# [Explorez vos données avec des algorithmes non supervisés](https://openclassrooms.com/fr/courses/4379436-explorez-vos-donnees-avec-des-algorithmes-non-supervises)

Table des matières

[Explorez vos données avec des algorithmes non supervisés 1](#_Toc57886857)

[**Comprenez pourquoi réduire la dimension de vos données** 3](#_Toc57886858)

[**Réduire les coûts** 3](#_Toc57886859)

[**Améliorer la qualité des modèles d'apprentissage** 4](#_Toc57886860)

[**Résumé** 5](#_Toc57886861)

[Calculez les composantes principales de vos données 7](#_Toc57886862)

[Analysez les composantes principales de vos données 7](#_Toc57886863)

[Calculer les composantes principales 7](#_Toc57886864)

[Comment choisir le nombre de composantes principales ? 9](#_Toc57886865)

[Résumé 11](#_Toc57886866)

[TP — ACP d’un jeu de données sur les performances d’athlètes olympiques 13](#_Toc57886867)

[Calcul des composantes principales 13](#_Toc57886868)

[Pourcentage de variance expliquée 14](#_Toc57886869)

[Contribution de chaque variable aux composantes principales 15](#_Toc57886870)

[Résumé 17](#_Toc57886871)

[Cherchez les variables latentes qui expliquent vos données 18](#_Toc57886872)

[L’ACP est une factorisation de la matrice des données 18](#_Toc57886873)

[Modélisation des relations de covariance entre les variables : l’analyse factorielle 18](#_Toc57886874)

[Cas des matrices non-négatives : la factorisation non-négative de matrices 19](#_Toc57886875)

[Résumé 21](#_Toc57886876)

[Partie 1 22](#_Toc57886877)

[Compétences évaluées 22](#_Toc57886878)

[ Question 1 22](#_Toc57886879)

[ Question 2 22](#_Toc57886880)

[ Question 3 24](#_Toc57886881)

[ Question 4 24](#_Toc57886882)

[ Question 5 25](#_Toc57886883)

[ Question 6 26](#_Toc57886884)

[ Question 7 27](#_Toc57886885)

[ Question 8 27](#_Toc57886886)

[Découvrez la réduction dimensionnelle non-linéaire 28](#_Toc57886887)

[Notion de non-linéarité 28](#_Toc57886888)

[Résoudre des problèmes non-linéaires 29](#_Toc57886889)

[Dans le cadre de la réduction dimensionnelle 33](#_Toc57886890)

[Résumé 35](#_Toc57886891)

[Utilisez une ACP avec un noyau 36](#_Toc57886892)

[Le problème : la complexité algorithmique 39](#_Toc57886893)

[Résumé 40](#_Toc57886894)

[Découvrez une variété qui conserve la structure globale 41](#_Toc57886895)

[MDS (Multidimensional Scaling) 41](#_Toc57886896)

[Une extension : Isomap 42](#_Toc57886897)

[Résumé 43](#_Toc57886898)

[Découvrez une variété qui favorise la structure locale 44](#_Toc57886899)

[Problème avec l'ACP et les méthodes globales 44](#_Toc57886900)

[LLE (Locally Linear Embedding) 45](#_Toc57886901)

[t-SNE (t-Stochastic Neighbour Embedding) 46](#_Toc57886902)

[Résumé 55](#_Toc57886903)

[Partie 2 56](#_Toc57886904)

[Compétences évaluées 56](#_Toc57886905)

[ Question 1 56](#_Toc57886906)

[ Question 2 56](#_Toc57886907)

[ Question 3 57](#_Toc57886908)

[ Question 4 57](#_Toc57886909)

[ Question 5 58](#_Toc57886910)

[ Question 6 59](#_Toc57886911)

[ Question 7 59](#_Toc57886912)

[**Découvrez l’intérêt des algorithmes de clustering** 60](#_Toc57886913)

[**Résumé** 62](#_Toc57886914)

[Définissez les critères que doit satisfaire votre clustering 63](#_Toc57886915)

[Résumé 68](#_Toc57886916)

[Partitionnez vos données avec un algorithme de clustering hiérarchique 70](#_Toc57886917)

[Résumé 74](#_Toc57886918)

[Partitionnez vos données avec l’algorithme du k-means 75](#_Toc57886919)

[Partitionnez vos données avec l'algorithme du k-means 75](#_Toc57886920)

[Résumé 78](#_Toc57886921)

[Partitionnez vos données avec DBSCAN 79](#_Toc57886922)

[Comment séparer des cercles imbriqués les uns dans l’autre ? 79](#_Toc57886923)

[Clustering par densité 80](#_Toc57886924)

[DBSCAN 81](#_Toc57886925)

[Avantages et inconvénients de DBSCAN 83](#_Toc57886926)

[Entraînez-vous à manipuler des algorithmes de clustering avec sklearn 84](#_Toc57886927)

[À vous de jouer 84](#_Toc57886928)

[Vérifiez votre travail 85](#_Toc57886929)

Dans le cours [Initiez-vous au machine learning](https://openclassrooms.com/courses/initiez-vous-au-machine-learning), vous avez découvert comment transformer une question que vous avez sur vos données en un problème d’apprentissage automatique non supervisé. Dans ce cours, vous apprendrez à choisir et utiliser les principaux algorithmes qui permettent de résoudre ces problèmes.

Vous découvrirez comment **réduire la dimension de vos données** grâce à des **techniques linéaires** comme l’analyse en composantes principales (**ACP**), ou des techniques non linéaires comme le très populaire **t-SNE**. Vous découvrirez aussi comment fonctionnent trois familles d’algorithmes de clustering : le **clustering hiérarchique**, **k-means** et le **clustering par densité**.

Suivez ce cours pour apprendre à réduire la dimension de vos données, mieux les visualiser ou pour **rendre vos algorithmes plus efficaces**, et pour découvrir comment segmenter automatiquement vos données, sans avoir à définir des classes a priori.



Ce cours a été créé en partenariat avec l'école CentraleSupélec

**Objectifs pédagogiques :**

* Comprendre à quelles questions un **algorithme d’apprentissage non supervisé** permet de répondre,
* Utiliser les principaux **algorithmes de réduction de dimension** non supervisé classique (**ACP**, **analyse factorielle**, **factorisation de matrice non négative**, **MDS**, **tSNE**),
* Choisir un algorithme de réduction de dimension non supervisé en fonction de vos besoins et des caractéristiques des données,
* Utiliser **les principaux algorithmes de clustering** (k-means, DBSCAN, clustering hiérarchique),
* Choisir un algorithme de clustering en fonction de vos besoins et des caractéristiques des données.

**Prérequis :**

Ce cours de Data Science se situe au croisement des mathématiques et de l'informatique. Pour en profiter pleinement, n'hésitez pas à vous rafraîchir la mémoire, avant ou pendant le cours, sur :

* **Python** pour le calcul numérique (numpy) et la création de graphiques (pyplot), que nous utiliserons dans les parties TP du cours,
* Quelques notions d'**algèbre linéaire** : manipulation de vecteurs, multiplications de matrices, normes, et valeurs/vecteurs propres,
* Quelques notions de **probabilités et statistiques**, telles que distribution de loi de probabilité et variance.
* [**Le cours d'initiation**](https://openclassrooms.com/fr/courses/4297211-evaluez-et-ameliorez-les-performances-d-un-modele-de-machine-learning), qui vous permettra de situer les algorithmes non supervisées au sein de l'ensemble des méthodes de machine learning

**Comprenez pourquoi réduire la dimension de vos données**

Nos données sont représentées sous la forme d'une matrice X de dimension n×p, où n est le nombre d'observations et p le nombre de variables les représentant. p est généralement un nombre assez grand, qui peut aller jusqu'à plusieurs dizaines de milliers dans certaines applications. C'est le cas par exemple lorsqu'on traite des images en haute résolution, et que chaque variable représente un pixel de cette image.

Dans cette partie, nous allons étudier des techniques non-supervisées permettant de **réduire ce nombre de variables.** Il s'agira de trouver mm variables, avec m<p, que nous allons choisir d'utiliser pour construire une nouvelle matrice , de dimension n×m pour représenter nos données.

**Visualiser les données**

Nous pouvons assez facilement représenter des données en 2 ou 3 dimensions. Mais au-delà, on est obligé de regarder les variables paires par paire (ou triplet par triplet). Ça devient assez vite difficile voire impossible quand le nombre de dimensions augmente. Si l'on pouvait représenter nos données avec 2 ou 3 dimensions seulement, ce serait beaucoup plus simple !

**Réduire les coûts**

Si nous pouvons représenter nos données en quelques dimensions, cela fait aussi moins d'information à stocker, ce qui réduit le **coût en espace mémoire.**

Par ailleurs, avoir moins de variables réduit la complexité des algorithmes d'apprentissage que nous pouvons utiliser, et donc les **temps de calcul.**

Enfin, si certaines variables sont inutiles, il n'est pas nécessaire de les obtenir pour de nouvelles observations : cela peut réduire le **coût d'acquisition** des données.

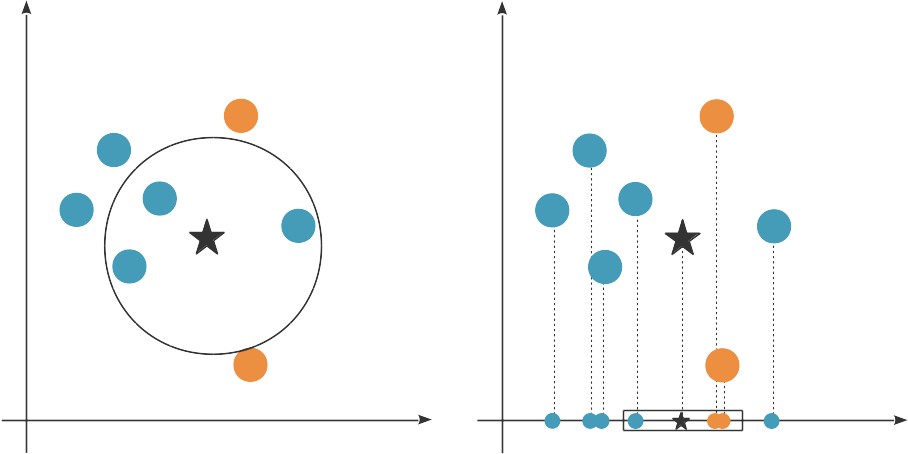


La réduction de dimension permet aussi de réduire les coûts

**Améliorer la qualité des modèles d'apprentissage**

En utilisant moins de variables, on peut construire des modèles avec moins de paramètres, donc plus simples, ce qui est généralement préférable pour **éviter le sur-apprentissage.**

De plus, **les variables non pertinentes peuvent induire l'algorithme d'apprentissage en erreur**. Prenons comme exemple l'algorithme des plus proches voisins, qui associe à une observation la même étiquette que celle de la majorité des k points d'entraînement les plus proches. Si l'on utilise la distance euclidienne, toutes les variables comptent autant dans le calcul des plus proches voisins. Mais si l'une de ces variables n'est pas pertinente, elle peut changer la définition du plus proche voisin, et introduire du bruit dans le modèle.



En utilisant les deux dimensions, les 3 plus proches voisins de l'étoile sont majoritairement bleus. En utilisant seulement la variable en abscisse, les 3 plus proches voisins sont majoritairement orange. Si la variable en ordonnée n'est pas pertinente, elle fausse l'algorithme.

Enfin, le **fléau de la dimension** (*curse of dimensionality* en anglais) est le terme que nous utilisons pour qualifier le fait qu'il est très difficile de faire de l'apprentissage en haute dimension. En effet, en haute dimension, on a besoin de beaucoup plus de points pour couvrir tout l'espace. Voir aussi le chapitre correspondant [dans le cours Initiez-vous au machine learning](https://openclassrooms.com/courses/initiez-vous-au-machine-learning/gerez-le-fleau-de-la-dimension).

Une autre façon de présenter le fléau de la dimension est de dire que toutes les observations sont loin les unes des autres, et il est très difficile de trouver ce qu'elles peuvent avoir de commun ou de différent. Les méthodes ou intuitions qui marchent en petite dimension ne marchent pas nécessairement en grande dimension.

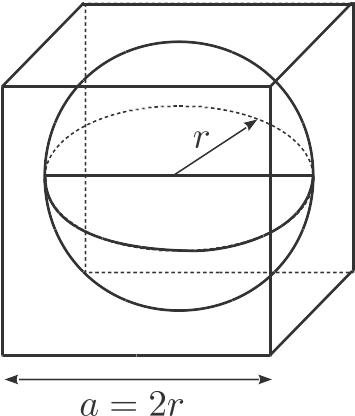
On en conclut que la réduction de dimension est nécessaire à la qualité de l'apprentissage.

Pour comprendre pourquoi je dis que toutes les observations sont loin les unes des autres, on peut se placer en dimension p et regarder la proportion du volume d'un hypercube comprise à l'extérieur de l'hypersphère inscrite dans cet hypercube.

En dimension 2, prenons un carré de côté a. Sa surface est . Le disque inscrit dans ce carré a pour surface . La surface entre ce disque et le carré vaut donc , et couvre donc une proportion   de la surface du carré.

En dimension p, prenons un hypercube de côté **a=2r** . Son volume est . Le volume d'une sphère de rayon r est donné . Par conséquent, la proportion qui nous intéresse vaut , qui tend vers 1 lorsque p tend vers l'infini.

La fonction **Γ** est une généralisation de la factorielle : si N est un entier, **Γ(N)=(N−1)!**.



Ici en dimension 3, on considère la proportion du volume du cube de côté a=2r située à l'extérieur de la sphère de rayon r inscrite dans ce cube.

**Résumé**

Réduire la dimensionalité des données, c'est-à-dire le nombre de variables utilisées pour les représenter, permet :

* de **faciliter la visualisation** des données ;
* de **réduire les coûts**de calcul, de stockage et d'acquisition des données ;
* d'**améliorer l'apprentissage** en construisant des **modèles moins complexes**, en éliminant les variables non pertinentes qui pourraient **fausser les prédictions** et enfin en réduisant le problème du **fléau de la dimensionalité.**

Dans la suite de cette partie, nous allons voir des techniques *linéaires* de réduction de dimension non-supervisée.

Lorsque les données sont *étiquettées*, il existe de nombreuses autres techniques permettant de réduire leur dimension. Certaines, comme la LDA (analyse discriminante linéaire, sur laquelle vous pouvez en lire plus [**ici (en anglais)**](http://sebastianraschka.com/Articles/2014_python_lda.html), sont des extensions des techniques d'*extraction de variables* dont nous allons parler dans cette partie : il s'agit de créer un petit nombre de nouvelles variables par lesquelles représenter les données. D'autres sont des techniques de *sélection de variables* : il s'agit non pas de construire de nouvelles variables, mais d'éliminer les variables les moins informatives.

## Calculez les composantes principales de vos données

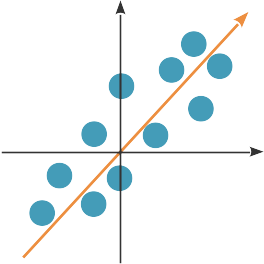
### 

### Analysez les composantes principales de vos données

Dans ce chapitre, nous allons parler d’une méthode très utilisée pour réduire la dimension d’un jeu de données : l’**analyse en composantes principales**, ou **ACP**. On parle aussi souvent de **PCA**, de son nom anglais Principal Components Analysis.

#### Objectif : maximiser la variance

Le but d'une analyse en composantes principales est de trouver une nouvelle base orthonormée dans laquelle représenter nos données, telle que la variance des données selon ces nouveaux axes soit maximisée.



Variance des données

La variance des données selon l'axe orange est grande. Si on projette les points sur cet axe, ils auront tous des coordonnées différentes ; en utilisant cet axe comme unique dimension, on réduit la dimension de nos données (de 2 à 1) mais on continue à pouvoir distinguer les points les uns des autres.

### Calculer les composantes principales

Supposons que nous ayons p variables : chaque observation est représentée par un vecteur dans , et nous avons n observations rassemblées dans une matrice .

Commençons par chercher une nouvelle direction, à savoir un vecteur , de norme 1, tel que la variance de nos données projetées sur cette direction soit maximale. La projection des données X sur .

Dans ce qui suit, nous allons supposer que X est centrée, c'est-à-dire que la moyenne de ses colonnes est 0. Si ce n'est pas le cas, on peut utiliser la transformation  où **μ**  est un vecteur contenant la moyenne de chaque colonne (variable).

La variance de  est égale à , soit, comme  (les données sont centrées), .

Appelons **Σ** la matrice de taille **p×p** égale à . C'est la matrice de covariance des données et une estimation de .

Nous cherchons donc **w1** tel que

*  est maximale
* ||w1||=1

**Σ** est, par construction, une matrice symétrique définie positive. Elle est donc diagonalisable par un changement de base orthonormé : , où  est une matrice diagonale dont les valeurs diagonales sont les valeurs propres de **Σ**, qui sont toutes positives.

Nous avons donc 

Posons **v=Qw1**, nous cherchons à maximiser . Comme **||w1||=1** et que Q est orthonormée, **||v||=1** et donc . La somme  est donc maximisée pour un vecteur **v** qui a une seule entrée à 1 et les autres à 0, l'entrée à 1 étant pour la valeur maximale des **λj**, autrement dit **la plus grande valeur propre de Σ.**

Par conséquent, **w1** est **le vecteur propre correspondant à la plus grande valeur propre de Σ.** C'est la **première composante principale de X**.

La **deuxième composante principale de X** doit vérifier les mêmes critères que **w1** : être de norme 1 et maximiser , mais lui est orthogonale. Il s'agit donc du vecteur propre de **Σ** correspondant à sa **deuxième plus grande valeur propre**.

Et ainsi de suite pour les autres composantes principales.

Nous pouvons donc utiliser comme nouvelles dimensions **la base formée par les vecteurs propres de la matrice de covariance des données.**

Il est possible d'obtenir cette base directement en calculant la décompostion en valeurs singulières (ou SVD) de X,   avec  orthogonale,  orthogonale, et  diagonale.

Les valeurs singulières de X, à savoir les entrées de D, sont les racines carrées des valeurs propres de **Σ**, tandis que les vecteurs singuliers de X sont les vecteurs propres de **Σ**.

En effet, .

L'intérêt d'utiliser une SVD de X plutôt qu'une décomposition en valeurs propres de est la stabilité numérique des méthodes implémentant la première.

### Comment choisir le nombre de composantes principales ?

La méthode que nous venons de décrire nous permet de construire autant de composantes principales que **Σ** a de vecteurs propres, soit autant que le nombre de descripteurs p de nos données. Nous n'avons pas encore réduit la dimension de nos données.

Pour ce faire, nous allons regarder la proportion de variance expliquée par chacune des composantes principales.

