Bi-clustering

[II.2415] Algorithmique et programmation avancée

DUMONT William - 8639
SOUVANHEUANE Gaëlle - 8654

Mercredi 30 Mai 2018

Table des matières

Introduction

Le **Bi-clustering** (ou **Co-clustering**) est une technique d'exploration de données qui partitionne simultanément les lignes et les colonnes d'une matrice de données. Cette technique permet ainsi de rassembler les données similaires en groupes appelés « **clusters** ».

Exemple :

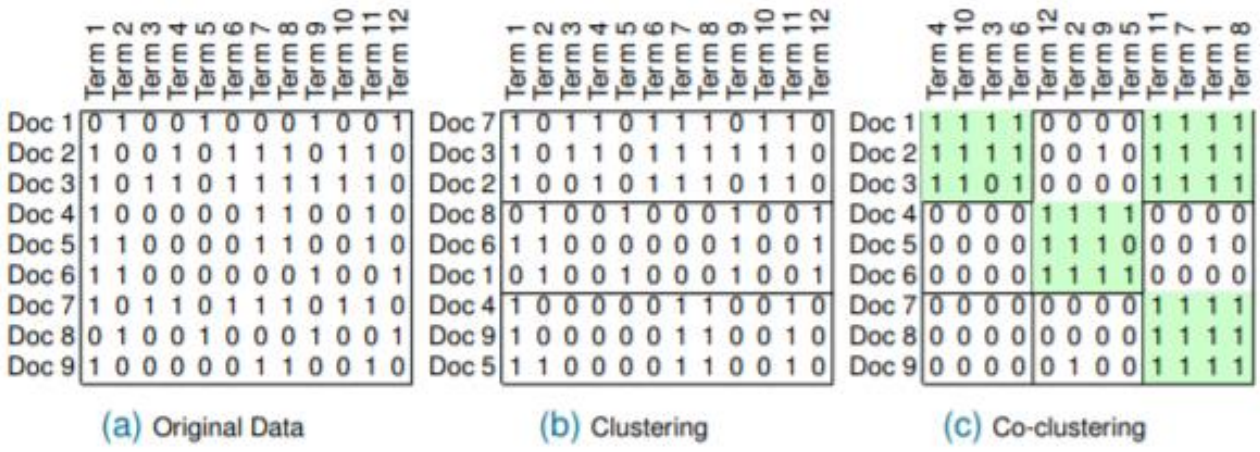

Figure 1- https://egc18.sciencesconf.org/data/pages/prix_EGC_Melissa_1.pdf

La *Figure 1* schématise le fonctionnement du **Co-clustering**. Cette technique est une extension du Clustering, qui rassemble les lignes de la matrice qui sont similaires *(b)*, puisqu'elle consiste à regrouper également les colonnes similaires afin de générer des clusters *(c)*.

I. Choix du langage

Nous avons choisi d'implémenter l'algorithme de bi-clustering en **Python** plutôt qu'en Java ou en C pour les raisons suivantes :

- Les librairies *numpy* et *scipy* offrent des fonctions intéressantes pour manipuler les matrices.
- La librairie *sklearn* propose des algorithmes clustering ainsi que des méthodes pour les tester ce qui nous a permis de comparer et d'évaluer notre algorithme
- La libraire *matplotlib* permet d'afficher les données simplement sous forme de graphe
- La syntaxe de Python (typage dynamique, compréhension de listes etc.…) est particulièrement adaptée aux mathématiques car elle permet d'avoir un code concis et donc de pouvoir se focaliser plus facilement sur l'algorithme en lui-même.

- Le sujet n'imposant pas de contraintes de performances, le fait que Python soit plus lent que le Java ou le C n'est donc pas un problème dans notre cas.

II. Choix de l'algorithme

Découvrant le clustering, nous avons choisi d'implémenter l'algorithme Spectral Co-clustering car c'est un algorithme de bi-clustering populaire pour lequel on peut facilement trouver des ressources en ligne.

Explication de notre algorithme Spectral Co-clustering :

Soit A la matrice originale de donnée

- Calculer R la matrice diagonale des degrés des lignes de A
- Calculer C la matrice diagonale des degrés des colonnes de A
- Calculer la matrice $A_{norm} = R^{-\frac{1}{2}} A C^{-\frac{1}{2}}$
- Calculer la SVD (Décomposition en valeurs singulières) de A_{norm} pour récupérer U et V, les matrices qui contiennent les vecteurs propres.
- Construire la matrice Z telle que $Z = \begin{pmatrix} R^{-\frac{1}{2}} U \\ C^{-\frac{1}{2}} V \end{pmatrix}$ en écartant la paire de vecteurs propres correspondant à la plus grande valeur propre.
- Effectuer n K-means sur Z
- Réarranger les clusters pour avoir une structure diagonale

III. Difficultés

Comme indiqué précédemment, nous avons été confrontés pour la première fois à un problème de clustering. Nous avons donc dû nous familiariser avec la théorie. Cette dernière étant assez complexe nous avons dû nous renseigner sur divers sujets tels que les valeurs propres et les vecteurs propres d'une matrice, la décomposition en valeurs singulières et le Laplacien.

