Chapitre 6 : Le clustering et ses applications

Lorsque les données que l'on étudie contiennent des milliers de points (un point pouvant être un contenu textuel, une image, un objet abstrait, une observation dans un tableau…), il devient vite rapidement impossible d'avoir une bonne idée du contenu global de sa base. Quels en sont les objets principaux ? Les plus représentatifs ? Les observations qui s'éloignent le plus de la norme attendue et qui mériteraient sans doute un examen plus minutieux. C'est pourquoi on a rapidement eu dans de nombreux domaines besoin de techniques permettant d'avoir rapidement une bonne idée des caractéristiques de ses données. Parmi ces stratégies, les méthodes de partitionnement de données (ou *clustering*) ont connu un grand succès et ce pour plusieurs raisons : elles sont souvent peu coûteuses en terme de calcul, fonctionnent sur tout type d'objet, et surtout sont globalement interprétables. Le clustering consiste à diviser ses données en paquets homogènes, de façon à regrouper des objets qui présentent des caractéristiques communes. Ainsi une bonne partition maximise les ressemblances entre objets qui sont placés dans un même groupe (similarité intra-classe) tout en maximisant la différence entre les objets d'un groupe et ceux d'un autre (inertie inter-classe). Disposer d'une segmentation d'une grosse base de donnée a divers usages fort pratique. Cela permet par exemple de la discrétiser ou de la compresser, on peut construire une base de donnée qui soit 10 fois plus petite tout en conservant la diversité et la distribution de la base de départ. Cela permet aussi de créer à moindre coût des algorithmes de classification (un nouvel objet appartient à la catégorie des objets du cluster des objets avec lequel il a été rangé). Et bien sur, cela facilite grandement l'analyse des gros volumes.

Si l'idée globale est assez intuitive (faire un gros tri de ses objets et regrouper ce qui va ensemble), l'implémentation n'est pas immédiate. En effet il convient déjà de définir ce que signifie « être similaire » pour les objets qu'on manipule. Et cela dépendra de l'usage que l'on veut faire des données que l'on a à notre disposition. Cela nécessite aussi de disposer d'un protocole pour regrouper les objets. Part on de petits paquets initiaux pour les agrandir ? Choisit on au départ le nombre de paquets que l'on souhaite pour ensuite répartir les points de donnée ? La question du partitionnement d'un dataset qui est au départ un problème de data science et d'analyse se convertit rapidement en une pure question d'algorithmique souvent détachée des données que l'on étudie.

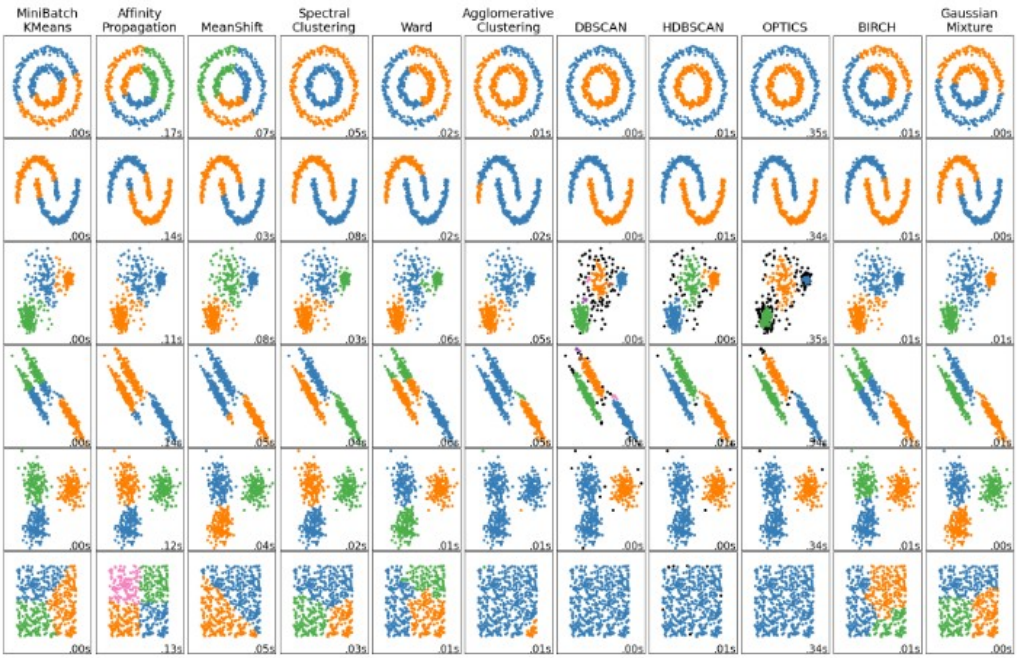
On présentera dans ce cours les algorithmes de clustering les plus connus et leurs cas d'usage privilégiés avant de discuter certaines difficultés posées par le clustering, notamment la gestion et l'interprétabilité des hyperparamètres de ces algorithmes et la problématique de la représentation des objets considérés. Enfin on présentera différents cas d'usage des techniques de clustering appliqué à l'analyse de données et de bases de connaissance, et notamment le Topic Modeling qui permet à moindre coût d'identifier rapidement les thèmatiques saillantes de corpus très importants.

I) Partitionner en 2025 : petite bibliothèque d’algorithmes

Le clustering est un problème ancien qui a connu une littérature abondante. Il s’agit à l’origine d’une question mathématique (algèbre), celle du partitionnement d’un ensemble en sous ensembles. On a pu rapidement énumérer le nombre de partitionnements possibles sur un ensemble de taille finie. Puis la question a évolué pour être capable de créer des partitions répondant à des caractéristiques spécifiques. Si on fournit une relation d’équivalence ou un ordre de grandeur sur un ensemble, alors on peut contraindre les partitions possibles. Lorsque l’on introduit un ordre de grandeur, les partitions peuvent être organisées différemment. Par exemple, si l’on veut partitionner un ensemble en fonction d’un critère quantitatif comme la taille, le poids ou une distance mesurée dans un espace métrique, il devient possible de définir des seuils ou des intervalles qui contraignent les regroupements possibles.

Avec ces bases mathématiques, le problème du partitionnement est devenu central dans de nombreux domaines comme la théorie des graphes, l’analyse des ensembles finis et la combinatoire. Les premières solutions visaient à énumérer systématiquement les partitions possibles d’un ensemble donné. Par exemple, le nombre de partitions d’un ensemble à n éléments est donné par les nombres de Bell, qui croissent de manière exponentielle avec n. Cette explosion combinatoire a rapidement posé des défis computationnels et a conduit les chercheurs à s’intéresser à des approches algorithmiques permettant de construire des partitions satisfaisant des critères spécifiques sans nécessairement explorer toutes les possibilités.

Avec le temps, cette problématique s’est étendue bien au-delà des ensembles finis et des relations simples. Dans les applications modernes, les données sont souvent complexes, multidimensionnelles et hétérogènes. Le défi n’est plus seulement de partitionner un ensemble, mais de le faire de manière optimale selon des critères définis, souvent en présence de contraintes fortes ou dans des espaces de grande dimension. On étudiera des algorithmes comme *k-means*, *hierarchical clustering* ou *spectral clustering* cherchent à maximiser une mesure de qualité des clusters (comme la similarité intra-classe ou l’inertie inter-classe) tout en respectant des contraintes de scalabilité et de robustesse.



Avant cela quelques remarques :

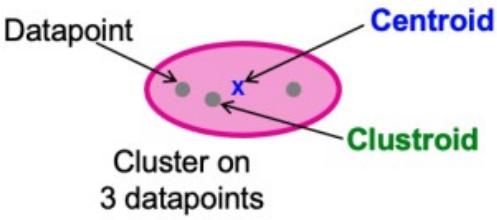
* Bien qu'il soit possible d'entraîner spécifiquement des modèles de ML pour la tâche de clustering, la plupart des méthodes existantes sont des méthodes non supervisées.