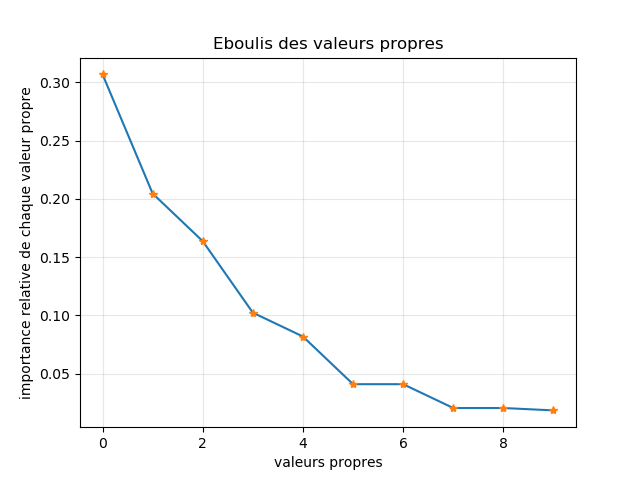
La variance totale du jeu de données est donnée par la somme des termes diagonaux de la matrice de covariance Σ (autrement dit la somme des variances de toutes les variables), soit sa trace. Or .

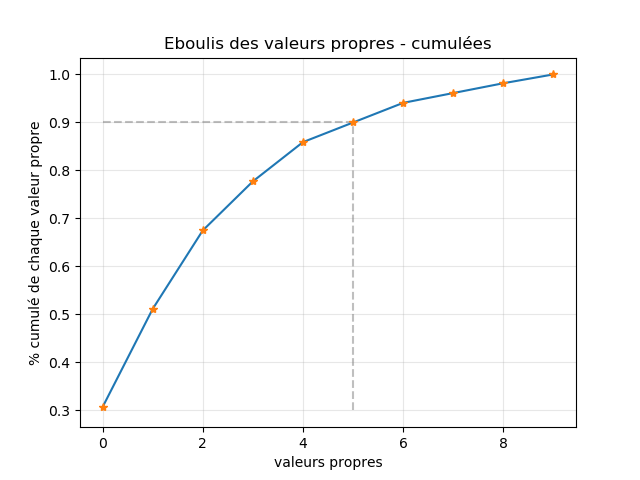
La variance totale est donnée par λ1+λ2+⋯+λp, et celle expliquée par les k premières composantes principales, par $ λ1+λ2+⋯+λk $. La proportion de variance expliquée par les k premières composantes principales est donc .

Nous pouvons maintenant regarder comment cette proportion évolue en fonction du nombre de composantes, et construire ce qu'on appelle en anglais un **scree plot**. Ce graphique présente la proportion de variance expliquée par la k-ième composante principale, ou par les k premières composantes principales, en fonction de k.

On peut l'utiliser pour choisir :

* le nombre de composantes principales qui explique un pourcentage de la variance que l'on s'est initialement fixé (par exemple, 90%)
* le nombre de composantes principales correspondant au « coude » du graphe, à partir duquel ajouter une nouvelle composante principale ne fait pas grande différence.

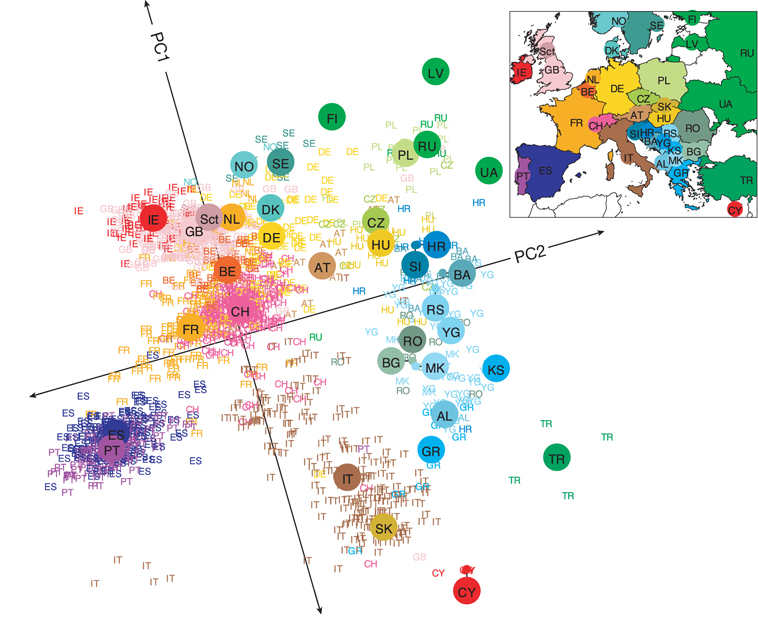


le pourcentage de variance expliqué par chacune des composantes principales. À partir de 5 composantes principales, ajouter une composante n'est pas très informatif.

le pourcentage cumulé de variance expliqué par chacune des composantes principales. Si on se fixe une proportion de variance expliquée de 90%, on peut se contenter de 5 composantes principales.

Une autre possibilité consiste à utiliser seulement deux ou trois composantes pour visualiser les données.

Un exemple de ce que peut faire une ACP : en 2008, John Novembre et ses collègues ont publié une analyse en composantes principales des génomes de 1387 Européens. La projection des données sur ces deux dimensions correspond assez bien à... la carte de l'Europe ! Ce résultat a été publié [**dans la revue Nature**](https://www.nature.com/nature/journal/v456/n7218/full/nature07331.html).



Les deux premières composantes principales des génomes de 1387 Européens correspondent à... une carte de l'Europe

Vous trouverez plus de détails par exemple sur [**ce billet de Science Étonnante**](https://sciencetonnante.wordpress.com/2011/05/16/comment-nos-genes-trahissent-nos-origines/).

### 

### Résumé

* Les composantes principales forment une base orthonormée et sont construites pour que les données aient une variance maximale selon ces nouveaux axes.
* Les composantes principales sont en fait les vecteurs propres de la matrice de covariance des données, classées par ordre décroissant de valeur propre correspondante.
* Pour choisir le nombre de composantes à utiliser, on regarde la proportion de la variance totale expliquée par k composantes.

## TP — ACP d’un jeu de données sur les performances d’athlètes olympiques

Nous allons maintenant faire une analyse en composantes principales des performances d'athlètes olympiques au décathlon.

Vous pouvez télécharger les données depuis l'URL suivante : [**http://factominer.free.fr/factomethods/datasets/decathlon.txt**](http://factominer.free.fr/factomethods/datasets/decathlon.txt)

Ce jeu de données contient 41 entrées, qui décrivent chacune les performances d'un athlète à une compétition de décathlon. Nous allons faire une ACP des dix variables décrivant les performances à chacune des épreuves du décathlon (100 mètres, saut en hauteur, lancer de poids, saut en hauteur, 400 mètres, 110 mètres haies, lancer de disque, saut à la perche, javelot, et 1500 mètres).

import pandas as pd

# charger les données

data = pd.read\_csv('decathlon.txt', sep="\t")

# éliminer les colonnes que nous n'utiliserons pas

my\_data = data.drop(['Points', 'Rank', 'Competition'], axis=1)

# transformer les données en array numpy

X = my\_data.values

Rappelez-vous : nous avons fait l'hypothèse que les données étaient centrées. Nous allons donc commencer par les standardiser.

from sklearn import preprocessing

std\_scale = preprocessing.StandardScaler().fit(X)

X\_scaled = std\_scale.transform(X)

### Calcul des composantes principales

Calculons maintenant les deux premières composantes principales :

from sklearn import decomposition

pca = decomposition.PCA(n\_components=2)

pca.fit(X\_scaled)

### Pourcentage de variance expliquée

pca.explained\_variance\_ratio\_ nous donne le pourcentage de variance expliquée par chacune des composantes.

print(pca.explained\_variance\_ratio\_)

print(pca.explained\_variance\_ratio\_.sum())

La première composante explique environ un tiers de la variance observée dans les données et la deuxième 17.3%. Au total, ces deux composantes expliquent 50% de la variance totale, en utilisant seulement un cinquième des dimensions initiales.

Nous pouvons représenter chaque athlète/compétition selon ces deux dimensions uniquement, et colorer chacun des points correspondant en fonction du classement de l'athlète lors de cette compétition.

# projeter X sur les composantes principales

X\_projected = pca.transform(X\_scaled)

# afficher chaque observation

plt.scatter(X\_projected[:, 0], X\_projected[:, 1],

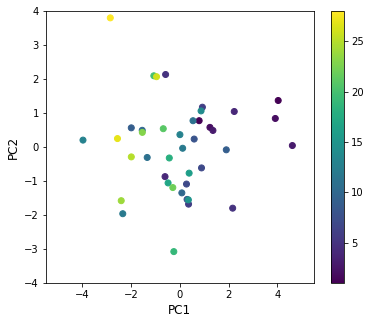
# colorer en utilisant la variable 'Rank'

c=data.get('Rank'))

plt.xlim([-5.5, 5.5])

plt.ylim([-4, 4])

plt.colorbar()

0

Données projetées sur les deux premières composantes principales et colorée par classement final.

Les bonnes performances (points bleu foncé) sont plutôt situées dans la partie droite du graphe (PC1 > 0) et les moins bonnes (points jaunes) plutôt dans la partie gauche (PC1 < 0).

### Contribution de chaque variable aux composantes principales

Pour mieux comprendre ce que capture ces composantes principales, nous pouvons utiliser pca.components\_, qui nous donne les coordonnées des composantes principales dans l'espace initial (celui à 10 variables). Nous allons afficher, pour chacune des 10 performances, un point dont l'abscisse sera sa contribution à la première PC et l'ordonnée sa contribution à la deuxième PC.

pcs = pca.components\_

for i, (x, y) in enumerate(zip(pcs[0, :], pcs[1, :])):

# Afficher un segment de l'origine au point (x, y)

plt.plot([0, x], [0, y], color='k')

# Afficher le nom (data.columns[i]) de la performance

plt.text(x, y, data.columns[i], fontsize='14')

# Afficher une ligne horizontale y=0

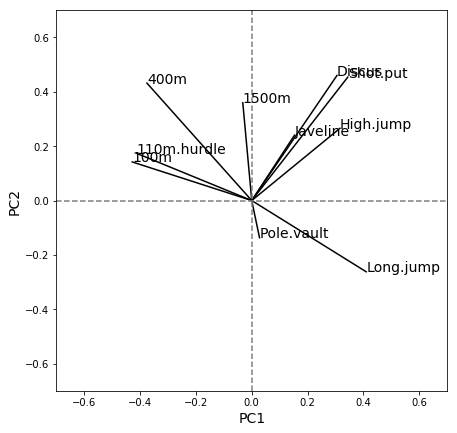
plt.plot([-0.7, 0.7], [0, 0], color='grey', ls='--')

# Afficher une ligne verticale x=0

plt.plot([0, 0], [-0.7, 0.7], color='grey', ls='--')

plt.xlim([-0.7, 0.7])

plt.ylim([-0.7, 0.7])



Contribution de chaque variable à chacune des deux composantes.

Les variables qui ont une contribution négative à la première composante principale correspondent aux disciplines pour lesquelles une bonne performance est représentée par un petit nombre (temps de course), et inversement pour les variables ayant une contribution positive. Cette composante permet de séparer les athlètes qui ont de très bonnes performances de ceux qui sont (relativement !) plus mauvais dans toutes les disciplines, comme nous l'avions remarqué.

La deuxième composante principale permet de séparer les athlètes plus forts (bonnes performances aux lancers) et moins endurants (basse performance aux 400m et 1500m) des autres.

Ce graphique nous permet aussi d'observer que certaines variables sont très corrélées, comme Discus (performance au lancer de disque) et Shot put (performance au lancer de poids).

### Résumé

L'ACP nous a permis de :

* représenter les données en deux dimensions ;
* établir des profils des athlètes ;
* mettre à jour des corrélations entre des variables.

## Cherchez les variables latentes qui expliquent vos données

Dans la plupart des cas, les variables que nous utilisons pour représenter nos données correspondent à un nombre assez grand de quantités que l'on peut facilement mesurer sur ces données. Par exemple, on représente une image par les valeurs RGB de chacun de ses pixels ; ou les goûts d'un client par l'ensemble des produits qu'il a acheté.

Cependant, le phénomène que l'on cherche à expliquer pourrait souvent être prédit à partir d'un plus petit nombre de variables, mais qui sont-elles plus difficiles à mesurer. Par exemple, plutôt que l'ensemble de ses transactions, une seule variable qui décrit si un client est végétarien ou non serait certainement très informative pour décider de s'il faut lui proposer des bons de réduction pour le rayon charcuterie.

La description « accessible » que nous avons de notre client, à savoir la liste de ses achats, est une fonction de cette variable (végétarien ou non) qui nous est cachée. On parle aussi souvent de **variable latente**. On peut considérer que le but de la réduction de dimension est de retrouver les variables cachées qui décrivent au mieux nos données, à partir des **variables observées** qui nous sont accessibles.

### L’ACP est une factorisation de la matrice des données

Soit  représentant n observations en p dimensions. Fixons-nous un nombre K de composantes principales calculées par une ACP. Ces K composantes peuvent être représentées par une matrice . Ainsi, la représentation réduite des n observations dans le nouvel espace de dimension K est donnée en projetant X sur les colonnes de W, autrement dit par .

La matrice  peut être interprétée comme la représentation latente de nos données, celle qu'on a cherché à découvrir grâce à l'ACP.

Comme les colonnes de W, étant les vecteurs propres de , sont des vecteurs orthonormés, on peut écrire . On a donc \_factorisé\_ la matrice de données X en deux termes : W qui définit de nouvelles dimensions, et H, que l'on appelle parfois les **facteurs** de X (« factors » en anglais), qui est la représentation latente des données.

### Modélisation des relations de covariance entre les variables : l’analyse factorielle

Considérons maintenant que nos données, c'est à dire les observations  qui forment les colonnes de X, sont générées par un modèle



où  est une représentation latente de **xi** et **ϵ**  du bruit gaussien : .

Supposons les données centrées en 0 : alors la moyenne de x, et donc de **Wh**, est 0, et sa covariance est celle de **Wh**, à savoir , plus celle de **ϵ**.



Si on considère que la covariance de **ϵ** est isotropique, autrement dit que , où **I** est la matrice identité. C'est cette formulation que l'on appelle **ACP probabiliste**, ou probabilistic PCA en anglais.

Plus généralement, supposons maintenant que la matrice de covariance du bruit **Ψ** est une matrice diagonale quelconque : **Ψ=diag(ψ1,ψ2,…,ψp)**. Selon ce modèle, les variables observées **(xj, j=1,…,pj=1,…,p)** sont conditionnellement indépendantes étant données les valeurs des variables latentes **(hk, k=1,…,K=1,…,K)**. Ainsi, les variables latentes expliquent les corrélations entre les variables observées, tandis que chaque **ψj** décrit la variance spécifique à la variable **xj** correspondante.

On peut alors utiliser les données pour estimer les valeurs de **Ψ** et W par maximum de vraisemblance. C'est ce que l'on appelle **l'analyse factorielle**.

Maximum de quoi ? L'estimation de paramètres par maximum de vraisemblance consiste à trouver les valeurs des paramètres qui maximisent... la vraisemblance, c'est à dire la probabilité d'obtenir les données que l'on a observées, étant donnés un choix de loi de probabilité (ici, la loi normale) et ses paramètres (ici, **Ψ** et W). [**La page Wikipedia**](https://www.wikiwand.com/fr/Maximum_de_vraisemblance) vous donnera plus de détails.

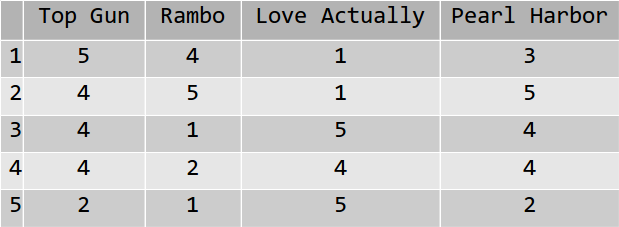
Pour plus de détails mathématiques, on pourra se référer à l'article **[Probabilistic Principal Component Analysis, Journal of the Royal Statistical Society, Series B (61), 1999](https://www.microsoft.com/en-us/research/publication/probabilistic-principal-component-analysis" \t "_blank)**

Dans l'analyse factorielle, nous ne faisons plus l'hypothèse que les nouvelles dimensions sont orthogonales. On peut ainsi en particulier se retrouver avec des dimensions dégénérées (des vecteurs colonnes de W dont toutes les entrées soient 0) et moins de dimensions qu'avec une ACP.

L'analyse factorielle est implémentée dans la classe [*Factor Analysis*](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FactorAnalysis.html) du modulescikit-learn.decomposition.

### Cas des matrices non-négatives : la factorisation non-négative de matrices

Prenons maintenant le cas où X représente les scores donnés par des spectateurs à des films.

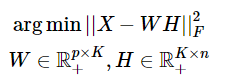


Un exemple de scores données par 5 spectateurs à 4 films, sur une échelle de 1 à 5.

Dans ce cas, une factorisation de  (p films, n spectateurs) sous la forme **WH** peut s'interpréter de la façon suivante :  représente les goûts des spectateurs selon K variables latentes (par exemple le genre du film, les acteurs, la période...), et  représente les films selon ces mêmes variables. Ainsi on pourrait expliquer le score donné par un spectateur à un film comme une combinaison du score donné « en interne » par le spectateur à différents aspects d'un film, et de la présence dans le film de ces mêmes aspects.

Cependant, cette interprétation est facilitée si, de même que les entrées de X, les entrées de W et de H sont toutes positives.

Dans ce cas, le problème de la réduction de dimension par factorisation de matrice est posé comme



 dénote la \_norme de Frobenius\_ de la matrice A ; il s'agit simplement de la racine carrée de la somme des carrés de ses entrées, . Ainsi,  compare les matrices X et WH entrée par entrée.

On appelle ce problème la **factorisation de matrice non-négative**, ou **NMF** pour « non-negative matrix factorization ». Ce problème peut être résolu par exemple par descente de gradient.

La **descente de gradient** est une méthode de minimisation applicable aux fonctions différentiables. Le gradient indique en effet la direction de plus forte pente de la fonction ; si on suit la direction opposée au gradient, la fonction va donc décroître. L'algorithme de descente de gradient consiste à choisir un point au hasard, y évaluer le gradient de la fonction à minimiser, se déplacer dans la direction opposée à ce gradient, et réitérer jusqu'à ce que le gradient soit (à peu près) nul et la fonction donc localement minimale. Attention : si le problème n'est pas convexe, le minimum atteint par cet algorithme peut ne pas être global (d'où le « localement » dans la phrase précédente).  Encore une fois, **[Wikipedia](https://www.wikiwand.com/fr/Algorithme_du_gradient" \t "_blank)** vous permettra de compléter vos connaissances sur le sujet !

Cette méthode est très intéressante pour les systèmes de recommandation, car elle s'applique aussi dans les cas où les données contiennent des valeurs manquantes. Par exemple, le deuxième spectateur n'a pas vu Top Gun. Quel score lui donnerait-on ? Lui recommanderait-on de regarder ce film ?

On remplace alors  par la somme sur les entrées connues de X des termes . On obtient ainsi une décomposition **WH** qui approxime X, et on peut utiliser les entrées correspondantes dans **WH** pour \_prédire\_ les valeurs manquantes de X.

La NMF est implémentée dans la classe [NMF](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html) du module `scikit-learn.decomposition`.

