Poly et Rbf. Et pour le poids, on passe de "distance" à "uniform" pour la Rbf et inversement pour Poly. Cependant, le gamma à considérablement augmenter pour la Rbf, passant de $10^{-6}$ à $10^{-3}$.

**Réduction de la dimension à 15 :**

| Kernels | Gamma | Poids | NbVoisins | CV | Scores |
|---|---|---|---|---|---|
| Poly | $10^{-4}$ | Distance | 10 | 7 | 0,96 |
| Rbf | $10^{-3}$ | Distance | 10 | 5 | 0,97 |
| Sigmoïde | $10^{-2}$ | Distance | 5 | 5 | 0,98 |
| Cosine | | Distance | 5 | 5 | 0,97 |

| Méthodes | Poids | NbVoisins | CV | Scores |
|---|---|---|---|---|
| PCA | Distance | 10 | 7 | 0,96 |
| LEM | Distance | 5 | 5 | 0,93 |
| LLE | Distance | 5 | 7 | 0,50 |
| Autoencodeur | Distance | 20 | 3 | 0,51 |

FIGURE 43 – Scores réduction à 15 dimensions

Pour cette réduction, on constate dans un premier temps, que tous les scores sont supérieurs ou égales à 50% et que certains ont légèrement baissés (PCA, Poly, LEM : 1%) et que le nombre de voisins à augmenter pour quasiment toutes les méthodes. Le noyau sigmoïde reste toujours la plus performante. Pour le poids, "distance" est la mieux adaptée.

**Réduction de la dimension à 2 :**

| Kernels | Gamma | Poids | NbVoisins | CV | Scores |
|---|---|---|---|---|---|
| Poly | $10^{-5}$ | Uniform | 15 | 7 | 0,57 |
| Rbf | $10^{-2}$ | Uniform | 20 | 3 | 0,64 |
| Sigmoïde | 30 | Distance | 20 | 7 | 0,58 |
| Cosine | | Uniform | 15 | 5 | 0,61 |

| Méthodes | Poids | NbVoisins | CV | Scores |
|---|---|---|---|---|
| PCA | Uniform | 15 | 7 | 0,57 |
| LEM | Uniform | 20 | 7 | 0,68 |
| LLE | Uniform | 20 | 7 | 0,77 |
| Autoencodeur | Distance | 15 | 7 | 0,47 |
| TSNE | Distance | 10 | 3 | 0,98 |

FIGURE 44 – Scores réduction à 2 dimensions

Pour cette réduction, le nombre de voisin de toutes les méthodes est au moins 10. On constate que le score a considérablement baissé pour la plupart des méthodes. Par exemple, sigmoïde passe de 98% à 58%. Néanmoins, celui de la LLE a augmenté pour passer à 77%. Pour le poids, toutes les méthodes à l'exception de sigmoïde et l'auto-encodeur la valeur optimale est "uniform". Et pour le gamma, sigmoïde passe de $10^{-2}$ à 30. La méthode la plus performante avec un score de 98% est la TSNE avec comme paramètres 10 pour le nombre de voisins de 3 pour le CV et distance pour le poids.

**Comparaison avec les résultats graphiques**
Pour cette réduction à 2 dimensions, nous pouvons comparer les résultats du tableau aux graphiques obtenus à la section expérimentations.

Pour le PCA, nous avons obtenu un score de 57%, en regardant la figure 20, on remarque que les classes ne sont quasiment pas séparées surtout les jaunes et les verts. Pour le LEM, certaines classes sont bien séparées des autres comme celle des bleus (figure 27), le score obtenu est 68%. Quant au LLE, étant quasiment la moins bonne en performance pour les grandes dimensions donne un score de 77%. Sur la figure 24, on peut voir que les classes sont séparées mais que certaines sont invisibles. En recompilant le programme, il s'avère que le score de la LLE change, probablement du à une initialisation aléatoire dans la méthode. Le plus grand score obtenu en dimension 2 est pour la TSNE, ce qui coïncide en performance avec les résultats graphiques.

Pour les méthodes à noyaux, graphiquement nous avons obtenu la meilleure figure avec le noyau **rbf**. Avec le même noyau, le score obtenu est 64%. On peut donc conclure que les méthodes à noyaux ont une bonne performance.