* On considérera dans les algorithmes présentés que l'on cherche à regrouper des objets de grande dimensionnalité (en des termes plus simples, de très longs vecteurs…)

* Il n'est pas évident de trouver un ordre optimal pour présenter ces algorithmes, et on les regroupe souvent selon l'intuition qui est derrière l'algorithme.

* Chacune de ces techniques a un comportement très différent selon aussi la nature des objets sur lesquels on les applique, c'est à dire qu'une méthode peut être très bien adaptée à un jeu de données, mais pas du tout à un autre.

Scikit learn implémente par exemple une grande variété d'algorithmes de clustering et présente clairement dans sa documentation[1] que leur efficacité dépend de la structure des données.



* Le but n'est pas de vous écraser sous une liste d'algorithmes mais de vous faire comprendre comment les principaux fonctionnent.

* La meilleure façon de comprendre est de visualiser.[2]

a) Les méthodes d'assignation de points

Une première grande famille d'algorithmes fonctionne en triant un à un les points dans différents clusters, puis en réajustant parfois la partition. On place les points dans le groupe qu'on pense être le plus adapté pour eux.

Les méthodes par assignation reposent pour la plupart sur la notion de centroïde. Un centroïde est un point qui est considéré comme le plus représentatif de son groupe. Ce peut être un point réel (celui qui est géométriquement le plus central dans le groupe, déterminé par la norme) ou un point fictif créé en faisant la moyenne de la représentation de chaque objet du groupe. L'idée est que chaque cluster est représenté par un centroïde et que l'on assigne des points dans un groupe en comparant ces points au centroïdes et en les mettant dans le groupe dont ils sont le plus similaire au centroïde. Un certain nombre de méthodes reposent sur ce principe.

La plus célèbre est incontestablement l'algorithme k-means qui fait partie du top 10 des algorithmes à connaître pour tout data scientist qui se respecte.

Dans l'algorithme k-means, on doit préciser le nombre de clusters (k) que l'on souhaite obtenir à la fin du processus. Il s'agit d'un algorithme non supervisé qui repose sur les propriétés géométriques de l'espace latent dans lequel sont représentés les objets. L'intuition mathématique

1 https://scikit-learn.org/1.5/modules/clustering.html
2 https://larevueia.fr/clustering-les-3-methodes-a-connaitre/ Les algos dynamiques de cette page sont très bien.

derrière l'algorithme est que des objets similaires seront plus proches géométriquement dans l'espace latent. On peut donc ramener le problème de clustering à un problème d'optimisation.

L'objectif de l'algorithme k means est de déterminer k centroids (points qui minimisent la distance cumulative dans le groupe), un pour chaque classe. L'algorithme procède comme suit :

*1 k centroïdes sont initialisés aléatoirement dans l'espace vectoriel. Ils ne correspondent pas à un objet en particulier.

*2 On calcule la distance entre chaque point et chaque centroïde et on assigne chaque point au centroïde le plus proche.

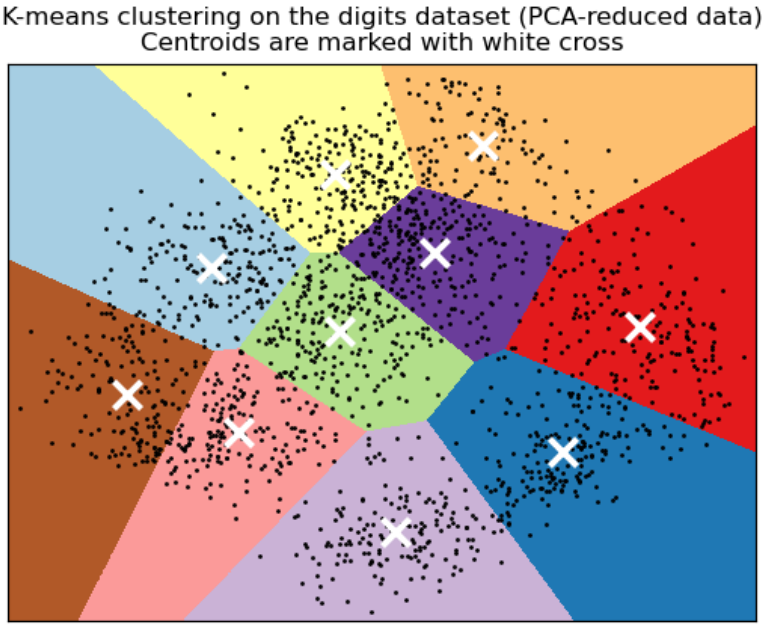
*3 On recalcule le centroïde de chaque cluster comme étant le point fictif minimisant la distance cumulative pour ses membres. Cela implique que les centroïdes changent de place, donc les points ne sont plus forcément dans le groupe du centroïde qui est le plus proche d'eux.

*4 On répète les actions 2 et 3 tant que des points sont changés de place. On s'arrête quand le système a atteint son équilibre, c'est à dire que le réajustement des centroïdes ne cause plus le changement de groupe d'aucun point.

Cet algorithme est très simple mais extrêmement efficace. Sa principale limitation est qu'il nécessite de connaître à l'avance le nombre de clusters que l'on veut obtenir. Or dans la réalité, quand on veut explorer la structure d'un dataset, on ne connaît pas son contenu ni en combien de groupe il faudrait la découper pour bien faire. Plusieurs méthodes ont été proposées pour déterminer la meilleure valeur de k pour un dataset, et parmi elles la méthode d'Elbow qui consiste à tester différentes valeurs de k de sorte à minimiser la somme des carrés des distances de chaque point à son centroïde sur l'ensemble du dataset.

$$WSS = \sum_{i=1}^{m} (x_i - c_i)^2$$

Représenter cette distance en fonction de k donne une courbe forcément décroissante, mais qui forme une sorte de coude (forme de la fonction inverse). Il faut choisir l'angle du coude comme valeur de k.



Voici une implémentation en python de Kmeans, à comprendre et méditer.

```python
import numpy as np

class KMeans:
    def __init__(self, n_clusters=3, max_iter=300, tol=1e-4):
        """
        Initialize the KMeans clustering algorithm.

        Parameters:
        - n_clusters: int, default=3
        - max_iter: int, default=300          Maximum number of iterations for the algorithm.
        - tol: float, default=1e-4  Tolerance for convergence. The algorithm stops if the change in cluster centers is less than this value.
        """
        self.n_clusters = n_clusters
        self.max_iter = max_iter
        self.tol = tol
        self.cluster_centers_ = None  # To store the final cluster centers

    def fit(self, X):
        """
        Perform k-means clustering on the input data.
        """
        # Randomly initialize the cluster centers
        n_samples, n_features = X.shape
        self.cluster_centers_ = X[np.random.choice(n_samples, self.n_clusters, replace=False)]

        for iteration in range(self.max_iter):
            # Step 1: Assign each sample to the nearest cluster center
            distances = np.linalg.norm(X[:, np.newaxis] - self.cluster_centers_, axis=2)  # Compute distances
            labels = np.argmin(distances, axis=1)  # Find nearest cluster center for each sample

            # Step 2: Calculate new cluster centers as the mean of assigned samples
            new_cluster_centers = np.array([
                X[labels == k].mean(axis=0) if len(X[labels == k]) > 0 else self.cluster_centers_[k]
                for k in range(self.n_clusters)
            ])

            # Step 3: Check for convergence (change in cluster centers is less than tol)
            shift = np.linalg.norm(self.cluster_centers_ - new_cluster_centers, axis=1).max()
            self.cluster_centers_ = new_cluster_centers

            if shift < self.tol:
                print(f"Converged in {iteration + 1} iterations.")
                break
        else:
            print("Reached maximum iterations without full convergence.")

        self.labels_ = labels

    def predict(self, X):
        """
        Predict the closest cluster for each sample in X. tantamount to a classification systme
        """
        distances = np.linalg.norm(X[:, np.newaxis] - self.cluster_centers_, axis=2)
        return np.argmin(distances, axis=1)
```

Résultat d’un clustering de quelques centaines de points en 10 clusters (k=10) projeté en dimension 2[3].

