Science des données

IFT799 – TP2-3

Amazon Book Reviews

UNIVERSITÉ DE SHERBROOKE

Elèves : Tahar AMAIRI et Corentin POMMELEC

Session : Automne 2022

Introduction :

Le code de ce TP est contenu dans un Jupyter Notebook. Pour son exécution, il nécessite plusieurs modules déjà très connus dans le domaine de la science des données : pandas, numpy, matplotlib etc.. Cependant certains le sont moins comme nltk : ces librairies peuvent être facilement installées à l'aide de "pip".

Le jeu de données que nous utilisons est celui de l'Amazon Book Reviews (ABR) contenant les revues données par les clients sur les livres qu'ils ont achetés sur Amazon. Celui-ci se trouve dans un fichier JSON contenant différentes informations sur les avis des clients qui vont nous être indispensables pour réaliser notre analyse descriptive. On y trouve par exemple l'ASIN (Amazon Standard Identification Number) du livre, sa note ou bien des indications temporelles sur l'avis (unix time, raw time). Pour charger le fichier JSON contenant toutes les reviews, nous allons le découper en "chunk" afin de pouvoir le lire car il est extrêmement lourd (~10 GB). Pour cela, il est possible en fonction de la puissance de votre ordinateur d'augmenter la taille des chunks et/ou leurs nombres. Dans le cadre de la rédaction de ce rapport, nous avons chargé 5 millions d'avis.

L'exploration des données, effectuée dans le TP1, nous a permis d'analyser le jeu de données et, ainsi, de mieux le comprendre. Cependant, le but reste de réaliser une analyse globale. Ainsi, dans la première partie du TP2&3 (a.) on cherche à exploiter des caractéristiques statistiques, caractérisant nos données, afin de réaliser une segmentation de ces dernières à partir de plusieurs stratégies de clustering.

Dans la deuxième partie (b.) on souhaite effectuer les mêmes étapes que dans la première partie (a.) mais en tenant compte de la totalité des données. Pour cela, nous réaliserons une sélection stratifiée pour faire en sorte qu'il y ait un nombre équitable de livres par catégorie de rating, lors de l'utilisation de nos algorithmes de segmentation. Nous pourrons ainsi comparer les résultats obtenus avec et sans équilibrage du nombre de livres par catégorie.

a.1) Construire la matrice de données X ∈ Rp×c avec c étant le nombre de caractéristiques statistiques extraites.

Pour construire la matrice de données X ∈ Rp×c avec c étant le nombre de caractéristiques statistiques extraites nous faisons appel à la fonction getX. Les caractéristiques que l'on souhaite extraire sont : le nombre de personnes ayant voté, la moyenne, l'écart type, et la médiane des scores, le nombre de fois qu'un livre a été apprécié, le nombre de fois que le livre n'a pas été apprécié et le nombre de fois que le choix a été neutre. Pour obtenir notre matrice X, on part du dataframe dfScore contenant pour chacun des livres le nombre d'avis obtenus pour chaque note :

	Ratings 1 ☆	Ratings 2 ☆	Ratings 3 ☆	Ratings 4 ☆	Ratings 5 ☆
000100039X	6	4	8	15	173
0001055178	0	4	2	10	2
0001473123	1	0	0	2	13
0001473727	0	0	0	0	7
0001473905	0	0	1	0	5

Premières lignes du dataframe dfScore

Nous pouvons désormais calculer pour chaque livre les statistiques énoncées précédemment et ainsi former notre dataframe dfX. Ce dernier contiendra p livres aléatoires avec p égale au nombre de livres dans la catégorie avec le minimum de livre, multiplié par trois (ceci est pour permettre une sélection stratifiée dans la partie b).

	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews
0811216780	4.600000	0.489898	5.0	5	0	0	5
0452274141	4.421053	0.935599	5.0	17	0	2	19
068982002X	4.000000	1.137593	4.0	11	5	1	17
0615699006	4.666667	0.471405	5.0	12	0	0	12
0609605844	3.375000	1.218349	3.0	7	5	4	16

Premières lignes du dataframe dfX

X est la conversion de dfX en un tableau numpy pour nous faciliter sa manipulation.

a.2) Effectuer une segmentation basée sur les k-moyennes avec k = 3.

Nous allons effectuer une segmentation basée sur l'algorithme de clustering k-means. Ce dernier va nous permettre de regrouper les livres de notre jeu de données en k=3 classes selon l'homogénéité des caractéristiques contenues dans la matrice X. K-means se divise en plusieurs étapes. La première est l'initialisation des centroïdes, qui correspondent aux centres initiaux de nos 3 classes. Pour la seconde étape, dans notre cas, nous utiliserons la distance euclidienne ou la similarité cosinus pour déterminer la mesure entre les livres et chaque centroïde. A partir de la mesure effectuée nous pouvons affecter chaque livre au centroïde le plus proche. La troisième étape est de réajuster les centroïdes en calculant les centres de gravité des différents groupes. Enfin, nous itérons les trois étapes précédentes tant que les livres sont affectés à de nouvelles classes après une itération.

