Projet de Visualisation d'Information

Analyse critique d'algorithme de réduction de dimension : UMAP

UE DEA
Master Bio-informatique
Université de Bordeaux
Novembre 2023

Auteurs :

Ait Abed Sara, Alkharrat Arij, Cros Esther & Guiberteau Yaëlle

Table des matières

1 Introduction

La réduction de dimensionnalité fait référence à la réduction de données de grande dimension en des données localisées dans une espace de plus petite dimension.

Ce terme désigne une opération importante dans de nombreux domaines et notamment dans celui de l'apprentissage automatique. L'apprentissage automatique est un champ d'étude de l'intelligence artificielle qui vise à donner aux machines la capacité d'« apprendre » à partir de données, via des modèles mathématiques. Ainsi, la réduction de dimensionnalité intervient dans ce contexte afin de réduire la complexité et d'améliorer les propriétés de stabilité et de robustesse des algorithmes [1]. Cette étape permet également de réduire l'espace de stockage nécessaire et de fluidifier les calculs.

L'emploi de divers algorithmes nécessitant le traitement de jeux de données massifs et complexes est de plus en plus commun. Dans ce sens, ce projet à pour but de comprendre, d'expliquer et de proposer une analyse critique d'un algorithme de réduction de dimension. Notre étude portera sur l'algorithme UMAP (Uniform Manifold Approximation and Projection) [2].

2 Présentation de UMAP

2.1 Fonctionnement

UMAP, pour *Uniform Manifold Approximation and Projection*, est un algorithme de réduction de dimension non linéaire présenté par McInnes et al. en 2018 [3]. Il cherche à apprendre la structure de variété (manifold en anglais) des données et à trouver une intégration de faible dimension qui préserve au mieux la topologie de cette variété. En mathématiques, une variété désigne un espace topologique abstrait, construit par recollement d'autres espaces simples.

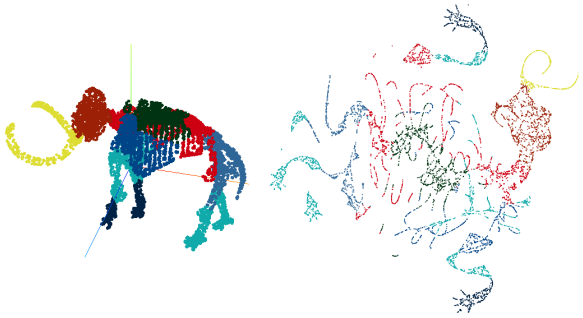

Figure 1 : Intégration UMAP d'un squelette de mammouth laineux en 3 dimensions dans un espaces en 2 dimensions [4].

Sommairement, UMAP construit d'abord un graphe en haute dimension qui rend compte de la structure de variété des données. Puis, il optimise un graphe à basse dimension pour que sa structure soit aussi similaire que possible et que les relations locales et globales entre les points soient préservées.

2.1.1 Construction d'un graphe en haute dimension

Le graphe initial en haute dimension est construit grâce à un complexe simplicial flou [5]. Pour comprendre ce qu'est un complexe simplicial, il faut au préalable définir ce qu'est un k-simplexe. Un k-simplexe est un objet k-dimensionnel formé en prenant la coque convexe de k+1 points indépendants. Une coque convexe est le plus petit polygone convexe entourant un ensemble de points. Un 0-simplexe est donc un point, un 1-simplexe est un segment de ligne, un 2-simplexe est un triangle et un 3-simplexe est un tétraèdre.

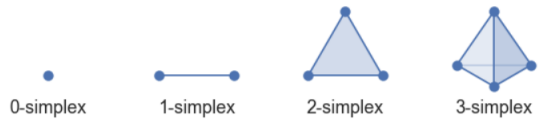

Figure 2 : Simplexes de dimension réduite.

Les k-simplexes sont les éléments de construction d'un complexe simplicial. En effet, un complexe simplicial est un ensemble de simplexes de différentes dimensions collés les uns aux

autres le long des faces. Ces éléments sont assemblés à partir d'une couverture ouverte de la variété. Une couverture ouverte correspond à une famille d'ensembles d'un espace topologique dont l'union contient un sous-ensemble donné. Dans un espace métrique, une couverture ouverte revient simplement à créer des sphères d'un rayon fixe autour de chaque point de données. Un simplexe est créé lorsque les rayons des points se chevauchent.

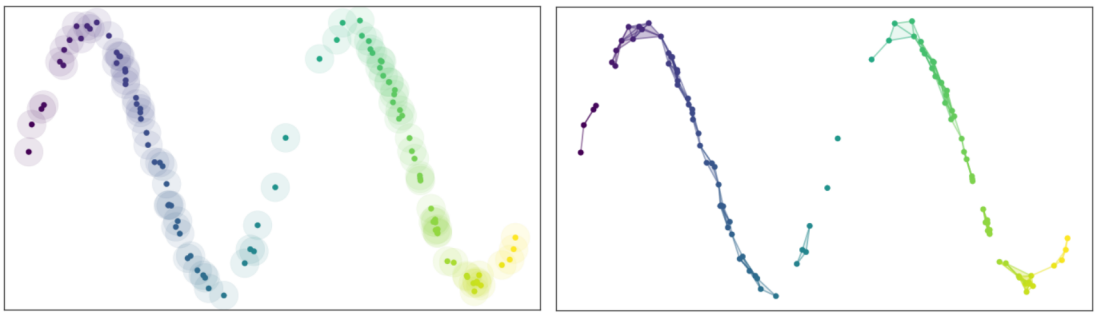

Figure 3 : A gauche, couverture ouverte de données distribuées sur une courbe sinusoïdale avec du bruit en 2 dimensions. A droite, complexe simplicial résultant de cette couverture ouverte.

La majeure partie de la variété est réellement capturée par les 0-simplexes et les 1-simplexes. Ainsi, la représentation topologique de la variété n'est en réalité qu'un graphe (combinaison de nœuds et d'arêtes) et la recherche d'une représentation de faible dimension revient à un problème d'agencement de graphe.

Toutefois, un problème se pose lors du choix du rayon des sphères de la couverture ouverte. Si le rayon est trop petit, le complexe simplicial est divisé en de nombreuses composantes connexes. Si le rayon est trop grand, le complexe simplicial se compose de quelques simplices de très haute dimension qui ne capturent pas la structure de la variété. En fait, cette théorie fonctionne correctement seulement si les données sont uniformément réparties sur la variété. Il suffirait alors de prendre la distance moyenne entre les points comme rayon. Or, dans la réalité, les données ne sont pas toujours uniformément distribuées et la notion de distance peut varier sur la variété.

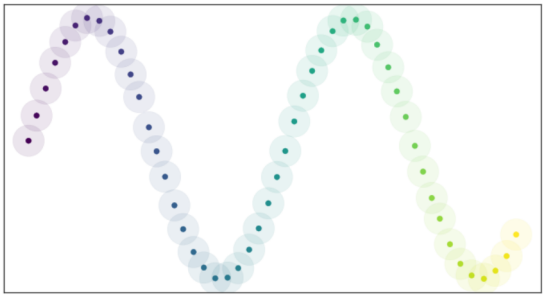

Figure 4 : Couverture ouverte données uniformément distribuées sur une courbe sinusoïdale en 2 dimensions.

