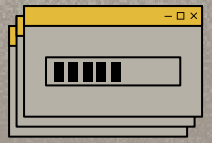
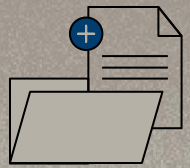
UFR UVSQ



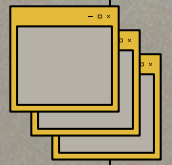
A new Topological

Biclustering using Topological Maps



BiTM Model

TABLE OF CONTENTS



Notre objectif

Durant ce projet nos objectifs étaient:

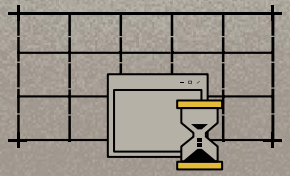
- Reprendre une ancienne version de l'algorithme **BITM** écrit en Scala .
- Comprendre son fonctionnement .
- Réécrire l'algorithme et faire en sorte qu'il compile avec les bonnes versions.
- Corriger certains bugs et modifier certaines fonctions et formules pour être en adéquation avec l'algorithme.
- Ajouter certaines fonctionnalités.

Pour ce faire, il était nécessaire d'introduire le paradigme MapReduce et Spark au projet, pour faciliter le traitement distribué sur des architecture modernes comme le cloud.

01

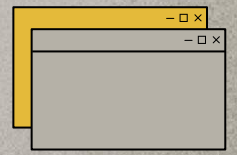
INTRODUCTION

Clustering et Biclustering

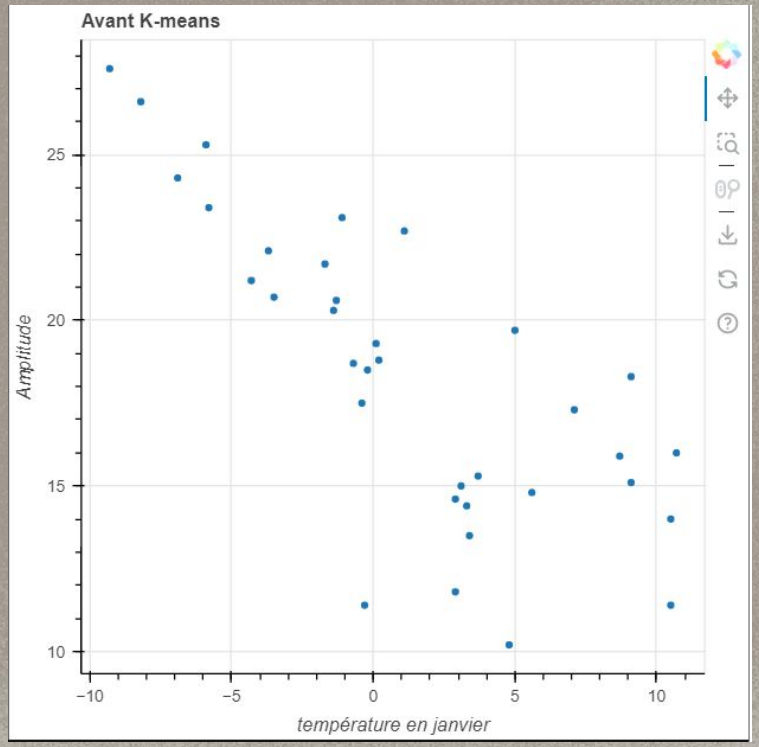


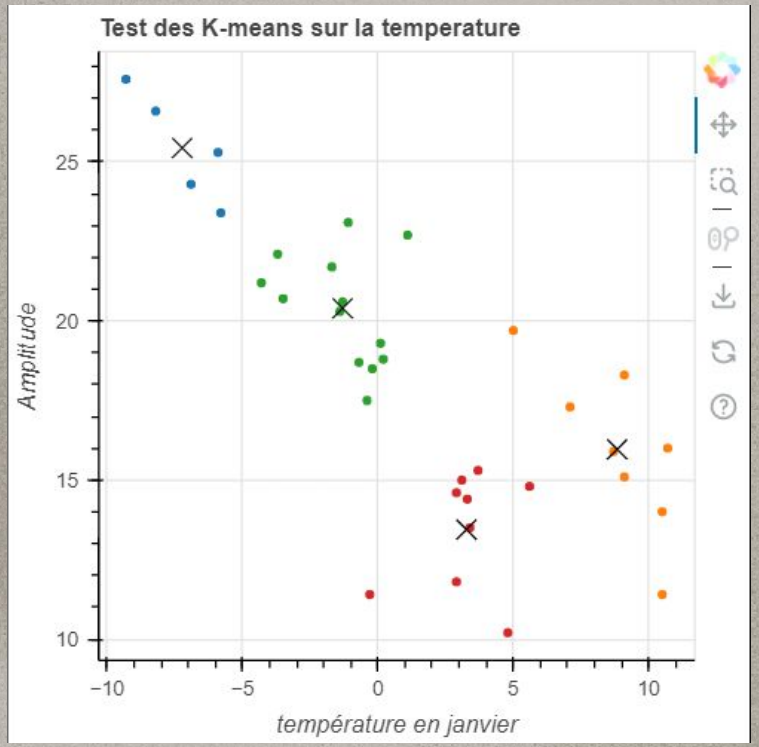
Clustering

- Le **clustering** est une technique d'apprentissage **non supervisé** qui regroupe des objets similaires dans des groupes ou “Clusters”.
- L’objectif est de **maximiser** la similarité intra-cluster et minimiser la similarité inter-cluster.
- **Avantages** : Scalable car les calculs peuvent se faire en parallèle.
- **Inconvénients** : Les valeurs aberrantes peuvent fausser les centroïdes des clusters et ce type d’algorithme est peu performant pour dimension très grande.



