

Mathématiques appliquées pour
l'Ingénierie, l'Industrie & l'Innovation



Analyse multi-temporelle par clustering sous contraintes

Stage de fin d'études

Benjamin Beyrie

5 mars 2018 - 31 août 2018

Superviseur:
M. Guillaume Pasero

Tuteur:
M. Fabrice Gamboa

Remerciements

Je voudrais tout d'abord remercier mon maître de stage, M. Guillaume Pasero qui m'a donné sa confiance et m'a accompagné tout au long de ces 6 mois de stage.

Je remercie également M. Fabrice Gamboa, mon tuteur référent de l'université Paul Sabatier.

Je tiens aussi à remercier l'équipe Orfeo ToolBox et les connaissances que j'ai fait à CS avec qui j'ai partagé ces 6 mois.

Je souhaite finalement remercier l'équipe pédagogique de la formation MApI³, qui nous soutiennent toute l'année au dépend de leur temps personnel.

Résumé

Avec l'essor des publications d'images satellites multi-temporelles en libre accès, le besoin de trouver des méthodes adaptées à ces nouveaux types de données bondit.

Ce stage a pour but d'étudier des algorithmes de clustering sous contraintes sur des images multi-temporelles, ainsi que la détection de changement avec des algorithmes de machine learning.

Table des matières

1 Présentation	1
1.1 Établissement d'accueil	1
1.2 Détails du stage	1
1.3 Acquisition des images	2
1.4 Format des images	3
1.5 Composants logiciels utilisés	3
2 Prétraitement des images multi-temporelles	4
2.1 Ré-échantillonnage en espace	5
2.2 Application des masques de nuage	5
2.3 Interpolation des valeurs manquantes	6
2.4 Lissage en espace	7
2.5 Concaténation des images date par date	7
2.6 Transformation en indices radiométriques	8
2.7 Normalisation des données	10
2.8 Lissage en temps	11
2.9 Image de référence	12
3 Clustering des images multi-temporelles sous contraintes	14
3.1 Clustering sous contraintes	14
3.1.1 Différents types de contraintes	14
3.1.2 Différentes familles d'algorithmes	14
3.1.3 Méthodes de clustering sous contraintes	15
3.1.4 Méthodes retenues	15
3.1.5 Création des contraintes	16
3.1.6 Algorithmes	17
3.2 Réduction de dimension non linéaire pour visualiser les données	19
3.3 Application des algorithmes COP-Kmeans et LCVQE	20
3.4 Clustering en utilisant une mesure de dissimilarité : Dynamic Time Warping	23
3.4.1 Raffinement par similarité inter-classes	27
3.4.2 Raffinement par similarité à des références temporelles	28
3.4.3 Clustering par similarité à des références temporelles	28
4 Détection de changements	29
4.1 Détection d'outliers (anomalies)	32
4.1.1 Algorithmes utilisés	32
4.2 Résultats des détections d'anomalies	33
4.3 Détection d'outliers en utilisant une matrice de dissimilarité	36

5 Discussions	37
5.0.1 Qualité de la carte de vérité terrain	37
5.0.2 Indices de texture	37
5.0.3 Séries d'images sur plusieurs années	37
5.0.4 Autres méthodes de clustering sous contraintes	37
6 Conclusion	39
Annexes	42

Chapitre 1

Présentation

1.1 Établissement d'accueil

CS Communication & Systèmes de Toulouse, situé zone de la Grande Plaine, appartenant au groupe du même nom, est une entreprise d'expertise en matière d'applications et de systèmes critiques et en fait le partenaire privilégié de grands secteurs économiques, notamment dans les domaines de la défense & de la sécurité, de l'espace, de l'aéronautique et de l'énergie.

Le stage s'est déroulé dans la branche "Espace" de l'entreprise.

Plusieurs projets ont lieu au sein de cette unité, j'ai eu le privilège d'être dans le même bureau que l'équipe OTB, acronyme de "Orfeo ToolBox", qui est un logiciel en open source codé en C++, destiné à faire du traitement d'images satellites.

1.2 Détails du stage

Le sujet du stage est "Analyse d'images multi-temporelles par clustering sous contraintes".

Avec l'ouverture en libre accès de nombreuses images prises par des satellites (missions Gaia, Sentinel-2, ASTER, ...), l'ère du traitement de l'imagerie satellite multi-temporelle prend son essor.

À mi chemin entre le traitement d'images et l'apprentissage non supervisé, l'objectif du stage est d'étudier des séries d'images satellites (Sentinel-2 dans notre cas) prises à un même endroit mais à des dates différentes, en utilisant des méthodes d'apprentissage non supervisé avec contraintes.