UMAP pallie à ce problème en remplaçant l'utilisation de sphères de rayon fixe par des sphères dont le rayon est défini par la distance locale. Cette distance locale est calculée pour chaque point en utilisant la géométrie riemannienne : la distance locale est approximée en

utilisant les k voisins les plus proches d'un point. Le choix du nombre de voisins k est crucial. Un petit k donne une estimation très locale de la métrique riemannienne tandis qu'un grand k donne des estimations basées sur des régions plus larges. Ceci est cohérent avec la théorie des graphes qui définit la connectivité avec une approche de k plus proches voisins.

Cette approche basée sur la métrique riemannienne nous permet alors d'avoir une couverture ouverte floue : l'appartenance d'un point à cette couverture n'est plus binaire mais plutôt une valeur continue entre zéro et un. Ainsi, la probabilité de présence d'un point dans une sphère d'un rayon donné diminue à mesure que l'on s'éloigne du centre de la sphère.

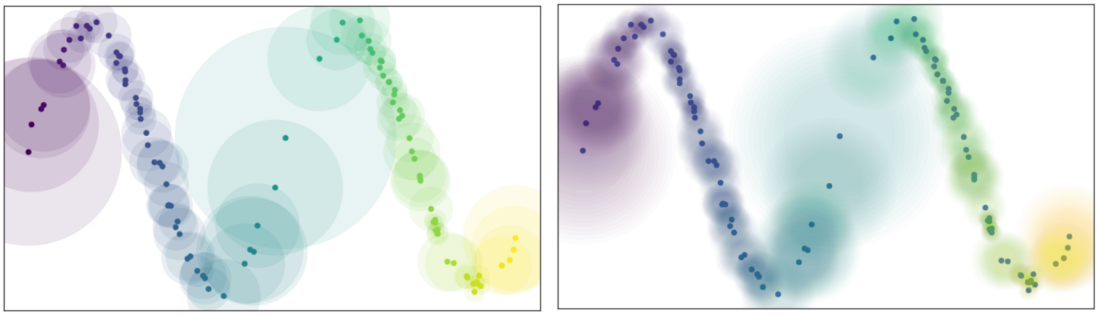

Figure 5 : A gauche, couverture ouverte avec un rayon déterminé par une métrique locale à chaque point dans un espace en 2 dimensions. A droite, couverture ouverte floue correspondante.

Un nouveau problème survient : les métriques locales entre les points sont incompatibles car chaque point a sa propre métrique locale. En effet, en partant du point a, la distance entre le point a et le point b n'est pas forcément égale à la distance entre le point b et le point a, en partant du point b. Pour déterminer quelle est la bonne distance, il faut se rappeler que le complexe simplicial flou n'est autre qu'un graphe avec des arêtes dirigées entre les points. Les arêtes portent des poids variables correspondant à la distance entre les points. Ainsi, entre deux points, il peut y avoir jusqu'à deux arêtes avec des poids incompatibles a et b. Afin de fusionner ces deux arêtes en une seule, on calcule le poids combiné $a + b - ab$. Les poids a et b représentent la probabilité qu'une arête existe et le poids combiné est alors la probabilité qu'au moins une des deux arêtes existe. Le graphe pondéré capturant la topologie de la variété associée des données en haute dimension est enfin construit.

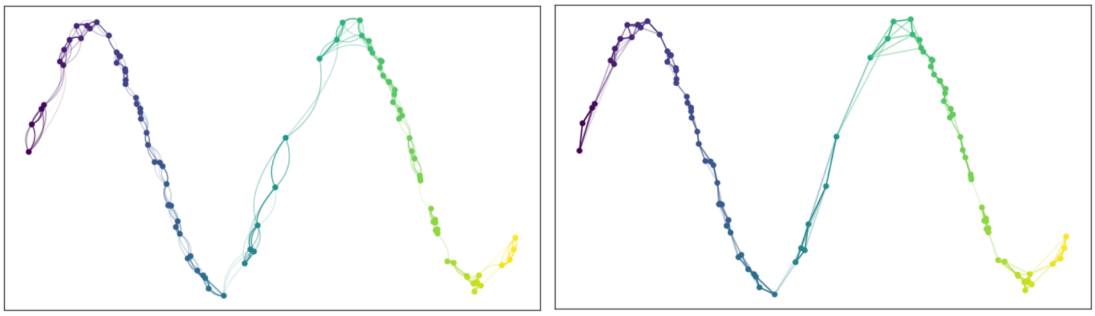

Figure 6 : A gauche, graphe avec des arêtes dont les poids sont incompatibles (dans un espace en 2 dimensions). A droite, graphe correspondant avec des arêtes fusionnées portant des poids combinés.

2.1.2 Optimisation d'un graphe en dimension réduite

Par la suite, il faut convertir le graphe en haute dimension en une représentation de dimension réduite ayant une structure topologique aussi similaire que possible. La première étape est de déterminer la structure du graphe en dimension réduite. La deuxième étape est de l'optimiser.

Pour la première étape, l'intuition serait de suivre la même procédure que celle précédemment utilisée pour trouver la structure de la variété de nos données en haute dimension. Cependant, la représentation en dimension réduite a une variété très particulière : c'est un espace euclidien de faible dimension dans lequel les données seront insérées. Aussi, la distance sur cette variété est une distance euclidienne standard et non une métrique variable. De plus, la propriété de la distance avec le voisin le plus proche utilisée précédemment doit être globalement vraie à travers la variété en dimension réduite.

La deuxième étape correspond à l'optimisation de la représentation en dimension réduite et est réalisée en comparant les structures topologiques des représentations en haute dimension et en dimension réduite. Précédemment, lors de la fusion de poids différents associés aux simplexes (arêtes), les poids représentaient la probabilité du simplexe existant. Ainsi, puisque les deux structures topologiques à comparer partagent les mêmes 0-simplexes (points), il suffit de comparer les deux vecteurs de probabilités (poids) des 1-simplexes. Ces 1-simplexes étant des variables de Bernoulli (le simplexe existe ou non) et la probabilité étant le paramètre d'une distribution de Bernoulli, l'entropie croisée peut être calculée.

Explicitement, si l'ensemble de tous les 1-simplexes possibles est E et que nous avons des fonctions de poids telles que $w_h(e)$ est le poids du 1-simplexe e en haute dimension et $w_l(e)$ est le poids de e en dimension réduite, alors l'entropie croisée sera :

$$\sum_{e \in E} \left(w_h(e) \log \left(\frac{w_h(e)}{w_l(e)} \right) + (1 - w_h(e)) \log \left(\frac{1 - w_h(e)}{1 - w_l(e)} \right) \right)$$

Le but est alors de minimiser l'entropie croisée. Le premier terme traduit une force attractive entre les points que e relie lorsque $w_h(e)$ est grand. Ce terme est minimisé lorsque $w_l(e)$ est grand. A l'inverse, le deuxième terme traduit une force répulsive entre les points que e relie lorsque que $w_h(e)$ est petit. Ce terme est minimisé lorsque $w_l(e)$ est petit. L'entropie croisée est donc une fonction de coût optimisée par une descente de gradient stochastique. Le processus d'attraction/répulsion, reflété par les poids sur les arêtes du graphe des données en grande dimension, fait que l'intégration de la représentation en dimension réduite rend compte relativement précisément de la topologie globale des données d'origine.