La librairie nltk fournit la classe KMeansClusterer qui comme son nom l'indique permet d'effectuer la segmentation telle que voulue. Nous créons donc l'objet clusterer en utilisant cette dernière avec comme attributs :

- `num_means=k`
  Pour indiquer le nombre de segments attendus. Dans notre cas, `k` sera égal à 3.
- `initial_means=centers`
  Pour spécifier les centroïdes initiaux. On lui donne comme valeur `centers` qui reçoit les centroïdes calculés grâce à la fonction, de la librairie scikit-learn, kmeans_plusplus. Cette dernière permet de sélectionner les centres de cluster initiaux en se basant sur les données pour accélérer la convergence.
- `distance=distance`
  Pour utiliser une distance en particulier lors de la mesure entre les coordonnées et les centroïdes.

Ensuite, le calcul est adapté sur nos données en appelant la fonction cluster, utilisé sur l'objet clusterer créé, avec la matrice X comme paramètre. Nous avons regroupé ces implémentation dans la fonction getKMeansLabels qui retourne le résultat de clusterer qui est un vecteur contenant les classes correspondantes à chaque livre de notre jeu de données. Par ailleurs, comme X est une matrice contenant des attributs de différents ordres de grandeurs, nous allons la standardiser à l'aide de la class StandardScaler de sklearn.

- **<u>En utilisant la distance euclidienne :</u>**

La segmentation par l'algorithme k-means en utilisant la distance euclidienne correspond à l'appel de la fonction getKMeansLabels en spécifiant que l'on souhaite effectuer l'algorithme avec la distance euclidienne, `euclidean_distance`. On stocke dans une variable le vecteur retourné par la fonction. On pourra par la suite utiliser ce vecteur afin de faire correspondre chaque livre à sa classe déterminée.

	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews	Labels
0811216780	4.600000	0.489898	5.0	5	0	0	5	1
0452274141	4.421053	0.935599	5.0	17	0	2	19	1
068982002X	4.000000	1.137593	4.0	11	5	1	17	0
0615699006	4.666667	0.471405	5.0	12	0	0	12	1
0609605844	3.375000	1.218349	3.0	7	5	4	16	0

<u>dfX labellisé avec k-means distance euclidienne (à titre d'illustration, non présent dans le code)</u>

- **<u>En utilisant la similarité cosinus :</u>**

La segmentation par l'algorithme k-means en utilisant la similarité cosinus correspond à l'appel de la fonction getKMeansLabels en spécifiant que l'on souhaite effectuer l'algorithme avec la similarité cosinus, `cosine_similarity`. De même, on conserve dans une variable le vecteur, retourné par la fonction, que l'on pourra utiliser afin de faire correspondre chaque livre à sa classe.

	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews	Labels
0811216780	4.600000	0.489898	5.0	5	0	0	5	1
0452274141	4.421053	0.935599	5.0	17	0	2	19	1
068982002X	4.000000	1.137593	4.0	11	5	1	17	0
0615699006	4.666667	0.471405	5.0	12	0	0	12	1
0609605844	3.375000	1.218349	3.0	7	5	4	16	0

dfX labellisé avec k-means similarité cosinus (à titre d'illustration, non présent dans le code)

a.3) Effectuer une analyse en composante principale et projeter les segments obtenus suivant les deux premières composantes principales.

