TP 2 Clustering

Youna Froger, Alice Gyde, Mathis Quais

29 Novembre 2024

1 Exercice du TP

Voici le lien du GitHub : https://github.com/younafroger/TAL$_H NM2$

2 Questions

1. What is clustering ?

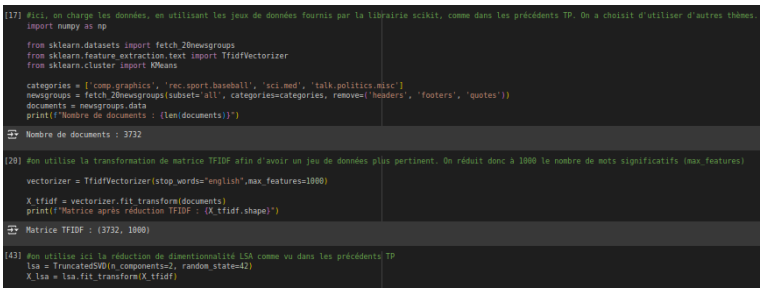
La clusteristion est le sujet de ce nouveau TP. C'est une opération effectué sur un jeu de données dans le cadre d'études statitstiques. Le but de la clusterisation est de réorganisé le jeu de données en plusieurs groupes en fonction de leur similitude. Pour ça, on utilise des algorithmes pré-définis qui ont chacuns leurs avantages/inconvéniants en fonction de la nature du jeu de données. Certains algorithmes sont efficaces lorsque les données sont très différentes, et d'autres fonctionnent avec des jeux de données d'avantages similaires.

2. Why and when should clustering be applied ?
Le clustering est une méthode utilisée lorqu'on a à traiter un jeu de données importants qui n'a pas été étiquetté au préalable manuellement.

3. What is the K-Means algorithm ?
L'algorithme K-Means fait partie des algorithme les plus utilisés en clusterisation. Il définit au hasard des centres de clusters, puis calcule la distance entre les centres et les individus, tout cela dans une boucle, afin de minimiser cette distance jusqu'à obtenir des clusters distincts. K-Means nécessite de choisir un bon nombre de clusters de départ (K).

2.1 Explications du code

Dans un premier temps, on charge et prépare les données textuelles importées de la librairie scikit-learn, comme dans le précédent TP. On choisit cette fois-ci d'utiliser d'autres thèmes de jeu de données. Les traitements ici sont de transformer la matrice du jeu de données avec la méthode TF-IDF, ce qui permet de rendre notre jeu de données plus pertinent, en réduisant le nombre de mots significatifs ($\max_features$)$à$1000.$Onréduitensuiteladimensionnalitéaveclaméthode LSA$.

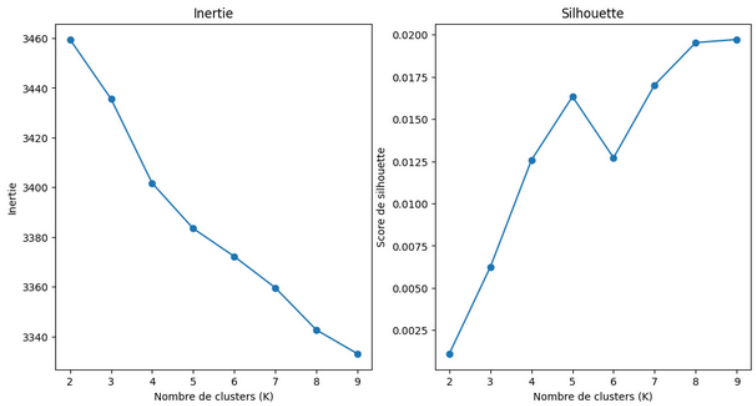


```
[17] #ici, on charge les données, en utilisant les jeux de données fournis par la librairie scikit, comme dans les précédents TP. On a choisit d'utiliser d'autres thèmes.
     import numpy as np

     from sklearn.datasets import fetch_20newsgroups
     from sklearn.feature_extraction.text import TfidfVectorizer
     from sklearn.cluster import KMeans

     categories = ['comp.graphics', 'rec.sport.baseball', 'sci.med', 'talk.politics.misc']
     newsgroups = fetch_20newsgroups(subset='all', categories=categories, remove=('headers', 'footers', 'quotes'))
     documents = newsgroups.data
     print(f"Nombre de documents : {len(documents)}")

→ Nombre de documents : 3732

[20] #on utilise la transformation de matrice TFIDF afin d'avoir un jeu de données plus pertinent. On réduit donc à 1000 le nombre de mots significatifs (max_features)

     vectorizer = TfidfVectorizer(stop_words="english",max_features=1000)

     X_tfidf = vectorizer.fit_transform(documents)
     print(f"Matrice après réduction TFIDF : {X_tfidf.shape}")

→ Matrice TFIDF : (3732, 1000)

[43] #on utilise ici la réduction de dimentionnalité LSA comme vu dans les précédents TP
     lsa = TruncatedSVD(n_components=2, random_state=42)
     X_lsa = lsa.fit_transform(X_tfidf)
```