1ère difficulté : Documentation majoritairement complexe car la plupart des ressources sont issues de thèses.

Pour implémenter l'algorithme nous nous sommes basés dans un premier temps sur l'algorithme théorique suivant :

Box 1. Spectral biclustering: the algorithm.

U: conditions; V: genes
$D_{N\times M}$: gene expression matrix.
Compute $R = \text{diag}(D \cdot \mathbf{1}_M)$ and $C = \text{diag}(\mathbf{1}_M{}^T D)$
Compute SVD of $R^{-1/2} D C^{-1/2}$
Discard the pair of eigenvectors corresponding to the largest eigenvalue
For each pair of eigenvectors u,v of $R^{-1}DC^{-1}D^T$ and $C^{-1}D^TR^{-1}D$ with the same eigenvalue do:
    Apply K-means to check the fit of u and v to stepwise vectors
    Report the block structure for the p couples u, v with the best stepwise fit.

Figure 2 - https://pdfs.semanticscholar.org/02ff/5813dcd8b4752b78dfdffb1a875bd95d0827.pdf

2ème difficulté : Traduire l'algorithme théorique en Python

La majeure partie de l'algorithme a été faite à partir de la version théorique, nous l'avons ensuite complété via le code source de la librairie *sklearn* (https://github.com/scikit-learn/scikit-learn/blob/a24c8b46/sklearn/cluster/bicluster.py) ainsi qu'une implémentation en *matlab* (http://adios.tau.ac.il/SpectralCoClustering/).

À ce stade nous avons un algorithme fonctionnel qui utilise cependant l'algorithme K-means de la libraire *sklearn*. Nous avons alors décidé d'implémenter nous-même cet algorithme. Pour cela nous avons fait des recherches sur la théorie puis nous avons intégré l'implémentation suivante : https://gist.github.com/iandanforth/5862470.

L'algorithme étant très aléatoire nous avons décidé de faire un K-means itératif et de prendre le meilleur résultat à chaque fois.

3ème difficulté : Formatter les données pour évaluer notre algorithme à l'aide de de la fonction *consensus_score* de la librairie sklearn.

IV. Tests

Pour tester notre algorithme, nous avons utilisé la fonction *make_bicluster* de la librairie *sklearn* afin de générer des clusters (*Figure 3*) sur une matrice de données 300x300 avec un bruit de 5 pour complexifier la tâche à l'algorithme. Nous avons ensuite mélangé ces données (*Figure 4*) puis nous avons utilisé notre algorithme sur cette matrice mélangée afin d'obtenir le résultat affiché sur la *Figure 5*.