3 https://scikit-learn.org/1.5/auto_examples/cluster/plot_kmeans_digits.html#sphx-glr-auto-examples-cluster-plot-kmeans-digits-py pour l’illustration

b) Les méthodes hiérarchiques / par agglomération

Une autre manière de constituer des paquets est de le faire par agglomération. À l'état initial, on considère que chaque point de la base est un cluster à lui tout seul (singleton), puis on fusionne ensemble les deux clusters les plus similaires jusqu'à atteindre une des conditions d'arrêts définies.

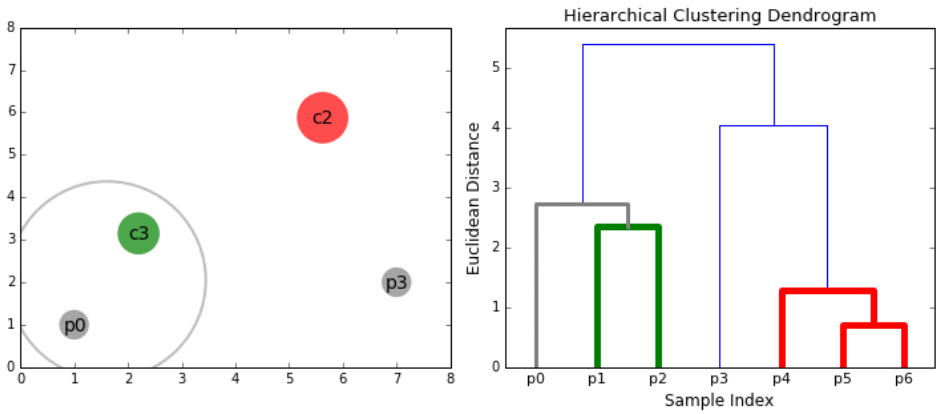
Cette approche très intuitive pose plusieurs question :

* Comment décider quel cluster est similaire à un autre quand il contient plusieurs points ? On prend le centroïde du cluster ? La moyenne ? Le score de similarité de deux représentants les plus similaires entre eux ? En mélangeant les options de min, max, moyenne on arrive à la proposition de 9 heuristiques différentes déjà … On pourrait aussi définir le score d'un matching inter clusters selon les propriétés intrinsèques (cohésion, distance inter-cluster avec les voisins) du potentiel cluster formé par la fusion). On doit donc définir une heuristique de similarité entre deux clusters.

* Comment définir les conditions d'arrêt de ce processus de fusion ? Un seuil numérique sur le score de similarité défini entre deux cluster ? S'arrêter quand la similarité inter-cluster tombe sous un certain seuil si on fait l'association ? Une taille de cluster à ne pas dépasser ? Un nombre min de clusters à conserver ? Il faut donc définir des conditions de terminaison, au risque de se retrouver sinon avec un unique cluster où tout est aggloméré.

* Souvent, tester toutes les combinaisons et calculer tous ces scores peut être computationellement bien plus cher que les méthodes par assignation. Pour une méthode par assignation, on tente d'assigner chaque point à un cluster (O(n*k) avec n nombre de points, k nombre de cluster pour chaque round d'assignation. Même si on a plusieurs rounds pour ajuster les placements, et que $k \leq n$ par définition on est à une complexité de O(n2) alors que les approches agglomératives doivent tester à chaque étape des combinaisons et associations possibles entre chaque paire, soit une complexité de O(n3). Avec des files de priorité, on peut réduire l'implémentation naïve à O(n2log(n)) mais cela reste beaucoup pour des datasets comprenant de nombreux points de grande dimension (la plupart des nôtres…).

Cette méthode présente plusieurs avantages théoriques comme pratique notables. Le premier est de prendre l'objet comme centre de la réflexion, l'autre est son interprétabilité car on comprend beaucoup mieux pourquoi chaque regroupement a été fait. On peut aussi résumer l'ensemble des fusions dans un diagramme nommé dendogramme (< grec dendron, arbre) pour une meilleure visualisation du déroulement de l'algorithme.



En face de cette méthode « bottom up », on trouve également la méthode « top down » équivalente qui consiste à partir d'un unique groupe dans lequel se trouvent tous les points, puis à le diviser jusqu'à satisfaire des conditions d'arrêt bien définies. Son implémentation est plus coûteuse, et trouver un critère de division impose de prédire les propriétés de toutes les combinaisons de groupes potentiellement créés. On parle de clustering par division.

Proposition d'exercice : implémenter un clustering hiérarchique en python.[4] Le but est de pouvoir l'utiliser avec des vecteurs denses de taille variable.

De quelles structures de données avez-vous besoin ?

Choisissez un critère de terminaison et une métrique de similarité inter cluster. Faites en sorte qu'on puisse facilement étendre votre code en implémentant d'autres métriques de similarité ou conditions de terminaison.

c) Les méthodes par densité

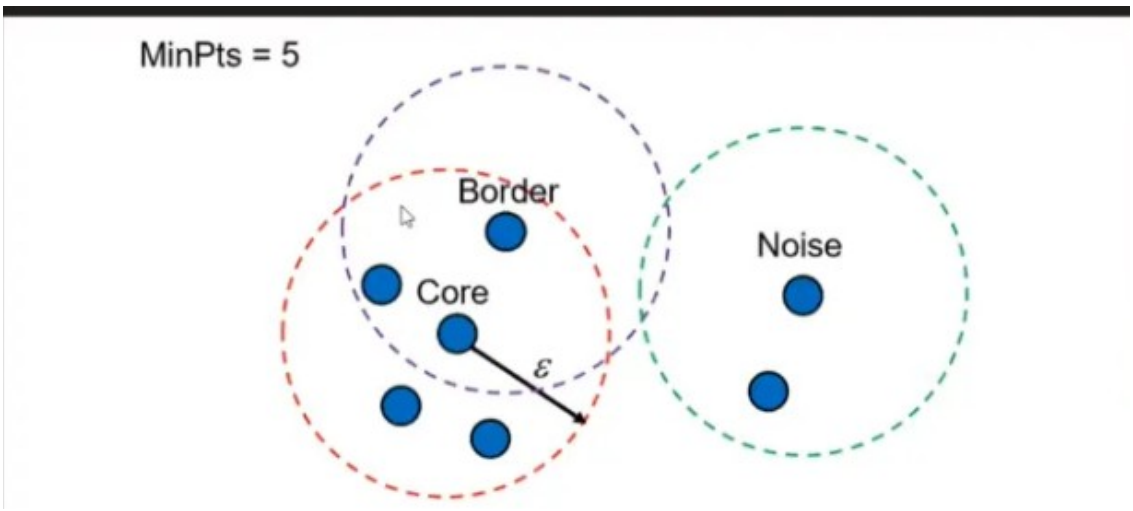
Une autre approche très connue mélange l'intuition d'une interprétation géométrique de l'espace latent avec la tentative de capturer des informations sur la structure de la base. L'idée est de définir des clusters en partant des zones où les points sont les plus densément situés. Ces méthode ont en plus le bon goût, contrairement aux précédentes, de permettre de repérer les points qui sont différents des autres.

L'algorithme le plus connu de cette famille est dit « density-based spatial clustering of applications with noise », DBSCAN.