D'un point de vue technique, le travail a été effectué sous Python pour plus d'aisance, avec aussi un peu de R. Un peu de C++ a été fait sur la fin du stage pour coder une méthode ne dépendant pas trop du sujet du stage. Cette dernière est exposée en annexe.

1.3 Acquisition des images

Les images sont issues d'un jeu de données des missions Sentinel-2, qui sont une série de satellites d'observation de la Terre de l'Agence Spatiale Européenne développés dans le cadre du programme Copernicus, programme financé par l'Union Européenne, dont les deux premiers exemplaires ont été mis en orbite en 2015 et 2017.

Pour récolter ces données, il suffit d'aller sur le site internet de l'ESA partie Sentinel-2 ou sur le site de Theia (<https://theia.cnes.fr/atdistrib/rocket/#/search?collection=SENTINEL2>). Une fenêtre s'ouvre alors, dans laquelle on peut découper dynamiquement un polygone d'une zone et en extraire toutes les images depuis le début des acquisitions, selon les caractéristiques voulues que l'on peut paramétriser dans la barre de recherche, comme par exemple :

- Le nom du satellite.
- Le taux de nuage.
- L'intervalle entre l'acquisition de la première image et la dernière image.
- Le numéro de l'orbite.

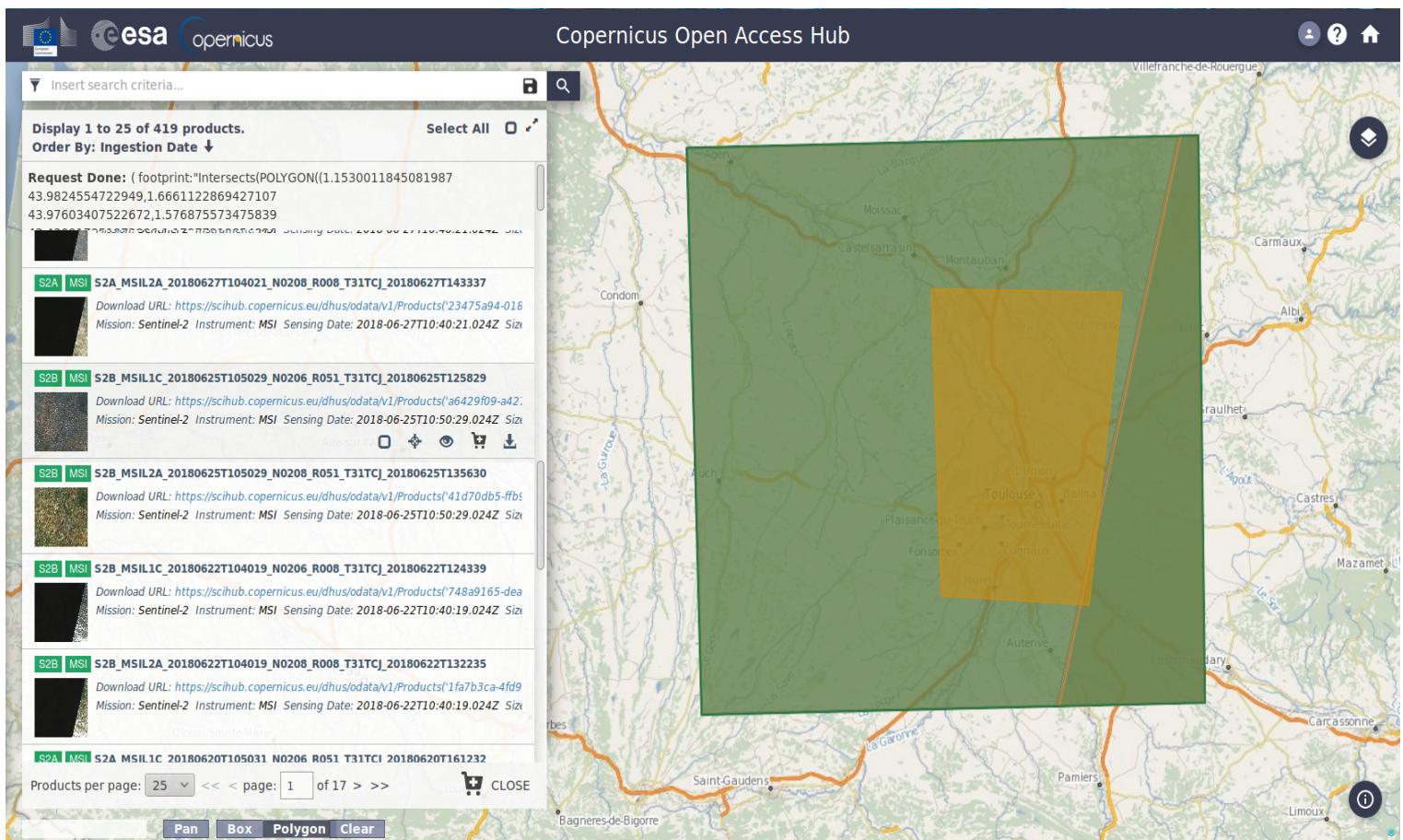


FIGURE 1.1 – Exemple de recherche d'images sur une tuile

1.4 Format des images

Une fois téléchargées, les images se présentent chacune dans un dossier séparé au format "Tag(ged) Image File Format" (tiff), dont l'avantage pour les utilisateurs est d'avoir en plus de l'image, ses caractéristiques de projection, mais se présentent aussi au format JPEG2000. Chaque dossier possède des images de trois résolutions différentes pour la zone sélectionnée, allant du domaine visible pour l'œil jusqu'à l'infrarouge court :

1. Résolution de 60 mètres : La bande 1 (443nm, violet), la bande 9 (940nm, proche infrarouge) et la bande 10 (1375nm, proche infrarouge).
2. Résolution de 20 mètres : La bande 5 (705nm, proche infrarouge), la bande 6 (740nm, proche infrarouge), la bande 7 (783nm, proche infrarouge), la bande 8a (865nm, proche infrarouge), la bande 11 (1610nm, infrarouge) et la bande 12 (2190nm, infrarouge).