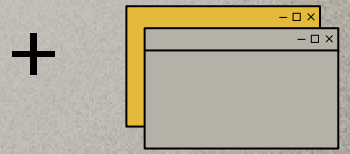
Clustering

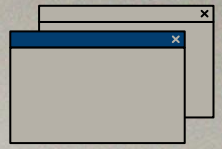

Avant K-means
Amplitude
température en janvier
25
20
15
10
−10
−5
0
5
10


Test des K-means sur la temperature
Amplitude
température en janvier
25
20
15
10
−10
−5
0
5
10

Co-clustering

- Le **co-clustering** est une variante du clustering qui organise simultanément les données en lignes et en colonnes.
- Cette méthode est particulièrement **utile** pour les matrices de données de grande taille, où les relations entre les objets peuvent être révélées en regroupant à la fois les objets “**lignes**” et les caractéristiques “**colonnes**”.
- Le **co-clustering** permet de trouver des **sous-matrices** cohérentes qui partagent des motifs similaires.



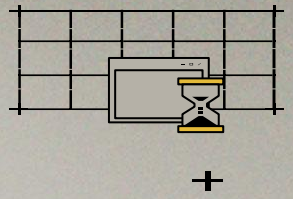


Co-clustering vs Clustering

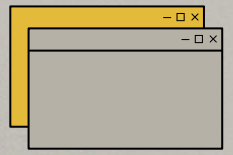
- Le **clustering** regroupe des objets similaires dans un même cluster. Ils sont généralement représentés par des points de données dans un espace vectoriel. Le **co-clustering** regroupe simultanément les lignes et les colonnes d'une matrice, créant une structure de bloc cohérente.

- Le clustering fonctionne avec des données simples. Le co-clustering est adapté aux données complexes et de grande taille. Il identifie des sous-structures cohérentes dans les données.

- Exemple d'application :
  - Clustering : Algorithme K-means
  - Co-Clustering : Algorithme BITM (biclustering utilisant des cartes topologiques)



Exemple Pratique

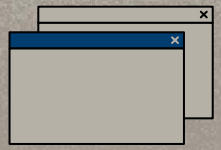


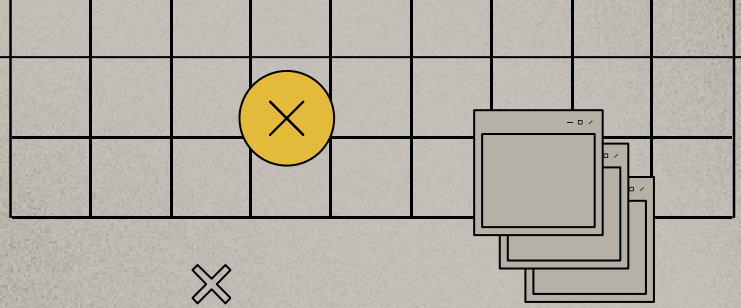
Clustering

Les entreprises l’utilisent pour segmenter leur clientèle en groupes homogènes en fonction de leurs de leurs préférences pour identifier des groupes de clients qui ont des préférences similaires afin cibler leurs préférences.

Co-clustering

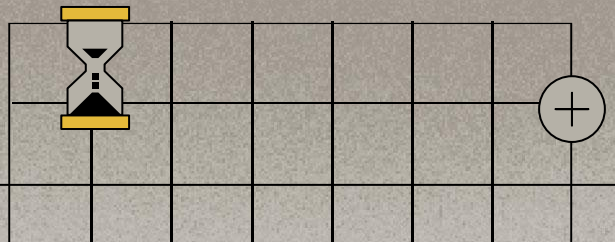
Peut être utilisé pour identifier des groupes de gènes exprimés de manière similaire dans les cellules d'un patient atteint d'une maladie particulière, mais qui ne sont pas exprimés de la même manière chez les patients non atteints.
Cela peut aider à identifier les voies biologiques impliquées dans la maladie et à développer de nouveaux traitements.





