**Clustering - Rapport v1**

**I. Import :**

**Import Packages**

Importation de tous les packages nécessaires au bon déroulement du code :

pandas, numpy, nltk, sklearn, matplotlib.pyplot.

**Import Data**

Fichiers importés:

- Bag of word (du groupe 3)

- Data.json

**II. Functions :**

1. **Première partie**

**Function optimal\_cluster**

*Paramètres d’entrée :*

- X, les données que l’on souhaite étudier

*Paramètre de sortie :*

- graphique 3D en mouvement des résultats

Cette fonction permet de représenter les articles sur les axes principaux d’une ACP (réduction de dimensions). C’est ce qui nous permet de faire une analyse visuelle.

**Function training**

*Paramètres d’entrée :*

- X, les données que l’on souhaite intégrer dans le modèle pour l’entraîner

- nb\_cluster : Le nombre de clusters

*Paramètre de sortie :*

- données ajustées par le modèle kmeans

Cette fonction permet d'entraîner un modèle sur les données X avec le nombre de clusters choisi (nb\_cluster) et renvoie le modèle entraîné.

**Function predict**

*Paramètres d’entrée :*

-X, les données que l’on souhaite labelliser

- model : le modèle choisi pour la labellisation de nos données

*Paramètre de sortie :*

*-* les étiquettes des données passées en paramètres

Cette fonction permet de prédire les étiquettes des données.

1. **Deuxième partie**

**Function compute\_distance**

*Paramètre d’entrée :*

- v: les éléments du cluster choisi

- km le modèle choisi pour créer nos clusters

- label : le label dominant

*Paramètre de sortie :*

-Une liste correspondant aux distances entre les éléments d’un cluster passé en paramètre et le centroïde du cluster avec le label sélectionné.

Cette fonction permet de calculer la distance cosinus entre les valeurs à l'intérieur du cluster et le label le plus dominant .

**Function obtain\_word**

*Paramètre d’entrée :*

- X: les données

*Paramètre de sortie :*

- le nombre optimal de clusters

Cette fonction permet de calculer le nombre optimal de clusters.

**Function whitelist\_content**

*Paramètre d’entrée :*

- List\_content: les contenus de tous les articles

- list\_whitelist: une liste de mots à supprimer de nos labels

*Paramètre de sortie :*

- le nombre optimal de clusters

Cette fonction permet d’obtenir le contenu d'un article à l'exception des mots présents dans la list\_whitelist

**Function get\_label\_title**

*Paramètre d’entrée :*

- data: un dataframe contenant toutes nos données.

- whitelist: une liste de mots à supprimer de nos labels

*Paramètre de sortie :*

* les thèmes de chaque cluster (composé des 3 mots les plus représentatifs)

Cette fonction permet de renvoyer le thème de chaque cluster .

1. **Troisième partie**

**function predict\_cluster**

*Paramètre d’entrée :*

- data: un dataframe contenant nos données.

*Paramètre de sortie :*

* Un dataframe qui contient l'identifiant de chaque article ainsi que leur label
* model: le modèle entrainé avec le nombre optimal de clusters

Cette fonction permet d'associer chaque article à son cluster.

**III. Méthode :**

Pour cette version v1, la problématique est d’automatiser tout le processus pour trouver le nombre k, de clusters optimal. Il faudra également générer de manière automatique des noms de thèmes pour chaque cluster.

Nous avons d’abord cherché un nouveau modèle ainsi qu’une nouvelle représentation, plus robustes pour nos données. Notre choix après analyse (voir annexe: clustering\_model) de plusieurs modèle avec différents paramètres et représentations, s’est fixé sur le modèle des k-médoïdes en utilisant une métrique cosine(distance cosinus) sur le bow du groupe 3 sur lequel on a effectué une ACP.