### Résumé

* L'ACP peut être vue comme une factorisation de la matrice de données X=WH, où W continent les nouvelles variables et H la représentation latente des données selon ces variables.
* Toute une famille de méthodes réduise la dimension des données par une décomposition approximative, X≈WH.
* L'analyse factorielle et l'ACP probabiliste sont issues de modèles génératifs utilisant la factorisation de la variable aléatoire x en Wh. Dans le cas de l'ACP probabiliste on modélise la covariance du bruit comme étant un multiple de l'identité. Dans le cas de l'analyse factorielle on estime la covariance du bruit indépendante dans chaque direction.
* L'**ACP** cherche à maximiser la variance de X selon des directions orthogonales.
* L'**analyse factorielle** cherche à modéliser la structure de la covariance des variables observées et ne définit pas nécessairement des axes orthogonaux.
* La **NMF (factorisation non-négative de matrice)** s'applique à des matrices X dont les entrées sont toutes positives et force les entrées de W et de H à être positives aussi. Elle permet aussi de prédire les valeurs manquantes de X.

# Partie 1

Bravo ! Vous avez réussi cet exercice !

### Compétences évaluées

* Choisir un algorithme de réduction de dimension non supervisé linéaire en fonction de vos besoins et des caractéristiques des données
* Utiliser les principaux algorithmes de réduction de dimension non supervisé linéaire classique
* Comprendre à quelles questions les principaux algorithmes de réduction de dimension non supervisé linéaire permettent de répondre

### Question 1

**Un Data Scientist fait une ACP sur ses données X, et trouve que les 3 premières composantes expliquent 80% de la variance de ses données. Comment ce nombre de 80% est-il calculé ?**

*Attention, plusieurs réponses sont possibles.*

* + 

C’est la somme des trois premiers termes de la diagonale de X, divisée par la trace de X.

* + 

C’est la somme des trois premières valeurs propres de , divisée par la trace de  .

* + 

C’est la somme des trois premières valeurs propres de , divisée par la somme de toutes ses valeurs propres.

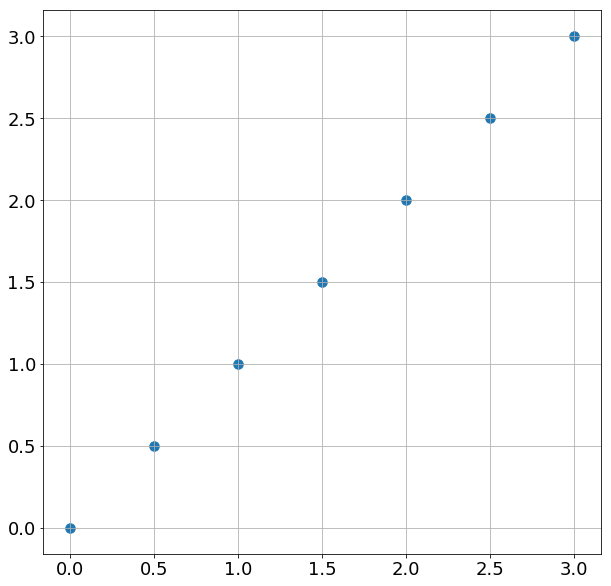
* + 

C’est la somme des trois premiers termes de la diagonale de , divisée par la trace de .

*La variance totale du jeu de données est la somme des variances attribuables à chaque variable. C’est donc la somme des termes diagonaux de la matrice de covariance**, qui est égale à la somme de ses valeurs propres. La j-ème composante principale est le j-ème vecteur propre de*Σ*. La variance due à cette j-ème composante est égale à la j-ème valeur propre*λj*. Ce nombre de 80% est donc égal à* λ1+λ2+λ3λ1+λ2+⋯+λp*.*

### Question 2

**Étant donnés les 7 points en deux dimensions de la figure ci-dessous, quelle est la première composante principale ?**



7 points en 2 dimensions

* + 
  + 
  + 

(1, 1)

* + 

(0, 1)

* + 

(0, √2 )

*La première composante principale est la direction de plus grande variance des données : c’est donc la direction de la diagonale qui passe par (0, 0) et (1, 1). De plus, la norme de cette composante doit être 1 : on obtient donc (√2, √2), ou (-√2, -√2).*

*Pour vous en convaincre, n’hésitez pas à utiliser le code Python suivant :*

*import numpy as np*

*X = np.array([[0, 0], [1, 1], [2, 2], [3, 3], [0.5, 0.5], [1.5, 1.5], [2.5, 2.5]])*

*from sklearn import decomposition*

*pca = decomposition.PCA(n\_components=1)*

*pca.fit(X)*

*print pca.components\_*

### Question 3

**Sur les mêmes données que pour la Question 2, la première composante trouvée par une analyse factorielle sera proportionnelle à celle trouvée par l'ACP. Quelle sera la deuxième composante ?**

* + 

(√(2/2),−√(2/2)

* + 

(1, -1)

* + 

(0, 0)

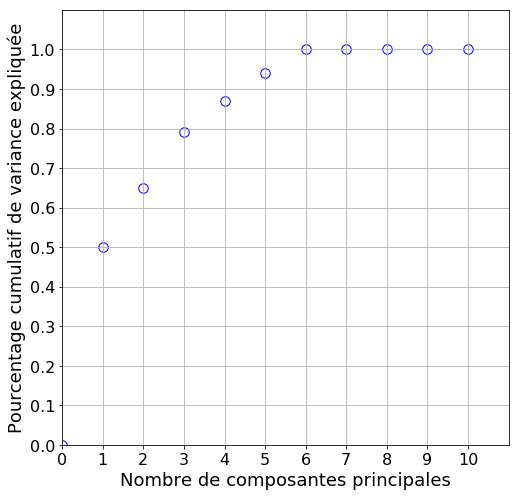
* + 

(√(2/2),√(2/2)

*Alors que l'ACP va forcer la deuxième composante à être orthogonale à la première, l'analyse factorielle ne s'impose pas cette contrainte. Comme les données sont parfaitement expliquées par la première composante, il n'y a pas de deuxième composante et elle est donc donnée par (0, 0).*

### Question 4

**Une Data Scientist calcule une ACP sur ses données et obtient  la figure ci-dessous. Combien de dimensions suffisent à représenter fidèlement les données ?**



Pourcentage de variance expliquée (cumulatif) en fonction du nombre de composantes principales.

* + 

1

* + 

3

* + 

6

* + 

10

*6 composantes suffisent à expliquer 100% de la variance et permettent donc de représenter fidèlement nos données.*

### Question 5

**Parmi les affirmations ci-dessous concernant la réduction de dimension non-supervisée, lesquelles sont vraies ?**

*Attention, plusieurs réponses sont possibles.*

* + 

Elle permet de faciliter la visualisation des données.

* + 

Elle fonctionne en minimisant l'erreur de prédiction d'un modèle.

* + 

Elle permet d’accélérer les temps de calcul.

* + 

Elle permet de découvrir des sous-groupes d'observations similaires dans les données.

*Réduire le nombre de dimensions permet 1) de mieux visualiser les données, 2) de construire des algorithmes (supervisés ou non) moins coûteux en temps de calcul, espace mémoire ou acquisition des données, 3) de construire des algorithmes (supervisés ou non) plus performant en éliminant les variables non-informatives, en facilitant la construction de modèle moins complexes et en réduisant les problèmes dus au fléau de la dimensionnalité (mais pas en minimisant directement une erreur de prédiction !).*

*La réduction non-supervisée du nombre de dimensions ne s'appuie pas sur des étiquettes. Elle ne s'intéresse pas non plus à la découverte de sous-groupes dans les données ; c'est le rôle du clustering.*

### Question 6

**On vous donne un ensemble d'articles de presse, représentés par une matrice X dont chaque ligne correspond à un mot-clé ("élections", "match", "finale", "bourse", "cinéma", "Dow Jones"...) et chaque colonne à un article. Quelle technique vous permettra de représenter les articles non pas dans l'espace des mots-clés mais dans celui des thèmes des documents (politique, économie, culture, sport...), en identifiant les mots-clés associés à chaque thème, et ce sans définir ces thèmes a priori pour les utiliser comme étiquettes ?**

* + 

ACP

* + 

NMF

* + 

Analyse factorielle

* + 

Classification

*La PCA, l'analyse factorielle et la NMF vont décomposer X en un produit WH. Dans le cas de la NMF, W et H seront positives et interprétables de la façon suivante : W (de dimension nombre de mots clés x nombre de thèmes) représente l'importance de chaque mot-clé pour chaque thème et H (de dimension nombre de thèmes x nombre d'articles) est la représentation que l'on recherchait des articles dans l'espace des thèmes.*

### Question 7

**Le modèle des Big Five en psychologie stipule qu'il existe 5 traits latents de personnalité découverts empiriquement (Ouverture, Conscienciosité, Extraversion, Agréabilité, Neuroticisme). Le travail d'analyse exploratoire le plus plausible effectué pour découvrir ces caractéristiques serait en utilisant :**

*Attention, plusieurs réponses sont possibles.*

* + 

Une analyse par composante principale

* + 

Une analyse factorielle

* + 

Une factorisation de matrice non-négative

*L'analyse factorielle postule justement que les données observées sont le fruit d'une combinaison linéaire de facteurs latents qu'il faut découvrir. Contrairement à l'analyse par composantes principales ou la NMF qui s'intéresse elle aux dimensions les plus explicative de la variance, mais capture par la même occasion l'erreur due à la variance globale, et au mieux approxime les facteurs latents.*

### Question 8

**Quelles sont les composantes principales des observations suivantes ?  
{(1,1) (3,1) (1,2) (2,0)}**

* + 

(-0.73517866, -0.6778734 ), (0.6778734, -0.73517866)

* + 

(-0.82192562, 0.56959484), (0.56959484, 0.82192562)

* + 

(0,1), (2,3)

*from sklearn.decomposition import PCA*

*data = array([ [1,3,1,2], [1,1,2,0] ]).T*

*pca = PCA(n\_components=2)*

*pca.fit(data)*

*Vous pouvez vérifier avec le code ci-dessus*

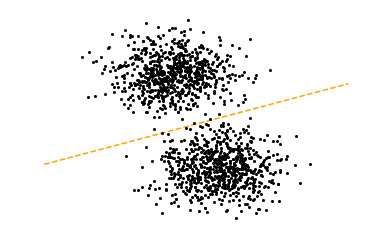
## Découvrez la réduction dimensionnelle non-linéaire

Bienvenue dans cette seconde partie sur l’apprentissage non-supervisé ! Une notion centrale sera présente dans cette section : la non-linéarité. J’avais déjà brossé le sujet dans le [cours d’initiation au machine learning](https://openclassrooms.com/courses/initiez-vous-au-machine-learning), mais l’idée ici est de renforcer la compréhension de cette notion afin de pouvoir aborder sereinement les différentes méthodes que nous explorerons par la suite, et que vous utiliserez au quotidien dans votre travail de data scientist.

### 

### Notion de non-linéarité

Imaginons que l’on se retrouve face à un problème de classification de la forme suivante :

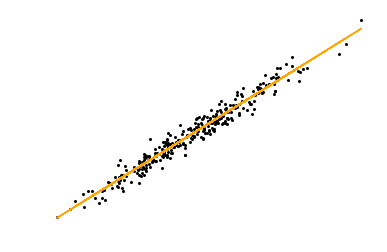
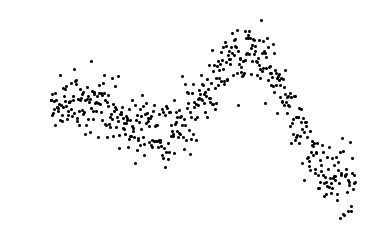


Pas de problème ici pour séparer les deux nuages de données par un hyperplan (une droite en l’occurrence) et donc classer facilement nos points. Prenons un autre exemple de classification :



Mmh, plus compliqué de séparer par une droite n’est-ce pas ? 😛

On peut observer de la même manière des régressions linéaires (on peut modéliser par une droite) vs non-linéaire



Vous vous en doutez, les problèmes rencontrés par les data scientists dans la vie réelle sont en général non linéaires. Comment traiter ce type de comportement ?

Une autre manière de voir les choses est de considérer que les méthodes linéaires ne capturent pas les propriétés géométriques du phénomène. Elles sont aplaties en quelque sorte.

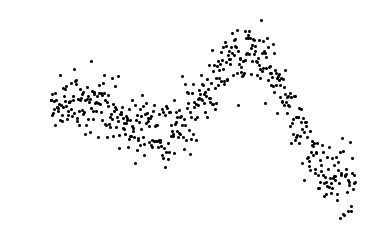
### Résoudre des problèmes non-linéaires

Il existe globalement deux manières d’aborder le sujet : de manière explicite ou de manière implicite.

#### La manière explicite

Traiter un problème non-linéaire de manière explicite signifie que l’on va créer des nouvelles features “à la main”, pour ensuite les utiliser dans des algorithmes linéaires. Ces features prennent en compte le comportement non-linéaire des données et “l’annulent” en quelques sorte, en augmentant la dimension du problème.

Par exemple, imaginez que vous voyez un jeu de données de points x=(x1, x2) qui ressemble à ça :



On pourrait ici se dire que le comportement a l’air [polynomial](https://fr.wikipedia.org/wiki/%C3%89quation_polynomiale), c'est à dire que les points d'observation se situent autour d'une courbe avec une équation de type , et l'on peut donc créer des features supplémentaires en conséquences, c’est à dire augmenter chaque point avec de nouvelles caractéristiques :



x\_augmented = np.array([x, x\*\*2, x\*\*3, x\*\*4, x\*\*5]).T

On peut ensuite effectuer une régression linéaire classique sur ce nouveau jeu de données de points z, qu'on affiche ensuite :

regr = linear\_model.LinearRegression()

regr.fit(x\_augmented, y)

plt.plot(x\_augmented[:,0], y, 'o', color='black', markersize=2)

plt.plot(x\_augmented[:,0], regr.predict(x\_augmented), '-', color='orange')

plt.axis('off')

plt.show()



En orange est représenté le résultat de notre régression linéaire. Plutôt pas mal !

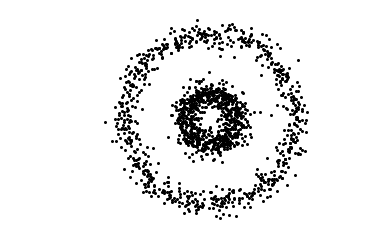
Les features polynomiales augmentent en taille de manière considérable. Si notre variable de départ x prenait une valeur entre 1 et 100, alors  entre 1 et 10000,  entre 1 et  etc. Attention à bien remettre à l’échelle entre elles vos différentes variables (par normalisation, etc.)

Les features polynomiales sont tentantes parfois mais il est facile de voir que si on n'a pas une vision globale du phénomène, on overfit assez rapidement notre jeu de données d’entraînement. Veillez bien à tester vos modèles en continu !

Là, j’ai présenté un modèle unidimensionnel donc tout va bien. Mais si je suis dans un modèle multidimensionnel, disons de 10 dimensions par exemple (donc avec 10 features), et que je souhaite utiliser les features quadratiques (c'est-à-dire x², x³,x⁴,... comme précédemment), j’aurais au minimum 10 x P features supplémentaires (où P est le nombre de degré polynomiaux dont j’ai besoin). Imaginez aussi que je souhaite connaître les interactions entre les différentes variables (x1, x2, x3, x4 , etc) on explose facilement notre nombre de features ! Heureusement pour nous, il existe des méthodes pour ne pas avoir à calculer explicitement toutes ces features, on voit ça dans la partie suivante 😉

 J’utilise ici des features polynomiales, mais c’est juste un exemple. L’idée à garder en tête c’est qu’avec des transformations de vos variables de départ (feature engineering) vous allez pouvoir “simplifier” votre travail de modélisation par la suite, jusqu’à parfois même la transformer en simple modélisation linéaire 🙂 - En général, il existe des méthodes automatiques (que l’on verra par la suite) pour trouver ces features importantes. C’est justement un des objectifs de la réduction dimensionnelle, que de trouver des features représentatives

 Prenons un autre exemple, de classification cette fois, avec un jeu de données (x1,x2) :

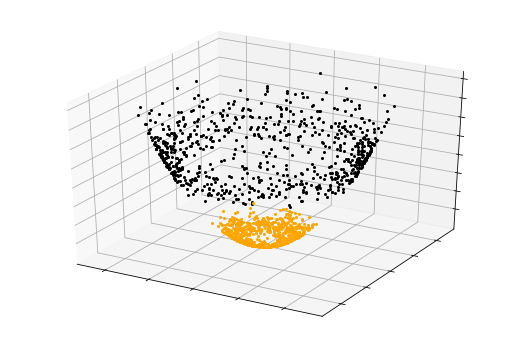


Si on imagine que l’on veut séparer les deux cercles ci-dessus, on peut naturellement créer la feature car l'[équation d'un cercle](https://fr.wikipedia.org/wiki/Cercle) (centré en (0,0) est ).

X\_augmented = np.array([X[:,0], X[:,1], X[:,0]\*\*2 + X[:,1]\*\*2).T

Dans ce nouvel espace de dimens

ion 3, on peut représenter notre jeu de données qui devient alors linéairement séparable (j’ai volontairement ajouté la couleur orange pour distinguer les deux cercles) :



On peut maintenant facilement imaginer un hyperplan qui sépare les deux classes.

#### La manière implicite

Bien, maintenant que l’on a vu les méthodes explicites, qui sont plus souvent utilisées dans un cadre supervisé d’ailleurs, on peut passer aux méthodes implicites. Comme évoqué plus haut, un ensemble de méthodes existent pour traiter les données au comportement non linéaire directement.

C’est notamment le cas des méthodes de noyaux (ou kernel methods), qui permettent de transformer certains algorithmes linéaires (telle que l’analyse par composante principale) en algorithmes non linéaires, sans avoir besoin d’effectuer des calculs dans cet espace de dimension supérieure. On va détailler un peu plus comment ça fonctionne dans le chapitre suivant.

Il existe d’autres familles de méthodes non supervisées (appelées graphical models), qui travaillent sur une structuration plus hiérarchique en général, a priori, des données à l’aide de variables latentes. On en aura un aperçu avec l’algorithme LDA ou les célèbres modèles de Markov cachés (HMM) dans un prochain cours sur le traitement du texte.

### 

### Dans le cadre de la réduction dimensionnelle

Dans la partie précédente, vous avez pu utiliser un algorithme très classique de réduction dimensionnelle appelé Analyse par Composantes Principales. Son objectif est de déterminer les directions dans lesquelles notre jeu de données possède le plus de variance et de “projeter” linéairement les données dans ces directions là (en général pas plus de 3 dimensions). Cette projection nous permet de mieux visualiser le comportement du phénomène observé.

Cette méthode est très efficace, je vous conseille d’en faire fréquemment lors de votre analyse exploratoire. Deux choses à noter cependant :

* C’est le comportement global du phénomène qu’on observe car on travaille sur la matrice de variance-covariance de tout le jeu de données directement.
* C’est une projection (plongement) linéaire (on transforme les données seulement par des opérations de la forme AX + b).

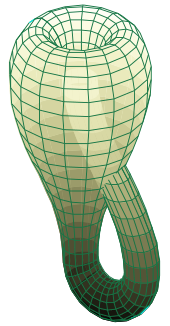
#### Plongement non-linéaire

On peut généraliser cette notion de factorisation de matrice (dont l’ACP fait partie) pour gérer des données qui réside sur une sous-variété non-linéaire (et non pas un sous-espace).