3. Résolution de 10 mètres : La bande 2 (490nm, bleu), la bande 3 (560nm, vert), la bande 4 (665nm, rouge) et la bande 8 (842nm, proche infrarouge).

On ignore dans la suite les images correspondantes à la résolution de 60 mètres car ces images sont d'une précision trop faible. À noter que les images de Sentinel-2 (niveau 2) sont offertes avec une correction atmosphérique.

1.5 Composants logiciels utilisés

- OTB :
 - ExtractROI : Extrait une région d'intérêt d'une image
 - ConcatenateImages : Concatène une liste d'images de mêmes tailles en une image multi-canaux
 - Smoothing : Applique un filtre de lissage à une image
 - RigidTransformResample : Applique une transformation paramétrique sur une image, comme un ré-échantillonnage ou un rotation par exemple
 - Autres applications testées : RadiometricIndices, ColorMapping, MultivariateAlterationDetector, HaralickTextureExtraction
- Python :
 - numpy & scipy : Gestion des tableaux et applications mathématiques
 - matplotlib, seaborn, bokeh : Outils d'aide à la visualisation
 - scikit-learn : Module pour le machine learning, ici utilisé pour le pré-traitement des images
 - rpy2 : Interface python pour le language R
 - pyod : Boîte à outils pour la détection d'outliers
 - tifffile : Gestion des images au format .tiff
 - os, sys, shutil : Module pour gérer le système de l'ordinateur
- R :
 - dtw : Implémentation de diverses déformations temporelles dynamiques (DTW)
 - conclust : Implémentations des algorithmes de clustering sous contraintes

Chapitre 2

Prétraitement des images multi-temporelles

Avant de commencer, il nous faut travailler sur les images pour les rendre plus lisibles. Pour réaliser ces étapes de pré-traitement on a utilisé plusieurs outils disponibles sur Python et sur l'OTB. L'image de référence utilisée est la suivante :



FIGURE 2.1 – Image résolution 10 mètres

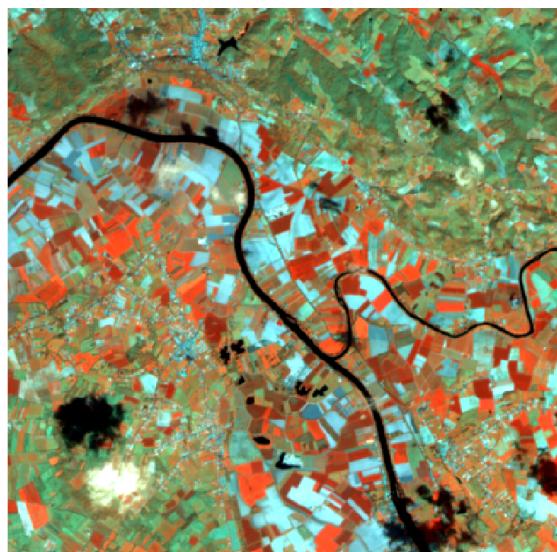


FIGURE 2.2 – Image résolution 20 mètres

On a sélectionné cette image de taille 800x800 et 400x400 selon la résolution, car elle contient entre autre des habitations, des champs, des forêts et une rivière, ce qui est intéressant pour notre étude du clustering. Cette image a été prise sous 11 dates différentes, allant de janvier 2017 à novembre 2017.