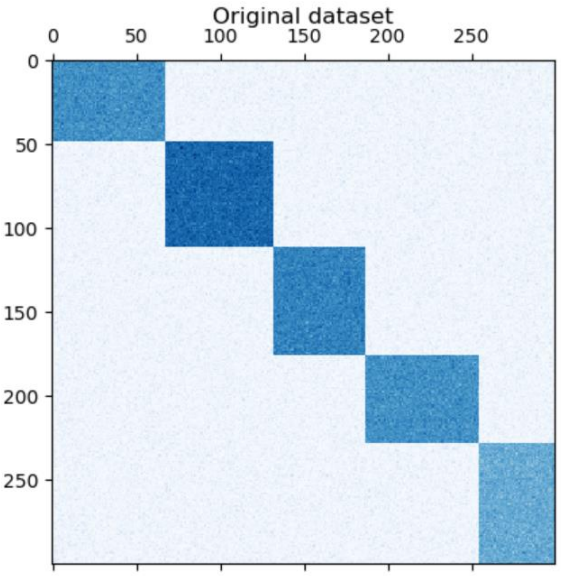

Figure 3 – Matrice de données originale

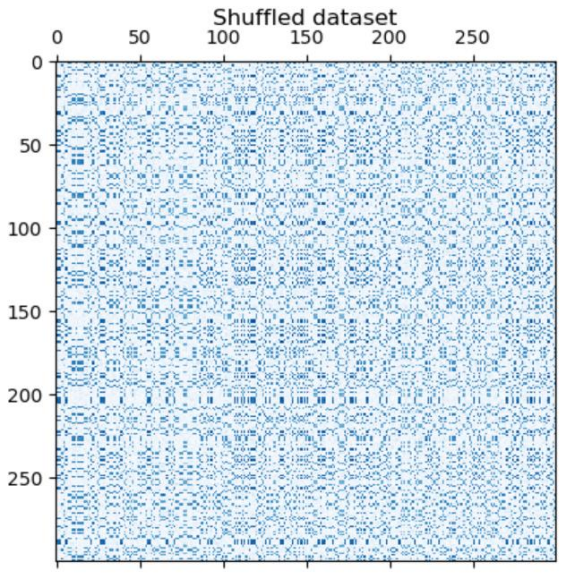

Figure 4 – Matrice de données mélangée

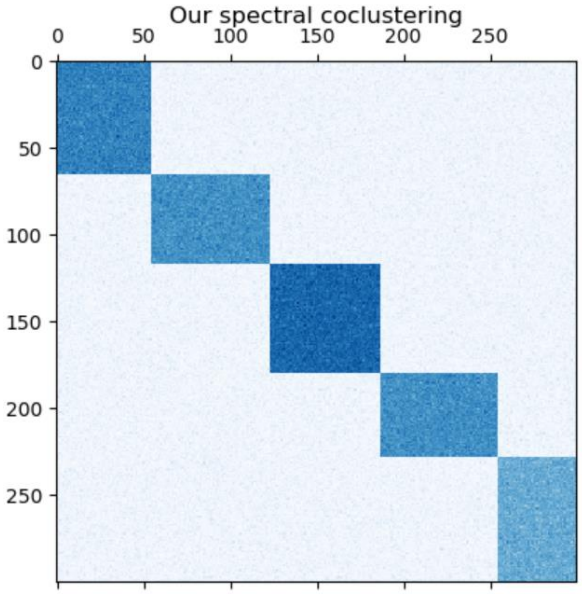

Figure 5 – Matrice à la sortie de notre algorithme

Nous observons que notre algorithme a retrouvé les clusters de la matrice d'origine de la *Figure 3*.

V. Performances

Nous avons effectué un benchmark afin de mesurer les performances de notre algorithme. Nous l'avons comparé avec la fonction *SpectralCoclustering* de la librairie *sklearn* en termes de temps d'exécution (*Figure 6*) et d'exactitude (*Figure 7*) pour trois itérations du K-means.