L'idée de départ est de diviser l'espace vectoriel entre régions denses et régions vides. On peut mesurer la densité autour d'un point en choisissant une distance et en regardant pour chaque point combien il a de points à une distance inférieure à ce seuil autour de lui. On peut donc choisir comme paramètre le nombre minimum de points pour qu'une zone soit considérée comme dense (minpts) ou encore cette distance maximale qui définit si deux points sont dans une zone dense ou non (noté epsilon , > 0). Ainsi un point va être assigné dans un cluster si autour de lui à une distance inférieure à epsilon il a minpts voisins ou s'il est voisin d'un nœud répondant à ces deux critères. Si un point n'est pas repéré alors il n'est pas placé dans un cluster. Cela fait que les points qui sont trop différents des autres (outliers) ne sont pas placés par défaut dans un cluster. Cela permet de limiter les « faux positifs » et d'avoir une conception plus précise du contenu du dataset. Mis à part les points situés dans une région dense et les points qui sont totalement déconnectés, il existe une troisième catégorie de points qui se situe dans les limbes de la connexion : ce sont les points qui ont moins de minpoints voisins dans le voisinage à distance epsilon mais qui sont voisins d'un point qui l'est. Ils se situent en d'autres termes à la frontière d'un cluster à une distance max de epsilon d'un point d'une région dense[5].

4 https://medium.com/@sachinsoni600517/mastering-hierarchical-clustering-from-basic-to-advanced-5e770260bf93

5 https://medium.com/@sachinsoni600517/clustering-like-a-pro-a-beginners-guide-to-dbscan-6c8274c362c4 illustration



On peut distinguer les points qui sont dans la même zone dense (ils sont à moins de epsilon et ont au moins minpts voisins) de ceux qui sont connectables par densité, c'est à dire si il y a une chaîne de points P1 … Pn tel que Pi et Pi+1 sont dans la même zone dense

L'algorithme DBSCAN part de ces définitions pour identifier des clusters.

* 1 D'abord selon epsilon et minpoints on classifie tous les points comme core point, border point ou bruit.

*2 Puis pour chaque core point non clusterisé on crée un nouveau cluster. Et on ajoute dedans tous les points qui sont connectables par densité à ce point.

*3 Pour les border points restants on les assigne au cluster du point clusterisé le plus proche, les point considérés comme du bruit et pas connectés ne sont pas assignés à un cluster et sont considérés comme des outliers.

Cette approche par densité a pour avantage comparé aux approches géométriques ou hiérarchiques de repérer les outliers du dataset. On a de plus pas besoin de préciser combien de cluster on veur, ce nombre est défini par les valeurs de epsilon et minpoints.

Néanmoins à cause de la philosophie de l'algorithme, certaines limites apparaissent. L'algorithme est très sensibles à la valeur des hyperparamètres et on peut obtenir un découpage très différent en changeant les valeurs. De plus si des zones de points sont très denses, l'algorithme va tout agglomérer sans identifier la spécificité. Pour y remédier on peut le modifier pour ajouter un nombre maximal de points par cluster ou on peut redécouper les clusters trop gros. Pour palier cette dernière limitation, un algorithme un peu plus complexe, HDBSCAN hierarchical density-based spatial clustering inclut la notion de hiérarchie des clusers pour mieux s'adapter à des ensembles de points qui présentent localement des densités différentes.

Idée d'exercice : faire des recherches sur HDBSCAN et expliquer la différence avec l'algorithme DBSCAN.

d) Le problème particulier du clustering sur les graphes

Le clustering sur les graphes pose des problèmes particuliers. Si on obtient un embedding des nœuds (soit en utilisant une représentation vectorielle de ce que l'objet représente, soit en utilisant une approche d'embedding adaptée aux graphes comme node2vec, alors on peut appliquer immédiatement toutes les techniques qu'on a vu précédemment. Néanmoins elles ont par rapport aux graphes une limitation importante : elles ne capturent pas du tout le fait que dans le graphe, certains des nœuds soient connectés par des arrêtes ![6] En bref on a pas de prise en compte directe de la connectivité du graphe alors que dans un graphe les relations sont des objets aussi importants que les nœuds.

C'est pourquoi certaines techniques de clustering ont été créées spécialement se basant sur la connectivité du graphe. De nombreuses approches existent mais je ne vais me concentrer que sur une seule qui repose sur la notion de modularité[7]. Ces méthodes sont souvent appliquées dans l'analyse des réseaux sociaux pour faire ce que l'on appelle de la détection de communautés, c'est à dire repérer les nœuds ou groupes de nœuds qui sont très densément connectés les uns aux autres.

Un des algorithmes les plus connus est l'algorithme dit de Louvain (2008)[8] qui crée des clusters dans un graphe à partir d'une métrique nommée Modularité (modularity) qui quantifie la qualité d'un groupe de nœuds par rapport au cas où les connexions du graphes seraient aléatoire. L'algorithme maximise le score de modularité des clusters qu'il désigne. Encore une fois le calcul d'un clustering est ramené à un problème d'optimisation.

Cet algorithme d'optimisation dépend donc de la définition de la quantité que l'on souhaite optimiser, en l'occurrence, la modularité. Ce score, compris entre -1/2 et 1 mesure la densité des arrêtes dans une communauté par rapport à la densité des arrêtes entre les communautés (remarquez la similitude de formulation avec la définition globale de la tâche de clustering). La modularité se calcule dans un graphe pondéré avec la formule suivante [9]:

6 Pour être pointilleux, certaines techniques d'embedding de graphe prennent en compte les voisins d'un nœud pour construire la représentation vectorielle du nœud. Mais l'information est potentiellement diffuse, le modèle d'embedding pas forcément adapté à effectuer des clusterings…

7 https://www.biz.uiowa.edu/faculty/nstreet/graphClustering.pdf pour les curieux

8 https://arxiv.org/abs/0803.0476

9 https://fr.wikipedia.org/wiki/M%C3%A9thode_de_Louvain que j'ai pris bien paginée de Wikipédia …

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j),$$

où :

- A_{ij} représente le poids de l'arête entre les nœuds i et j ;
- k_i et k_j sont la somme des poids des arêtes liées aux nœuds i et j respectivement ;
- m est la somme de l'ensemble des poids des arêtes du graphe ;
- c_i et c_j sont les communautés de nœuds ;
- δ est une fonction delta.

delta : fonction identité de Kronecker, = 1 si les deux sont égaux, 0 sinon. En gros on calcule la modularité en additionnant ensemble que les informations des nœuds qui sont dans la même communauté.

L'algorithme procède ainsi :

*1 Comme pour le clustering hiérarchique on initialise chaque nœud comme étant dans sa propre communauté / cluster.

*2 Puis on évalue le gain de modularité en fusionnant une communauté avec une communauté voisine (entendons connectée par des arrêtes). On calcule donc ce que chaque déplacement créerait en terme de modularité. (calcul de Delta Q occasionné par la suppression du nœud i de sa communauté d'origine et son ajout à une communauté j)

*3 On fusionne les communautés dont on a prévu que la fusion donnerait une bonne modularité.