UMAP est un outil très puissant et offre plusieurs avantages par rapport à t-SNE bien que les deux algorithmes soient assez similaires. En effet, UMAP fait preuve d'une vitesse accrue et d'une meilleure préservation de la structure globale des données.

2.2 Impact des différents paramètres

Plusieurs hyperparamètres permettent de configurer l'algorithme de UMAP. Ceux-ci peuvent avoir un impact significatif et distinct sur le résultat de l'intégration dans l'espace réduit. Les quatre principaux hyperparamètres sont : `n_neighbors`, `min_dist`, `n_components` et `metric`.

2.2.1 `n_neighbors`

Ce paramètre correspond au nombre de points voisins les plus proches utilisés pour construire le graphe initial en haute dimension. Il influence la manière dont UMAP fait un compromis entre les structures locale et globale des données. Pour ce faire, `n_neighbors` limite la taille du voisinage local que UMAP examine lors de l'apprentissage de la structure de la variété des données en haute dimension. Avec des valeurs faibles, UMAP se concentre sur un voisinage resserré de chaque point et rend compte de la structure très locale et ce au détriment de la vue d'ensemble. Avec des valeurs élevées, UMAP examine des voisinages plus larges de chaque point et représente donc la structure globale tout en perdant les détails les plus fins. La valeur par défaut de `n_neighbors` est 15.

2.2.2 `min_dist`

Le paramètre `min_dist` équivaut à la distance minimale entre les points dans l'espace de dimension réduite. Il contrôle le degré d'intégration de regroupement des points par UMAP. Avec des valeurs faibles, on obtient une intégration plus groupée, ce qui peut être utile si on veut conserver une structure topologique plus fine. Avec des valeurs plus élevées, on a une dispersion plus uniforme des points ce qui permet de préserver la structure topologique générale. La valeur par défaut de `min_dist` est 0.1.

2.2.3 `n_components`

Ce paramètre correspond à la dimension de l'espace réduit dans lequel les données seront intégrées. La valeur par défaut de `n_components` est 2.

2.2.4 `metric`

Le paramètre `metric` définit la manière dont la distance est calculée entre les points dans l'espace de haute dimension. UMAP peut fonctionner avec des métriques diverses : des métriques de type Minkowski, des métriques spatiales (normalisées ou non), des métriques angulaires et de corrélation ou des métriques pour les données binaires. De plus, UMAP accepte aussi les métriques personnalisées (compilées en mode nopython avec Numba) définies par l'utilisateur. La valeur par défaut de `metric` est "euclidean".

3 Implémentation

3.1 Générateur de données

Dans le but de faire une analyse quantitative de l'algorithme UMAP, le groupe a commencé par implémenter des générateurs de données en haute dimensions. Cette étape permet d'obtenir des données synthétiques utilisées pour tester l'algorithme étudié. A ce stade, l'objectif est de concevoir des données semblables à des données réelles et proposant une certaine diversité. Afin d'imiter cela nous nous sommes basées sur trois distributions.

La première distribution est la distribution uniforme. Celle-ci, selon les paramètres et les caractéristiques données, va former une distribution de valeurs uniformes dans un intervalle [0,1]. Cette distribution est implémentée grâce à la fonction `random.uniform` de numpy. La deuxième distribution est la distribution gaussienne, le générateur crée des points symétriques par rapport à une moyenne. La distribution gaussienne a pour particularité de favoriser les valeurs centrales au détriment des valeurs extrèmes. Celle-ci à une forme de cloche qui est déterminée par la moyenne et l'écart type. La troisième distribution implémentée correspond à une distribution gaussienne regroupée en communautés. On retrouve les mêmes paramètres que précédemment avec en plus le nombre de communautés. Ces deux dernières distributions ont été implémentées avec la fonction `make_blobs` de scikit-learn.

Les générateurs de données sont implémentés dans deux fichiers : `Generateur_Uniform.py` pour la distribution uniforme et `Generateur_Gaussian_Cluster.py` pour les distributions gaussienne et en communautés. Ces fichiers se décomposent tous deux en plusieurs fonctions. Une fonction `data_generation` génère les données synthétiques. Pour les distributions gaussienne et en communautés, les données sont normalisées avec la fonction `MinMaxScaler()` de scikit-learn qui permet d'avoir des valeurs dans l'intervalle [0,1]. Cela est fait afin de faciliter l'analyse ultérieure. La fonction `parameters_testing` fait appel aux fonctions précédentes et a deux objectifs. Elle permet d'abord de créer de la répétition dans les jeux de données puis de donner une nouvelle combinaison de paramètres à chaque itération de la fonction. La fonction `file_names` sauvegarde les données générées dans des fichiers .csv. Elle permet d'attribuer des noms de fichiers spécifiques à chaque combinaison de paramètres utilisés pour la génération et de sauvegarder les fichiers dans les répertoires correspondants. Des noms de colonnes sont pré-établis par la fonction `columns_names` afin de faciliter le traitement ultérieur des fichiers.

Nous allons à présent accorder une attention particulière aux divers paramètres d'entrée des générateurs de données. La distribution uniforme utilise quatre paramètres. Le paramètre *low* spécifie la borne inférieure de l'intervalle de valeurs des données et est fixé à 0. Le paramètre *high* quant à lui correspond à la borne supérieure et est fixé à 1. Le paramètre *size* comprend deux sous-paramètres : *n_samples* qui correspond au nombre d'échantillons générés et *n_features* qui correspond au nombre de caractéristiques par échantillon. Les deux autres distributions sont générées par la même fonction et ont donc les mêmes paramètres. Le paramètre *n_samples* spécifie le nombre total de points générés (répartis équitablement entre les communautés lorsqu'il y en a plusieurs). Le paramètre *n_features* correspond au nombre

de caractéristiques pour chaque échantillon. Le paramètre *centers* donne le nombre de centres et donc de communautés à générer. Enfin, le paramètre *cluster_std* précise l'écart-type des communautés. Le paramètre *random_state* détermine la génération de nombres aléatoires et est fixé à 0. La particularité de la distribution en communautés se trouve dans le nombre de centre : si ce paramètre vaut 1, alors on a une distribution gaussienne.

Le produit cartésien de ces paramètres fait que nous obtenons 12 500 fichiers au total dont 7500 pour la distribution en communautés, 3 750 pour la distribution gaussienne et 1250 pour la distribution uniforme.