Hmm… Tu as déjà évoqué la notion de variété, mais ça me parait un peu ésotérique comme notion…

Bien que la notion de variété géométrique (ou manifold en anglais) en mathématique soit assez rigoureuse, elle est utilisée en machine learning un peu plus librement pour désigner des espaces plus généraux que les espaces euclidiens, dans lequel se trouve notre phénomène.

Une variété, pour donner un peu plus de contexte, est définie comme la généralisation de la notion de courbe. Son comportement peut notamment être explicité au voisinage des points qui la composent.



Bouteille de Klein

Cette bouteille de Klein est un objet en 3 dimensions, mais sa surface est-elle, en 2 dimensions. Supposons que nous ayions des données à 3 dimensions. On peut les représenter par des points dans cet espace à 3 dimensions. Supposons en plus que tous ces points soient situés sur la surface de la bouteille de Klein. On peut alors dérouler (ou aplatir) la surface de cette bouteille. Une fois la surface applatie, on se retrouvera avec les points qui pourront être décrits en 2 dimensions au lieu de 3 originellement. L'action de dérouler la bouteille est une tranformation. Plus tard, on apellera cette transformation fonction **Φ**. Très souvent, l'action de dérouler sera très complexe (imaginez-vous dérouler cette bouteille qui a une forme tellement bizarre !), comme nous le verrons au chapitre suivant.

Cette notion sera plus facile à appréhender pour vous avec la pratique.

Ok donc pour reprendre, pourquoi tu parles de variété déjà ?

Comme je le disais plus haut, l’idée est de généraliser des algorithmes qui travaillent dans des espaces euclidiens pour passer sur des variétés, donc des espaces non-linéaires.

Dans le contexte de l’apprentissage non-supervisé, l’objectif est de trouver une variété de dimension (bien) inférieure à la dimension originelle dans laquelle se trouvent les données. On l'appelle Espace de Redéfinition. On peut ainsi représenter nos données plus facilement ou les simplifier en terme de features représentatifs, prédire en parcourant ce manifold… de manière beaucoup plus simple.

De manière un peu plus formelle, on a nos points de départs X=[x1,x2,x3,...,xN] où chaque point est de dimension P et on essaie de trouver une application : X → Y, où Y=[y1,y2,...,yN]sont des points qui approximent X dans un espace de dimension bien inférieur. Par exemple, de dimension 2 pour une visualisation.

#### Global vs Local

Dans cet ensemble de méthodes de réduction dimensionnelles non-linéaires, on peut encore séparer les algorithmes en deux catégories : ceux qui travaillent sur la structure globale vs ceux qui se concentrent sur la structure locale.

Hmm, je pense qu’intuitivement je comprends ce que ça veut dire mais concrètement ?

Concrètement, les méthodes s’intéressant à la structure globale s’intéressent aux paires de distances entre les points et les considère comme toutes équivalentes alors que les méthodes s’intéressant à la structure locale vont pondérer les paires de distances (relativement) proches avec plus d’importance.

Il existe encore une autre distinction dont je n’ai pas parlé c’est paramétrique vs non-paramétrique. En réalité dans ce cours on utilisera que des méthodes non-paramétriques.

### Résumé

* Les données non-linéaires sont abordables de manière explicites en effectuant directement un feature engineering sur les observations de départ, ou bien de manière implicite en utilisant des méthodes spécifiques à la non-linéarité. On retrouvera parmi ces méthodes les méthodes à noyaux et les méthodes à plongement dans des variétés.
* Les données à grandes dimensions reposent souvent sur une variété non-linéaire de dimension bien inférieure (ou bien sont proches de celle-ci).
* Les méthodes de réduction dimensionnelles non linéaires que nous allons étudier permettent ainsi de retrouver une approximation de ces variétés et de bien mieux représenter ces données dessus.
* On s’intéresse aux informations liées à la structure du phénomène représenté, mais c’est parfois suffisant de n'observer que les paires de distances entre les points dans l’espace original. C’est le cas d’une partie des méthodes que nous allons étudier.

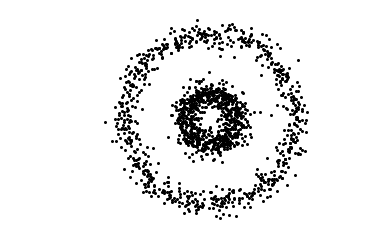
## Utilisez une ACP avec un noyau

Dans ce chapitre, nous allons étudier un modèle non-supervisé appelé kPCA. Hmm, beaucoup trop de lettres en commun avec PCA pour que ce soit une coïncidence, n’est-ce pas 😛.

La lettre supplémentaire ‘k’ désigne ici un objet mathématique appelé kernel (noyau en français). Nous sommes donc face à un algorithme qui se nomme kernel principal component analysis. L’ajout de ce kernel va nous permettre d’appliquer notre modèle à des données dites non-linéaire. C’est ce fameux passage d’un espace euclidien vers une sous-variété dont nous avons parlé dans le chapitre précédent.

 Ce modèle est moins utilisé en réalité dans sa forme première. Il permet cependant de bien illustrer la notion de passage d’un algorithme linéaire à un algorithme non-linéaire, et donc ce qu’apporte la non-linéarité en pratique.

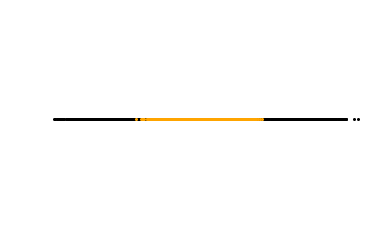
Si on reprend l’exemple du chapitre précédent des deux cercles imbriqués :



Ici, appliquer simplement une PCA directement ne pourra pas donner un résultat pratique, puisque la transformation effectuée est linéaire. Pour rappel, l’objectif d’une analyse par composantes principales est de réduire la dimension du jeu de donnée en conservant un maximum d’information, c’est à dire de variance. En d’autres termes, nous voulons réduire la dimension des données, tout en gardant autant que possible les informations qui permettent de distinguer les observations.

En trouvant les directions qui possèdent le plus de variance des observations, on peut représenter avec quelques axes (2 ou 3) la majorité de la variance globale du jeu de données. Encore une fois, c’est une transformation linéaire que l'on effectue sur les observations.

Ci-dessous, j’ai représenté la première composante principale d’une PCA linéaire appliquée au jeu de données. Pour mieux comprendre, j’ai mis en orange la seconde classe (celle du petit cercle) de manière artificielle.



Comme on peut le voir, une transformation linéaire sur les composantes principales ne peut clairement pas distinguer les deux classes, car ce qui les sépare n’est pas linéaire. Elle écrase donc la propriété géométrique de cercle du phénomène observé. 😐

On a besoin de changer d’espace et de passer dans un espace non-linéaire **ω**, dans lequel on pourrait appliquer notre PCA tranquillement. C’est ce qu’on a fait dans la partie précédente justement pour ce problème en ajoutant la feature . Pour l’exercice, appelons la transformation d’un point dans cet espace hypothétique



Le problème, c’est que, souvent, cet espace est trop complexe à définir ou bien même de dimension infinie, et qu’on ne peut donc pas effectuer nos calculs en utilisant **ϕ** (on peut dire qu’il est “intractable” en anglais, ce qui veut dire qu’il y a très rapidement une explosion du temps de calcul). Du coup, comment faire ? Et bien nous faisons appel à ce fameux noyau (ou kernel), à l’origine d’une technique très utilisée qui s’appelle le “kernel trick” (astuce du noyau en français).

Cette technique sera détaillée dans un autre cours sur les Machines à vecteurs de support, dont elle sera un élément principal.

#### Intuition du kernel trick

Notre fonction **ϕ** est trop complexe. Cependant, si on détaille la construction de l’algorithme (ce qu’on ne fera pas ici), on se rend compte qu’on peut formuler notre méthode uniquement à l’aide de produits scalaires de points **xi.xj**. Ce produit est valable dans l’espace euclidien classique. Si on veut effectuer les mêmes calculs sur l’espace de plus petite dimension, appelé espace de redescription, ce serait donc sur les produits ϕ(xi)ϕ(xj).

Aussi, l’idée du kernel trick est de laisser tomber la fonction **ϕ** et l’espace de redescription pour s’intéresser au produit **ϕ(x)ϕ(y)** à la place. Dans ce cas on va représenter ce produit par une fonction **noyau (ou kernel)** **K(xi,xj)**.

En faisant quelques calculs sur la matrice de variance-covariance, on peut exprimer la projection d’une observation **xi** dans cet espace en utilisant seulement la matrice **K=K[xi][xj]**.

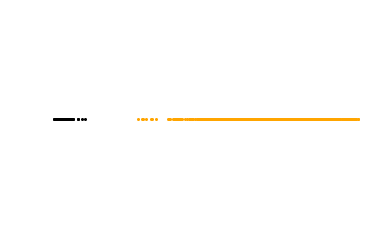
On aura donc le même genre d’opérations que PCA pour réduire les dimensions mais en utilisant cette matrice K(x,x) ainsi que les coefficients **α** calculés à partir des valeurs et vecteurs propres de la matrice de variance-covariance C.

[Pour en savoir plus sur la formulation de l’algorithme, rendez-vous sur l’article originel](http://pca.narod.ru/scholkopf_kernel.pdf).

Qu’est-ce que ça donne sur l’exemple plus haut? Et bien si on effectue une kernel PCA avec un kernel gaussien (défini par , et un **γ** =10, on obtient la composante principale suivante :

kpca = KernelPCA(n\_components=1, kernel='rbf', gamma=10)

X\_spca = kpca.fit\_transform(X)



On visualise ci-dessus la composante qui possède le plus de variance dans l’espace dans lequel se trouve le kernel. On peut voir qu’ici la géométrie du modèle a bien été capturée, et qu’on peut bien distinguer les deux classes aisément. Top !

Alors, plusieurs questions se posent :

Comment est défini ce kernel, si on ne connait même pas ϕ ni ω ?

En fait il existe plusieurs familles de fonctions kernel pratiques et qui permettent de résoudre la plupart des problèmes. Il faut visualiser ses données et avoir une petite idée de leur comportement afin de pouvoir prendre une décision. L’idée générale, c’est qu’un kernel doit permettre de représenter la similarité entre les deux points sur lesquels on effectue le produit. On peut même ne pas utiliser de données numérique (par exemple des données textes) tant que cette notion de similarité entre les observations est respectée.

Voici une liste des kernel les plus utilisés :

* Linéaire - équivalent à une PCA classique
* Polynomial
* Gaussien (RBF) - le noyau gaussien RBF représente le plus usité, dont les cas ci-dessus sont en réalité des cas particuliers
* Laplacien

Pourquoi ça fonctionne ? Ca parait un peu facile de ne pas utiliser l’espace final, non ?

Il existe aussi une théorie sous-jacente qui tourne autour de ce qu’on appelle espaces de Hilbert (notamment le théorème de Mercer), qui permet de justifier / vérifier que sous certaines conditions, on peut en effet utiliser un Kernel pour y effectuer un plongement, sans tout expliciter. [Je vous invite à suivre ce cours si vous êtes intéressés par le sujet](https://alex.smola.org/teaching/cmu2013-10-701/kernels.html).

Quelles sont les limitations?

En réalité, comme je l’ai dit au début, cet algorithme n’est pas le plus utilisé. En effet, un des problèmes principaux des méthodes à noyaux sont qu’elles nécessitent le stockage en mémoire de la matrice K(x,y) sur les données afin de pouvoir être utilisées. Comme la taille nécessaire pour le stockage augmente au carré de la taille des données, on peut vite se retrouver avec une utilisation de la mémoire trop importante. Il existe surtout maintenant des algorithmes non supervisés plus efficaces, que l’on va voir dans les prochains chapitres. L’idée ici était d’illustrer ce plongement avec noyau qui est une notion très importante puisqu’on vit/visualise naturellement dans un espace euclidien.

En réalité, kPCA est quand même une sorte de méthode générale englobant les autres algorithmes que nous allons étudier dans la suite de cette partie, qui sont en fait des cas particuliers, traités de manière algorithmiquement intelligente, avec des approximations bien placées. kPCA est juste rarement utilisé dans sa forme première.

### Le problème : la complexité algorithmique

La bonne nouvelle, c’est que la complexité algorithmique de l’ACP avec noyau ne grandit pas avec la dimension de l’espace de plongement implicite sur lequel on travaille. C’est l’avantage par rapport aux représentation explicites dont on parlait dans le chapitre précédent.

L’ACP possède une complexité algorithmique de où D représente le nombre de features/dimensions originales. L’ACP avec noyau en revanche puisqu’on doit travailler avec K(x,y) possède une complexité de  où n représente le nombre de points d’observations utilisé. De même en complexité espace on est sur  puisqu’on conserve toute la matrice K contrairement à l’ACP classique qui est en 

### Résumé

* Le plongement d’un algorithme linéaire dans une variété non-linéaire peut s’effectuer à l’aide du kernel trick
* Les méthodes à noyaux permettent de ne pas avoir à définir la variété dans laquelle on travaille pour appliquer nos algorithmes
* La réduction dimensionnelle non-linéaire par kPCA n’est pas la plus utilisée, mais elle illustre la transition entre les méthodes linéaires et non-linéaires

## Découvrez une variété qui conserve la structure globale

Une première famille de méthodes de réduction dimensionnelles non-linéaire est celle des méthodes dites globales. Comme je l’ai expliqué dans le chapitre précédent, cette famille permet de traiter le jeu de données avec un focus sur les distances entre les paires de points, qu’ils soient distants ou proches.

### MDS (Multidimensional Scaling)

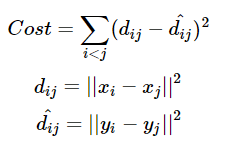
MDS (Multidimensional Scaling) est une méthode de réduction dimensionnelle qui utilise les distances entre les points. C’est encore une méthode qui vous aidera à expliquer les similarités et dissimilarités observées dans votre jeu de données, en les représentant dans un espace géométrique de dimension inférieure à celle de l’espace initial

#### Intuition du fonctionnement

Pour cela, le MDS s’intéresse à la matrice **Dij** de distances entre les points. A la différence de l’analyse en composantes principales qui traite uniquement la matrice de variance-covariance (corrélation) entre les variables, MDS permet d’utiliser n’importe quel type de mesure de similarité comme “**distance**” (dont la distance euclidienne bien sûr 😉).

A l’aide de cette matrice de distances D, une procédure MDS va essayer de réarranger les observations dans un espace de dimension inférieure (2 en général pour la visualisation) pour se rapprocher au mieux des distances dans l’espace d’origine.

Comme d’habitude dans les algorithmes de machine learning, il faut minimiser une fonction objectif (ou fonction de coût) héhé. Dans notre cas présent, l’objectif est de minimiser l’erreur entre **les distances approximées** dans l’espace de dimension inférieure, et **les distances réelles** dans l’espace de départ.



L’objectif est de retrouver les vecteurs **yi** dans l’équation ci-dessus qui formeront les points de notre nouvelle visualisation.

Ces manipulations m’ont l’air assez linéaires, non?

Bien vu effectivement 😛 En effectuant cette MDS de base avec les distances euclidiennes (appelée **MDS métrique**) on est équivalent aux résultats d’une ACP linéaire, sous sa forme duale. Ça veut dire deux choses. La première c’est que l’ACP conserve les propriétés de distances ce qui est une bonne chose. La seconde, c’est que MDS est un algorithme linéaire...

En réalité, la non-linéarité va apparaître en modifiant notre définition de la mesure utilisée pour le calcul des distances, pour la généraliser à des fonction **monotones** (constamment croissantes ou décroissantes), même si ce ne sont pas des mesures au sens formel du terme (qui respectent l’inégalité triangulaire). On est alors dans la famille de méthodes MDS non-métriques.

Scikit par exemple, implémente une régression monotonique, qui conserve **l’ordre** des points, en conservant les inégalités des valeurs dans la matrice D. Par exemple, si  alors le plongement doit conserver l’inégalité .

Pour cela, il suffit de spécifier metric=False dans la fonction de scikit sklearn.manifold.MDS.

### Une extension : Isomap

Imaginez que je veuille trouver la distance sur un globe qui sépare Paris de Los Angeles. Je ne vais pas utiliser la distance euclidienne directement, ce serait une erreur. Je vais plutôt considérer la Terre comme une variété et calculer la distance sur sa surface

Intuitivement, une sous-variété se rapproche d’un espace linéaire quand on s’intéresse au voisinage d’un point. Par exemple pour cette distance, si je m’intéresse à une ville proche de Paris, je pourrais considérer une ligne droite qui les sépare, donc une distance euclidienne.

Maintenant, si j’effectue cette approximation de ville en ville jusqu’à Los Angeles, j’aurais une bonne approximation de la “vraie” distance qui sépare les deux villes.

Les variétés dont on parle ici sont considérées lisses et convexes, sinon ça ne fonctionne plus trop. Et on n’est pas très mathématiquement rigoureux, car on s’intéresse à des méthodes relativement robustes dans tous les cas… (merci les chercheurs).

Ainsi, l’algorithme Isomap a été créé dans l’objectif de favoriser encore plus la structure locale ainsi que la capture de la non-linéarité de la variété sous-jacente au phénomène, qui reste difficile à détecter à l’aide d’un MDS. Isomap a été imaginée avec le type d’approximation que je viens d’effectuer pour estimer la distance Paris-Los Angeles

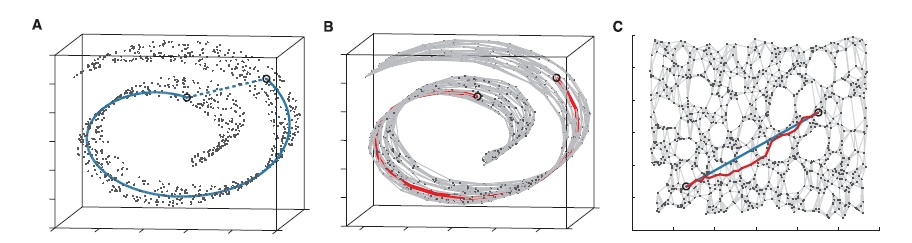
Il fonctionne en trois étapes :

1. Créer un graphe de voisins (Neighbourhood Graph). Plusieurs méthodes sont possibles. On peut simplement dire que les K voisins les plus proches sont connectés. Ou bien que tous les points dont la distance est inférieure à ϵϵ sont connectés.

Attention à choisir une distance assez grande pour être sûr que tous les points ont bien des voisins

2. Calculer la distance **géodésique** entre chaque paire de points. La définition de distance géodésique utilisée ici est le chemin le plus court dans le graphe, pondéré par la distance entre les points. Il y a des algorithmes pour effectuer ce calcul de manière efficace.

3. Appliquer l’algorithme du MDS à cette matrice de distances géodésique.



L’exemple célèbre du swissroll sur lequel est appliqué l’algorithme de l’isomap. La 3ème image montre en rouge la distance géodésique finale

### 

### Résumé

L’algorithme Isomap est une méthode pratique lorsqu’on veut conserver la structure géométrique globale du phénomène, à petite et à grande échelle. C'est un des algorithmes à tester lorsqu'on est face à un problème non-linéaire à explorer.