L’algorithme

BITM



Entrées :

Les données D
Les prototypes G (initialisation)
tf : le nombre maximal d'itérations

Sorties :

La matrice d'affectation Z,W
Les prototypes G

Initialisation :

Affecter une valeur initiale à tf et à G

Tant que t ≤ tf, faire :
Pour tous les xi ∈ D, faire :

Phase d'affectation des observations :

Assigner chaque observation **xi** au prototype le plus proche **gk** en utilisant la fonction d'affectation définie dans l'équation

$$\phi_w(x^j) = \arg\min_l \sum_{i=1}^{N} \sum_{k=1}^{K} z_i^k (\pi_r^l x_i^j - g_r^l)^2$$

SLIDE EXPLICATIVE

Phase d'affectation des caractéristiques :

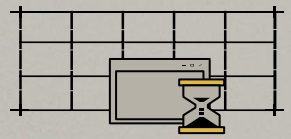
Assigner chaque caractéristique **xj** au prototype le plus proche **gl** en utilisant la fonction d'affectation définie dans l'équation:

$$\phi_w(x^j) = \arg\min_l \sum_{i=1}^{N} \sum_{k=1}^{K} z_i^k (\pi_r^l x_i^j - g_r^l)^2$$

Phase de quantification :

Mettre à jour les vecteurs de prototype en utilisant l'expression définie dans l'équation:

$$g_r^l = \frac{\sum_{k=1}^{K} K^T(\delta(k,r)) \sum_{x_i^j \in B_k^l} x_i^j}{\sum_{k=1}^{K} K^T(\delta(k,r)) \sum_{x_i^j \in B_k^l}}$$



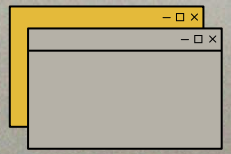
Fin de la boucle pour tous les xi ∈ D

Mettre à jour la valeur de T
{ T varie de Tmax jusqu'à Tmin }

t++

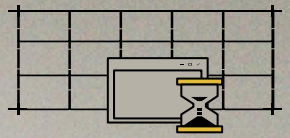
Fin de la boucle tant que

Retourner la matrice d'affectation Z,W et les prototypes G



03

UpDate PROGRAMME



TopoFactor

Calcul du voisinage avec la formule donnée.

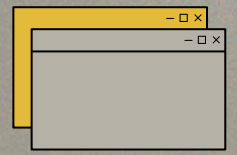
Avec **t** l'itération courante et **itérations** le nombre total d'itérations .

$$T = T_{max}\left(\frac{T_{min}}{T_{max}}\right)^{\frac{t}{iterations}}$$

Nous utilisons la fonction ci-dessous pour définir le voisinage :

$$K^T\left(\sigma\left(c_r, c_s\right)\right) = e^{\frac{-\sigma(c_r, c_s)}{T}}$$

T représente la température qui diminue graduellement en fonction de **Tmax** et **Tmin**, afin de contrôler la taille du voisinage qui influence une cellule donnée sur la carte.



MapperRowAffectation

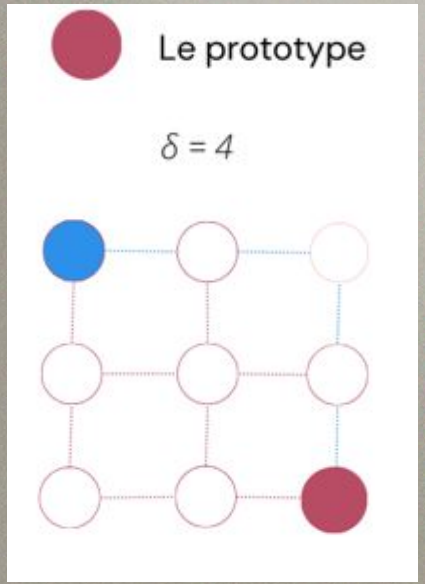
Calcul d'une distance échiquier:

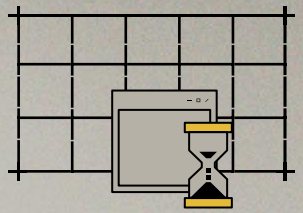
La clé de l'algorithme BiTM est le calcul des distances

On a donc changé la fonction de distance, afin de calculer celle-ci entre le meilleur neurone et l'ensemble des clusters de tel sorte à respecter le formule:

abs(f(rowbestN) - f(i) +g(rowbestN)-g(i))

Avec **f** une fonction qui récupère les coordonnées en ligne et **g** une fonction qui récupère les coordonnées en colonne



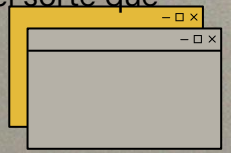


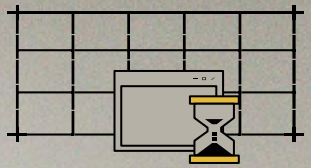
Clusters Colonnes

L'exécution du code de **départ** prend un nombre de clusters prédéfini entre lignes et colonnes. L'affectation aléatoire pour chaque colonne pouvait donc aller de **0** à **nombre de clusters** . Idem pour les lignes.

Comme il n'y **pas forcément** de raison pour que le nombre de cluster soit le même entre les lignes et les colonnes et pour éviter d'avoir des clusters **vides** sur les colonnes on a du :

- Créer une nouvelle **variable** pour spécifier le nombre de clusters sur les colonnes de tel sorte que l'affection aléatoire se fasse entre **0** et cette **nouvelle variable**.