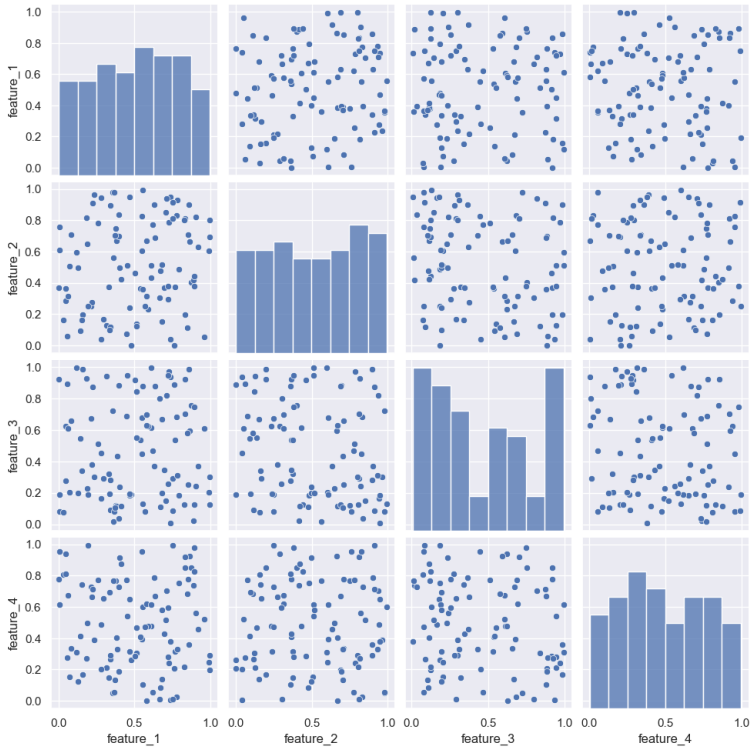

Figure 7 : Pairplot de 100 données avec 4 caractéristiques distribuées uniformément.

3.2 UMAP

Dans le fichier `UMAP.py`, la fonction `UMAP_dimension_reduction` applique l'algorithme de UMAP aux données précédemment générées en combinant diverses autres fonctions. Ces données étant enregistrées dans des fichiers .csv, le script va itérer sur ces fichiers dont les noms ont été récupérés avec la fonction `get_files`. Pour chaque fichier, son contenu est d'abord chargé dans une dataframe pandas. Puis pour toutes les combinaisons de paramètres de UMAP possibles, on lance la dimension de réduction UMAP sur la dataframe des données en haute dimension avec la fonction `UMAP_red`. La sortie de UMAP est normalisée avec la fonction `MinMaxScaler()` comme précédemment et est enregistrée dans un fichier .npy. La fonction `np_path` va formater le nom de ce fichier avec les paramètres des données en haute dimension ainsi que les paramètres utilisés pour la réduction de dimension UMAP puis le concaténer au chemin dans lequel on souhaite enregistrer les données en dimension réduite.

Il faut s'attarder sur les combinaisons de paramètres de UMAP. En effet, en exécutant UMAP avec une variété d'hyperparamètres, on se fait une meilleure idée de la façon dont l'intégration en dimension réduite est affectée par ces paramètres. Nous avons choisi de fixer deux paramètres : le paramètre *metric* aura la valeur "euclidean" et le paramètre *n_components* sera égal à 2. Par ailleurs, deux autres paramètres varient dans notre application de UMAP : le paramètre *min_dist* est égal à 0.1 ou à 0.5 et le paramètre *n_neighbors* est égal à 5 ou à 10. Nous nous sommes cependant rendu compte que nous avions fait un choix peu judicieux pour les valeurs de *n_neighbors*. En effet, les deux valeurs sont assez faibles et cela fait que UMAP se concentre trop sur la structure locale des données en comparaison à la structure globale. Nous aurions dû ajouter des valeurs plus élevées pour ce paramètre (autour de 50 voire 70 voisins). Nous aurions ainsi également pu étudier l'impact de ce paramètre sur la déformation des données lorsque UMAP se concentre sur la structure globale.

Le produit cartésien des paramètres de UMAP qui varient vaut 4. En le multipliant au nombre des fichiers de données en haute dimension, nous devrions obtenir 50 000 intégrations (aussi appelées plongements) dans UMAP sur lesquels mener une étude quantitative de la déformation des données. Or, au vu du temps que cette étape prends et du peu de temps qu'il restait, nous avons fait le choix de ne finalement pas considérer les données de la distribution en communautés pour notre étude. Nous obtenons donc 15 000 plongements pour la distribution gaussienne et 5000 plongements pour la distribution uniforme à cette étape. Nous appliquerons nos métriques sur ces intégrations afin de quantifier la déformation des données induite par UMAP.

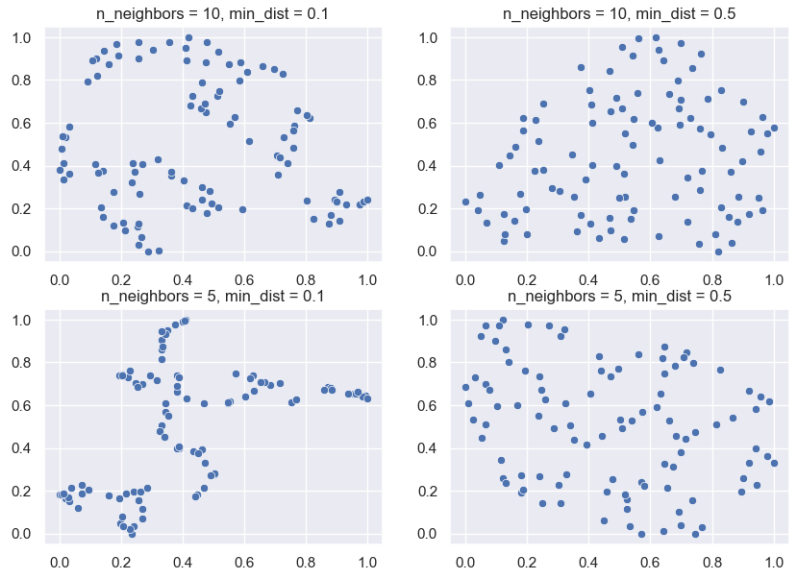

Figure 8 : Intégration UMAP de 100 données en 4 dimensions dans un espace en 2 dimensions, pour différentes valeurs des paramètres n_neighbors et min_dist.

3.3 Métriques

Les métriques jouent un rôle crucial dans l'évaluation de la qualité de la représentation réduite des données. Nous avons choisi d'en implémenter deux que nous présenterons en

suivant. Cette diversité de métriques permet après l'exécution de UMAP d'avoir plusieurs points de vue pour quantifier et analyser la manière dont l'algorithme déforme les données. Ces deux métriques sont implémentées dans le script `METRICS.py`.