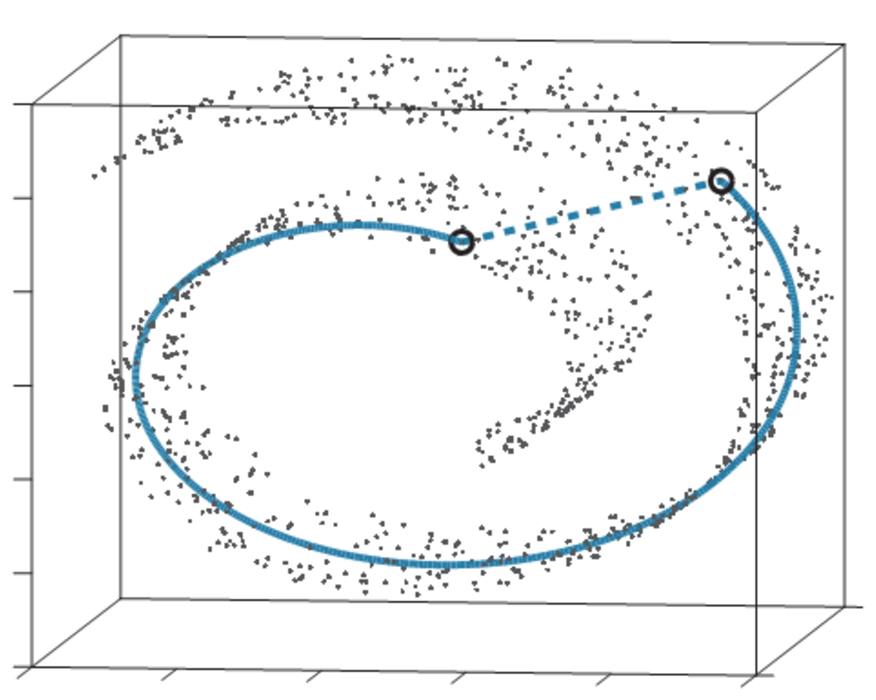
## Découvrez une variété qui favorise la structure locale

Dans ce chapitre nous allons étudier en détail les méthodes qui favorisent la structure locale du phénomène observé. L’objectif n’est donc pas de retrouver la variété globale d’origine mais de souligner certains comportements des données à l’échelle locale.

### Problème avec l'ACP et les méthodes globales

Comme on l’a vu précédemment, l’analyse par composantes principales essaie de déterminer la structure sous-jacente d’un jeu de données. Pour rappel, l’ACP maximise la variance en minimisant l’erreur en distance euclidienne entre les données originales et la projection.

Parce qu’on utilise une erreur quadratique, seulement les grandes distances/variations entre les données sont magnifiées, et “écrasent” rapidement les petites variations.



Les points ici sont proches dans l’espace euclidien (en pointillé la distance euclidienne) mais éloignés si on considère le jeu de données (en trait plein la distance “selon les données”)

L’idée des algorithmes de ce chapitre est de considérer la variété (manifold en anglais) tout entière, et se concentrer sur la structure locale des données, en mettant en exergue cette fois les paires de distances euclidiennes entre les points proches.

On gagne ainsi deux avantages principaux :

* En travaillant seulement à l’échelle locale, on travaille avec beaucoup moins de points sur nos calculs, donc un gain en complexité algorithmique non-négligeable
* On aura une meilleure représentation d’un plus grand nombre de variétés, dont la géométrie générale est trop éloignée d’une représentation euclidienne pour pouvoir être utile. En se libérant de cette contrainte générale, on ouvre ainsi la possibilité de représentation plus fidèle sur une plus large classe de variété.

### LLE (Locally Linear Embedding)

On commence à y être habitué, on veut trouver une représentation de dimension inférieure de nos points **X=[x1,x2,...xN]**. Cette fois ci en revanche, on ne va travailler que sur le voisinage des points.

La technique de LLE (qui est apparu dans le même journal que Isomap d’ailleurs), consiste à effectuer deux optimisations.

La première, c’est dans l’espace d’origine à grande dimension, trouver comment reconstruire chaque point à l’aide de ses K plus proches voisins, en minimisant la fonction suivante :



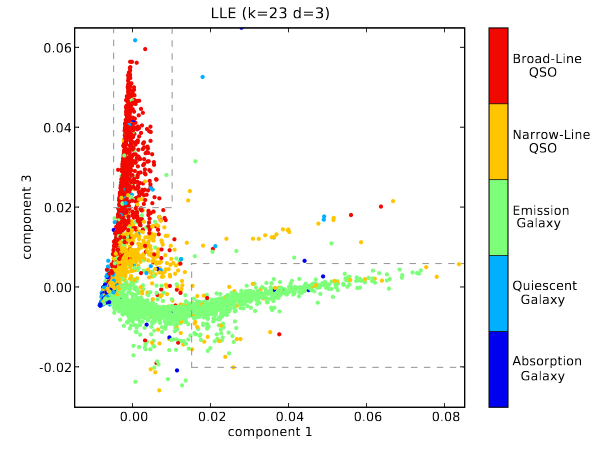
Où V(X) représente la matrice des K voisins de X. On obtient ainsi la matrice W.

A l’aide de cette matrice, on veut trouver les points Y dans la variété de dimension inférieure en minimisant :



Et voilà ! C’est facile hein 😀

C’est une première méthode, assez classique de visualisation de données non-linéaire. En général, la variété se retrouve représenté le long d’axe qui représentent les degrés de libertés principaux du phénomène.

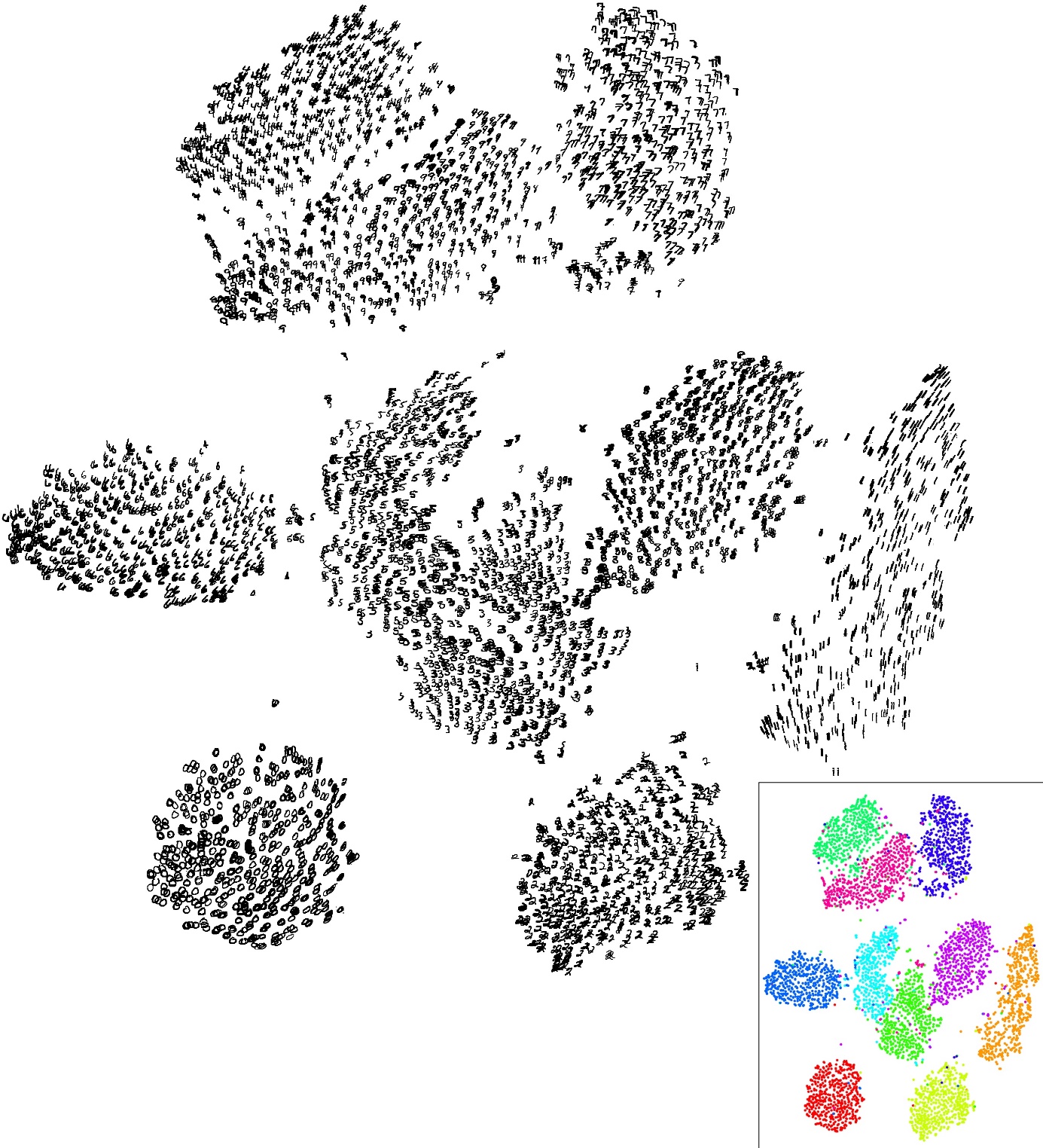


LLE sur des images de spectroscopie

### t-SNE (t-Stochastic Neighbour Embedding)

Dans cette partie, nous allons étudier une des méthodes de visualisation les plus usitées en ce moment, t-SNE, pour “t-distributed Stochastic Neighbor Embedding”. Le t désigne une distribution utilisée dans l’algorithme (pour loi t de Student, on verra plus tard pourquoi).

En bref, cette technique permet de visualiser des données de (très) grandes dimensions, en effectuant un plongement (embedding en anglais) dans une variété de plus petite dimension, généralement 2 ou 3 pour pouvoir repérer des caractéristiques intéressantes du phénomène à modéliser. Elle est devenue populaire pour sa capacité à générer des visualisations très parlantes.



#### Intuition du fonctionnement

Comme je l’ai dit plus haut, le t-SNE met l’accent sur les petites distances. Le concept général de l’algorithme est de considérer chaque point de données séparément, et d’assigner une probabilité conditionnelle (un poids) gaussienne à chacun des autres points en fonction de leur distance par rapport à ce point.

Une fois toutes ces probabilités calculées **Pij**, on peut plonger dans une dimension inférieure ce jeu de données en calculant une nouvelle distribution **Qij** (qui est une t-student distribution cette fois, d’où le “t”), qui minimise la [KL divergence](https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence)entre P et Q.

L’idée est de trouver un espace de dimension inférieure mais qui respecte une distribution proche en KL divergence de la distribution d’origine.

Encore une fois, la probabilité met l’accent sur la structure locale, puisqu’elle met plus de poids sur les petites distances entre les points du jeu de données.

On peut par exemple appliquer t-SNE sur le jeu de données de visages (Olivetti dataset).

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_olivetti\_faces

from sklearn import (manifold, datasets, decomposition, ensemble,discriminant\_analysis, random\_projection)

from matplotlib import offsetbox

olivetti = fetch\_olivetti\_faces()

targets = olivetti.target

data = olivetti.data

images = olivetti.images

# fonction pour afficher une partie des images sur la visualisation 2D

def plot\_embedding(X, title=None):

x\_min, x\_max = np.min(X, 0), np.max(X, 0)

X = (X - x\_min) / (x\_max - x\_min)

plt.figure(figsize=(15, 15))

ax = plt.subplot(111)

if hasattr(offsetbox, 'AnnotationBbox'):

shown\_images = np.array([[1., 1.]])

for i in range(data.shape[0]):

dist = np.sum((X[i] - shown\_images) \*\* 2, 1)

if np.min(dist) < 2e-3:

continue

shown\_images = np.r\_[shown\_images, [X[i]]]

props={ 'boxstyle':'round', 'edgecolor':'white'}

imagebox = offsetbox.AnnotationBbox(offsetbox.OffsetImage(images[i], cmap=plt.cm.gray, zoom=0.5), X[i], bboxprops=props)

ax.add\_artist(imagebox)

if title is not None:

plt.title(title)

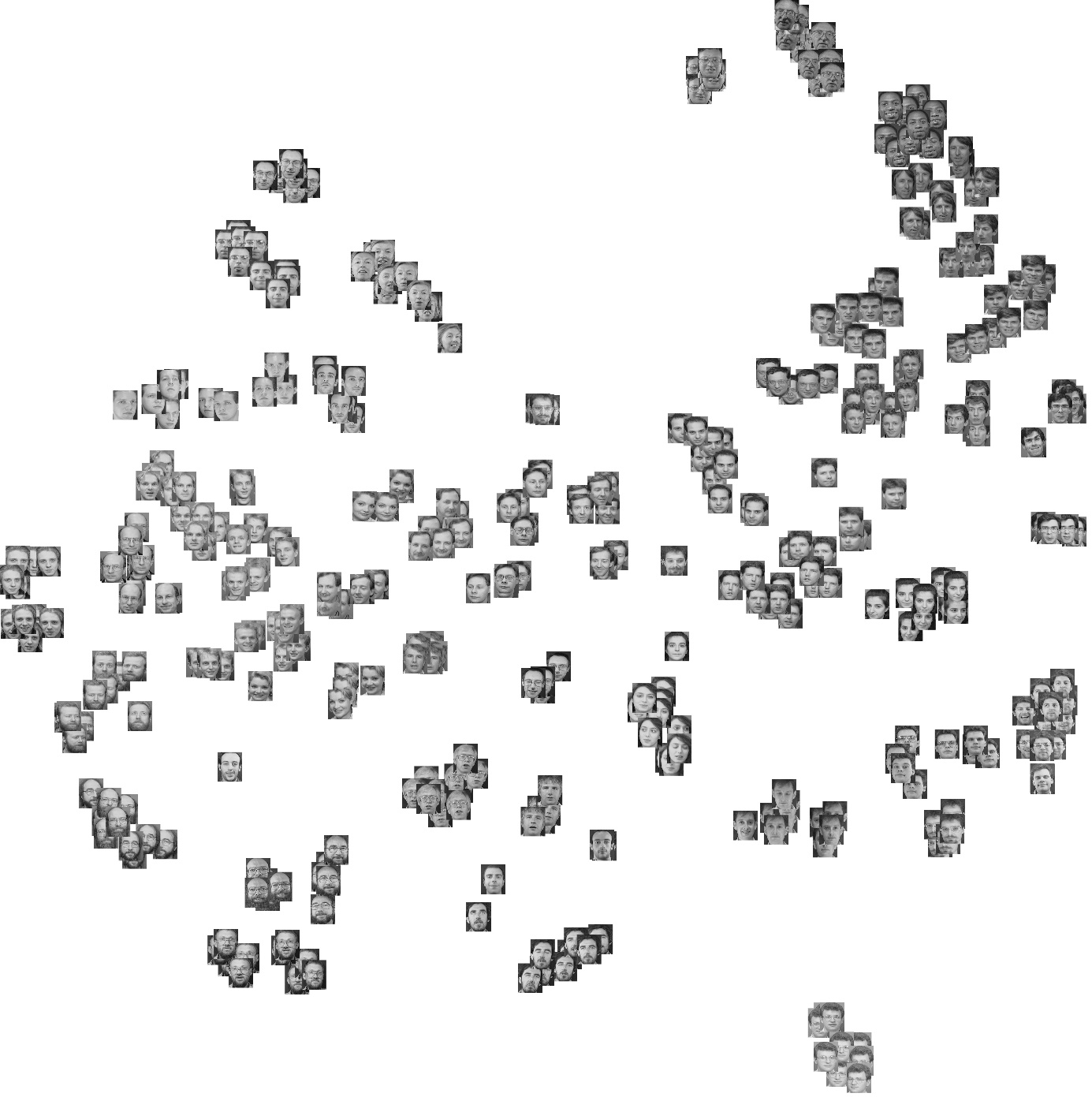
X = data

tsne = manifold.TSNE(n\_components=2, perplexity=40, n\_iter=3000, init='pca')

X\_tsne = tsne.fit\_transform(X)

plot\_embedding(X\_tsne, "Principal Components projection of the digits")

plt.show()



Comme on peut le voir, t-SNE permet de distinguer différents clusters, associés au différents visage, sans supervision ! Vous pouvez essayer de reproduire une visualisation similaire à l’aide de différents hyperparamètres (perplexité, nombre d’itérations)

#### Utilisation en pratique

Maintenant que vous avez une intuition du fonctionnement du t-SNE, on va voir comment l’utiliser en pratique et dans quelles circonstances il sera le plus pertinent.

Comme l’algorithme du t-SNE est une recherche de minimum local (pas global), on peut tomber facilement sur des résultats assez différents si on exécute plusieurs fois l’algorithme.

 Attends mais ce n’est pas censé être un algorithme non-supervisé ? Je ne suis pas censé appuyer sur un bouton magique et avoir mon résultat ?

Même si c’est un algorithme non-supervisé, il demande de développer une petite intuition de son fonctionnement et des différents comportements sur des cas simples afin de pouvoir comprendre un peu mieux les visualisations créées et les interpréter correctement.

Je ne fais que survoler ici les différents points d’attention que vous devrez avoir en utilisant cet algorithme. Pour plus de détail, je vous recommande notamment [**cet excellent article**](http://distill.pub/2016/misread-tsne/) sur le site distill.pub

##### Perplexité

Une notion que je n’ai pas évoqué dans l’explication sur le fonctionnement du t-SNE est la notion de perplexité, que vous allez devoir fixer pour utiliser l’algorithme. La perplexité représente en fait la taille de la variance de **pij**. Grosso modo elle représente la balance entre prise en compte de la structure globale et locale c’est à dire une estimation du nombre de voisins “proches” (relativement à l’ensemble du jeu de données) que possède chaque point.

Typiquement, vous pouvez tester une valeur de la perplexité entre 5 et 50 à partir de 5-10k observations.