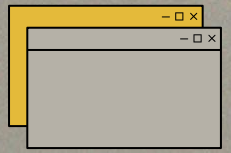
Clusters Lignes

Quant à l'**affectation des lignes**

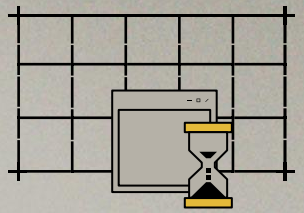
Le code de départ bouclait sur le nombre de **neurones sur les lignes** pour rechercher le meilleur neurone or ceci ne correspond pas à l'algorithme **BiTM** .

De plus cela avait engendré des résultats peu concluants .

On a donc changé la boucle de telle sorte que **la recherche du meilleur neurones** se fasse en parcourant **le nombre de neurones** plutôt que **le nombre de neurones sur les lignes.**



Run Programme



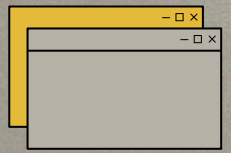
+

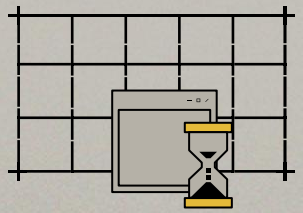
+ Une fonctionnalité **supplémentaire** ajoutée à notre programme est lors du lancement de celui-ci

En effet on a le choix entre:

- Un lancement Random avec un choix aléatoire de prototype au début de l'algorithme.
- Un lancement récupère les résultats d'une précédente exécution.

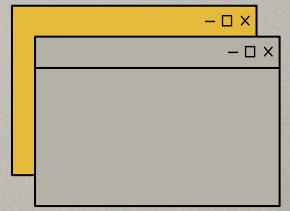
+



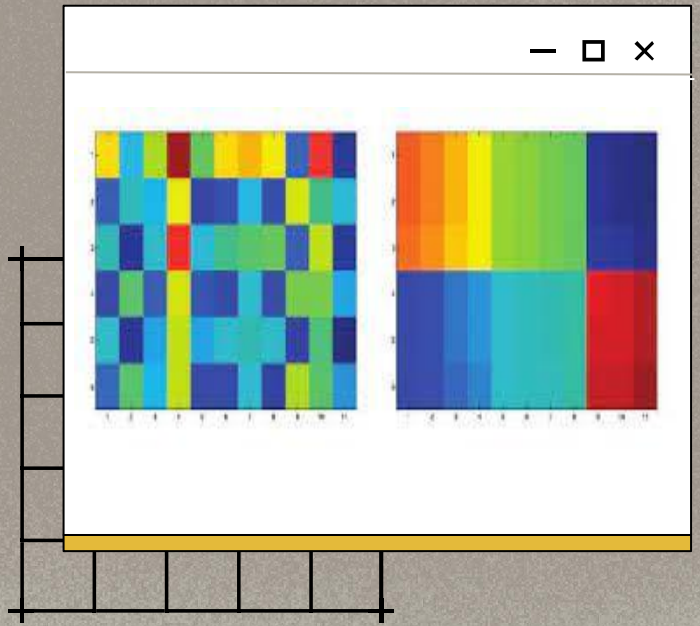


04

Affectations



VISUALISATIONS



On a visualiser les résultats obtenu grâce à notre algorithme par deux façons:

- Visualisation du Dataset organisé en entier
- Heatmap: visualisation des clusters sur cartes

Un code Python a été implémenté afin d'avoir les visu qui suivent.

Bibliothèque utilisé:

- pandas
- seaborn
- Matplotlib

OUTILS DE VISUALISATIONS

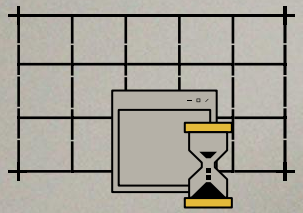
Dataset

Entrée De l'algorithme:
waveform-5000

Heatmap

Sortie: affectations en
NB clusters de map

Affectation



+

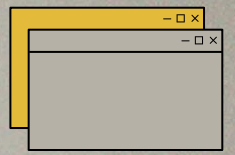
Deux méthodes à ne surtout pas confondre:

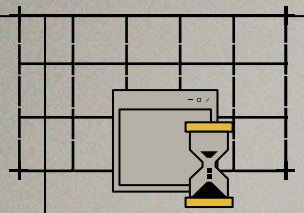
+

Affectation: réorganise le Dataset et le retourne.

Cette méthode réorganise le dataset en fonction de l'affectation des **lignes** et des **colonnes** afin de regrouper les attributs qui **se ressemblent** le plus et le retourne.