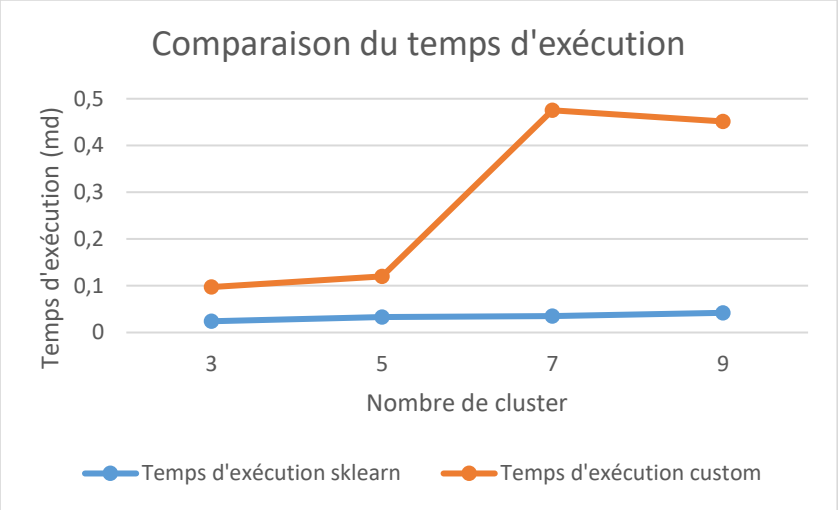

Figure 6 – Graphe de comparaison du temps d'exécution des algorithmes

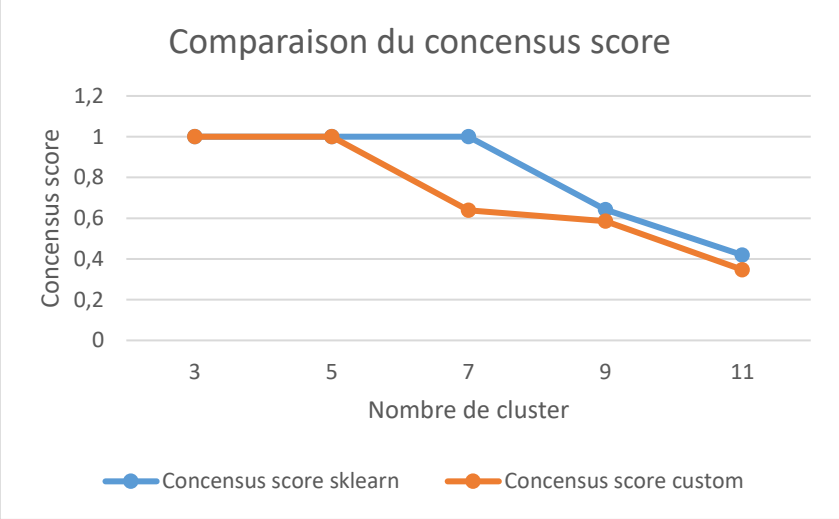

Figure 7 – Graphe de comparaison de l'exactitude du bi-clustering

Nous remarquons que le temps d'exécution de notre algorithme croît plus rapidement que celui de l'algorithme de sklearn. Nous pouvons expliquer cela par notre algorithme K-means qui n'est pas bien optimisé.

Concernant l'exactitude de notre algorithme, elle semble être similaire à celle de sklearn dans l'ensemble. Cependant la précision de notre algorithme dépend du nombre d'itération de

K-means : plus ce nombre est grand, plus l'algorithme a des chances de trouver une meilleure solution mais le temps d'exécution en sera impacté.

Complexité : $O(i.e.m.n.k)$

i : nombre de k-means à effectuer

e : déplace les centroïdes des clusters e fois tant que l'erreur est différente de 0 (bornée à 300 pour éviter une éventuelle boucle infinie)

m : nombre de colonnes de la matrice initiale

n : nombre de lignes de la matrice initiale

k : nombre de clusters

VI. Lancer le programme

L'algorithme est disponible sur le répertoire git suivant : https://github.com/wildum/SpectralCoClustering. Il nécessite d'avoir Python3 ainsi que les librairies *numpy*, *matplotlib*, *scipy* et *sklearn* d'installées. La librairie *sklearn* n'est pas utilisée par notre spectrale co-clustering mais nous l'utilisons pour créer les données, évaluer notre algorithme et comparer avec le spectrale co-clustering fourni. La commande pour le lancer est « py main.py ».

Bibliographie

- https://www.youtube.com/watch?v=mnDC6hWWbwY&t=2s
- http://www.escience.cn/system/download/85300
- http://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html
- https://fr.wikipedia.org/wiki/Classification_double
- https://en.wikipedia.org/wiki/Biclustering
- https://fr.wikipedia.org/wiki/Partitionnement_spectral
- https://fr.wikipedia.org/wiki/D%C3%A9composition_en_valeurs_singuli%C3%A8res
- Powerpoint « La Décomposition en Valeurs Singulières – Ensimag »
- https://www.youtube.com/watch?v=P-LEH-AFovE
- https://www.youtube.com/watch?v=mBcLRGuAFUk
- https://www.youtube.com/watch?v=P5mlg91as1c&t=2s
- https://www.youtube.com/watch?v=cOUTpqlX-Xs
- https://egc18.sciencesconf.org/data/pages/prix_EGC_Melissa_1.pdf
- http://www.kemaleren.com/post/spectral-biclustering-part-2/
- https://calculatedcontent.com/2012/10/09/spectral-clustering/
- https://tel.archives-ouvertes.fr/tel-00935278/document
- http://cs-people.bu.edu/evimaria/cs565-13/co-clustering.pdf
- https://hal.archives-ouvertes.fr/hal-01469546/document
- https://pdfs.semanticscholar.org/02ff/5813dcd8b4752b78dfdffb1a875bd95d0827.pdf
- http://math.bme.hu/~marib/prezentacio/bollabecs.pdf
- https://pdfs.semanticscholar.org/ff97/09af96c13c523a2825fbced1eada7c843ac2.pdf
- http://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html
- https://fr.wikipedia.org/wiki/Matrice_laplacienne
- https://en.wikipedia.org/wiki/Laplacian_matrix
- https://gist.github.com/iandanforth/5862470
- https://github.com/scikit-learn/scikit-learn/blob/a24c8b46/sklearn/cluster/bicluster.py
- http://statweb.stanford.edu/~owen/courses/321/readings/spect-biclust.pdf
- https://mubaris.com/2017/10/01/kmeans-clustering-in-python/
- http://adios.tau.ac.il/SpectralCoClustering/