3.3.1 Majorisation du stress

La première métrique est la majorisation du stress [6] et quantifie la préservation des distances entre les points dans l'espace en haute dimension par rapport à leur représentation dans l'espace de dimension réduite. La formule de la majorisation du stress est la suivante :

$$\sum_{i<j} w_{ij}(\|X_i - X_j\| - d_{ij})^2$$

$\|X_i - X_j\|$ correspond à la distance entre les points i et j dans l'espace de dimension réduite. d_{ij} correspond à la distance entre les points i et j dans l'espace de haute dimension. w_{ij} est un poids associé à la distance d_{ij} et valant $1/d_{ij}^{\alpha}$. α est une constante qui, en fonction de sa valeur, va influencer le poids de manière à ce qu'il favorise soit les petites soit les grandes distances d_{ij}. Si on prend $\alpha = 0$, tous les poids valent 1 et ni les grandes ni les petites distances ne sont favorisées. Si $\alpha \geq 1$, le poids w_{ij} diminue lorsque la distance d_{ij} augmente et inversement. Ainsi, dans ce cas, les petites distances sont favorisées. Si $0 < \alpha < 1$, le poids w_{ij} augmente lorsque la distance d_{ij} diminue et inversement. Dans ce cas de figure, ce sont donc les grandes distances qui sont favorisées. La formule calcule donc la somme des différences pondérées etre les distances dans les deux espaces (haute dimension et dimension réduite) pour chaque paire de points i et j avec $i < j$. La différence des distances est élevée au carré afin d'en amplifier l'effet dans la formule.

En théorie plus la majorisation du stress est basse, meilleure est la préservation des distances entre les points. Une conservation parfaite entre les deux espaces donnerait une majorisation du stress égale à 0.

Dans le script `METRICS.py`, la fonction `dist_matrix` calcule, dans un array numpy d, une matrice de distances entre les points en haute dimension. d[i][j] renvoie la distance entre les points i et j. La fonction `weight_matrix` calcule, dans un numpy array w, une matrice de poids en se basant sur une matrice de distance précédemment calculée et une constante α. w[i][j] renvoie le poids associé à la distance entre les points i et j dans l'espace en haute dimension. Dans notre code, nous fixons la valeur de α à 2 afin de favoriser les petites distances. Enfin, la fonction `stress_majo` calcule la majorisation du stress d'un plongement en faisant appel aux fonctions mentionnées précédemment. Dans notre code, nous normalisons la majorisation du stress par le nombre de paires de points afin de pouvoir, par la suite, comparer les valeurs de cette métrique calculée pour des plongements ayant différents nombres d'échantillons.

3.3.2 Indice de Jaccard moyen pour les k plus proches voisins

La seconde métrique est l'indice de Jaccard moyen pour les k plus proches voisins. Elle permet de quantifier le recoupement des k plus proches voisins de tous les points entre

l'ensemble en haute dimension et celui en dimension réduite. Soit A l'ensemble des k plus proches voisins d'un point p dans l'espace en haute dimension et B l'ensemble des k plus proches voisins de p dans l'espace de dimension réduite. Alors l'indice de Jaccard [7] pour les k plus proches voisins de p s'écrit :

$$Jaccard(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Cet indice correspond donc au rapport entre la cardinalité (nombre d'éléments d'un ensemble) de l'intersection de A et B et la cardinalité de l'union entre A et B. Cet indice varie entre 0 et 1. Si les relations de voisinage sont parfaitement conservées, c'est-à-dire si les k plus proches voisins de p sont les mêmes dans les deux espaces, alors cet indice vaut 1. En revanche, si p n'a aucun voisin en commun entre les deux espaces, alors l'indice de Jaccard vaut 0.

Dans le script METRICS.py, la fonction `k_neighbors` renvoie la liste des indices des k plus proches voisins de chaque point d'un ensemble de données, qu'il soit en haute dimension ou en dimension réduite. La fonction `jaccard_neighbors` calcule dans un premier temps l'indice de Jaccard, décrit ci-dessus, pour chaque échantillon et le sauvegarde dans la liste `jaccard_index`. Dans un second temps, la fonction `jaccard_neighbors` calcule l'indice de Jaccard moyen en faisant le rapport entre la somme de tous les indices de Jaccard contenus dans la liste `jaccard_index` et la longueur de cette liste (correspondant au nombre d'échantillons). Cet indice moyen est ensuite multiplié par 100 afin d'obtenir un pourcentage. Dans notre code, nous avons fixé le nombre de k plus proches voisins à considérer pour l'indice de Jaccard moyen à 7.

3.3.3 Sauvegarde des métriques

Pour chaque plongement d'un répertoire, la fonction `deformation_quantif_metrics` de `METRICS.py` va, dans un premier temps, récupérer ses paramètres de génération des données et de UMAP grâce à la fonction `get_params_np`. Dans un second temps, les métriques du plongement seront calculées en faisant appel aux fonctions décrites ci-dessus. Enfin, les paramètres et les métriques sont sauvegardés dans un fichier .csv. Les colonnes de ce fichier correspondent aux paramètres de génération des données et de UMAP ainsi qu'aux deux métriques. Chaque ligne correspond à un plongement différent. Ce fichier va permettre de mener l'analyse quantitative et de visualiser les métriques en fonction de divers paramètres.

3.4 Main

Un script *MAIN.py* regroupe le lancement de tous les scripts codés lors de ce projet. Il commence par charger les librairies permettant d'intéragir avec le système d'exploitation puis les fonctions principales des scripts. Viens ensuite la définition du répertoire de base, c'est-à-dire le répertoire regroupant le dossier principal du projet avec tous les scripts. Celui-ci est affecté à la variable *directory* et sera à changer avant le lancement. Après cela des contrôles sur les dossiers de sorties sont effectués. Les dossiers dans lesquels on retrouvera les données produites sont créés s'ils n'existent pas déjà à l'emplacement *directory*. Pour faciliter la lecture du code et l'utilisation dans les divers scripts nous avons mis en place, les variables d'environnement suivantes : *DATA_GENERATION_PATH*, *DATA_UMAP_PATH*

et *DATA_METRICS_PATH*. Pour finir, vient le lancement des fonctions principales des divers scripts avec d'abord la génération des données selon les trois distributions puis la réduction de dimension avec UMAP et enfin le calcul des métriques de quantification de la déformation. Par manque de temps, nous n'avons pas pu collecter les métriques pour toutes les intégrations. Toutefois, nous en avons récolté suffisamment pour les distributions uniforme et gaussienne pour mener une analyse quantitative.

4 Analyse quantitative

4.1 Distribution uniforme

Nous avons obtenu des métriques pour 2809 plongements ayant une distribution uniforme. Dans un premier temps, nous allons étudier la majorisation du stress sur les paramètres suivants : le nombre d'échantillons *n_samples*, le nombre de caractéristiques *n_features*, la distance minimale *min_dist* et le nombre de voisins *n_neighbors* utilisés dans UMAP .