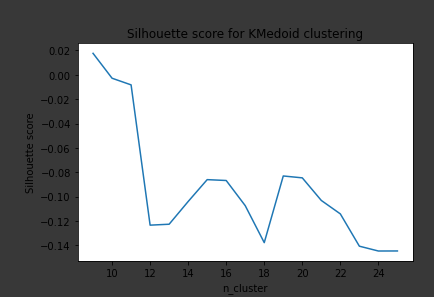
Pour trouver la valeur optimale de k nous utilisons le calcul sur le score de la silhouette. Ce dernier correspond à la différence entre la distance moyenne des documents du même cluster et la distance moyenne avec les points des autres clusters. Le nombre optimal de clusters qui en ressort est 9.

**IV. Modèles :**

1. **k-Means**

Méthode de partitionnement des données .

Etant donné des documents, la problématique est de les regrouper en k groupes appelés clusters. On cherche à regrouper le nombre optimal de clusters en affichant la silhouette. Dans notre cas , la silhouette donne comme nombre de clusters optimal 9 comme visualisé sur la figure ci-après.



1. **k-Médoïdes**

Méthode représentant le document le plus central d’un jeu de données.

L’algorithme des k-médoïdes est un algorithme de partitionnement plus robuste vis-à-vis des données aberrantes que celui des k-means. Il minimise l’erreur quadratique moyenne qui est la distance entre les points correspondant aux articles et le point central (médoïde). Contrairement aux centroïdes de l’algorithme Kmeans, le médoïde existe.

1. **Birch**

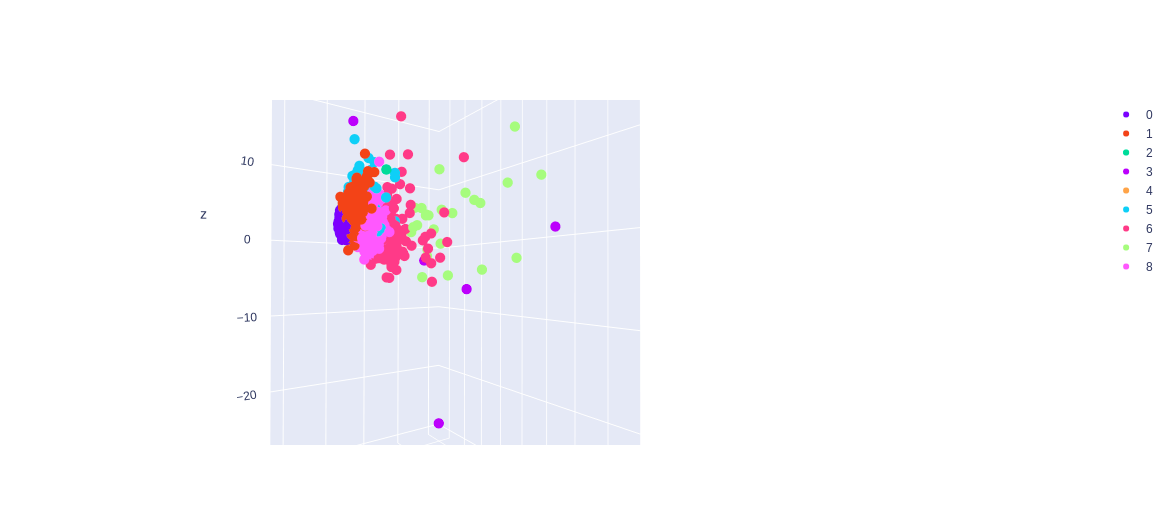
Birch est un algorithme d’exploration de données non supervisées utilisées pour effectuer un clustering hiérarchique sur des ensembles de données particulièrement volumineux .

Un avantage de cet algorithme est sa capacité à regrouper de manière incrémentielle et dynamique des points de données , ceci dans le but de produire la meilleure qualité de clustering pour un ensemble de données de ressources .