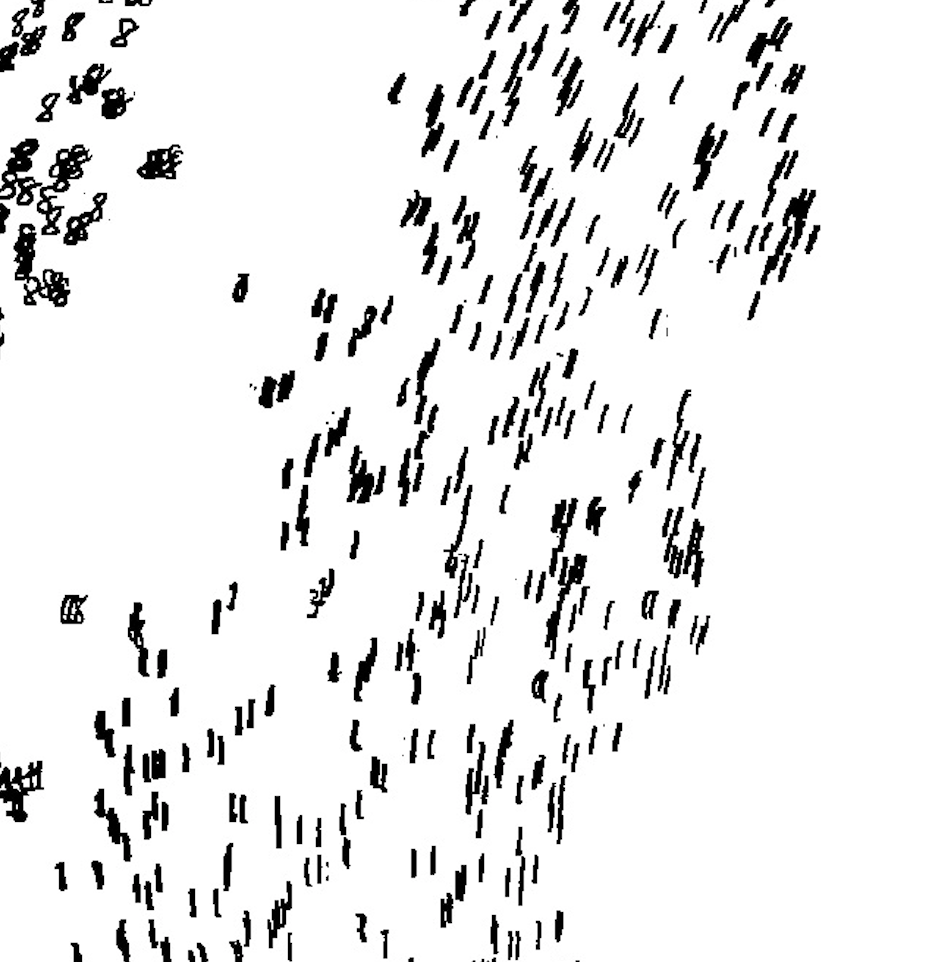
##### Visualisation du dataset

t-SNE permet de visualiser la structure des données à très hautes dimension, afin de pouvoir orienter par la suite la modélisation.

Alors, que représente cette visualisation du jeu de données ?

Et bien la principale chose qu’on peut distinguer sur le t-SNE c’est les différents clusters créés, qui peuvent représenter plusieurs catégories de données qui ont été détectées. Comme les chiffres sur l’image plus haut par exemple.

La seconde chose qui peut arriver, c’est que l’ordre relatif des différentes observations peut représenter une feature sous-jacente. Par exemple, j’ai zoomé ci-dessous sur le cluster de “1” d’une visualisation t-SNE des données MNIST. Comme on peut le voir, les 1 sont organisés selon leur orientation (de légèrement penché vers la gauche à légèrement penché vers la droite).



Si t-SNE vous fournit un clustering satisfaisant (avec des groupes assez bien séparés) par rapport à une PCA linéaire classique, cela veut dire que vous pourrez utiliser des algorithmes supervisés favorisant cette structure locale (comme le SVM que nous étudierons dans un autre cours).

##### Validation de features

Visualiser les données, c’est pas mal pour développer une intuition sur notre dataset. Une autre application assez utilisée du t-SNE c’est de valider les features que l’on vient de créer.

En effet, si vous avez créé des features qui vous semblent représentatives des distinctions entre les données, cette distinction doit se retrouver lors de la visualisation du t-SNE. Si vos features ne semble pas se retrouver groupées de manière significative, c’est peut être qu’elles ne sont pas si représentative des similarités/distinctions entre les données qui vous permettront de les modéliser.

Le t-SNE dans ce sens, permet de valider que les features créées sont bien représentatives de la topologie du jeu de données, et permettront potentiellement une modélisation correcte du phénomène.

#### Potentiels problèmes

##### Problèmes d'interprétation

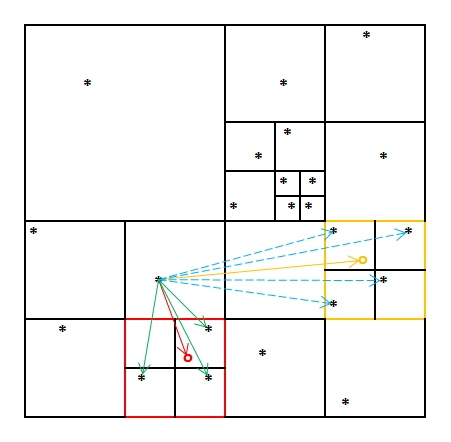
Il existe pléthore d’erreurs d’interprétation possible, sur tous les autres aspects du phénomène : la taille des cluster n’est pas significative, ni la distance entre les différents clusters, difficile de détecter la topologie (i.e. si une catégorie de points est contenue dans l’autre). Attention à ne pas surinterpréter les visualisations.

Je vous conseille l'excellent article sur le sujet : <http://distill.pub/2016/misread-tsne/>

##### Problèmes de temps de calcul

t-SNE possède une complexité quadratique  au nombre de points N, en temps et espace. Ce qui veut dire que ça devient rapidement inutilisable au-delà de quelque milliers/dizaine de milliers de points.

Heureusement, une technique a été développée depuis qui partitionne l’espace pour optimiser le calcul des différentes **pij**. Cette méthode utilise [la méthode de Barnes-Hut](https://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation) qui permet de partitionner l’espace afin d’optimiser le calcul des distances entre chaque point qui est le plus lourd en termes de calculs. En utilisant cette méthode d’approximation, t-SNE est applicable sur un gros jeu de données, et passe à une complexité . C’est en fait la méthode recommandée, qui est déjà implémentée dans la fonction t-SNE de scikit 😱



##### Données métriques

Il faut faire attention à la nature des données évaluées. En effet, t-SNE n’est pratique que lorsqu’il existe une symétrie entre les données. C’est à dire que si un point A est proche de B et B est proche de C, alors A est proche de C (l’inégalité triangulaire est respectée). Il existe une variante du t-SNE pour ce genre de besoin, à l'aide de plusieurs visualisations [(l'article de référence ici)](https://lvdmaaten.github.io/publications/papers/MachLearn_2012.pdf)

Quand est-ce que ce n’est pas le cas ?

Un cas simple qui n’est pas représentable par un t-SNE classique sont les données texte. Imaginons que le mot “roi” est proche de “reine” et que “reine” est proche du mot “femme”, ça ne veut pas dire que “roi” est proche de “femme” ! t-SNE ne fera pas cette distinction. Faîtes donc bien attention à la signification des relations entre les observations.

##### Pas de préservation de la structure globale

Comme je l’ai dit plus haut, l’algorithme t-SNE a été construit spécifiquement pour mettre en exergue la structure locale du jeu de données. C’est pourquoi il perd facilement la structure globale du dataset, qui peut aussi être utile à visualiser.

L’idée dans ce cas est soit d’utiliser plusieurs visualisations (une t-SNE pour observer la structure locale, et PCA ou Isomap pour la structure globale par exemple). Une autre solution est d’initialiser l’algorithme t-SNE par une PCA avant de le lancer, ce qui permet de conserver de base un peu plus de la structure globale (avec l'option init=’pca’)

### Résumé

On a vu dans ce chapitre l’algorithme t-SNE qui est une des méthodes les plus utilisées pour visualiser des données à grandes dimensions non-linéaires afin de repérer des structures locales intéressantes pour le travail de modélisation.

Attention encore une fois à bien développer une intuition de ce que peut apporter une visualisation t-SNE et à bien en faire plusieurs exécutions avec différents hyperparamètres. Il faut tester un bon nombre de fois ces méthodes avec des datasets différents pour bien les prendre en main.

# Partie 2

Bravo ! Vous avez réussi cet exercice !

### Compétences évaluées

* Choisir un algorithme de réduction de dimension non supervisé non linéaire en fonction de vos besoins et des caractéristiques des données
* Utiliser les principaux algorithmes de réduction de dimension non supervisé non linéaire classique
* Comprendre à quelles questions les principaux algorithmes de réduction de dimension non supervisé non-linéaire permettent de répondre

### Question 1

**Pour effectuer une réduction dimensionnelle non-linéaire, on peut utiliser**

* + 

l'algorithme du MDS métrique

* + 

le t-SNE, qui mettra l'accent sur la structure locale

* + 

le kPCA avec le kernel  

*Le MDS métrique est un algorithme de réduction dimensionnelle non-linéaire.*

*De même, le kernel est en fait juste le produit scalaire classique (noyau linéaire), ce qui fait que le kPCA avec ce noyau est linéaire.*

### Question 2

**Quelle feature serait la plus optimale pour effectuer une classification des observations ci-dessous ?**

****

* + 

Polynomial du second degré

* + 

Logarithmique

* + 

Une combinaison des deux

* + 

Aucune des deux

*On a simplement besoin d'une droite pour séparer les deux classes*

### Question 3

**Effectuer une réduction dimensionnelle non-linéaire permet de**

*Attention, plusieurs réponses sont possibles.*

* + 

Trouver des nouvelles features plus représentatives des données

* + 

Explorer les données car elle permet de visualiser certains comportements du phénomène plus simplement

* + 

Traiter des problèmes non-linéaires comme des problèmes linéaires

*En diminuant le nombre de dimensions, on trouve quelles sont celles qui représentent au mieux les données, qui peuvent ainsi être de nouvelles features exploitables par la suite et aussi mieux les visualiser.*

### Question 4

**Le kernel trick est une méthode permettant de...**

*Attention, plusieurs réponses sont possibles.*

* + 

ne pas avoir à expliciter l'application de l'espace de départ dans l'espace d'arrivée.

* + 

de revenir facilement dans l'espace de départ

* + 

d'effectuer rapidement une réduction dimensionnelle pour l'algorithme t-SNE

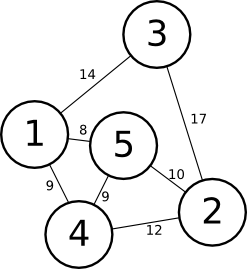
* + 

d'utiliser différentes fonctions comme produit scalaire, ce qui permet d'utiliser des méthodes linéaires sur des problèmes non-linéaires

*En formulant les algorithmes seulement avec des produits scalaires, on n’a pas besoin d'expliciter l'application de l'espace de départ dans l'espace d'arrivée, juste le résultat de ce produit. On utilise des méthodes linéaires en remplaçant le produit scalaire classique par une fonction noyau pour traiter des problèmes non-linéaires. t-SNE n'est pas une méthode qui utilise le kernel trick.*

### Question 5

**Pour l'algorithme de l'isomap, calculer le plus court chemin sur un graphe pondéré donne la distance géodésique. Quelle est la distance géodésique entre le point 3 et le point 4 sur le graphe ci-dessous ?**

****

* + 

3 - 1 - 4

* + 

3 - 1 - 5 - 4

* + 

3 - 2 - 5 - 4

* + 

3 - 2 - 4

*En calculant la valeur des différent trajets possible, la distance la plus courte est celle entre 3 et 4*

### Question 6

**Pourquoi serait-ce plus intéressant d'utiliser t-SNE par rapport à une analyse par composantes principales avec noyau ?**

* + 

Pour mieux observer la non-linéarité des données

* + 

Pour effectuer une réduction dimensionnelle

* + 

Pour favoriser la structure locale vis à vis de la structure globale

*Au dela des performances supérieures du t-SNE, les deux algorithmes effectuent des réductions dimensionnelles non-linéaires. La différence principale étant que le t-SNE se concentre sur la structure locale des observations.*

### Question 7

**t-SNE nous permet de visualiser les données de grandes dimensions afin de**

* + 

découvrir une structure dans les données que l'on ne verrait pas autrement

* + 

transformer les données pour les rendre plus réalistes

* + 

représenter fidèlement les données observées de départ

*La t-SNE permet uniquement de découvrir une structure existant au niveau local des données en réduisant assez le nombre de dimensions pour pouvoir les visualiser, sans pour autant conserver toute les spécificités (topologies, distances, symétrie, etc) de la structure originale.*

**Découvrez l’intérêt des algorithmes de clustering**

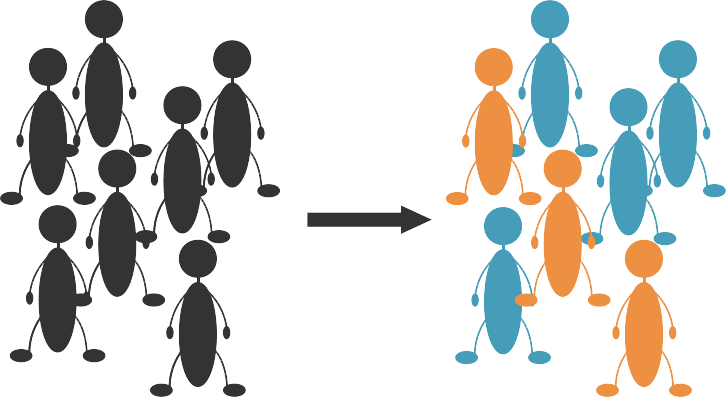
Les algorithmes de clustering permettent de **partitionner** les données en **sous-groupes**, ou *clusters*, de manière non supervisée. Intuitivement, ces sous-groupes regroupent entre elles des observations **similaires**. Les algorithmes de clustering dépendent donc fortement de la façon dont on définit cette notion de *similarité*, qui est souvent spécifique au domaine d'application.

**Comprendre les données**

Les algorithmes de clustering sont le plus souvent utilisés pour une analyse **exploratoire** des données.

Il s'agit par exemple d'identifier :

* des clients qui ont des comportements similaires (**segmentation de marché**);
* des utilisateurs qui ont des usages similaires d'un outil ;
* des communautés dans des réseaux sociaux ;
* des motifs récurrents dans des transactions financières.



Le clustering permet d'identifier des groupes d'utilisateurs similaires.

**Visualiser les données**

En plus d'un algorithme de réduction de dimension qui permet de visualiser les données en deux ou trois dimensions, on peut utiliser un algorithme de clustering pour former des sous-groupes de ces points, ou clusters.

On peut ainsi représenter visuellement les relations entre les points. Par ailleurs, on peut aussi, au lieu de représenter l'intégralité des données, afficher uniquement un point représentatif par cluster.



Une fois des clusters identifiés, on peut visualiser les données en utilisant uniquement un représentant par cluster.

**Inférer des propriétés des données**

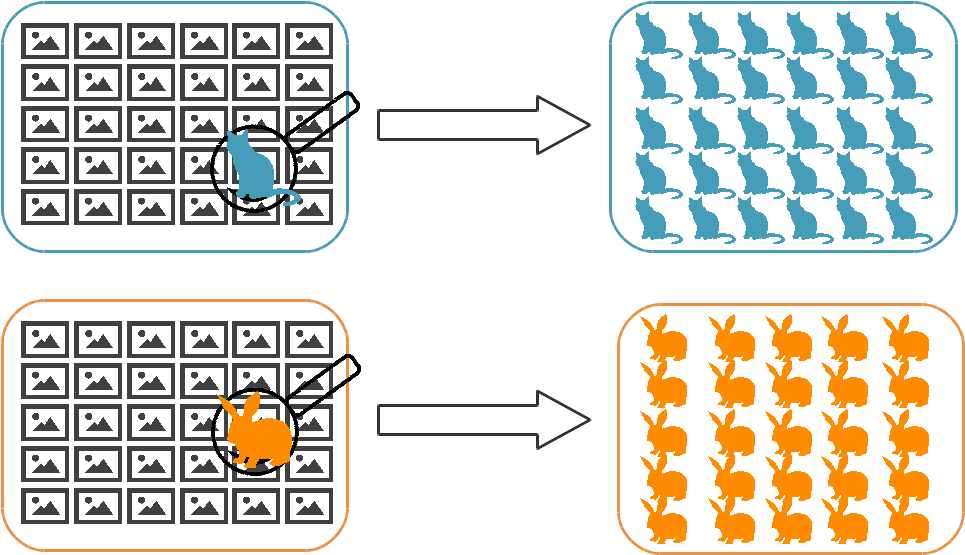
Cette utilisation des algorithmes de clustering est particulièrement utile dans les cas, fréquents, où il est coûteux d'étiqueter les données.

Prenons l'exemple de l'annotation d'une large banque d'images. Annoter chacune de ces images par ce qu'elle représente peut-être un travail long et fastidieux, au point d'ailleurs que les personnes qui l'effectuent peuvent involontairement introduire des erreurs par inattention ou fatigue. Il est moins coûteux et peut-être même plus efficace de laisser un algorithme de clustering regrouper entre elle les images similaires, puis de ne faire intervenir un opérateur humain qu'au moment d'assigner une étiquette à une classe d'images.

Ainsi, on peut utiliser des algorithmes de clustering pour **étendre** à tous les points du même cluster une propriété de l'un de ces points (dans l'exemple précédent, l'objet représenté.).

Ils s’agissent par exemple d'identifier :

* des images similaires, susceptibles de représenter le même objet, le même animal ou la même personne ;
* des textes similaires, susceptibles de parler du même sujet ;
* dans une image, les points qui appartiennent au même objet (on parle alors plus spécifiquement de **segmentation**).



Une fois les images regroupées dans des clusters, il suffit d'identifier que l'image du haut représente un chat pour inférer que toutes les images du cluster du haut représentent vraisemblablement des chats.

Dans la suite de cette partie, nous allons définir plusieurs critères à optimiser pour définir une partition intéressante des données, et les utiliser pour dériver quelques-uns des algorithmes de clustering les plus connus : clustering hiérarchique, k-means et DBSCAN.

**Résumé**

* Les algorithmes de clustering permettent de partitionner un jeu de données en sous-groupes d'observations similaires ;
* Ils peuvent être utilisés pour
* - mieux comprendre les données ;
* - faciliter la visualisation des données ;
* inférer des propriétés des données.

## Définissez les critères que doit satisfaire votre clustering

Dans le cas d'algorithmes non supervisés, le but de l'algorithme est moins évident à définir que dans le cas d'algorithmes supervisés, où il y a une tâche claire à accomplir. Le succès du modèle est donc plus subjectif. Le fait que la tâche soit plus difficile à définir n'empêche pas qu'il existe un large éventail de mesures de la performance d'un algorithme supervisé !

#### Distances et similarités

Faire un clustering, c'est regrouper ensemble les points les plus proches, ou les plus semblables. Le concept de clustering repose fortement sur ceux de distance et de similarité.

Ces concepts vont nous être très utiles pour formaliser à quel point deux observations sont proches les unes des autres ; à quel point une observation est proche d'un cluster ; et à quel point deux clusters sont proches les uns des autres.

Une distance est formellement définie comme une fonction , qui vérifie trois propriétés :

* d(x,x)=0 (auto-similarité)
* d(u,v)=d(v,u) (symétrie)
* d(u,v) ≤ d(u,x)+d(x,v) (inégalité triangulaire).

Les exemples les plus utilisés de distances sont

* la distance euclidienne : 
* la distance de Manhattan : , ainsi appelée parce qu'elle correspond en deux dimension à la distance parcourue par un taxi dans les rues de Manhattan, qui sont toutes soit parallèles soient perpendiculaires les unes aux autres.

Il s'agit de cas particulier de la distance de Minkowski, qui repose sur la norme **ℓq** :



On peut utiliser une distance pour une définir une similarité : plus deux points sont distants, moins ils sont similaires, et inversement. On peut par exemple transformer une distance d en similarité s de la façon suivante :  

Une autre façon courante de définir une similarité est d'utiliser la corrélation de Pearson entre les variables :



où  représente la moyenne des entrées de u.