+





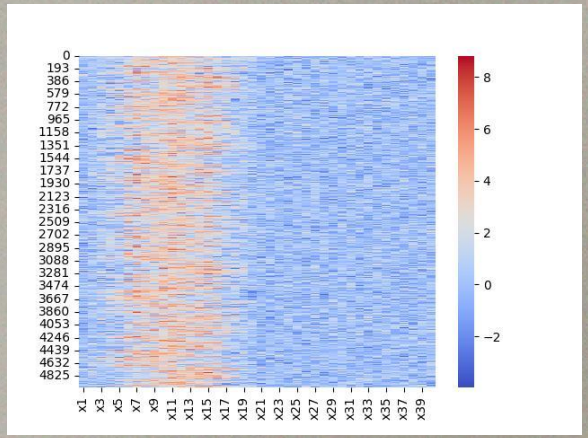
+

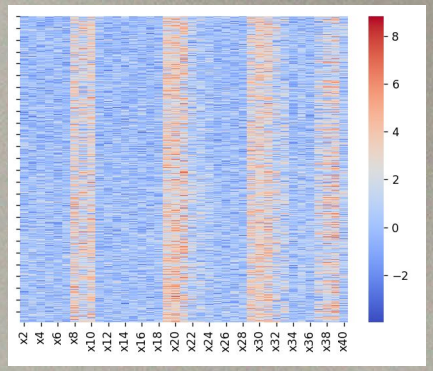
+

Output de affectation

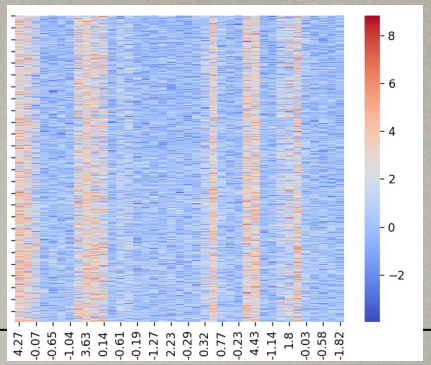
Ici on peut observer la visualisation de Waveform avant et après passage dans la fonction d'organisation **Affectation**

Waveform

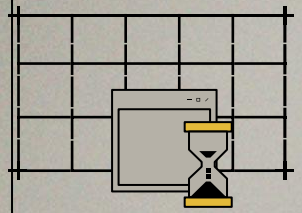




Dataset organisé en 4 clusters



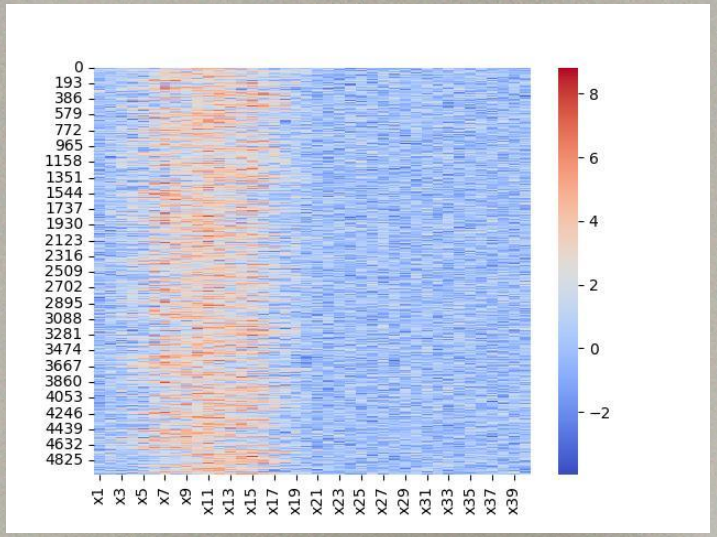
Dataset organisé en 5 clusters

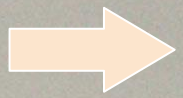


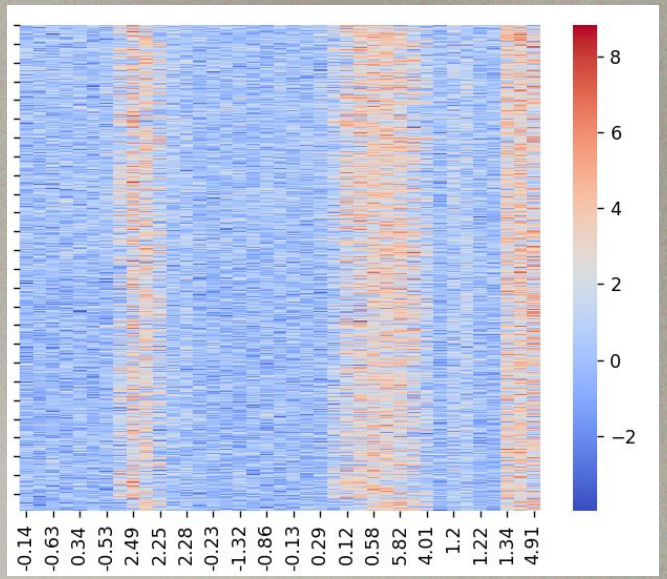
+

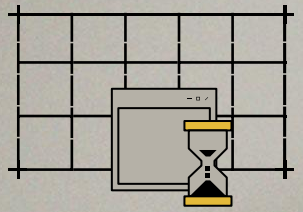
+

Dataset organisé en 3 clusters







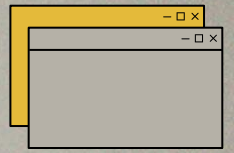


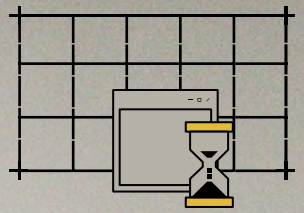
AffectationS

AffectationS:

Gère quant à elle l'affectation de chaque **ligne** au cluster le plus **proche.**

Ici chaque **ligne** sera affectée à **une sous carte** : valeur variante de 0 au nombre de clusters.