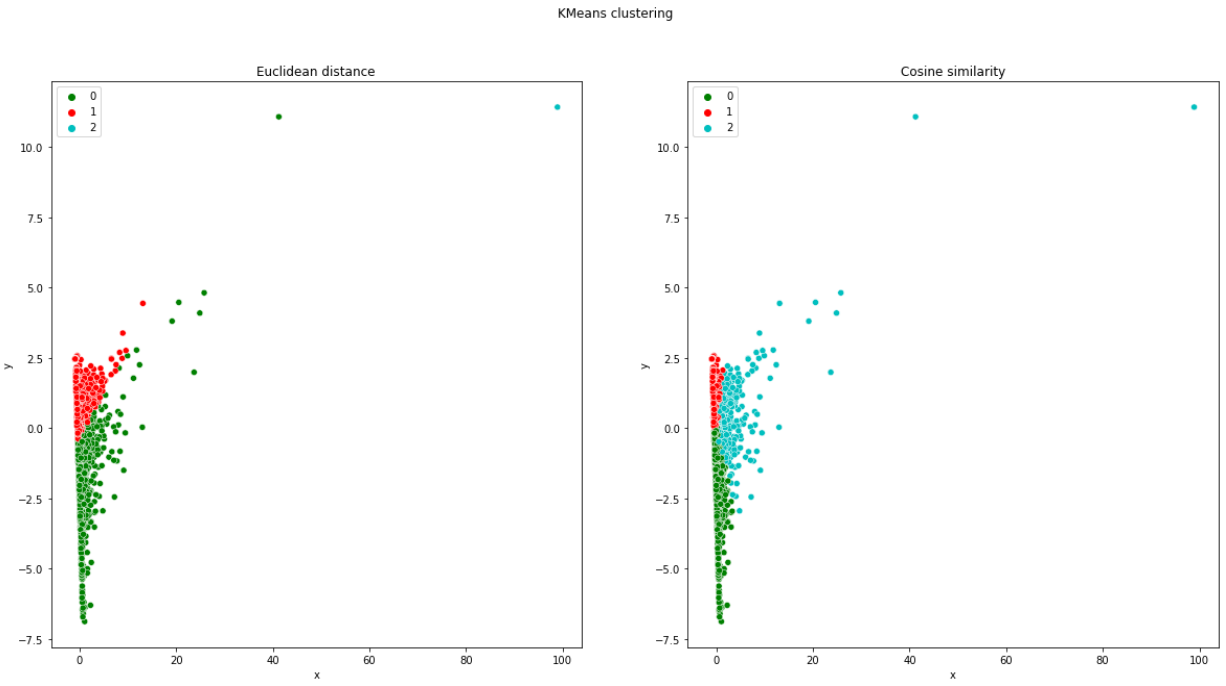
Nous réalisons une analyse en composante principale à l'aide d'un objet de la classe PCA, implémentée dans scikit-learn. Nous appelons la fonction fit_transform de la classe PCA qui va nous permettre d'ajuster le modèle et d'appliquer la réduction de dimensionnalité, à partir, et sur notre matrice XStand. Cette dernière correspond à la matrice X que l'on a standardisée.

```
The explained variance for each dimension : [0.50462694 0.33613477]
```

Variance expliquée par la PCA

Comme nous l'indique l'attribut explained_variance_ratio_ de l'objet PCA près de 85% de l'information totale du modèle est expliquée par les deux premiers composants principaux. Il est donc cohérent de n'en conserver que deux.

Ensuite, nous affichons les nouvelles coordonnées obtenues, suite à la réduction de dimensionnalité, à l'aide de la fonction plotCluster, créée par nos soins, qui utilise les outils de visualisation de données fournis par les librairies matplotlib et seaborn. Plus précisément, nous créons deux affichages distincts, un avec la segmentation obtenue avec k-means utilisant la distance euclidienne et l'autre avec k-means utilisant la similarité cosinus. La segmentation est repérable grâce au code couleur vert, rouge et cyan.



Représentation des deux stratégies de segmentation des coordonnées par k-means

Pour nous aider à interpréter ces résultats nous utilisons des métriques statistiques que l'on récupère via notre fonction displaySegmentationMetrics. Cette dernière affiche la moyenne ou la médiane des différentes caractéristiques représentant les différentes classes obtenues. On utilise la médiane ou la moyenne selon ce qui est le plus adapté à la caractéristique traitée. On obtient ces deux tableaux :

Metrics of k-means clustering, euclidean distance :

Cluster	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews	Number of books
0	3.750892	1.166739	3.944616	7.0	2.0	2.0	12.0	2266
1	4.547449	0.678119	4.904132	10.0	1.0	0.0	11.0	3364
2	3.490709	1.461977	4.000000	2134.0	644.0	1043.0	3821.0	1

Metrics of k-means clustering, cosine similarity :

Cluster	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews	Number of books
0	3.787822	1.158522	4.021254	7.0	2.0	2.0	10.0	2376
1	4.599848	0.619901	4.934080	9.0	1.0	0.0	10.0	2814
2	4.210415	0.973779	4.537415	89.0	13.0	7.0	114.0	441

Nous pouvons désormais effectuer notre interprétation de l'affichage de nos coordonnées segmentées. Premièrement, on remarque que la représentation, dans le plan 2D, de la classe représentée par la couleur rouge est très similaire, en terme de position et d'effectif, entre la segmentation via la distance euclidienne (affichage à gauche) et celle via la similarité cosinus (affichage à droite). On remarque bien cette forte similarité dans les métriques caractérisant les différentes classes ci-dessus.

De même, les coordonnées des livres appartenant à la classe de couleur verte sont groupées aux mêmes endroits quelque soit le type de k-means utilisé. Cependant, il y a un effectif relativement plus grand avec l'utilisation de la distance euclidienne. Cela se retrouve de nouveau dans le tableau des métriques affichées, qui sont très similaires à l'exception du nombre de livres.