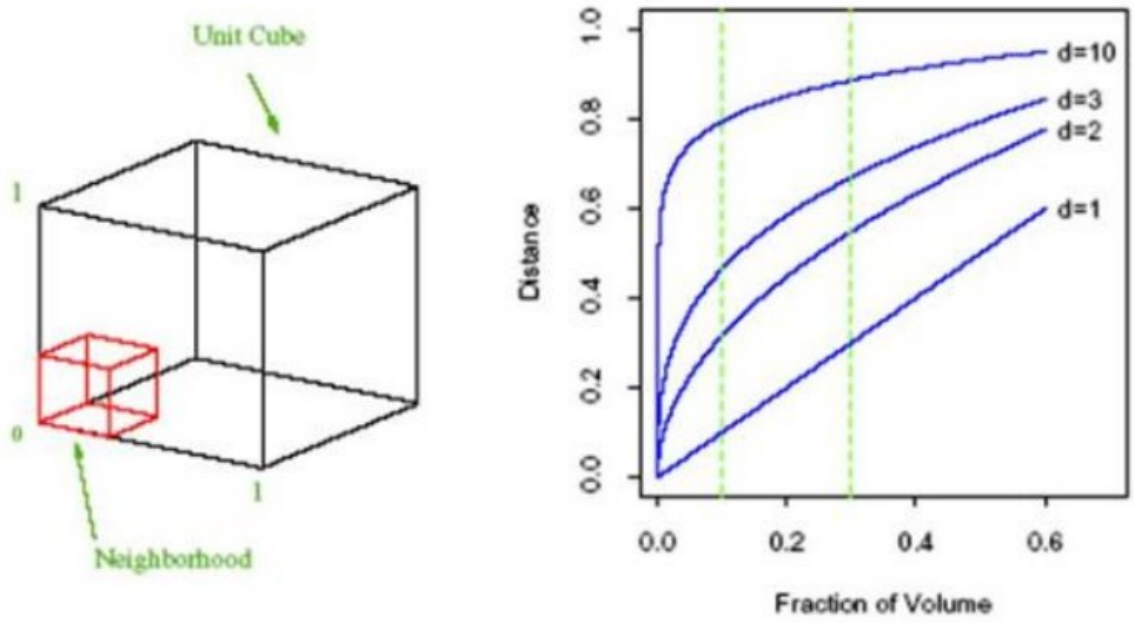
*4 On répète les deux dernières étapes jusqu' à ce que l'on ne puisse plus améliorer la modularité.

Proposition d'exercice : démontrer que cet algorithme a une complexité en O(nlogn).

Cette méthode est comme les autres, truffées de défauts , et le choix des paramètres de l'algorithme est très important. Il ne faut pas oublier qu'un algorithme est avant tout une proposition pour résoudre un problème. On peut l'adapter à des graphes orientés, non pondérés, optimiser son temps de calcul… Le côté séduisant de Louvain est que les arrêtes sont au coeur.

II) Considérations et problèmes

Le clustering semble être une tâche facile (intuitif), surtout si les objets sont de faible dimension et contiennent peu de points. Mais dès que le nombre de dimensions augmente, le résultat devient tout de suite difficilement interprétable, mais également beaucoup plus coûteux en données. En effet, plus la dimension augmente, plus on doit examiner de données pour trouver un voisinage autour du point.[10]



1. Définition des critères de similarité

Le fondement du clustering repose sur l'idée de regrouper des objets similaires. Cependant, définir la notion de "similarité" ou de "proximité" n'est pas toujours trivial, surtout quand on traite d'objets abstraits.. Cette définition dépend largement du domaine d'application et des données manipulées. C'est l'occasion de faire une piqûre de rappel sur les différentes métriques de similarité entre deux vecteurs[11].

On peut avoir des intuitions différentes de la similarité selon la façon dont on considère les clusters :

* Si on considère les clusters comme des points (interprétation géométrique du clustering, idée de centroïde) on utilise les distances classiques dans un espace euclidien, des mesures telles que la distance euclidienne

la distance de Manhattan peuvent suffire. Cf structure en grille des villes américaine, la distance n'est pas la diagonale mais la somme des distances sur chaque dimension du vecteur.

$$d(A,B) = |X_B - X_A| + |Y_B - Y_A|$$

10 https://web.stanford.edu/class/cs246/slides/05-clustering.pdf pour le graphique
11 https://medium.com/@gshriya195/top-5-distance-similarity-measures-implementation-in-machine-learning-1f68b9ecb0a3

En revanche, lorsqu'il s'agit de contenus plus complexes comme des textes ou des images, des mesures adaptées comme la similarité cosinus ou des métriques basées sur l'apprentissage automatique peuvent être nécessaires.

* Si on considère les clusters comme des vecteurs, on peut mesurer leur distance avec mesure de distance classique entre deux vecteurs, à savoir :

La distance euclidienne (calcul de la « diagonale » dans l'espace vectoriel) , encore appelée l2 norm

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

La similarité cosinus

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \ \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \ \sqrt{\sum_1^n b_i^2}}$$

* Enfin on peut considérer les cluster comme des ensembles et utiliser des métriques telles que la similarité de Jaccard qui sert à comparer deux ensembles en comparant leur « pourcentage d'intersection » par rapport au nombre total de points (ratio du cardinal de l'intersection sur le cardinal de l'union) .

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

Je vous fais grâce des dizaines d'autres normes qui existent (Cadek, Minkowski…) [12]mais il y a vraiment beaucoup de façons de mesurer la distance entre deux objets…
Il est également important de noter que le choix d'une métrique de similarité influence directement les résultats du clustering. Une mauvaise définition de la similarité peut entraîner des regroupements peu significatifs, voire inutiles. De plus, certains types de données, comme celles ayant une forte dimensionnalité, peuvent nécessiter des approches spécifiques pour éviter des problèmes comme la "malédiction de la dimensionnalité".

12 https://towardsdatascience.com/17-types-of-similarity-and-dissimilarity-measures-used-in-data-science-3eb914d2681 En voilà déjà 17. Qui dit mieux ?

2. La valse des algorithmes et des hyperparamètres

Comme nous l'avons vu, le choix de l'algorithme est déterminant dans la qualité du clustering. Si les clusterings par assignation de points sont très efficaces lorsque les clusters sont de forme convexe et que les points sont regroupés, si la forme du cluster est étrange, les méthodes par densité ou hiérarchie obtiennent souvent de meilleurs résultats.

La plupart des algorithmes de clustering nécessitent de définir un ensemble d'hyperparamètres qui influencent leur comportement. Par exemple, l'algorithme *k-means* requiert que l'on spécifie le nombre de clusters *k*. Pourtant, déterminer cette valeur n'est pas toujours évident. Des approches comme la méthode de l'élbow ou la silhouette score permettent de guider ce choix, mais elles ne garantissent pas toujours une segmentation optimale, surtout lorsque les données sont très hétérogènes.D'autres algorithmes, comme DBSCAN, nécessitent la définition de paramètres tels que la distance maximale entre deux points pour qu'ils soient considérés comme voisins (*eps*) et le nombre minimal de points dans un voisinage (*minPts*). Ces choix peuvent être délicats, car ils dépendent souvent des caractéristiques intrinsèques des données, parfois difficiles à anticiper sans exploration préalable.

En tout cas, là où la littérature s'accorde, c'est à dire qu'il n'y a pas de « recette miracle » pour déterminer quel algorithme et encore moins quel jeu de paramètre choisir par rapport à son problème. Il y a des tendances selon le cas d'usage (Louvain est mieux pour l'analyse de réseaux) ou le faciès de l'espace vectoriel qu'on veut traiter(K means pour le cas où les clusters sont assez bien délimités) mais pas de règle ou méthode définitoire. Ce qui rend les choses assez complexes car il faut un minimum de compétences en data science et une bonne compréhension des algorithmes pour pouvoir prendre en charge un clustering d'une base de données.