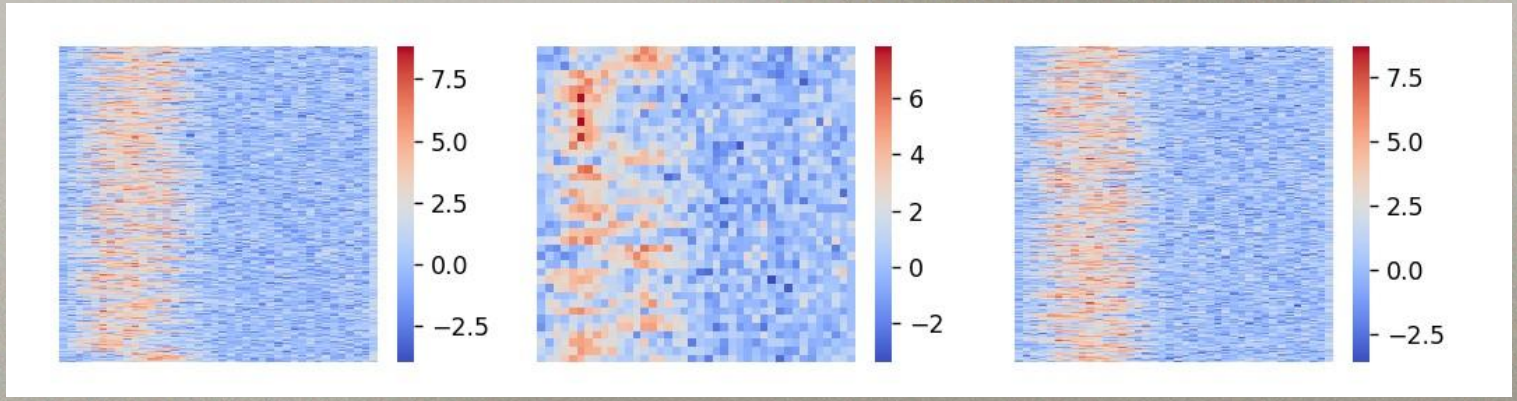




Voyons ça:

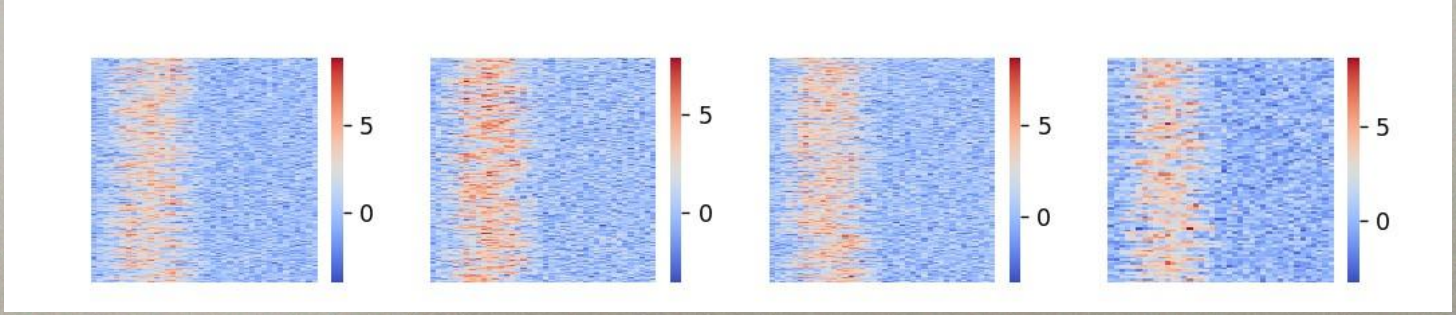
\+ Suit 2 executions:

Pour le Dataset **Waveform**
Le nombre de clusters choisi est 9
Le nombre d'itérations est 100



Pour le même Dataset

Le nombre de clusters choisi est 16
Le nombre d'itérations est 100



05

Co-Clustering Performances

Clustering performance comparison

Pour tester la qualité des résultats de notre algorithme on calcule **3 indices** de pureté :

NMI

ARI

ACC

Pour mesurer la qualité de notre algorithme, nous calculons des indices :

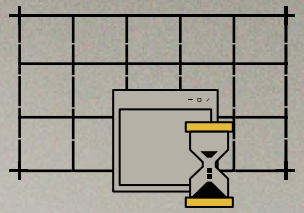
- L'exactitude **ACC** : mesure la proportion de données **correctement** attribuées à leur classe d'origine par rapport à toutes les données.

- L'information mutuelle normalisée **NMI**: mesure la quantité d'information **partagée** entre les clusters trouvés par l'algorithme BiTM et les classes réelles des données.

  Les 2 indices sont compris entre 0 et 1 , où 0 indique une classification aléatoire et 1 la correspondance parfaite.