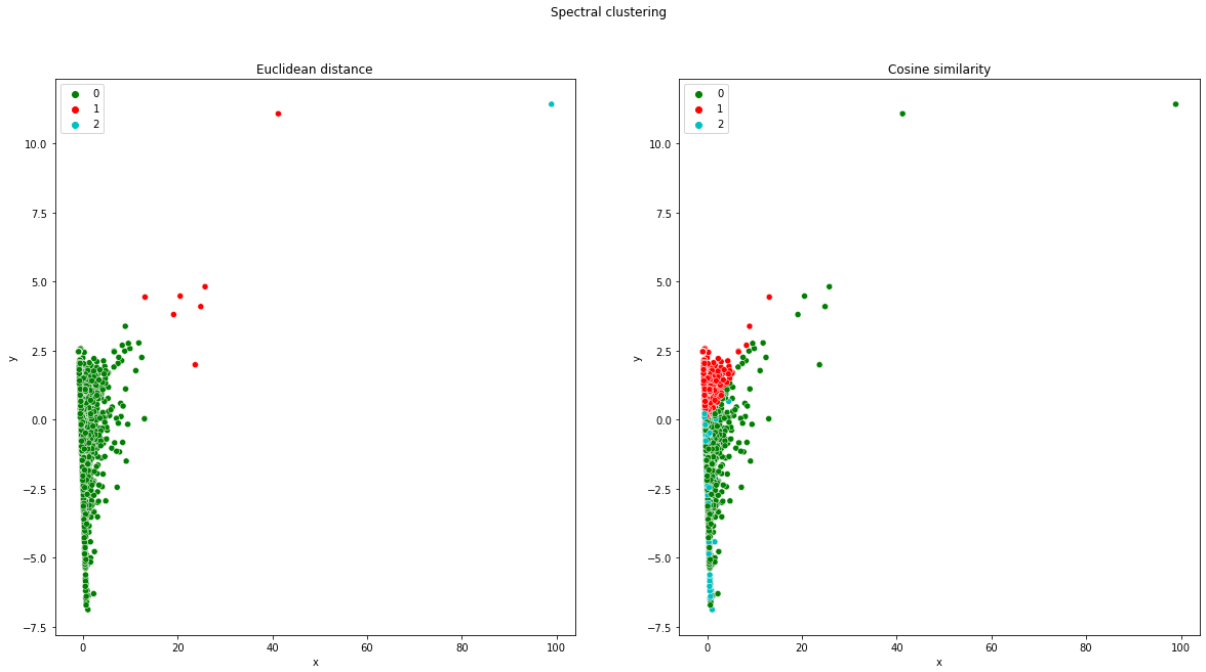
Enfin, on remarque pour la classe représentée par la couleur cyan qu'elle est significativement plus présente dans l'affichage du k-means avec la similarité cosinus que dans celui avec la distance euclidienne. On remarque des très hautes valeurs, pour la classe de couleur cyan, dans le tableau des métriques pour le k-means avec la distance euclidienne. Ces valeurs sont inhabituelles comparées à ses autres classes et possèdent visiblement un nombre d'éléments très réduit (1 livre) de par les caractéristiques originales que la classe possède. Du côté du k-means avec similarité cosinus, l'utilisation de cette mesure a eu pour effet de limiter la présence potentielle de valeurs extrêmes et ainsi de créer une classe de couleur cyan avec des métriques caractéristiques équilibrées, comme pour ses autres classes. Cela permet, comme on peut le voir, en comparaison avec k-means utilisant la distance euclidienne, d'attribuer un plus gros échantillon de livre à la classe cyan.

a.4) Effectuer une segmentation par le clustering spectral sur deux dimensions en prenant k = 3.

Pour effectuer le clustering spectral, nous devons tout d'abord effectuer une projection spectrale des données X. Pour cela, nous allons utiliser la fonction `spectralProjection` : celle-ci aura besoin d'une matrice X et d'une métrique pour pouvoir calculer la matrice de distance M en interne. Pour calculer cette dernière, on utilisera la fonction `pairwise_distances` de sklearn, cependant, celle-ci ne peut pas calculer la similarité cosinus. Par conséquent, on calculera à la place la distance cosinus et on utilisera la propriété suivante pour passer à la similarité : sim_cos = 1 - dist_cos.
Finalement, à l'aide de numpy on calcule les valeurs propres et les vecteurs propres qui leurs sont associées : cependant, on effectuera un tri en prenant en compte uniquement les valeurs propres positives pour construire la matrice $P \cdot \sqrt{D}$.

En prenant maintenant les deux premières colonnes de cette dite matrice, on obtient les nouvelles coordonnées des livres sur le plan spectral. Il suffit maintenant de faire un clustering kmeans en prenant ces points. Ici, il n'est pas nécessaire de standardiser X pour kmeans car on fournit à la fonction `spectralProjection` XStand. En effet, la projection spectrale dépend de M et étant une matrice de distance, elle dépend tout comme kmeans d'une standardisation pour obtenir de bons résultats.