Hyperparamètre	K-Means	DBSCAN	Agglo. Clust.	Mean Shift	GMM	Spectral Clust.	Description
`n_clusters`	✅		✅		✅	✅	Nombre de clusters à former
`init`	✅						Méthode d'initialisation des centres (`k-means++` , `random`)
`max_iter`	✅			✅	✅		Nombre maximal d'itérations avant convergence
`tol`	✅						Seuil de tolérance pour l'arrêt de l'algorithme
`eps`		✅					Distance maximale entre deux points pour être voisins
`min_samples`		✅					Nombre minimal de points pour former un cluster
`metric`		✅	✅				Type de distance utilisée (`euclidean` , `manhattan` , etc.)
`linkage`			✅				Méthode de fusion des clusters (`ward` , `complete` , `average`)
`affinity`			✅			✅	Mesure de similarité utilisée (`euclidean` , `cosine` , etc.)
`bandwidth`				✅			Largeur de la fenêtre de recherche pour Mean Shift
`n_components`					✅		Nombre de gaussiennes dans le modèle
`covariance_type`					✅		Structure de la matrice de covariance (`full` , `diag` , etc.)
`n_neighbors`						✅	Nombre de voisins pour construire le graphe de similarité

Ce problème de paramétrage se rapproche au final de décisions d'architecture que l'on doit prendre au moment de créer un réseau de neurones pour traiter un problème de machine learning. Quelle architecture ? Combien de couches ? Quelle fonction de loss choisir ? Quels

hyperparamètres pour l'entraînement ? Ainsi, inspirés par les travaux sur la grid search en ML, certains ont proposé un approche similaire qui consiste à tester divers clusterings jusqu'à trouver le meilleur sur le dataset.[13] Cela nécessite de définir ce qu'est un bon clustering (et est souvent fait selon la définition canonique : minimisation de la distance intra cluster, maximisation de la distance inter cluster, avec quelques choix d'heuristiques). A noter de plus qu'un clustering mathématiquement optimal sur le dataset n'est pas forcément un clustering opérationnellement optimal et que tout dépend vraiment du cas d'application. En bref, choisir la méthode de clustering reste encore un grand casse tête.

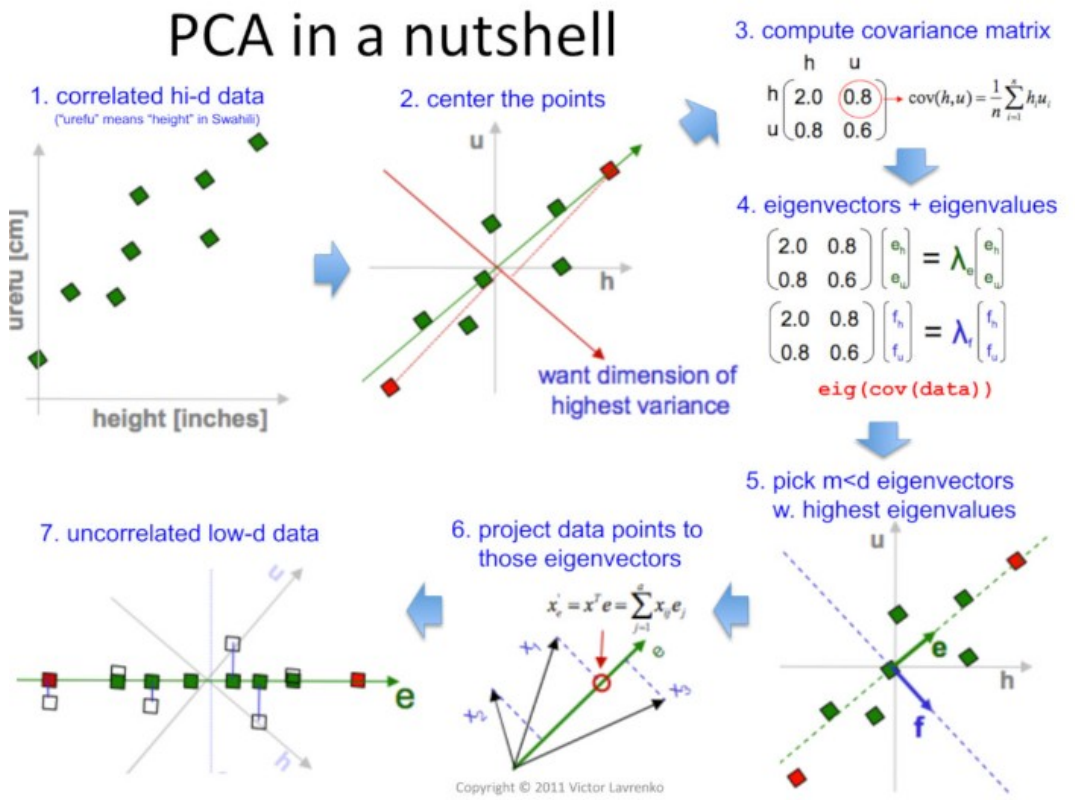
3. Représentation des données et interprétabilité

La qualité du clustering dépend également de la manière dont les objets sont représentés. Une représentation inadaptée peut réduire considérablement la capacité des algorithmes à identifier des regroupements pertinents. Par exemple, lorsque l'on manipule des données textuelles, la transformation des textes en vecteurs via des approches comme TF-IDF ou des embeddings sémantiques (par exemple, *word2vec*, *BERT*) joue un rôle crucial dans le succès de la segmentation. De plus, dans certains cas, les données brutes doivent être prétraitées pour être utiles au clustering. Cela peut inclure des étapes comme la normalisation, le traitement des valeurs manquantes. Ces opérations peuvent non seulement améliorer la qualité des clusters, mais aussi rendre les résultats plus faciles à interpréter.

Une autre difficulté de l'application du clustering est sa difficile interprétation. Comment vérifier à l'oeuil nu que un clustering fait sur des vecteurs à 256 dimensions est pertinent et significatif ? Pour pouvoir mieux interpréter le résultat d'un clustering, on effectue souvent une étape supplémentaire qui s'appelle la réduction de dimensionnalité : On transforme un vecteur de très grande taille en un vecteur de taille plus petite sans que cela perturbe la distance relative des vecteurs les uns avec les autres. Cela permet également de réduire l'espace de stockage nécessaire aux vecteurs, mais aussi les coûts de calcul des algorithmes ! Si cette réduction de dimension est faite en deux ou trois dimensions, le clustering devient alors visuellement interprétable. La visualisation et l'interprétabilité des données sont d'ailleurs les champs d'application principaux de ces techniques mathématiques.

13 https://www.sciencedirect.com/science/article/pii/S2590005622000406 par exemple.

Il existe pour se faire plusieurs méthodes, dont les plus célèbres sont UMAP et PCA[14] (Principal Components Analysis) que l'on va présenter briévement. Cette méthode statistique transforme un ensemble de données multivariées en un nouvel espace où les dimensions (ou "composantes principales") sont non corrélées et ordonnées par leur importance. Mathématiquement, cela revient à trouver une nouvelle base vectorielle dans laquelle la variance des données est maximisée, tout en minimisant la redondance.



Le tout est très algébrique. Pour y parvenir, on commence par centrer les données autour de leur moyenne pour garantir que l'origine du nouvel espace est au centre des données. Ensuite, on calcule la matrice de covariance des données, qui capture les relations entre les dimensions. À partir de cette matrice, on effectue une décomposition en valeurs propres : les valeurs propres (qui représentent la variance expliquée par chaque direction) et leurs vecteurs propres associés (qui donnent les directions des nouvelles composantes) sont extraits. Les vecteurs propres associés aux plus grandes valeurs propres forment les nouvelles dimensions principales, car elles expliquent la majorité de la variance. Les données initiales sont ensuite projetées dans cet espace réduit, permettant une simplification tout en conservant au maximum l'information essentielle.[15]

En résumé l'intuition est de regarder quels composants du vecteur ou quelle combinaison linéaire de composants du vecteurs sont les plus informatifs.

Projeter ses vecteurs en 2D avant un clustering permet d'avoir un clustering interprétable à l'oeil nu, bien qu'il y ait un risque qu'une forte réduction de dimensionnalité ne dégrade la précision. Il y a donc un trade of entre performance et interprétabilité de tous ces algorithmes.

14 https://arxiv.org/pdf/1804.02502
15 https://devopedia.org/principal-component-analysis pour l illustration.