- L'indice de Rand ajusté **ARI** : mesure la **similitude** entre les clusters trouvés par l'algorithme BiTM et les classes réelles des données.

  Il est compris entre -1 et 1, où
  1 indique une correspondance parfaite entre les clusters et les classes réelles,
  0 indique une correspondance aléatoire,
  -1 indique une correspondance complètement opposée.

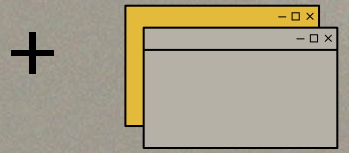


Le **RI** mesure la similarité entre deux partitions de clustering en calculant :

- Le nombre de paires d'observations qui sont affectées à la **même** classe dans les deux partitions (vrai positif)
- Le nombre de paires d'observations qui sont affectées à des classes **différentes** dans les deux partitions (vrai négatif).

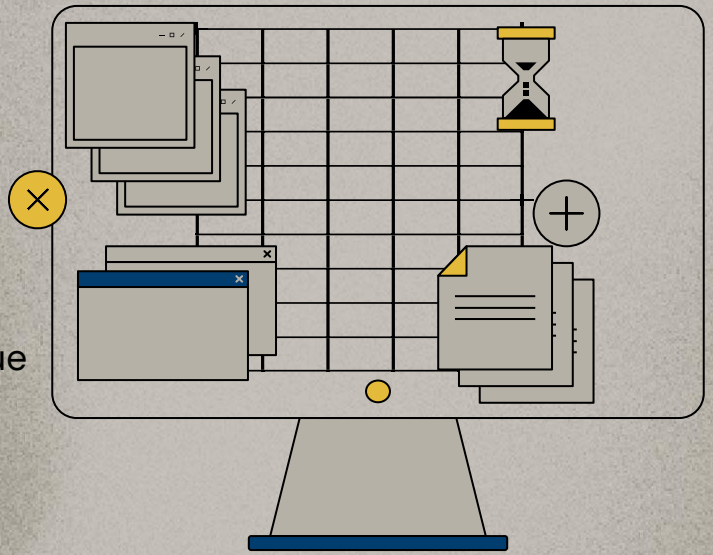
Le **RI** varie de -1 : **pas de similitude** a 1 **similitude parfaite**.

```
 ARI =  0.0005084170051013902
 NMI =  0.0008424056674870993
 ACC =  0.1564
RI =  0.5189929985997199
```



REMARQUES

Après exécution du code il est pertinent de noter que

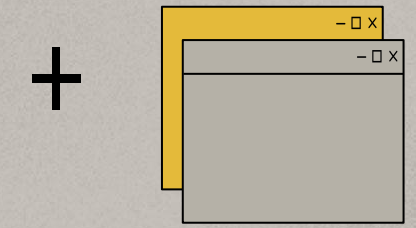


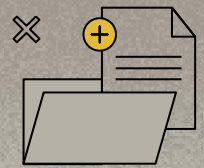

VERSIONS
!!!

Versions

Les versions utiliser pour ce projet sont:

Scala: 2.12
Spark: 2.4.3
Java: 8

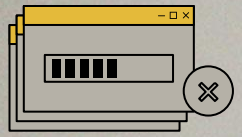


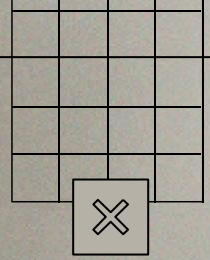


<u>À noter :</u>

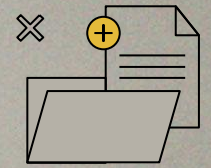
Meilleurs **performances** obtenu en utilisant:

Scala: 2.13.10
Spark: 3.3.2
Java: 17





CO-CLUSTERING MULTI VUES



ALOI Dataset

Le dataset **ALOI** est utilisé pour évaluer des méthodes de co-clustering **multi-vues**.

En effet, chaque image de texture peut être vue comme une source de données ou “ **vue**”, avec des caractéristiques spécifiques à chaque pixel de l'image.

En combinant les différentes vues, le co-clustering **multi-vues** peut être utilisé pour regrouper les pixels de toutes les images de textures en groupes cohérents de textures.

Le co-clustering **multi-vues** peut également être utilisé pour découvrir des relations entre les différentes textures en considérant simultanément plusieurs vues de chaque texture.