Voici le résultat qu'on obtient en utilisant le plan PCA (de manière à comparer les deux méthodes) :



Représentation des deux stratégies de segmentation des coordonnées par clustering spectral

Au niveau visuel, on remarque une très grande différence concernant la distance euclidienne. En effet, le clustering spectral a considéré tout le conglomérat de points au niveau de x = 0 comme un seul cluster vert là où kmeans l'a partagé en deux clusters. Concernant le cluster numéro 2, on obtient la même chose pour les deux méthodes c'est-à-dire un très petit cluster.

Pour la similarité cosinus, on remarque que certains clusters se chevauchent (le vert et le bleu notamment) alors qu'avec kmeans les frontières des clusters étaient séparées. Cependant, il ne faut pas oublier qu'on effectue un affichage via le plan PCA et non le plan spectral : cela signifie que sur ce dernier, on aurait eu un meilleur affichage (on pourra vérifier cela dans la prochaine question).

Etudions maintenant la particularité de chacun de ces clusters :

Cluster	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews	Number of books
0	4.226783	0.874751	4.518051	9.0	1.0	1.0	11.0	5623
1	4.276686	0.899434	4.428571	957.0	150.0	79.0	1142.0	7
2	3.490709	1.461977	4.000000	2134.0	644.0	1043.0	3821.0	1

```
Metrics of spectral clustering, cosine similarity :
```

Cluster	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews	Number of books
0	3.760241	1.265320	4.101472	8.0	2.0	2.0	13.0	1902
1	4.594602	0.649958	4.977499	10.0	1.0	0.0	11.0	2911
2	4.002156	0.767505	3.850244	7.0	2.0	0.0	9.0	818

Nous pouvons effectuer ici, globalement, les mêmes remarques et interprétations qu'avec les segmentations k-means. Cela se traduit par le fait que pour la distance euclidienne, le classement des livres se fait principalement par le nombre de revues avec des valeurs extrêmes qui sont rares mais qui imposent un classement à part entière. En conséquence nous avons très peu d'individus appartenant à deux des trois classes.
Tandis qu'avec la similarité cosinus les valeurs sont équilibrées et l'on peut remarquer que le classement se fait principalement selon la moyenne et les autres métriques la reliant.
De plus, les moyennes sont suffisamment distinctes pour tenter d'attribuer aux différentes classes une appartenance à une catégorie de score. Cependant, le nombre d'excellents scores comparé aux nombre de scores moyens et faibles est trop élevé, dans toutes les classes, pour pouvoir attribuer ces classes à une autre catégorie que celle des livres avec un score élevé.

a.5) Utiliser la métrique du coefficient de silhouette et de l'information mutuelle pour évaluer la qualité de la segmentation effectuée.

Le coefficient de silhouette est une mesure de qualité concernant un clustering. Elle permet de savoir si un point a bien été classé ou pas grâce à un calcul entre la cohésion et la séparation de celui-ci. Une valeur négative signifie que le point est plus proche du groupe l'avoisinant que le sien : il est donc mal classé. Au contraire, une valeur positive signifie qu'il est très proche de son groupe et il est donc bien classé. Finalement, une valeur proche de 0 signifie qu'il est entre deux clusters. Pour calculer cette mesure, nous allons utiliser la fonction `getSilhouetteScore` utilisant en interne la fonction `silhouette_score` de sklearn qui effectuera une moyenne sur tous les points. `getSilhouetteScore` a besoin des labels pour chaque cas de figure mais aussi des matrices sur lesquels nous avons effectué kmeans respectivement. Ces matrices vont être stockées sous forme de liste dans le paramètre XList : par exemple, pour calculer le score silhouette du clustering spectral avec la similarité cosinus, nous allons fournir les coordonnées spectrales de la similarité cosinus. En effet, il faut fournir les mêmes matrices qu'on a fourni à kmeans car

pour obtenir le score silhouette il faut aussi effectuer des mesures de distance se basant sur les données fournies.

Par conséquent, on précise aussi qu'on a utilisé la bonne métrique lors du calcul de la silhouette, i.e que pour la distance euclidienne nous avons fourni cette même distance à la fonction `silhouette_score` et pour la similarité nous avons fourni la distance cosinus. En effet, le calcul de la silhouette se base sur une distance et non sur une similitude qui peut renvoyer des valeurs négatives.

L'information mutuelle quant à elle permet de mesurer la concordance de deux affectations, en ignorant les permutations. Une valeur très proche de 1 signifie une parfaite concordance entre les deux affectations. Pour cela, nous avons besoin de fournir à quel cluster chaque livre appartient en se basant sur une croyance/information extérieure. Par conséquent, pour rester cohérent avec le TP1, nous allons utiliser la moyenne bayésienne. En effet, comme nous l'avons vu durant le TP1, elle permet d'obtenir des résultats satisfaisants pour grouper les livres. Nous utiliserons exactement les mêmes paramètres pour effectuer ce calcul à l'aide de la fonction `getTrueLabels`, i.e : C = 20, m = 4.0, a = 4.0 et b = 3.5.