III) Applications

Même si l'on en entend bien moins parler que du deep learning, les algorithmes de clustering sont parmi les techniques d'apprentissage non supervisé les plus fréquemment utilisées et ont reçu un vaste panel d'applications de la plus théorique à la plus pratique. On verra théoriquement comment on peut se servir d'un clustering pour effectuer d'autres tâches avant de voir l'usage des clustering en NLP et étudier le cas particulier du topic modelling.

a) Un vaste panel applicatif

Les techniques de clustering trouvent des applications dans un large éventail de domaines, grâce à leur capacité à extraire des connaissances utiles à partir de vastes ensembles de données. Voici quelques détournements techniques, ou comment les algorithmes sont détournés à d'autres usages :

1. Analyse exploratoire et réduction de la complexité des données.

Le clustering est particulièrement utile pour obtenir une vue d'ensemble sur des ensembles de données massifs. Par exemple, dans une base de clients, segmenter les individus en groupes basés sur leurs comportements d'achat peut révéler des catégories homogènes, comme les acheteurs fréquents, les clients saisonniers ou les consommateurs occasionnels. On regroupe des individus objets selon leur comportements habituels ce qui peut aider à prédire des évolutions communes.

2. Détection d'anomalies

Une autre application majeure du clustering est la détection des observations qui s'écartent de la norme. En regroupant les données similaires, les points qui ne s'intègrent pas bien dans les clusters formés peuvent être identifiés comme des anomalies. Cela est particulièrement utile dans des domaines comme la cybersécurité, où ces anomalies pourraient correspondre à des intrusions ou des comportements suspects, ou dans la finance pour identifier des fraudes potentielles.

3. Compression et discrétisation des données

Dans le traitement de données volumineuses, il peut être bénéfique de réduire la taille d'un dataset tout en conservant une représentation fidèle de sa diversité. Par exemple, en regroupant les données en clusters et en sélectionnant un représentant par groupe, il est possible de travailler avec une version condensée de la base, ce qui réduit les coûts de calcul pour des analyses ultérieures.

4. Génération de modèles supervisés à moindre coût

Le clustering peut également servir d'étape préliminaire pour construire des modèles supervisés. Par exemple, en utilisant les clusters comme labels provisoires, il devient possible de générer un jeu de données annoté à moindre coût. Cela peut être particulièrement utile lorsqu'il est difficile ou coûteux d'obtenir des annotations manuelles.

Pour construire un pseudo système de classification à partir d'un faible jeu de données annotées (par exemple analyse de sentiments ou catégorisation de documents) : on réalise un clustering en s'assurant que les objets appartenant à la même catégorie se retrouvent dans des

clusters homogènes (on évite à tout prix que des observations avec des labels différents se retrouvent dans le même cluster). Pour catégoriser un nouvel objet, on s'inspire du fonctionnement des algorithmes de clustering basés sur la géométrie : l'inférence est aussi rapide, on passe par le modèle d'embedding puis on assigne le nouveau point au cluster dont il est le plus proche (interprétation géométriqe) et on attribue au point la catégorie des membres de sont clusters.

C'est une technique entièrement non supervisée (suppression de la phase d'entraînement) qui, de façon surprenante obtient souvent des résultats aussi satisfaisants que des modèles d'IA pour une fraction du coût de développement (nécessite un embedding, figé ou contextuel) à partir d'un modèle, puis au lieu de l'entraînement l'application d'un algorithme de clustering (en général peu coûteux vu la taille des datasets). On est bien sûr pas à l'abri d'erreur dus au cas limites (élément à mi chemin entre deux catégories mais géométriquement plus près de la mauvaise, outliers…)

Ainsi, le clustering, bien que conceptuellement simple, se révèle être un outil polyvalent aux applications multiples, offrant des solutions à de nombreux problèmes complexes d'analyse et de compréhension des données.

b) Un cas d'usage particulier en NLP : le topic modelling

Le topic modeling est une branche clé du traitement automatique du langage naturel (NLP) qui vise à identifier les thèmes principaux présents dans une collection de documents. Cet outil permet d'obtenir rapidement une vision globale du contenu d'une base de données textuelle, en extrayant les thématiques saillantes. Initialement développé avec des techniques statistiques classiques, le topic modeling s'est enrichi grâce aux avancées en science des données, au machine learning et, plus récemment, à l'arrivée des modèles de langage de grande taille (LLMs). Aujourd'hui, le topic modeling constitue un véritable framework modulaire, combinant diverses approches et outils.

La démarche du topic modeling peut être décomposée en plusieurs étapes principales :

1. Représentation vectorielle des documents

La première étape consiste à convertir chaque document textuel en une représentation vectorielle. Cette transformation peut être effectuée par des méthodes statistiques classiques, telles que le Bag of Words (BOW) ou le TF-IDF, qui capturent les fréquences ou l'importance des termes dans les documents. Toutefois, avec l'émergence des modèles de fondation, il est désormais courant d'utiliser des embeddings contextuels générés par des modèles comme BERT ou ses dérivés. Ces techniques modernes permettent de représenter les documents dans des espaces vectoriels riches en informations sémantiques, capturant non seulement les mots eux-mêmes, mais aussi leurs relations contextuelles.

2. Regroupement des représentations similaires

Une fois les documents représentés sous forme vectorielle, la deuxième étape consiste à regrouper les documents similaires en fonction de leurs caractéristiques communes. Différentes approches peuvent être employées :

*** Latent Dirichlet Allocation (LDA)**

LDA repose sur une technique d'inférence bayésienne et considère qu'un document est composé d'une distribution de topics, et qu'un topic est lui-même constitué d'une distribution de mots. Cet algorithme identifie ces distributions en s'appuyant sur les co-occurrences des mots dans les documents. Bien qu'efficace, LDA est non déterministe : les résultats peuvent varier d'un run à l'autre.

* **Non-Negative Matrix Factorization (NMF)**

NMF utilise une décomposition matricielle pour modéliser les relations entre les documents et les topics. L'hypothèse centrale est qu'un document peut être représenté comme une combinaison linéaire de topics, et que chaque topic peut être décrit comme une combinaison linéaire de termes. Ce modèle produit une matrice où les lignes correspondent aux topics et les colonnes aux documents.

* **BERTTopic**

BERTTopic illustre l'intégration des modèles de fondation dans le topic modeling. Cette méthode exploite les embeddings contextuels produits par BERT ou des modèles similaires pour regrouper les documents selon leur contenu sémantique. Le processus comprend plusieurs étapes :

1. **Conversion des documents en vecteurs** : Chaque document est encodé en un vecteur dense à l'aide de modèles comme Sentence Transformers.
2. **Réduction de la dimensionnalité** : Pour faciliter le clustering, les dimensions des vecteurs sont réduites (par exemple, avec des techniques comme UMAP).
3. **Clustering des vecteurs** : Les vecteurs ainsi réduits sont regroupés à l'aide d'algorithmes de clustering, tels que HDBSCAN, qui identifie les clusters de documents similaires.
4. **Extraction des topics** : Une variante du TF-IDF, appelée Class-Based TF-IDF, est utilisée pour extraire les termes les plus représentatifs de chaque cluster. Ces termes permettent de définir les topics. Les distributions de topics par document et par cluster sont ensuite fusionnées pour produire la liste finale des thématiques identifiées.

Les algorithmes de clustering jouent un rôle essentiel dans des frameworks modernes comme BERTTopic, où ils servent à structurer les représentations sémantiques en groupes cohérents. Ces clusters deviennent alors une base solide pour extraire des topics précis et interprétables. En combinant des embeddings contextuels puissants, une réduction dimensionnelle efficace et des algorithmes de regroupement performants, le topic modeling moderne offre une analyse riche et adaptable, capable de traiter des corpus de grande taille tout en capturant des thématiques complexes.