**Calculs des scores sur la base de test**
Dans cette partie nous calculerons les scores des différents algorithmes sur la base de test avec les paramètres optimaux obtenus.
Les résultats sont reportés dans le tableau ci-dessous

| | | **Paramètres optimaux** (base d'apprentissage) | | | | | |
|---|---|---|---|---|---|---|---|
| Dimensions | Méthodes optimales | Gamma | Distance | NbVoisins | CV | Score base d'apprentissage | Score base de test |
| 60 | Sigmoïde | $10^{-2}$ | Distance | 5 | 5 | 0,98 | 0,93 |
| 30 | Sigmoïde | $10^{-2}$ | Distance | 5 | 5 | 0,98 | 0,94 |
| 15 | Sigmoïde | $10^{-2}$ | Distance | 5 | 5 | 0,98 | 0,94 |
| 2 | LLE | | Uniform | 20 | 7 | 0,77 | 0,60 |
| 2 | Sigmoïde | 30 | Distance | 20 | 7 | 0,58 | 0,56 |
| 2 | Rbf | 10-2 | Uniform | 20 | 3 | 0,64 | 0,62 |
| 2 | Cosine | | Uniform | 15 | 5 | 0,61 | 0,58 |
| 2 | LEM | | Uniform | 20 | 7 | 0,68 | 0,70 |
| 2 | TSNE | | Distance | 10 | 3 | **0,98** | **0,93** |

FIGURE 45 – Scores sur la base de test

Comme on peut le voir, nous obtenons un score de plus de 90% pour la méthode à noyau sigmoïde avec les dimensions 60, 30 et 15. Pour la dimension 2, la meilleur méthode est la TSNE avec un score de 98% sur la base d'apprentissage et 93% sur la base de test. Remarquons que ces scores sont quasi-identiques à ceux obtenus sur les données initiales

**Remarques générales sur la réduction des dimensions** :
Sur la base d'apprentissage, suite aux différents calculs de scores effectués ci-dessus, il se relève que la méthode à noyau sigmoïde est la plus performante pour des réductions à de grandes dimensions avec un score de l'ordre de 98%. On remarque un même score pour les réductions à 60 et 30 dimensions, pour cette méthode, mais que ce dernier s'abaisse très fortement lorsqu'on passe à la dimension 2. On constate également que la valeur du poids passe de "distance" pour les grandes dimensions à "uniform" pour les très petites pour quasiment toutes les méthodes. Quant à la valeur de gamma, elle passe de $10^{-2}$ à 30 pour le sigmoïde. Et enfin la valeur du nombre de voisin passe de 5 pour les grandes dimensions au double voire triple pour les très petites.

Sur la base de test, un score de plus de 90% a été obtenu pour la méthode à noyau sigmoïde pour les réductions en grande dimension. Pour la dimension 2, la méthode la plus performante est la TSNE.

Nous pouvons ainsi conclure que les méthodes les plus performantes sont :
La méthode à noyau **sigmoïde** pour les dimension 60, 30 et 15 avec les paramètres suivants :
**gamma** $= 10^{-2}$**, poids = distance**.
Pour la dimension 2, la **TSNE** avec les paramètres :
**poids : distance, CV : 3, NbVoisins : 10**.

# 6 Conclusion

L'objectif de notre projet est la visualisation de données et la réduction de dimensionalité en utilisant quelques méthodes de projection. Après les descriptions de celles-ci, nous avons effectué des premiers tests pour évaluer l'influence des paramètres ou non. Suite à ces tests, nous avons remarqué que certains paramètres ont bien des effets sur l'algorithme associé comme le gamma ou le noyau pour les méthodes à noyaux, alors que d'autres méthodes sont plutôt insensibles (sur nos bases de données considérées) à certains paramètres comme le solveur du PCA. Par la suite, nous avons expérimenté les méthodes sur deux bases de données afin de comparer leur capacité de décomposition des données en clusters. Dans cette partie, nous avons également évaluer la qualité des figures grâce au coefficient de silhouette, et grâce aux graphiques obtenus avec le knn, nous avons pu visualiser les zones de décisions correspondants à chaque classe, ou du moins lorsque celle-ci est visible. Sur la première base, la meilleure figure parmi les meilleurs est obtenue avec la méthode TSNE, sur laquelle on pouvait distinguer toutes les classes, cependant quelques parties de ces classes sont mélangées avec les autres. Pour la deuxième base, quasiment toutes les méthodes ont retrouvé une classe : **USA**. Pour aller plus loin dans les comparaisons, nous avons calculé les scores de chaque méthode avec les paramètres optimaux obtenus. Ainsi, pour les grandes dimensions la méthode la plus performante est la méthode à noyau sigmoid, et pour les petites la TSNE. Dans cette partie, nous avons remarqué que pour les grandes dimensions, les scores obtenus pour les meilleures méthodes sur la base d'apprentissage et de test sont quasi-identiques à ceux obtenus sur la base initiale. Par contre, en basse dimension, certaines méthodes comme la sigmoid perdent en performance.

De manière générale, les méthodes les moins bonnes graphiquement et numériquement sont les auto-encodeurs et la LLE et les plus performantes sont les méthodes à noyau et la TSNE.

Il faut noter que du à des initialisations aléatoires dans certaines méthodes, les résultats peuvent être différents en utilisant le même paramétrage lors de la reproduction des expériences.

# 7 Bibliographie

[1] Roweis, S. & Saul, L. Nonlinear dimensionality reduction by locally linear embedding
[2] van der Maaten, L.J.P. t-Distributed Stochastic Neighbor Embedding
[3] Laplacian Eigmaps for dimensionality reduction and data representation, Mikhail Belkin, Partha Niyogi
[4] Wikipédia