Pour rappel :

- C : le nombre d'avis fixé comme seuil de confiance
- m : A priori sur la moyenne des notes
- [a;b] : l'intervalle dans lequel un livre est catégorisé comme mauvais, moyen ou bon.

Maintenant que nous avons nos "vrais" labels, on peut utiliser la fonction `getNMI` utilisant en interne la fonction sklearn `normalized_mutual_info_score`. Ici, nous allons calculer plutôt la version normalisée de l'information mutuelle permettant d'avoir un résultat entre [0;1]. En outre, nous n'allons pas utiliser la version ajustée non plus car nous avons un nombre de clusters très petit (k = 3).
Finalement, voici les résultats que nous obtenons :

	Distance euclidienne		Similarité cosinus	
Approche	Silhouette	Infor. Mut. Norm.	Silhouette	Infor. Mut. Norm.
KMeans	0.40	0.55	0.59	0.39
Spectrale	0.97	0.003	0.65	0.38

On remarque que l'approche spectrale a une meilleure silhouette que k-means et cela est sans surprise. En effet, si on compare visuellement les deux méthodes dans le cas de la distance euclidienne, on remarque que pour kmeans les frontières entre les clusters se chevauchent là où pour le clustering spectral, les clusters sont très bien séparés. Cependant, concernant l'information mutuelle, kmeans produit un meilleur résultat là où le clustering spectral produit une affectation totalement différente que celle basée sur la moyenne bayésienne (car résultat très proche de 0). Par ailleurs, on remarque que les résultats dans le cas de la similarité cosinus sont plutôt proches. Finalement, pour les deux approches, la similarité cosinus produit des résultats satisfaisants et un bon compromis concernant les deux métriques de performance.

b.1) Quel serait le risque de prendre aléatoirement un sous-ensemble de données pour effectuer les tâches de la partie a) ?

Le risque de prendre aléatoirement un sous-ensemble de données pour effectuer les tâches de la partie a) est que certaines catégories soient sous représentées voire pas représentées du tout et que d'autres soient par conséquent sur représentées. Cela a une influence sur le travail que l'on effectue sur ces données et peut, par exemple dans notre cas, biaiser les étapes d'exploration des données, d'analyse prédictive et par conséquent les interprétations que l'on en ferait.

b.2) En procédant par une sélection stratifiée, s'assurer que toutes les catégories soient représentées dans notre sous-ensemble.

Comme nous avons construit la matrice X avec p livres aléatoires tel que $p = C_{min} \times 3$ avec C_{min} le nombre minimum de livres entre les 3 catégories présentées dans l'énoncé, il est très facile d'obtenir une version stratifiée. Pour cela, il suffit de grouper les livres présents dans dfXAll (le data frame contenant tous les livres) par catégorie puis de choisir C_{min} livres aléatoirement pour chacune des catégories. Pour vérifier si on a bien une stratification de notre sous-ensemble, on peut vérifier si la somme des livres appartenant à chacune des catégories équivaut bien à p (cela est effectué par l'assert dans le code).

b.3) Refaire les étapes a-2) à a-5). Comparer les résultats obtenus dans les tableaux contenant les métriques du coefficient de silhouette et de l'information mutuelle.

Comme nous avons créé des fonctions pour la majorité des questions, il est très simple d'effectuer toutes les étapes de la question a). En effet, il suffit de faire exactement la même chose mais avec la matrice X stratifiée.

Voilà les résultats que nous obtenons :

- **PCA :**

Nous réutilisons la classe PCA avec la fonction fit_transform pour ajuster le modèle et appliquer la réduction de dimensionnalité, à partir de, et sur notre matrice XStandStrat obtenue suite à la sélection stratifiée et une standardisation.