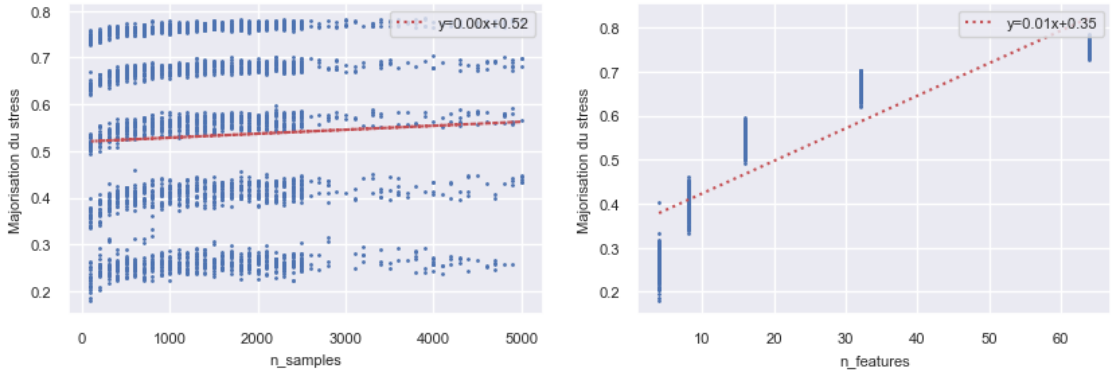

Figure 9 : Scatterplot de la majorisation du stress en fonction de différentes valeurs des paramètres n_samples et n_features.

Nous pouvons voir sur la figure 9 que la majorisation du stress a tendance à augmenter avec le nombre d'échantillons et le nombre de caractéristiques. Ainsi, plus il y a d'échantillons ou de caractéristiques moins les relations de distance entre les points seront conservées.

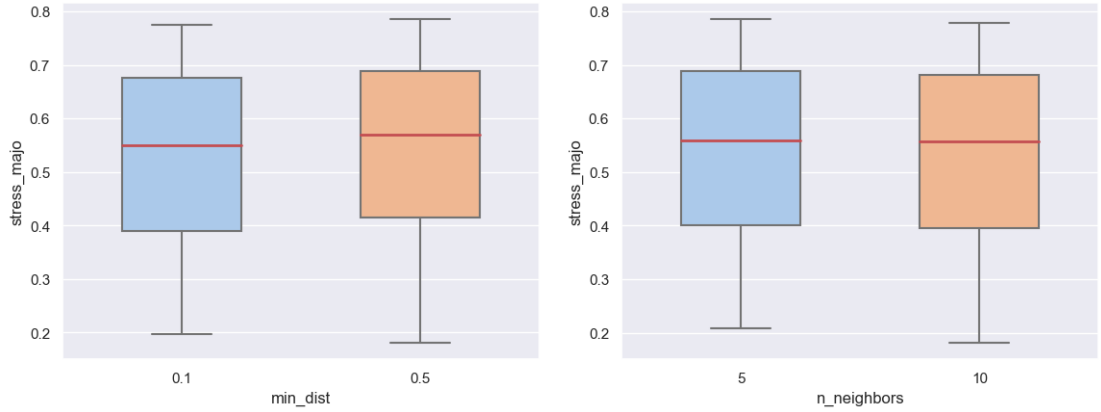

Figure 10 : Boxplot de la majorisation du stress en fonction de différentes valeurs des paramètres min_dist et n_neighbors.

La figure 10 nous indique que la majorisation du stress est légèrement plus élevée lorsque *min_dist* vaut 0.5 que lorsqu'elle vaut 0.1. En effet, la médiane et les quartiles du boxplot pour $min\_dist = 0.1$ sont inférieurs à ceux du boxplot pour $min\_dist = 0.5$. Les boxplots pour les deux valeurs de *n_neighbors* (5 ou 10) sont très similaires avec une médiane d'environ 0.55. La conservation des relations de distance entre les points semble donc sensible à la distance minimale utilisée dans UMAP mais pas au nombre de voisins considérés.

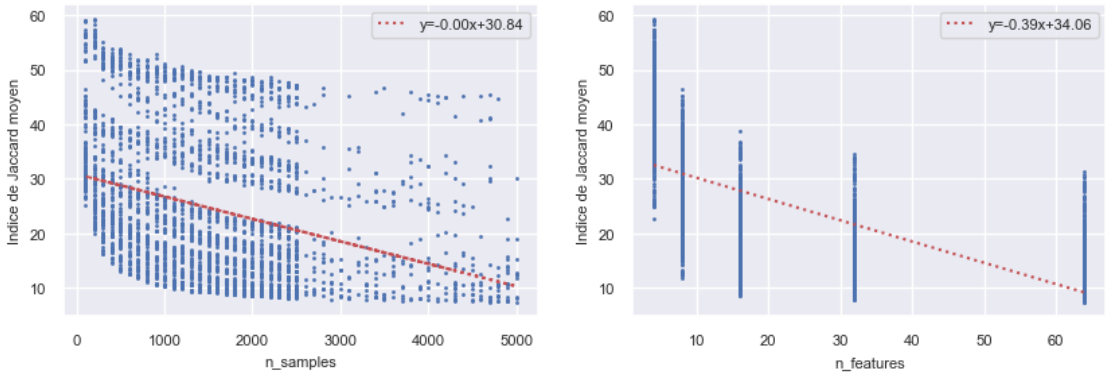

Figure 11 : Scatterplot de l'indice de Jaccard moyen pour les 7 plus proches voisins en fonction de différentes valeurs des paramètres n_samples et n_features.

Sur la figure 11, nous voyons que l'indice de Jaccard moyen pour les 7 plus proches voisins a tendance à diminuer avec le nombre d'échantillons et le nombre de caractéristiques. Plus il y a d'échantillons ou de caractéristiques moins les relations de voisinage entre les points sont conservées. L'ordonnée à l'origine de la courbe de tendance pour *n_features* est inférieure à celle pour *n_samples*. La déformation des distances entre les points est donc plus sensible à l'accroissement des caractéristiques qu'à celui des échantillons.

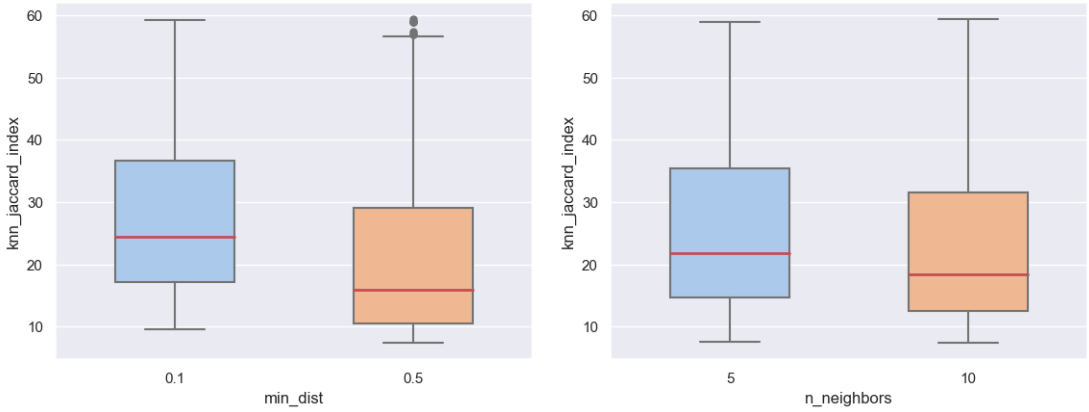

Figure 12 : Boxplot de l'indice de Jaccard moyen pour les 7 plus proches voisins en fonction de différentes valeurs des paramètres min_dist et n_neighbors.