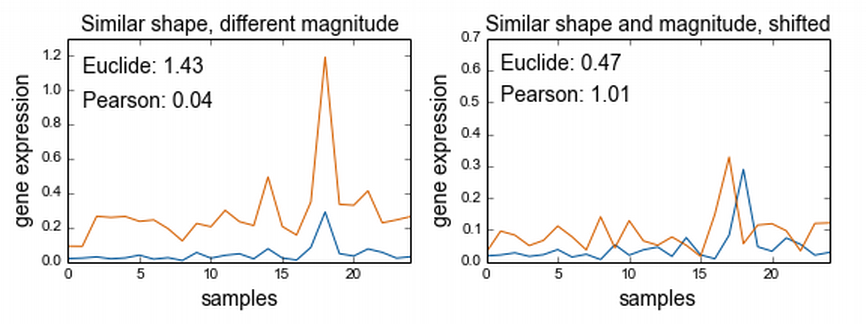
Si les données sont centrées, , on peut réécrire la corrélation de Pearson comme



qui est le cosinus de l'angle formé par les vecteurs u et v. On appelle ainsi parfois cette similarité la similarité cosinus, ou cosine similarity en anglais.

On peut étendre ce concept et définir une similarité en utilisant un noyau : 

Une particularité de la corrélation de Pearson est de prendre en compte la forme des signaux plutôt que leur amplitude. À l'inverse, la distance euclidienne prend surtout l'amplitude en considération. On fera donc attention au choix de la distance !

**

La courbe bleue et la courbe orange représentent chacune une variable, mesurée sur 25 observations. À gauche, les profils de ces deux variables sont similaires, mais pas leurs amplitudes : l'inverse de la corrélation de Pearson est faible, la distance euclidienne grande. À l'inverse, à droite, les variables ont des formes et amplitudes similaires, mais sont décalées l'une par rapport à l'autre. L'inverse de la corrélation de Pearson est grand, la distance euclidienne faible.

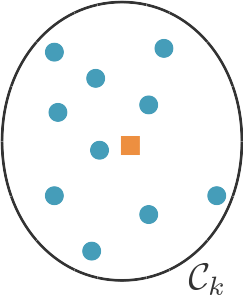
#### Forme des clusters

Nous voulons de nos clusters qu'ils soient

* resserrés sur eux mêmes : deux points qui sont proches doivent appartenir au même cluster
* loin les uns des autres : deux points qui sont éloignés doivent appartenir à des clusters différents.

Notons d la distance que nous allons utiliser, par exemple la distance euclidienne.

On appelle centroïde d'un cluster le barycentre des points de ce cluster : .



Le centroïde (en orange) du cluster $C\_k$

On définit ainsi l'homogénéité (notée **Tk** pour tightness en anglais) d'un cluster comme la moyenne des distances de chacun des points contenus dans ce cluster au centroïde .

Un cluster resserré aura une hétérogénéité plus faible qu'un cluster de points épars.

Pour caractériser non pas un cluster, mais l'ensemble des clusters, on peut calculer la moyenne des homogénéités de chaque cluster :



Nous avions dit plus haut aussi vouloir que les clusters soient loin les uns des autres. Pour quantifier cela, on définit la séparation de deux clusters comme la distance entre leurs centroïdes : . Encore une fois, on peut calculer la moyenne de ces quantités sur l'ensemble des paires de cluster **(k,l)** obtenues :



Nous avons maintenant deux critères à optimiser. Pour nous faciliter la tâche, nous pouvons les regrouper en un seul critère, l'indice de Davies-Bouldin. L'idée de cet indice est de comparer les distannce intra-cluster (c'est l'homogénéité), que l'on veut faibles, aux distances inter-cluster (la séparation), que l'on veut grandes. Pour un cluster k, cet indice est donné par  et est d'autant plus faible que tous les clusters sont homogènes (le numérateur de cette fraction est petit) et que tous sont bien séparés (le dénominateur est grand). Encore une fois, on peut calculer un indice de Davies-Bouldin global en moyennant les indices de Davies-Bouldin de tous les clusters :



Une autre façon de quantifier à quel point un clustering répond à ces deux exigences (homogénéité et séparation) est de mesurer ce que l'on appelle le coefficient de silhouette. Pour un point x donné, le coefficient de silhouette s(x) permet d'évaluer si ce point appartient au « bon » cluster : est-il proche des points du cluster auquel il appartient ? Est-il loin des autres points ? Pour répondre à la première question, on calcule la distance moyenne de x à tous les autres points du cluster C\_k auquel il appartient : .

Pour répondre à la deuxième, on calcule la plus petite valeur que pourrait prendre a(x), si x était assigné à un autre cluster : .

Si x a été correctement assigné, alors a(x) < b(x). Le coefficient de silhouette est donné par .

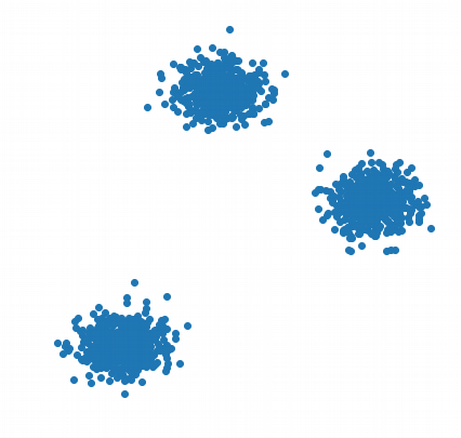
Il est donc compris entre -1 et 1, et d'autant plus proche de 1 que l'assignation de x à son cluster est satisfaisante. Pour évaluer un clustering, on peut calculer son coefficient de silhouette moyen.

Dansscikit-learn, le coefficient de silhouette se calcule avec [sklearn.metrics.silhouette\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html" \t "_blank).

#### Stabilité des clusters

Un autre critère important est la stabilité du clustering : si je lance l'algorithme plusieurs fois sur les mêmes données avec une initialisation différente, ou sur des sous-ensembles différents des données, ou encore sur les mêmes données légèrement bruitées, est-ce que j'obtiens les mêmes résultats ?

Ce critère est particulièrement pertinent pour choisir le nombre de clusters : si le nombre de clusters choisi correspond à la structure naturelle des données, le clustering sera plus stable que si ce n'est pas le cas. Sur l'image ci-dessous, un algorithme qui cherche à déterminer 3 clusters va raisonnablement retrouver les trois groupes que l'on voit. Mais si on lui demande de déterminer 4 clusters, la répartition dans ces 4 clusters sera plus aléatoire et ne sera pas nécessairement deux fois la même. C'est une façon de déterminer que 3 est un meilleur nombre de clusters que 4.



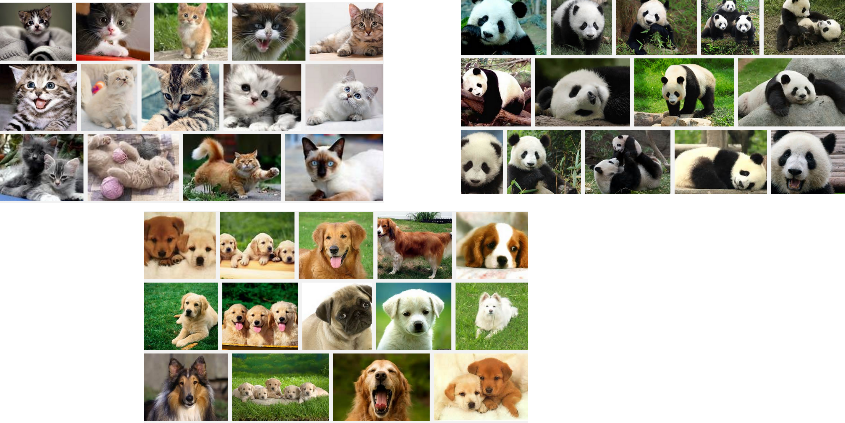
Combien de clusters ?

#### Compatibilité avec des connaissances spécifiques au domaine

Très souvent, on va aussi évaluer un algorithme de clustering « à l'œil », et regarder si les clusters proposés ont du sens. Les textes regroupés dans ce cluster parlent-ils tous du même sujet ? Les images dans ces deux clusters représentent-elles des sujets différents ?

Pour faire ça plus proprement, on peut travailler sur un jeu de données (par exemple, un sous-ensemble des données qui nous intéressent) sur lequel on connaît une partition raisonnable des données. On va ensuite comparer cette partition avec celle retournée par notre algorithme de clustering.

Par exemple, on peut travailler avec 1000 images étiquetées par l'objet qu'elles représentent : chien, chat, ou panda.



Chien, chat, ou panda ?

Il s'agit maintenant d'évaluer si les groupes formés par l'algorithme de clustering correspondent à ceux définis a priori. Facile ! C'est comme d'évaluer un algorithme de classification multi-classes. Mais pas si vite : si on s'intéresse à ce que les mêmes objets appartiennent au même cluster, que ce cluster soit le premier, le deuxième ou le k-ième importe peu.

Il faut donc utiliser des mesures de performance spécifiques, qui permettent d'évaluer la concordance de deux partitions du jeu de données. Vous en trouverez une liste dans [scikit-learn.metrics](http://scikit-learn.org/stable/modules/clustering.html" \l "clustering-evaluation" \t "_blank).

Un exemple de ces mesures est l'indice de Rand. L'indice de Rand est la proportion de paires de points **(x1, x2)** qui sont groupées de la même façon dans les deux partitions : soit parce que, dans les deux cas, x\_1 et x\_2 appartiennent au même cluster, soit parce que, dans les deux cas,  x\_1 et x\_2 appartiennent à des clusters différents.

L'indice de Rand peut être gonflé artificiellement en prédisant beaucoup de clusters : les paires de points appartenant à des clusters différents seront nombreuses, et il y aura de grandes chances que deux points étiquetés différemment soient dans deux clusters différents. L'indice de Rand ajusté (ARI, pour Adjusted Rand Index) corrige cet effet en normalisant l'indice de Rand (RI) :



où E(RI) est l'espérance de la valeur de l'indice de Rand, autrement dit l'indice obtenu en partitionnant les données au hasard. Cet index ajusté est proche de 0 pour un clustering aléatoire et égal à 1 uniquement quand le clustering correspond exactement à la partition initiale. Dansscikit-learn, on peut le calculer grâce àsklearn.metrics.adjusted\_rand\_score.

### Résumé

Pour évaluer un algorithme de clustering, on peut s'intéresser à :

* la forme des clusters qu'il produit (sont-ils denses, bien séparés). On utilise ici souvent le coefficient de silhouette ;
* la stabilité de l'algorithme ;
* la compatibilité des résultats avec des connaissances spécifiques au domaine, que l'on peut évaluer à l'aide de mesures d'enrichissement.

## Partitionnez vos données avec un algorithme de clustering hiérarchique

Dans le chapitre précédent, nous avons défini les critères que doivent satisfaire les clusters que l'on forme : nous cherchons à faire en sorte de séparer des clusters homogènes. Cependant, optimiser exactement une mesure comme le coefficient de silhouette ou l'index de Davies-Bouldin est très intensif en termes de temps de calcul. Nous sommes donc obligés d'adopter des heuristiques.

Dans ce chapitre, nous allons parler d'algorithmes de clustering hiérarchique, qui cherchent à créer des clusters homogènes bien séparés par récurrence.

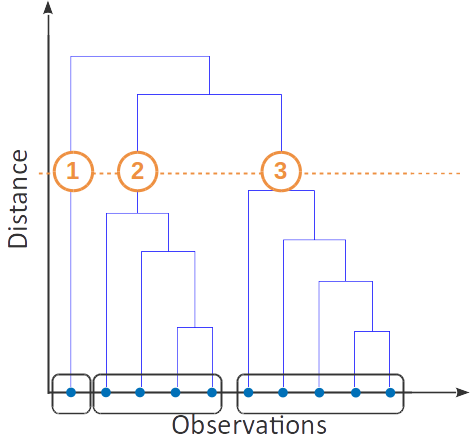
#### Partition hiérarchique des observations

Dans le cas du clustering agglomératif (ou bottom-up), on commence par considérer que chaque point est un cluster à lui tout seul. Ensuite, on trouve les deux clusters les plus proches, et on les agglomère en un seul cluster. On répète cette étape jusqu'à ce que tous les points appartiennent à un seul cluster, constitué de l'agglomération de tous les clusters initiaux.

L'approche inverse, le clustering divisif (ou top-down), consiste à initialiser avec un unique cluster contenant tous  les points, puis à itérativement séparer chaque cluster en plusieurs, jusqu'à ce que chaque point appartiennent à son propre cluster.

#### Dendrogrammes

Un dendrogramme est un arbre dont les feuilles sont les points d'un jeu de données. Chaque nœud de l'arbre représente un cluster (les feuilles sont des clusters contenant un point chacun). Les clusters qui ont le même parent sont agglomérés pour former ce cluster parent. La longueur des U est proportionnelle à la distance entre les deux clusters qu'elle connecte.



Un exemple de dendrogramme. En coupant au niveau de la ligne horizontale orange, on obtient 3 clusters.

Pour les grands jeux de données, vous pourrez vous contenter de visualiser uniquement la partie supérieure du dendrogramme, pour éviter d'avoir à représenter tous les points.

#### Distances entre deux clusters

Mais comment calculer la distance entre deux clusters ? Il existe pour cela plusieurs méthodes, dites méthodes de lien (linkage methods) car elles permettent de lier les clusters entre eux.

On peut dire que deux clusters sont proches si deux de leurs points sont proches. Ainsi, la distance entre deux clusters est la distance minimale entre deux points, l'un appartenant au premier cluster et l'autre au deuxième :

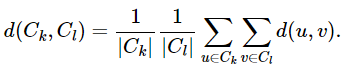


* C'est ce qu'on appelle le lien simple, ou single linkage en anglais.

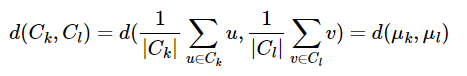
Mais on peut préférer penser que deux clusters sont proches si tous leurs points sont proches les uns des autres. Ainsi, la distance entre deux clusters est la distance maximale entre deux points, l'un appartenant au premier cluster et l'autre au deuxième :



* C'est ce qu'on appelle le lien complet, ou complete linkage en anglais. On peut aussi définir le lien moyen, dans lequel la distance entre deux clusters est la distance moyenne entre toutes les paires de points deux à deux :



Cette dernière distance, le lien moyen, est aussi parfois appelée UPGMA pour Unweighted Paired Group Method with Arithmetic mean. Elle contraste avec le lien centroïdal (centroid linkage), qui est non pas la moyenne des distances entre paires de points appartenant chacun à un des clusters, mais la distance entre les moyennes des points de chaque cluster, autrement dit la distance entre les deux centroïdes :



* Le lien centroïdal est aussi appelé UPGMC pour Unweighted Paired Group Method with Centroid.

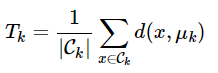


Lien simple, lien complet,  lien centroïdal (UPGMC), et lien moyen (UPGMA, difficile à représenter !)

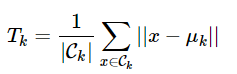
La distance **d** dans les formules précédentes n'est pas forcément la distance Euclidienne. Rappelez-vous la discussion que nous avons eu à ce sujet dans [**le chapitre précédent**](https://openclassrooms.com/courses/explorez-vos-donnees-avec-des-algorithmes-non-supervises/definissez-les-criteres-que-doit-satisfaire-votre-clustering).

#### Clustering de Ward

Les méthodes que je viens de présenter permettent de garantir la séparation des clusters : les points éloignés les uns les autres ne sont agglomérés dans le  même cluster que tardivement. Mais quid de l'homogénéité ? Nous allons maintenant essayer d'agglomérer les clusters de sorte à maximiser l'homogénéité du résultat. Rappelez-vous, la mesure d'homogénéité



* Si on utilise la distance euclidienne pour d, alors



* Remplaçons dans cette équation  ; l'idée est conceptuellement similaire, mais on mesure maintenant la variance du cluster  (on parle aussi d'inertie).

La variance (ou inertie) inter-cluster d'un clustering est alors donnée par



* Le clustering de Ward est une méthode de clustering hiérarchique qui choisit, à chaque étape, d'agréger deux clusters de sorte à minimiser l'augmentation de la variance inter-cluster que cette agrégation entraîne.

Le clustering hiérarchique est implémenté dans le module [cluster](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html) de scikit-learn.

##### Avantages et inconvénients du clustering hiérarchique

Le clustering hiérarchique a l'avantage qu'il n'est pas nécessaire de définir le nombre de clusters à l'avance (on explore toutes les possibilités). Cependant, cela ne fait que repousser cette décision. Celle-ci peut se faire sur la base d'un dendrogramme (cela suppose de pouvoir bien visualiser le dendrogramme, ce qui sera plus aisé sur un petit jeu de données). On peut aussi évaluer les différentes partitions trouvées (une pour chaque valeur possible de nombre de clusters) avec une mesure telle que le coefficient de silhouette ou l'indice de Davies-Bouldin.

Par contre, sa complexité algorithmique est lourde. À chaque itération, pour décider quels clusters joindre, nous aurons besoin des distances deux à deux entre toutes les paires de points du jeu de données. Les calculer a un coût quadratique en temps de calcul ; les stocker, un coût quadratique en espace mémoire. Pour cette raison, le clustering hiérarchique est plus adapté aux échantillons contenant un faible nombre d'individus.

### Résumé

* Le clustering hiérarchique permet de partitionner un jeu de données de manière hiérarchique.
* À chaque étape, on agrège les deux clusters les plus proches
* La distance entre clusters peut être calculée de la façon suivante :
  + Lien simple : la distance entre deux clusters est celle entre les deux points les plus proches.
  + Lien complet : la distance entre deux clusters est celle entre les deux points les plus éloignés.
  + Lien centroïdal : la distance entre deux clusters est celle entre les deux centroïdes.
  + Lien moyen : la distance entre deux clusters est la distance moyenne entre les points des deux clusters.
  + Clustering de Ward : la distance entre deux clusters est calculée de façon à minimiser la variance inter-cluster.
* On peut visualiser une partition hiérarchique des données avec un dendrogramme.

## Partitionnez vos données avec l’algorithme du k-means

### 

### Partitionnez vos données avec l'algorithme du k-means

Supposons maintenant que, plutôt que d'explorer tous les nombres de clusters possibles de 1 (contenant tous les points) à n (contenant chacun un seul point), nous nous fixons à l'avance le nombre K de clusters. En pratique, on pourra ainsi se contenter d'explorer une fourchette de valeurs de K, qui sera déterminée en fonction des besoins de nos applications. Dans le cas de la segmentation de marché, on commencera en général par s'intéresser à une segmentation grossière, avec peu de clusters (dans ce cas, des groupes de clients aux profils similaires), quitte à la raffiner ultérieurement. Si l'on analyse de larges banques de données de photographies, que l'on espère pouvoir regrouper par sujets, le nombre de clusters voulus peut-être bien plus élevés.

Nous avons établi dans les chapitres précédents que de minimiser la variance intra-clusters  permet de définir des clusters homogènes. Étant donné K, nous allons donc chercher à répartir les points **x1, x2, …, xn** en K groupes  de telle sorte que  soit minimale. C'est le problème que la méthode dite du k-means cherche à résoudre.