Le topic modeling classique avait pour objectif principal de déterminer les topics d'un corpus, ce qui, en pratique, revient souvent à une tâche d'extraction de mots-clés, mais appliquée à l'ensemble d'une base de données. Cependant, les récents progrès dans l'intelligence artificielle, et en particulier l'apparition des modèles de langage de grande taille (LLMs), ont radicalement enrichi les capacités du topic modeling, ouvrant de nouvelles perspectives.Les modèles modernes permettent d'aller bien au-delà de l'identification basique des topics en intégrant des approches innovantes :

Définition des topics via les embeddings

Les embeddings associés aux topics offrent une représentation dense et sémantique qui peut être utilisée pour extraire les extraits de texte les plus représentatifs où ces topics apparaissent. Ces extraits sont ensuite transformés en définitions claires grâce à l'utilisation des LLMs. Cela offre une interprétation enrichie et directement utilisable des topics identifiés, facilitant la compréhension du contenu d'un corpus.

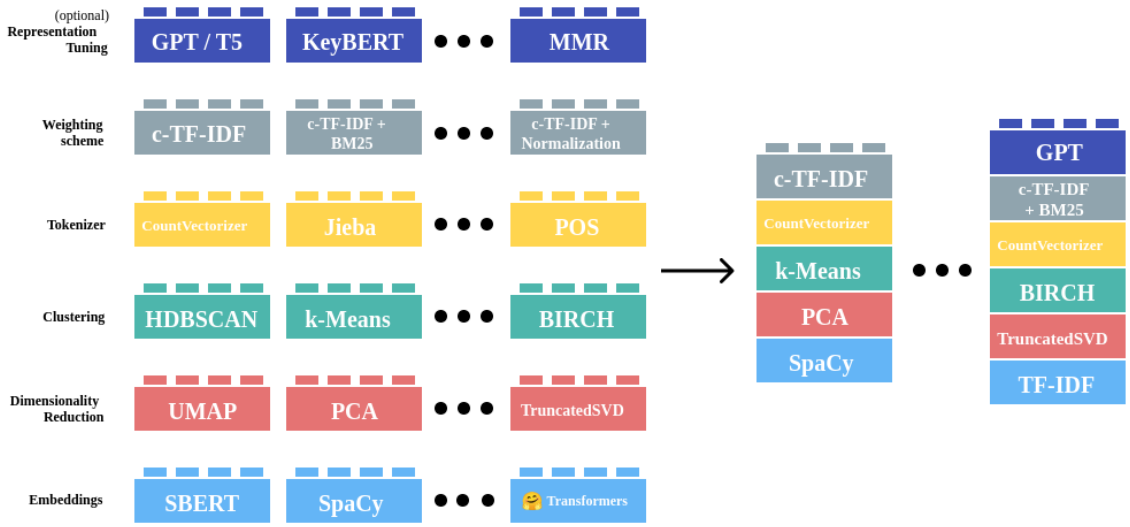
Résumé des clusters

Les LLMs peuvent également être exploités pour résumer le contenu de chaque cluster de documents :

- Une **approche par Map-Reduce** peut être utilisée pour résumer les documents à plusieurs niveaux : les documents individuels d'abord, puis les clusters en tant qu'ensemble.
- Alternativement, un **résumé basé sur les documents représentatifs** de chaque cluster permet de produire une synthèse concise et pertinente.

Ces fonctionnalités rendent désormais possible une représentation organisée et structurée d'une base de connaissances. Les données deviennent plus exploitables, facilitant leur transformation et regroupement, et ouvrant la voie à de nouvelles perspectives d'analyse.

Un exemple clé de cette évolution est la librairie **BERTTopic**, créée par Martin Grootendorst. Initialement développée dans le cadre d'un papier de recherche, BERTTopic a progressivement évolué pour devenir une solution modulaire et polyvalente, permettant de combiner différentes approches algorithmiques pour le topic modeling.[16]



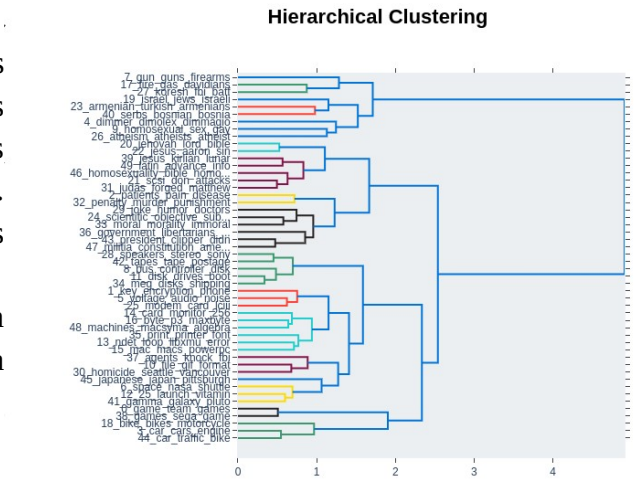
BERTTopic offre une :

- **Flexibilité algorithmique** : L'utilisateur peut choisir parmi plusieurs modèles d'embeddings et algorithmes de clustering pour extraire les topics de manière adaptée à ses données.
- **Extraction hiérarchique des topics** : BERTTopic permet de créer des représentations hiérarchiques, facilitant la compréhension des relations entre les topics.
- **Modularité** : La librairie est extensible, permettant d'expérimenter avec des méthodes variées et d'adapter le pipeline aux besoins spécifiques de l'utilisateur.

16 https://maartengr.github.io/BERTopic/index.html

BERTTopic intègre également des outils de visualisation puissants, facilitant l'exploration des résultats :

1. **Matrices de similarité entre topics** : Ces matrices permettent d'identifier les relations sémantiques entre topics, notamment les synonymes ou des relations de type « is a ». Cela peut enrichir les connaissances dérivées du corpus.
2. **Hiérarchie des topics** : Une visualisation hiérarchique aide à mieux comprendre la structure des topics et leurs liens contextuels.



TODO faire exemple de code avec BERTTopic sur le corpus lotr

Les problématiques du topic modeling s'inscrivent pleinement dans celles de la construction automatique d'ontologies à partir de corpus documentaires, tant par leurs objectifs que par les outils techniques employés. Le topic modeling vise à identifier des thèmes récurrents dans une base de données textuelle, à en analyser les structures et à en tirer des représentations organisées. Ces thèmes, ou topics, peuvent être interprétés comme des prémices d'une ontologie, dans laquelle ils apparaissent soit comme des catégories abstraites englobant plusieurs concepts, soit comme des instances spécifiques à intégrer à l'ontologie. La capacité du topic modeling à dégager des relations implicites ou explicites entre différents topics renforce cette correspondance.

Par ailleurs, les approches modernes, comme celles proposées par BERTTopic, introduisent une dimension hiérarchique dans la structuration des topics. Cette hiérarchie permet non seulement de regrouper les documents autour de thématiques centrales, mais aussi de révéler les relations de subordination ou de dépendance entre ces thématiques. Cette organisation hiérarchique préfigure directement la structuration conceptuelle d'une ontologie, où les concepts sont liés entre eux par des relations de type « est un » ou « partie de ». Ainsi, les clusters et les topics identifiés peuvent être vus comme des premiers niveaux d'abstraction dans la modélisation des concepts du domaine.

Le rapprochement entre topic modeling et ontologies est également renforcé par l'utilisation d'outils similaires. Les embeddings, les algorithmes de clustering et les modèles d'inférence probabiliste, comme LDA ou NMF, participent à la fois à la structuration thématique des données et à l'identification des relations conceptuelles. À cela s'ajoutent les avancées permises par les LLMs, qui offrent des capacités inédites d'interprétation, de synthèse et de contextualisation, permettant d'associer à chaque topic une définition ou un extrait textuel représentatif, utile dans le cadre d'une ontologie. Ces outils permettent donc de passer d'une analyse exploratoire de corpus à une structuration plus formelle et exploitable des connaissances, unifiant ainsi les objectifs du topic modeling et de la construction automatique d'ontologies.