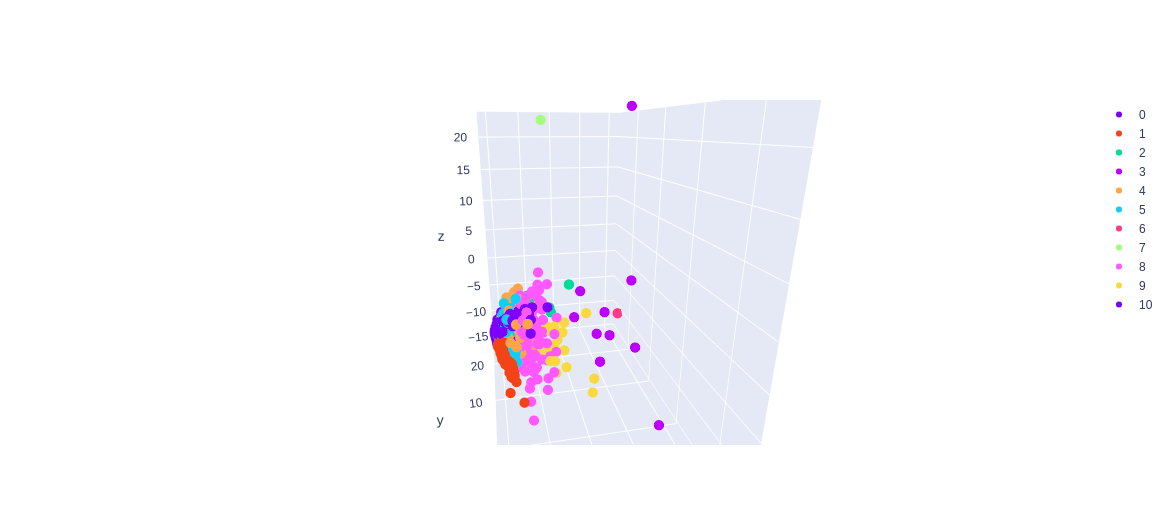
**V. Analyses faites sur différentes méthodes :**

Nous avons représenté les données en 3D sur différents modèles afin d’en déduire le modèle optimal. Les modèles les plus efficaces qui ont découlés de notre analyse sont (k\_means, kmédoïde, birch).

Ci-dessous une vue sur les différents modèles :

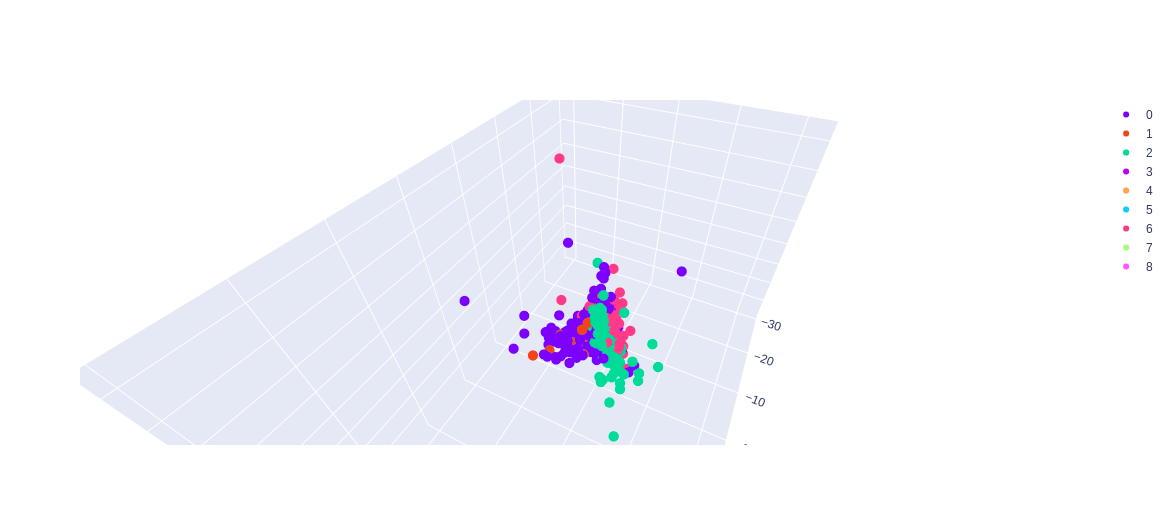
1. **k-Means :**

|  |
| --- |
| paramètres: nb\_cluster=9 |
| nombre d’article par cluster:  0 4318  1 1892  8 727  2 238  6 167  5 160  7 25  3 5  4 1 |