#### Algorithme de Lloyd

Il n'est pas possible de résoudre ce problème de manière exacte. Nous sommes donc obligés d'utiliser une heuristique.

Dans le chapitre précédent, nous avons utilisé le clustering de Ward ; cependant, il a tendance à être coûteux en temps de calcul. Le clustering de Ward partitionne les données de manière hiérarchique, mais nous avons dit ici nous intéresser à une unique partition en K clusters.

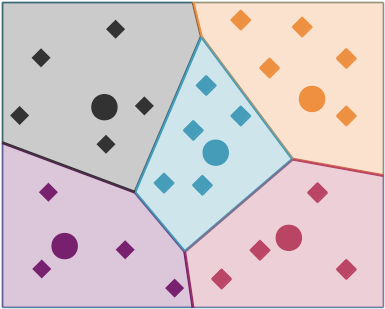
L'algorithme de Lloyd procède de la façon suivante. On commence par choisir aléatoirement K centroïdes parmi nos observations. On associe ensuite chaque point au centroïde dont il est le plus proche, formant ainsi K clusters. On peut maintenant recalculer le centroïde de chaque cluster (son barycentre), et recommencer l'opération jusqu'à ce que l'algorithme converge.

Il s'agit d'une stratégie gloutonne. L'algorithme converge en général très rapidement, mais peut tomber dans un minimum local. Pour cette raison, il peut être utile de le relancer plusieurs fois et d'évaluer la variance intra-cluster pour chacune de ces répétitions.

Dans scikit-learn, le k-means est implémenté dans cluster.KMeans. Par défaut, l'algorithme est répété 10 fois (paramètre n\_init).

#### Forme des clusters du k-means

Dans le résultat du k-means, les points appartenant à un cluster C\_k sont plus proche du centroïde μkμk que de n'importe quel autre centroïde. Cela implique que l'algorithme du k-means partitionne l'espace selon une tessellation de Voronoi. En particulier, cela implique aussi que les clusters obtenus sont convexes. On ne peut donc pas, avec l'algorithme du k-means, obtenir de cluster en forme de croissant de lune, d'anneau, etc.

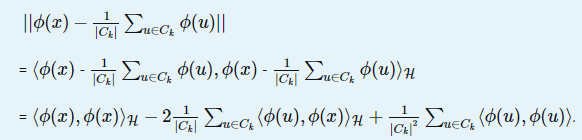


Tesselation de Voronoi en 5 cellules. Chaque disque représente un centroïde

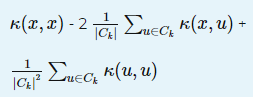
On peut pallier cette limitation grâce au k-means à noyau, ou kernel k-means en anglais. N'hésitez pas à vous rafraichir la mémoire sur les noyaux dans le [**chapitre 2 de la partie 2 de ce cours**](https://openclassrooms.com/courses/explorez-vos-donnees-avec-des-algorithmes-non-supervises/plongez-dans-une-variete-mds-et-tsne) !

Supposons que l'on envoie les données dans un espace de redescription **H** grâce à une application **ϕ**. On peut appliquer l'algorithme du k-means aux données transformées {ϕ(x1), …, ϕ(xn)} en ne faisant apparaître les ϕ(xi) que dans des produits scalaires ⟨ϕ(xi),ϕ(xj)⟩, que l'on peut donc remplacer par le noyau correspondant à ϕ, κ(xi,xj). Par définition, κ associe ⟨ϕ(u)ϕ(v)⟩ à (u, v).

En effet, à chaque itération du k-means, nous avons besoin de connaître la distance entre tous les points du jeu de données, et les K centroïdes, c'est à dire, pour tout x et pour tout . Si l'on envoie les points x dans un espace de redescription **H** grâce à une application ϕ, et que l'on applique l'algorithme du k-means dans cet espace de redescription, on aura donc besoin d'être capable de calculer, pour tout x et pour tout k, la quantité



Cette quantité peut donc s'écrire



Le kernel trick s'applique donc.

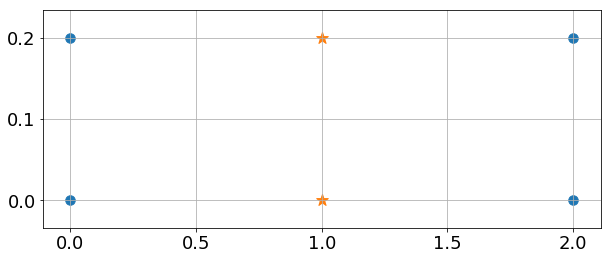
Dans scikit-learn, le kernel k-means est implémenté dans cluster.KernelKMeans.

La tessellation de Voronoi intervient aussi quand on parle de l'algorithme du plus proche voisin, qui est un algorithme d'apprentissage supervisé. Pour plus de détails sur la tesselation de Voronoi, vous pouvez vous référer à **[Wikipedia](https://www.wikiwand.com/fr/Diagramme_de_Vorono%C3%AF" \t "_blank)**.

#### k-means++

Un des inconvénients du k-means est que l'initialisation se fait aléatoirement. L'algorithme n'est donc pas déterministe, et on peut obtenir des résultats différents en le faisant tourner plusieurs fois. Et certains de ces résultats peuvent être très mauvais, c'est-à-dire très éloignés de la solution optimale que l'on obtiendrait si l'on pouvait résoudre exactement notre problème de minimisation de variance intra-cluster !

On peut voir ça sur un exemple simple :



Initialisé avec les deux étoiles, le k-means (K=2) va créer deux clusters horizontaux.

Si on applique le k-means, initialisé avec les 2 étoiles orange, aux 4 points bleus, les deux clusters initiaux seront {(0, 0), (2, 0)} et {(0, 0.2), (2, 0.2)} et ne bougeront pas. Cependant, la variance intra-cluster du clustering {(0, 0), (0, 0.2)}, {(2, 0), (2, 0.2)} est bien plus faible et ce serait donc un résultat plus désirable.

Pour éviter ce problème, on peut utiliser la variante k-means++ du k-means. Dans cette variante, au lieu d'initialiser les clusters au hasard, on va créer les centroïdes initiaux de sorte à les « éparpiller » le plus possible dans les données. Plus précisément, on choisit un premier centroïde aléatoirement parmi les points du jeu de données. Puis on calcule la distance D entre ce centroïde et chacun des autres points.

On choisit ensuite un deuxième centroïde de telle sorte qu'un point x a une probabilité d'être choisi proportionnelle à , et donc d'autant plus grande que x est loin du premier centroïde. On répète cette opération (calcul de la distance D au k-ième centroïde puis utilisation de cette distance pour sélectioner le (k+1)-ème) jusqu'à avoir K centroïdes. On applique ensuite le k-means avec cette initialisation.

Dans scikit-learn, l'initialisation par défaut d'une instance de cluster.KMeansse fait avec kmeans++ (paramètre init).

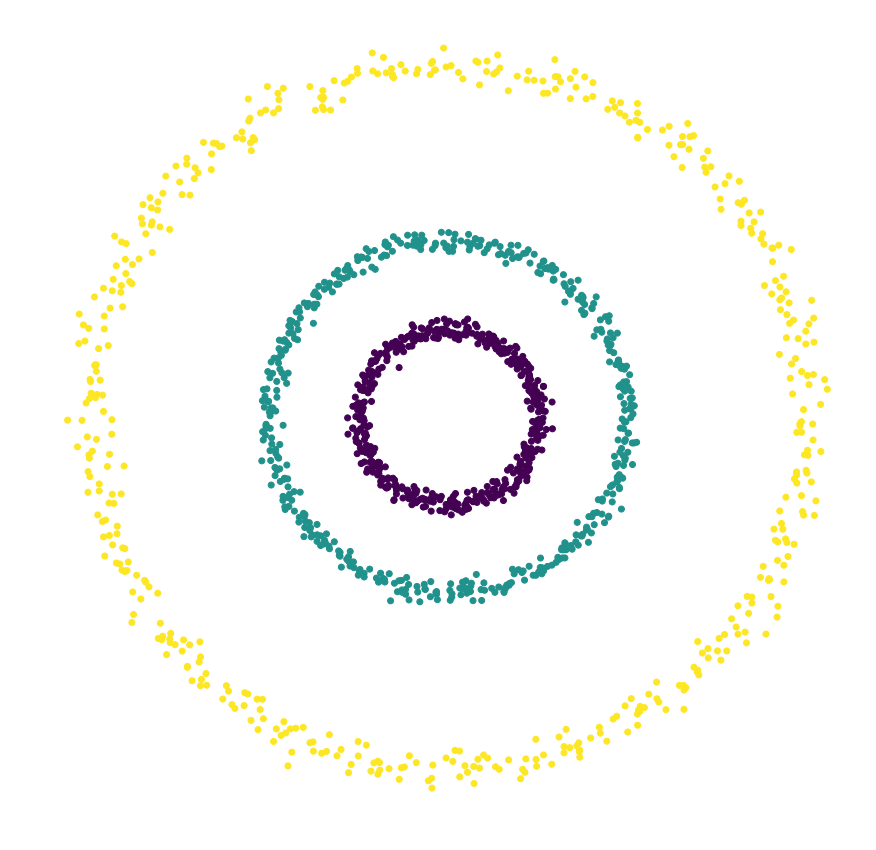
### Résumé

* L'algorithme du k-means permet de rechercher efficacement une partition des données dont la variance intra-cluster est minimale
* Cependant il s'agit d'une approche heuristique qui peut retourner un minimum local plutôt que global
* L'initialisation kmeans++ ainsi que des répétitions multiples permettent de mitiger ce problème

## Partitionnez vos données avec DBSCAN

### Comment séparer des cercles imbriqués les uns dans l’autre ?

Quel algorithme de clustering utiliser pour former des clusters non-convexes, comme des demi-lunes ou des cercles imbriqués les uns dans les autres ? Dans la partie précédente, nous avons vu que le k-means forme nécessairement des clusters convexes, ce qui n'est pas le cas de ce genre de figure. Par exemple, sur l'image ci-dessous, un segment reliant un point en haut du cercle extérieur à un point en bas du même cercle pourra contenir des points des deux autres cercles/clusters).

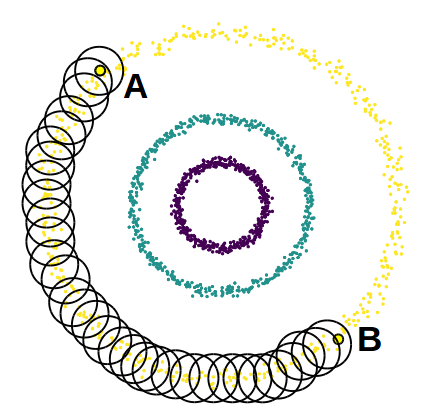


Quel algorithme de clustering utiliser pour former trois clusters correspondant aux trois cercles de ces données ?

Nous avons vu précédemment que le kernel k-means permet de résoudre ce genre de problème. Mais ce n'est pas la seule façon de faire !

### Clustering par densité

Les méthodes de clustering par densité sont basées sur l'observation suivante : bien que les deux points que j'ai mentionnés plus haut (notés A et B sur la figure ci-dessous) ne puissent pas être liés l'un à l'autre par un segment sans intersecter les deux autres clusters, on peut créer un chemin pour passer de l'un à l'autre de proche en proche en restant à l'intérieur du même cluster.



On peut connecter A à B par des petits voisinages (les cercles) contenant uniquement des points du cluster extérieur.

Un peu de vocabulaire maintenant...

Étant donné un nombre réel positif **ϵ**, on appelle **epsilon-voisinage** d'un point x l'ensemble des points du jeu de données dont la distance à x est inférieure à **ϵ** :



J'utilise ici une distance d quelconque : ce n'est pas nécessairement la distance euclidienne (même si ça l'est sur mes figures).

Étant donné un entier naturel , on dit que x est un **point intérieur** (core point en anglais) si son **ϵ**-voisinage contient au moins  points : .

On dit maintenant que deux points u et x sont **connectés par densité** si l'on peut passer de l'un à l'autre par une suite d'**ϵ**-voisinages contenant chacun au moins  points. Autrement dit, il existe une suite de points intérieurs v1, v2, … vm tels que v1 appartient à l'ϵ-voisinage de u, v2 appartient à l'ϵ-voisinage de v1, et ainsi de suite, jusqu'à ce que x appartienne au voisinage de vm. On dit aussi alors que x est **atteignable par densité** depuis u.

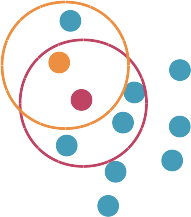
Pourquoi tous ces vocabulaires ? Parce que nous allons maintenant créer des clusters de points atteignables par densité les uns depuis les autres.

### DBSCAN

L'algorithme **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) a été introduit en 1996 dans ce but. Cet algorithme est très utilisé, raison pour laquelle il a obtenu en 2014 une distinction de contribution scientifique ayant résisté à l'épreuve du temps.

DBSCAN itère sur les points du jeu de données. Pour chacun des points qu'il analyse, il construit l'ensemble des points atteignables par densité depuis ce point : il calcule l'epsilon-voisinage de ce point, puis, si ce voisinage continent plus de n\_min points, les epsilon-voisinages de chacun d'entre eux, et ainsi de suite, jusqu'à ne plus pouvoir agrandir le cluster. Si le point considéré n'est pas un point intérieur, c'est à dire qu'il n'a pas suffisamment de voisins, il sera alors étiqueté comme du bruit. Cela permet à DBSCAN d'être robuste aux données aberrantes puisque ce mécanisme les isole.

Un point initialement considéré comme du bruit par DBSCAN peut être ultérieurement rattaché à un autre cluster : il peut appartenir au voisinage d'un autre point, qui lui contient plus de n\_min points, comme sur la figure suivante



Dans l'image ci-dessus, si , le point orange n'est pas un point intérieur car son epsilon-voisinage ne contient pas suffisamment de points. Par contre, il appartient à l'epsilon-voisinage du point rose, qui lui contient plus de n\_min points ; ainsi le point orange sera assigné au même cluster que le point rose plutôt que d'être considéré comme du bruit.

L'algorithme de DBSCAN est le suivant :

1. Prendre un point x qui n’a pas été visité

2. Construire N = voisinage(eps, x)

3. If |N| < n\_min:

Marquer x comme bruit

Else:

Initaliser C = {x}

agrandir\_cluster(C, N, eps, n\_min)

Ajouter C à la liste des clusters

Marquer tous les points de C comme visités

4. Repeat 1-3 until tous les points ont été visités

La procédure agrandir\_cluster est donnée par

agrandir\_cluster(C, N, eps, n\_min):

For u in N:

If u n’a pas été visité:

N’ = N(eps, u)

If |N’| >= n\_min:

N = union(N, N’)

If u n’appartient à aucun autre cluster:

Ajouter u à C

### Avantages et inconvénients de DBSCAN

L'algorithme DBSCAN est difficile à utiliser en très grande dimension : souvenez-vous [du fléau de la dimensionalité](https://openclassrooms.com/courses/explorez-vos-donnees-avec-des-algorithmes-non-supervises/comprenez-pourquoi-reduire-la-dimension-de-vos-donnees#/id/r-4446149). Les boules de rayon epsilon et de grande dimension ont tendance à ne contenir aucun autre point.

Le choix des paramètres ϵ et  peut aussi être délicat : il faut veiller à utiliser des paramètres qui permettent de créer suffisamment de points intérieurs (ce qui n'arrivera pas si  est trop grand ou ϵ trop petit). En particulier, cela signifie que DBSCAN ne pourra pas trouver des clusters de densités différentes.

DBSCAN a le grand avantage d'être efficace en temps de calcul sans requérir de prédéfinir le nombre de clusters. Enfin, comme je l'ai dit au début de ce chapitre, il permet de trouver des clusters de forme arbitraire.

## Entraînez-vous à manipuler des algorithmes de clustering avec sklearn

### À vous de jouer

Pour vous entraîner, réalisez cet exercice étape par étape. Une fois terminé, vous pouvez comparer votre travail avec les pistes que je vous propose.

Le but de cette activité est de faire un clustering d’images représentant des chiffres manuscrits, en python. Nous allons utiliser des images extraites d’un très célèbre jeu de données, MNIST. Pour cela, définissez le chemin custom\_data\_home vers le répertoire où vous voulez le télécharger :

custom\_data\_home = <mon répertoire>

puis utilisez le code python suivant :

from sklearn import datasets

Si le message ci-dessus renvoie une erreur INTERNAL SERVER ERROR, c’est parce que le serveur `mldata.org` depuis lequel fetch\_mldata tente de télécharger des données ne fonctionne pas. Vous pouvez alors alternativement télécharger les données depuis <https://github.com/amplab/datascience-sp14/raw/master/lab7/mldata/mnist-original.mat> dans le répertoire <custom\_data\_home>/mldata.

Les données MNIST sont assez lourdes (70000 images). Pour cette activité, vous pouvez travailler avec une fraction du jeu de données : une image sur 50.

X = mnist.data[::50, :]

y = mnist.target[::50]

Chaque observation de ce jeu de données est une image de 28 pixels par 28 pixels, dont l’étiquette y[i] est le chiffre qu’elle représente, et les 784 features le niveau de gris (entre 0 et 255) du pixel correspondant (car 28 x 28 = 784). Pour visualiser une de ces images, par exemple celle d’index 42, vous pouvez utiliser le code suivant :

sample\_idx = 42

sample\_image = np.reshape(X[sample\_idx, :], (28, 28))

plt.imshow(sample\_image, cmap='binary')

Pour cet exercice il vous est demandé :

1. d’effectuer un partitionnement de X en 10 clusters, avec l’algorithme de clustering de votre choix
2. de visualiser le résultat de ce clustering en deux dimensions (obtenues par exemple grâce à tSNE, après scaling des données)
3. d’évaluer la qualité de ce partitionnement, d’une part intrinsèquement (sans utiliser y) et d’autre part en le comparant aux chiffres représentés par les images (en utilisant y).

**Coup de pouce :** Pour faciliter la visualisation des résultats, vous pouvez représenter chaque point par le chiffre qu’il représente vraiment, coloré selon le cluster auquel il appartient. Pour cela, vous pouvez utiliser pour le i-ème point le code suivant :

plt.text(dimension1[i], dimension2[i], # à vous de définir ces dimensions !

        str('%d' % y[i]),  # le point i est représenté par son chiffre

        color=plt.cm.Set2(myclust.labels\_[i]/10.)

        )

Dans ce code, myclust est un objet cluster de scikit-learn. N’hésitez pas à changer la colormap Set2, cf <https://matplotlib.org/users/colormaps.html>

Vérifiez-bien que vous avez les éléments suivants :

* la réduction de dimension est fonctionnelle (par exemple grâce au tSNE) ;
* les données ont été scalées avant la réduction de dimension ;
* l'algorithme de clustering approprié a été utilisé et retourne 10 clusters.

### Vérifiez votre travail

Voici un [exemple](https://s3-eu-west-1.amazonaws.com/static.oc-static.com/prod/courses/files/explorez-vos-donnees-avec-des-algorithmes-non-supervises/corrige.py.zip) pour vous permettre de vérifier votre travail.