La figure 12 nous montre que la médiane de l'indice de Jaccard moyen pour les 7 plus proches voisins du boxplot pour $min\_dist = 0.1$ vaut environ 25 et celle du boxplot pour $min\_dist = 0.5$ vaut environ 15. De plus, les quartiles du boxplot pour $min\_dist = 0.1$ sont supérieurs à ceux du boxplot pour $min\_dist = 0.5$. L'indice de Jaccard moyen pour les 7 plus proches voisins est donc plus élevé lorsque *min_dist* vaut 0.1 que lorsqu'elle vaut 0.5. On observe le même comportement avec les boxplots pour les deux valeurs de *n_neighbors* (5 ou 10). En effet, la médiane de l'indice de Jaccard moyen pour les 7 plus proches voisins du boxplot pour $n\_neighbors = 5$ vaut environ 22 et celle du boxplot pour $n\_neighbors = 10$ vaut environ 18. Ainsi, la conservation des relations de voisinage entre les points semble donc sensible à la distance minimale et au nombre de voisins utilisés dans UMAP. Il nous faudrait toutefois tester plus de valeurs des paramètres *min_dist* et *n_neighbors* afin d'étudier la manière exacte dont ils déforment les relations de distance et de voisinage entre les points.

4.2 Distribution gaussienne

Les résultats des métriques pour la distribution gaussienne sont présentés ici. Ces métriques concernent 1034 plongements. L'interprétation de ces métriques va permettre d'évaluer les performances du modèle UMAP pour la réduction de données ayant ce type de distribution.

La première métrique, la majoration du stress, est représentée ci-dessous en fonction de deux paramètres (le nombre d'échantillons et le nombre de caractéristiques).

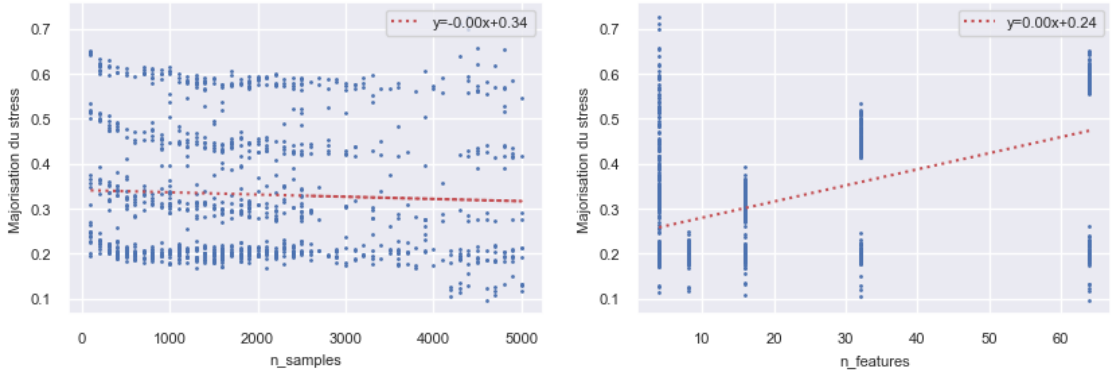

Figure 13 : Scatterplot de la majorisation du stress en fonction de différentes valeurs des paramètres n_samples et n_features.

On observe une tendance décroissante pour la courbe représentant la métrique en fonction du nombre d'échantillons. Cela suggère que lorsque le nombre d'échantillons augmente le stress diminue. Nous pouvons déduire de cela que l'algorithme fonctionne de manière plus stable et robuste avec un nombre élevé d'échantillons. Le second graphique évaluant la métrique en fonction du nombre de caractéristiques révèle une tendance croissante de cette dernière. Cela laisse penser que l'ajout de caractéristiques augmente la majorisation du stress. Il semble donc que la conservation des relations de distance entre les points soit corrélée à l'augmentation du nombre d'échantillons et anti-corrélée à l'accroissement du nombre de caractéristiques.

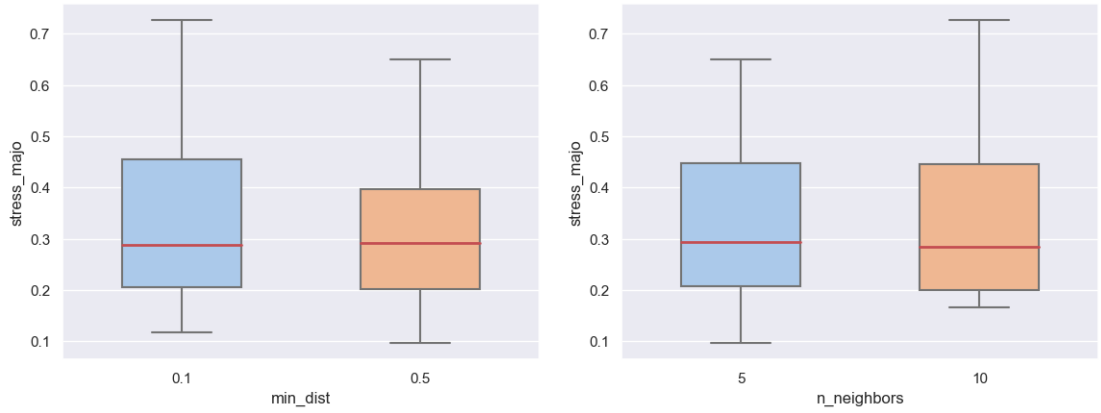

Figure 14 : Boxplot de la majorisation du stress en fonction de différentes valeurs des paramètres min_dist et n_neighbors.

La majorisation du stress a également été représentée en fonction de la distance minimale et du nombre de voisins utilisés dans UMAP. De ce fait, la figure 14 nous indique que les deux valeurs de distance minimale étudiées (0.1 et 0.5) ne diffèrent pas de manière significative. En effet, leurs médianes se trouvent toutes les deux aux alentours de 0.3. Une différence est observable pour le troisième quartile des boxplots. Celui-ci se situe vers 0.45 pour 0.1 alors qu'il est de 0.4 pour 0.5. Cela suggère que la dispersions des données dans la partie supérieure est plus importante pour la distance de 0.1. De même, les boxplots du nombre de voisins utilisés dans UMAP ont une médiane relativement similaire (autour de 0.3) Ainsi, la déformation des données, en terme de relations de distance entre les points, ne semble pas sensible à la distance minimale et au nombre de voisins dans cette situation.

La seconde métrique étudiée a été représentée par des graphiques évaluant l'impact du nombre d'échantillons, du nombre de caractéristiques, de la valeur de la distance minimale et du nombre de voisins.