Cette partie du code va calculer des métriques importantes dans le cadre de la clusterisation : Le score de silhouette et l'inertie. Puis on projete les résultats sur deux graphiques pour les analyser. L'inertie sert à mesurer à quel point les clusters sont proches.

```
[32] #ici on va calculer le score de silouhette et l'inertie. Ces métriques permettent de calculer le nombre optimal de clusters.
     inertia = []
     silhouette = []
     K_range = range(2, 10)
     for k in K_range:
         kmeans = KMeans(n_clusters=k, random_state=42)
         labels = kmeans.fit_predict(X_tfidf)
         inertia.append(kmeans.inertia_)
         silhouette.append(silhouette_score(X_tfidf, labels))

     plt.figure(figsize=(12, 6))
     plt.subplot(1, 2, 1)
     plt.plot(K_range, inertia, marker='o')
     plt.title('Inertie')
     plt.xlabel('Nombre de clusters (K)')
     plt.ylabel('Inertie')

     plt.subplot(1, 2, 2)
     plt.plot(K_range, silhouette, marker='o')
     plt.title('Silhouette')
     plt.xlabel('Nombre de clusters (K)')
     plt.ylabel('Score de silhouette')
     plt.show()
```

Dans le graphique obtenu, il faut trouver l'endroit où la courbe forme un "coude" pour trouver le moment où l'ajout de cluster ne change plus grand-chose à la clusterisation. Ici, on remarque qu'il n'y a pas de "coude" marqué. Le calcul de l'inertie ne nous apport pas beaucoup d'informations.



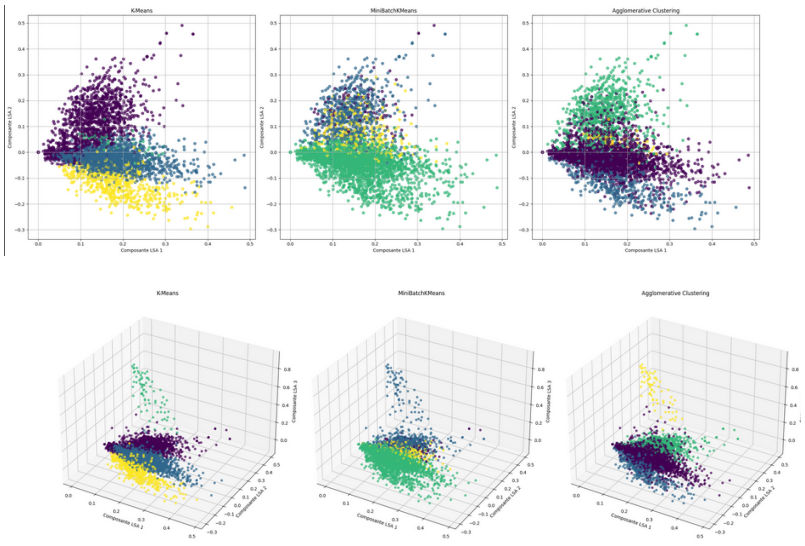
Le score de silhouette est une métrique qui mesure à quel point les données dans un cluster sont proches, comparé à leur distance avec les données d'autres clusters. La valeur obtenue est comprise entre -1 , 0 et 1. Un score de silhouette de 1 est optimal, et signifie que les clusters sont bien séparés. Une valeur s'approchant de 0 signifie que les clusters ont tendance à se chevaucher. Et plus on s'approche d'une valeur de -1, moins la clusterisation n'est claire et les données ont été mal assignée dès le début à un centre aléatoire. Le graphique montre une courbe, mais le score est très bas (0.02) montrant que les clusters sont mal séparés. L'hypothèse serait que notre jeu de données ne présente pas de clusters très nets, les données doivent être trop similaires.

```
optimal_clusters = 4
kmeans = KMeans(n_clusters=optimal_clusters, random_state=42)
clusters = kmeans.fit_predict(X_tfidf)


mb_kmeans = MiniBatchKMeans(n_clusters=optimal_clusters, random_state=42, batch_size=100)
mb_clusters = mb_kmeans.fit_predict(X_tfidf)


agglo = AgglomerativeClustering(n_clusters=optimal_clusters)
agglo_clusters = agglo.fit_predict(X_tfidf.toarray())
```

Cette partie du code permet l'application de plusieurs algorithmes de clusterisation différents afin de pouvoir voir les différences entre eux. Le premier est l'algorithme K-Means comme évoqué précédemment. Pour chacun des algorithmes, on indique que le nombre de clusters idéal est de 4 (puisqu'on sait qu'on a 4 thèmes différents). On répète la même opération avec l'algorithme MiniBatchKMeans et Agglomerative Clustering. Les deux premiers algorithmes (K-Means et MiniBatchKMeans) sont similaires, et essaient de trouver les clusters plutôt compacts, et sphériques. Agglomerative Clustering, est différents et

souvent utilisés pour des jeux de données aux clusters de forme plus complexe, pas forcément sphérique, et est souvent utilisé lorsque les clusters sont moins bien marqués.
Maintenant, on passe à l'étape de génération de graphiques, d'abord en 2D puis nous avons suggéré de faire un graphique en 3D afin de mieux voir les clusters. La génération des graphiques reprend les codes précédemment utilisés dans les autres TP.

2.2 Analyse des graphiques et conclusion



En 2D, les trois algorithmes utilisés nous montrent une séparation des clusters qui se chevauchent beaucoup au centre. Les trois algorithmes montrent des graphiques très similaires, avec 3 grands clusters qui se chevauchent.

En 3D, c'est l'algorithme Agglomerative Clustering et le K-Means qui se trouvent être les plus révélants, puisque le cluster vert (K-Means) et jaune (AC) ne sont plus mêlés à tout le reste, et se détachent du groupe central où l'on aperçoit tout de même une séparation en 3 groupes distincts. On peut en déduire que 3 clusters semblent très similaires (bleu, violet et vert) et pourraient correspondre à 3 thèmes de jeu de données qui serait proche en termes de mots utilisés. Et les clusters vert (K-Means) et jaune (AC) se détachent du reste, bien qu'un peu éparse, ils indiqueraient un thème tout à fait différent.