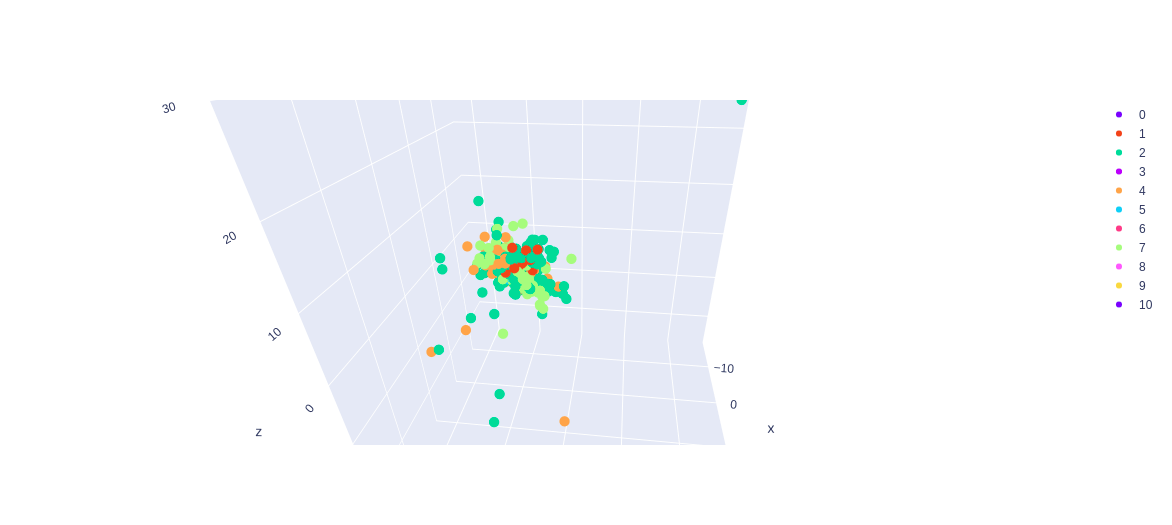


|  |
| --- |
| paramètres: nb\_cluster=11 |
| nombre d’article par cluster:  6 4432  3 1162  0 939  2 509  9 238  10 123  5 102  4 16  1 10  7 1  8 1 |