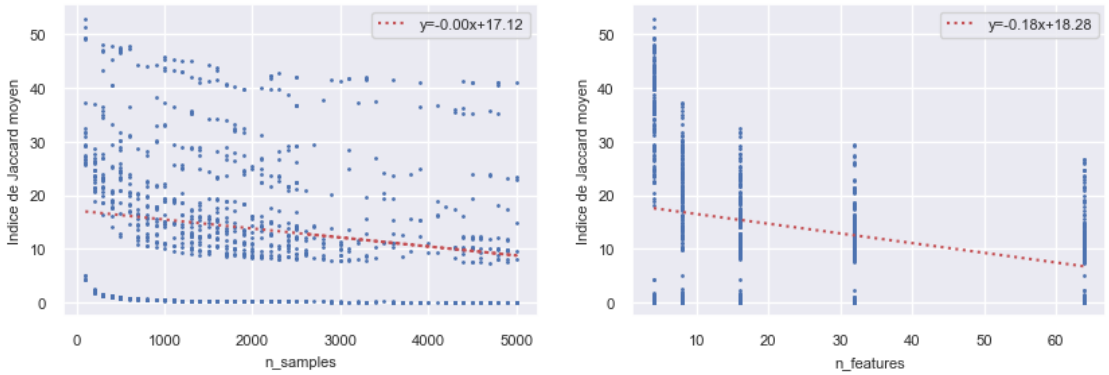

Figure 15 : Scatterplot de l'indice de Jaccard moyen pour les 7 plus proches voisins en fonction de différentes valeurs des paramètres n_samples et n_features.

Sur la figure 15, il est important de constater que l'indice de Jaccard moyen diminue en fonction du nombre d'échantillons et du nombre de caractéristiques. Plus il y a d'échantillons et de caractéristiques moins les relations de voisinages sont conservées.

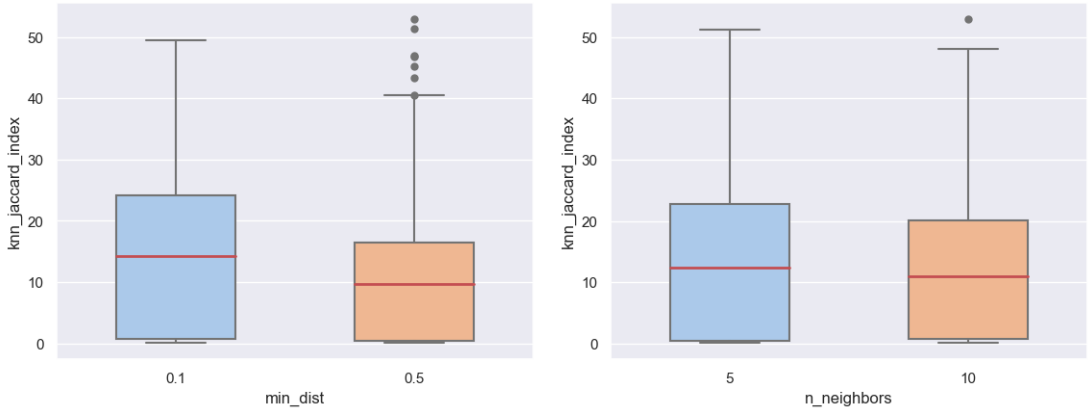

Figure 16 : Boxplot de l'indice de Jaccard moyen pour les 7 plus proches voisins en fonction de différentes valeurs des paramètres min_dist et n_neighbors.

Enfin, la distance minimale semble avoir un effet significatif sur l'indice de Jaccard moyen. En effet, les médianes des boxplots pour $min\_dist = 0.1$ et $min\_dist = 0.5$ ne sont pas équivalentes. Elle sont de 15 et 10 respectivement. L'indice de Jaccard moyen est donc plus élevé pour une distance minimale plus basse. L'étendue des boxplots révèle une variabilité plus importante dans les résultats pour $min\_dist = 0.1$. Cela peut indiquer la présence de plus de données extrêmes ou de sous-groupes distincts dans la distribution. Le nombre de voisins semble avoir un impact moindre sur l'indice de Jaccard moyen. En effet, pour un choix de 5 ou 10 voisins, la variation de l'indice de Jaccard moyen n'est que de 2. Le premier quartile se révèle cependant plus élevé pour 5 voisins (23 contre 20), révélant ainsi une possible meilleure conservation des relations de voisinage entre les points pour un nombre de voisins fixé à 5.

5 Conclusion

Pour conclure, nous observons des comportements assez similaires de nos métriques entre les distributions uniforme et gaussienne. En général, les relations de distances entre les points ont tendance à être déformées avec l'augmentation du nombre d'échantillons ou de caractéristiques. Les relations de voisinage entre les points sont elles aussi dénaturées par l'augmentation de ces paramètres. En ce qui concerne le paramètre *min_dist*, une faible valeur de ce dernier semble mieux préserver les relations de voisinage entre les points mais semble aussi légèrement déformer les relations de distance entre les points. Enfin, le paramètre *n_neighbors* ne semble pas avoir d'effet sur les relations de distance entre les points. En revanche, celui-ci a un réel impact sur les relations de voisinage entre les points : plus le nombre de voisins considérés dans UMAP est grand moins les relations de voisinage seront conservées. Toutefois, notre indice de Jaccard moyen ne concerne que les 7 plus proches voisins. Elle ne nous donne ainsi des informations que sur un voisinage relativement proche des points. Il serait donc intéressant, de tester une nouvelle métrique avec une valeur plus élevée pour les k plus proches voisins.

Plusieurs points d'amélioration de notre implémentation sont possibles. Tout d'abord, nous pourrions étudier la déformation induite l'algorithme de UMAP avec les paramètres suivants :

- *n_components* $= 1$, pour étudier la réduction d'espaces en haute dimension en espaces à une dimension (ce qui n'est autre qu'une droite).
- *n_components* $= 2$ et *n_features* $= 2$, pour vérifier que l'intégration d'un espace bidimensionnel dans un espace bidimensionnel n'apporte aucune déformation des données.
- *n_neighbors* avec plus de valeurs et des valeurs plus élevées (50-70 voisins), pour étudier l'impact de ce paramètre sur la déformation induite par UMAP lorsque l'algorithme se concentre sur la structure globale des données.
- *min_dist* avec plus de valeurs, pour étudier l'impact de ce paramètre sur la déformation induite par UMAP.

Par ailleurs, nous avions envisagé d'intégrer une troisième métrique permettant de comparer les écarts-types intra- et inter-communautés. Cependant, ayant choisi de ne plus travailler avec la distribution en communautés, cette métrique n'était plus pertinente pour notre étude.

Références

[1] Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2(Mar) :499–526, 2002.

[2] Junyue Cao, Malte Spielmann, Xiaojie Qiu, et al. The single-cell transcriptional landscape of mammalian organogenesis. *Nature*, 566 :496–502, 2019. doi : 10.1038/s41586-019-0969-x. URL https://www.nature.com/articles/s41586-019-0969-x.

[3] Leland McInnes, John Healy, and James Melville. Umap : Uniform manifold approximation and projection for dimension reduction. 2 2018. URL https://arxiv.org/abs/1802.03426v3.

[4] Understanding umap. URL https://pair-code.github.io/understanding-umap/.

[5] Umap : Uniform manifold approximation and projection for dimension reduction — umap 0.5 documentation. URL https://umap-learn.readthedocs.io/en/latest/index.html.

[6] Emden Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. volume 3383, 09 2004. ISBN 978-3-540-24528-5. doi : 10.1007/978-3-540-31843-9_25.

[7] Luciano Da and Fontoura Costa. Further generalizations of the jaccard index. 2022. URL https://hal.science/hal-03384438v4.