Multi

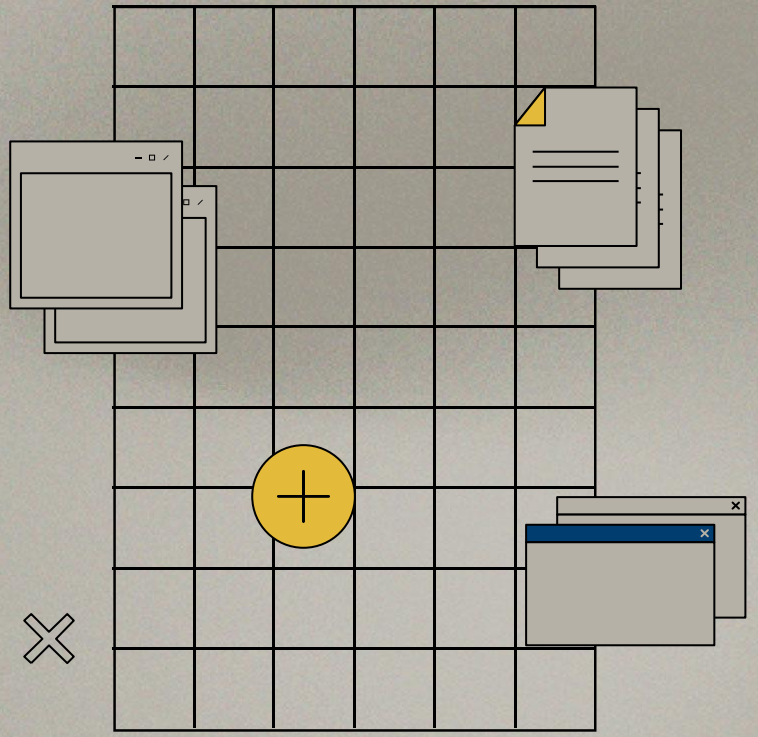
1000

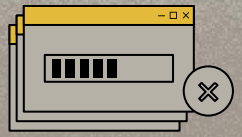
VUES

Images

ALOI

DataSet





+

On a lancé notre algorithme sur un autre jeu de données **d’images**

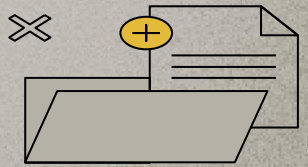
+

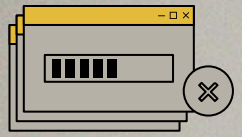
Du fait de la grande taille des données on a utilisé une analyse en composantes principales “**PCA**” pour **réduire la dimensionnalité** des données.

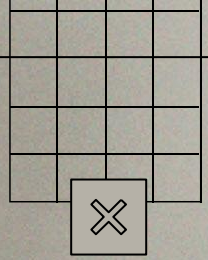
En effet une image est encodée en 108 variables et chaque variable contient un vecteur de 768 éléments .

Notre **ACP** réduit la dimensionnalité comme suit:

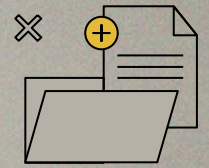
- Pour chaque ligne est associée un vecteur de 108 variables.
- Pour chaque variable, on réduit le nombre d'éléments de 768 à 53.







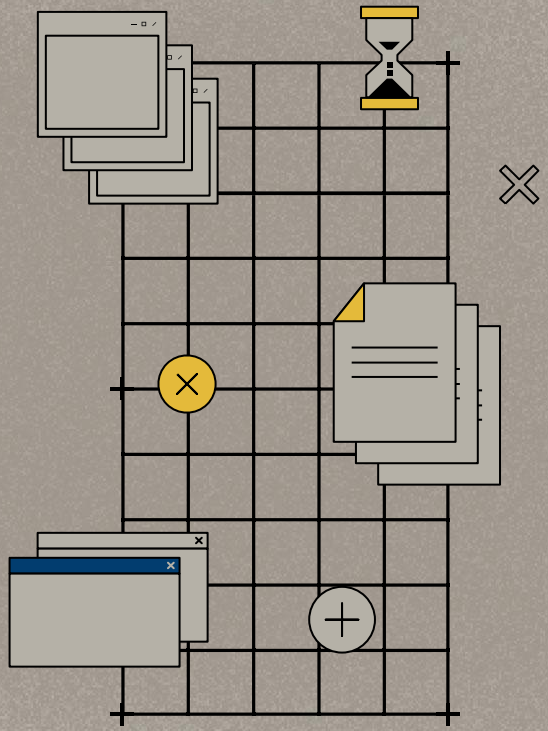
Conclusion



Améliorations

Plusieurs axes **d'améliorations** et autres modifications reste possible pour ce projet:

- Faire un **co-clustering** multi-vues.
- DataSet ALOI: Faire correspondre les clusters et retrouver les **objets associés**.
- Passer sur des versions Scala/Spark plus récentes et plus **performantes.**
- Approfondir l'étude sur la possibilité d'introduire plus de **réduction** et donc **améliorer** les performances de l'algorithme .
- Approfondir l'étude des scores pour **évaluer** au mieux l'algorithme.
- Faire un **dashboard** englobant l'ensemble des visualisations et analyses effectuées pour un dataset .

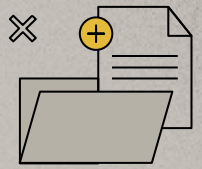


Merci!

Avez-vous des questions ?

Dahmani Lydia
Cherifi Wissem
Ka William
Labdi Wassim
Lopes Fernandes Dylan

RESOURCES



GitHub

- https://github.com/TugdualSarazin/spark-clustering
- https://github.com/unsupervise/ACOMI
- https://github.com/unsupervise/NPLBM

Doc

- Apprentissage massivement distribué dans un environnement Big Data: Tugdual Sarazin
- Waveform: https://datahub.io/machine-learning/waveform-5000
- Jeu de données images: https://aloi.science.uva.nl/