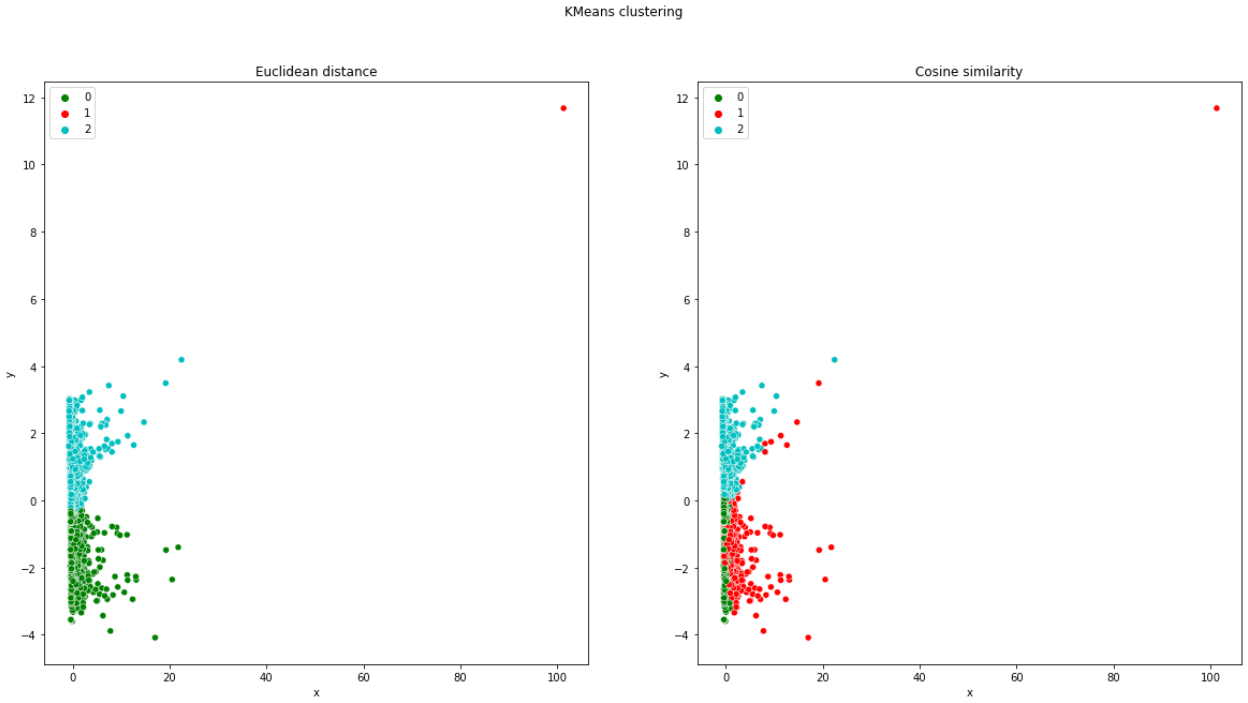
```
The explained variance for each dimension : [0.47236577 0.34372357]
```

Variance expliquée par la PCA

Ici l'attribut explained_variance_ratio de l'objet PCA indique près de 82% de l'information totale du modèle expliquée par les deux premiers composants principaux. De nouveau on peut donc utiliser cette projection pour afficher nos données et les clusters.

- **KMeans :**

En appliquant l'algorithme k-means avec la distance euclidienne et la similarité cosinus sur les données avec sélection stratifiées, nous obtenons ces nouveaux affichages :



Représentation des 2 stratégies de segmentation par k-means avec sélection stratifiée

Metrics of k-means clustering, euclidean distance :

Cluster	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews	Number of books
0	4.186082	0.800031	4.303709	7.0	2.0	0.0	9.0	2211
1	2.419686	1.583406	2.000000	990.0	350.0	2165.0	3505.0	1
2	2.784078	1.233324	2.551185	2.0	3.0	3.0	8.0	3419

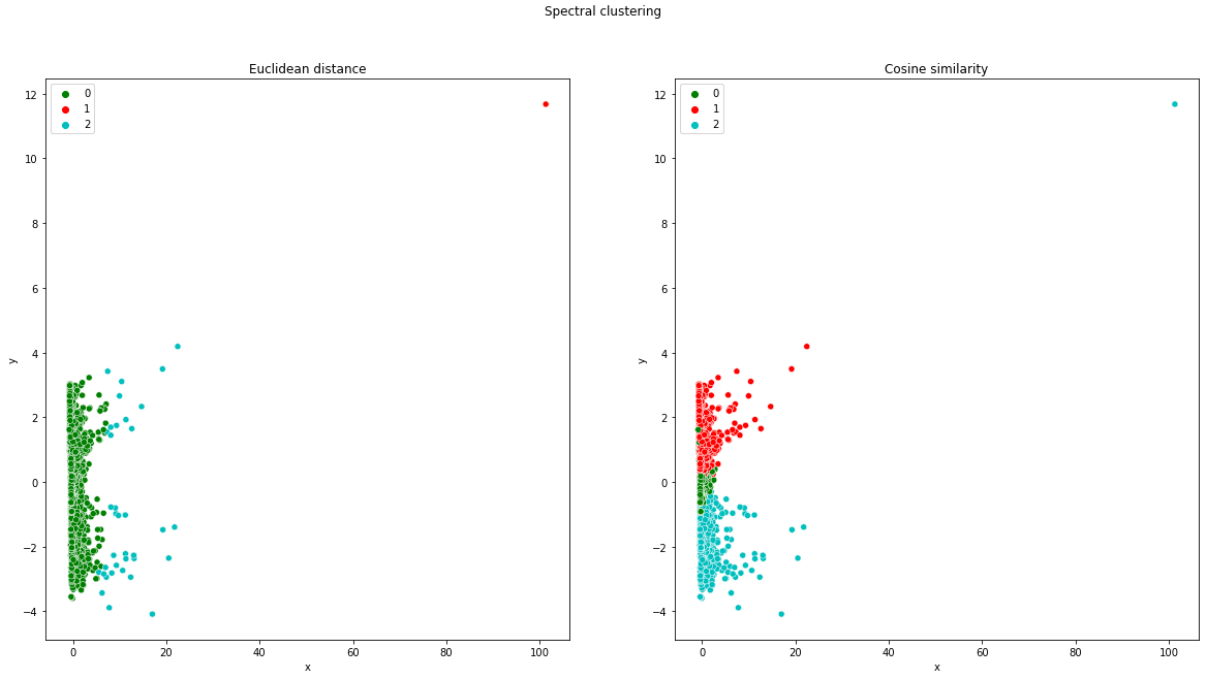
Metrics of k-means clustering, cosine similarity :