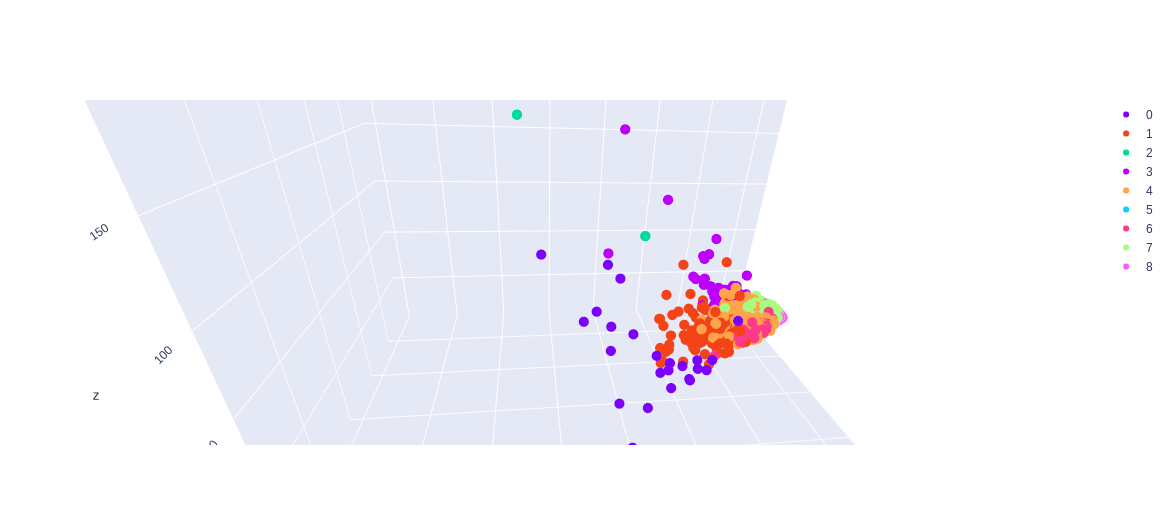
1. **k-Médoïdes**



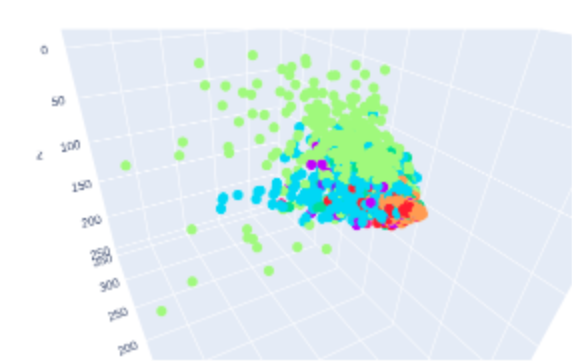
|  |
| --- |
| paramètres: n\_clusters=9, metric='cosine', random\_state=0 |
| nombre d’article par cluster:  8 3362  3 708  0 707  7 607  6 575  2 425  5 413  4 385  1 351 |



|  |
| --- |
| paramètres: n\_clusters=11, metric='cosine', random\_state=0 |
| nombre d’article par cluster:  10 2211  8 1150  2 717  9 704  3 573  4 560  7 422  5 365  1 328  6 318  0 185 |

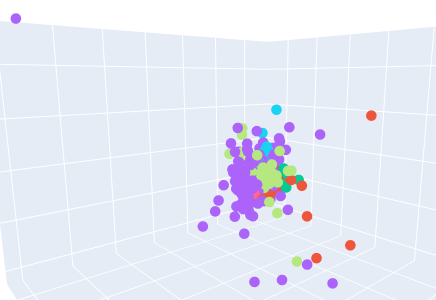
1. **Birch**

|  |
| --- |
| paramètres: n\_clusters=9 |
| nombre d’article par cluster:  8 3853  7 2317  4 889  1 245  3 124  6 74  0 24  5 4  2 3 |

1. **AgglomerativeClustering**

|  |
| --- |
| paramètres: n\_clusters=9, affinity='cosine', linkage='complete' |
| nombre d’article par cluster:  4 3744  5 986  7 984  1 808  2 432  3 250  0 230  8 81  6 18 |

Notre modèle est le k-médoïdes avec pour metric cosine (n\_clusters=9, metric='cosine', random\_state=0 ) dont la représentation 3D est la suivante :



**VI. Résultat :**

Dans chaque exécution du code, la numération des clusters différent, mais les tailles des clusters restent constants, la taille des clusters qu’on a obtenue est:

|  |  |
| --- | --- |
| 0 | 3400 |
| 1 | 711 |
| 2 | 710 |
| 3 | 573 |
| 4 | 570 |
| 5 | 424 |
| 6 | 419 |
| 7 | 376 |
| 8 | 350 |

**Conclusion :**

Notre v1 est meilleure que la v0 en termes de répartition des mots dans chaque cluster, ainsi que dans la représentation 3D . Les résultats ne sont pas parfaits, nos documents sont très proches les uns des autres car ils abordent les mêmes sujets . C’est pourquoi notre représentation 3D montre un gros tas de documents et peu d’articles très différents des autres .