Cluster	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews	Number of books
0	3.990928	0.682549	3.903157	4.0	2.0	0.0	7.0	1869
1	4.077497	1.124551	4.508424	15.0	2.0	2.0	21.0	831
2	2.705276	1.288639	2.455988	2.0	2.0	4.0	8.0	2931

On remarque visuellement que les démarcations des classes de couleurs cyan et verte sont encore plus nettes qu'avec les livres pris aléatoirement. Cependant, il y a toujours le problème de biais d'échelle présent pour la distance euclidienne. Du côté de k-means avec similarité cosinus les classe 0 et 1 sont devenues très proches au niveau de la moyenne de scores, cela se remarque visuellement par la confusion des deux groupes. Par contre, elles peuvent être départagées grâce à leur nombre de reviews. Elles semblent toutes deux correspondre à des scores proches d'excellents, mais elles se différencient car la classe 1 possède un nombre d'avis plus conséquent que la classe 0. La classe 2 quant à elle semble désigner les livres avec des scores principalement faibles et relativement peu d'avis. On saura remarquer visuellement le nombre conséquent de livres appartenant à cette classe.

- **Spectrale :**

En appliquant l'algorithme de clustering spectral avec la distance euclidienne et la similarité cosinus sur les données avec sélection stratifiée, nous obtenons ces nouveaux affichages :



Représentation des 2 stratégies de segmentation par clustering spectral avec sélection stratifiée

Metrics of spectral clustering, euclidean distance :

Cluster	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews	Number of books
0	3.331905	1.062750	3.234358	3.0	2.0	2.0	8.0	5594
1	2.419686	1.583406	2.000000	990.0	350.0	2165.0	3505.0	1
2	3.764263	1.127264	4.027778	252.0	43.0	46.5	344.5	36

Metrics of spectral clustering, cosine similarity :

Cluster	Average score	Std	Median	Number of excellent ratings	Number of medium ratings	Number of low ratings	Number of reviews	Number of books
0	3.297648	0.785094	2.963519	2.0	4.0	1.0	6.0	1165
1	2.705569	1.335343	2.451394	2.0	2.0	4.0	8.0	2726
2	4.344522	0.823223	4.658046	9.0	1.0	1.0	11.0	1740

Ici encore, les résultats obtenus grâce à la distance euclidienne sont biaisées. Cependant, cette fois il semble que nous ayons les résultats de segmentation les plus intéressants pour nommer les différentes classes. En effet, du côté du clustering spectral avec similarité cosinus on observe trois classes dont le nombre d'avis et de livres par classe est relativement équilibré et pour lesquelles les scores moyens sont nettement distincts. Nous pouvons donc tenter de leur attribuer une catégorie de score précise. La classe 0 correspond aux livres avec un score moyen, la classe 1 aux livres avec un faible score et la classe 2 aux livres avec un score élevé.

- **Performance :**

	Distance euclidienne		Similarité cosinus	
Approche	Silhouette	Infor. Mut. Norm.	Silhouette	Infor. Mut. Norm.
KMeans	0.39	0.41	0.52	0.29
Spectrale	0.94	0.008	0.77	0.45

Finalement, au niveau des performances, on remarque une légère baisse concernant k-means alors que le clustering spectrale a eu une amélioration. En effet, en termes de compromis, ce dernier combiné avec la similarité cosinus produit de très bons résultats.

b.4) Si l'on associe les étiquettes respectives l1, l2 et l3 aux catégories 1-2, 3 et 4-5. Quelle stratégie utiliser pour retrouver les étiquettes des données restantes.

Pour retrouver les étiquettes des données restantes, c'est-à-dire celles qui n'ont pas encore été classées, nous pouvons nous baser sur les centroïdes, définies lors de la segmentation k-means effectuées sur les données déjà classées. En effet, nous pouvons mesurer la similarité cosinus entre chaque données restantes et les différents centroïdes et faire correspondre, à chaque donnée, la classe pour laquelle la mesure obtenue à la plus grande similarité. Nous utiliserions la similarité cosinus et pas la distance euclidienne afin de ne pas avoir de problème d'échelle de grandeur.