2.1 Ré-échantillonnage en espace

Tout d'abord, en passant par la librairie OTB, on a appliqué un sur-échantillonnage de facteur 2 sur nos bandes correspondantes aux résolutions de 20 mètres, avec un filtre bicubique.

L'opération de convolution correspondant à ce filtre est le suivant :

$$W(x) = \begin{cases} (a+2)|x|^3 - (a+3)|x|^2 + 1 & \text{si } |x| \leq 1 , \\ a|x|^3 - 5a|x|^2 + 8ax - 4a & \text{si } 1 \leq |x| \leq 2 , \\ 0 & \text{sinon} \end{cases}$$

Avec le paramètre a généralement fixé à -0.75 ou -0.5.

2.2 Application des masques de nuage

Les images Sentinel-2 sont fournies avec plusieurs masques de nuage pour chaque date, correspondant à des nuages de différentes sortes (nuages cirrus, nuages denses, nuages de chaleur, ..). Ils sont représentés par une image binaire ou ternaire, avec 0 si le pixel n'a pas de nuage, et 1 ou 2 si il y a un nuage ou ombre. Pour appliquer les masques, il suffit de sommer tous les masques de nuages donnés, de ne garder qu'une valeur de masque à 0 (si aucun nuage n'est présent) ou 1 (si un nuage est présent) et de multiplier le résultat par les images correspondantes pixel à pixel à cette date.

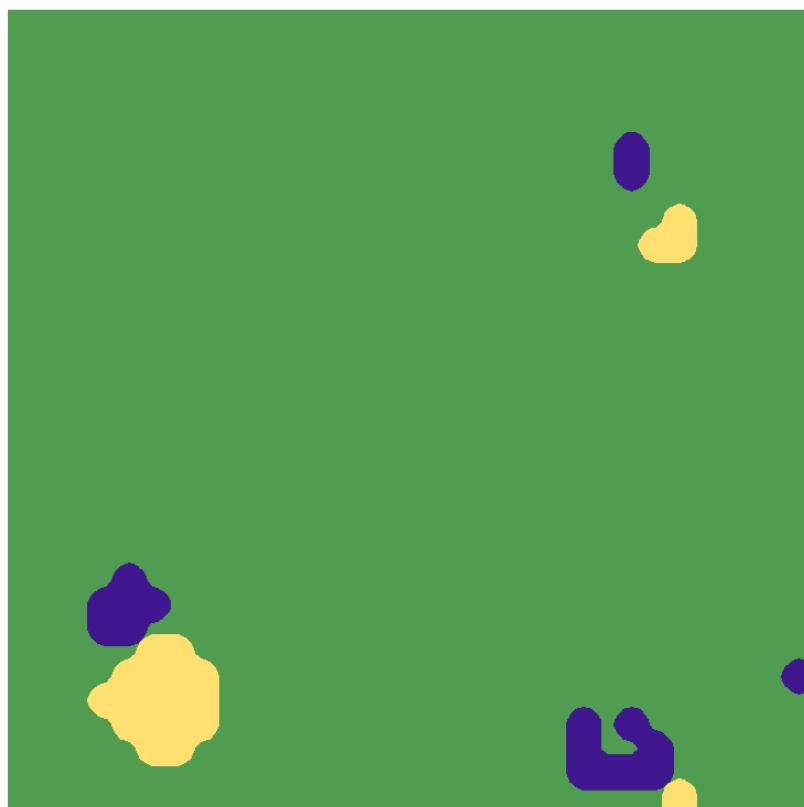


FIGURE 2.3 – Exemple d'un masque de nuage

2.3 Interpolation des valeurs manquantes

L'interpolation des points manquants sur une image multi-temporelle peut se faire de plusieurs façons. Parmi les familles que l'on peut tester, du plus simple au plus complexe, on a :

1. l'interpolation en espace, qui a le désavantage de ne pas réussir à combler de grands vides si les nuages sont trop présents.
2. L'interpolation en temps, qui a l'avantage d'utiliser les connaissances futures et passées pour combler les vides, seul problème qui peut mais n'est pas encore arrivé, d'avoir un nuage tout le temps au même endroit dans quel cas il serait impossible d'interpoler de cette façon.
3. L'interpolation en temps et en espace, qui combine les deux méthodes précédentes, et donc comble les lacunes de chacune. Son temps d'exécution reste long.
4. Méthode d'attribution des valeurs manquantes avec un algorithme de type machine learning (missForest qui est un dérivé des randomForest par exemple). Ces méthodes sont gourmandes en mémoire, et marchent bien pour des données manquantes de façon complètement aléatoire (MCAR), ce qui n'est pas notre cas ici car c'est nous qui avons créé les données manquantes avec les masques de nuages.

On a choisi la méthode d'interpolation linéaire en temps, qui est un bon compromis entre temps de calcul et qualité des résultats.

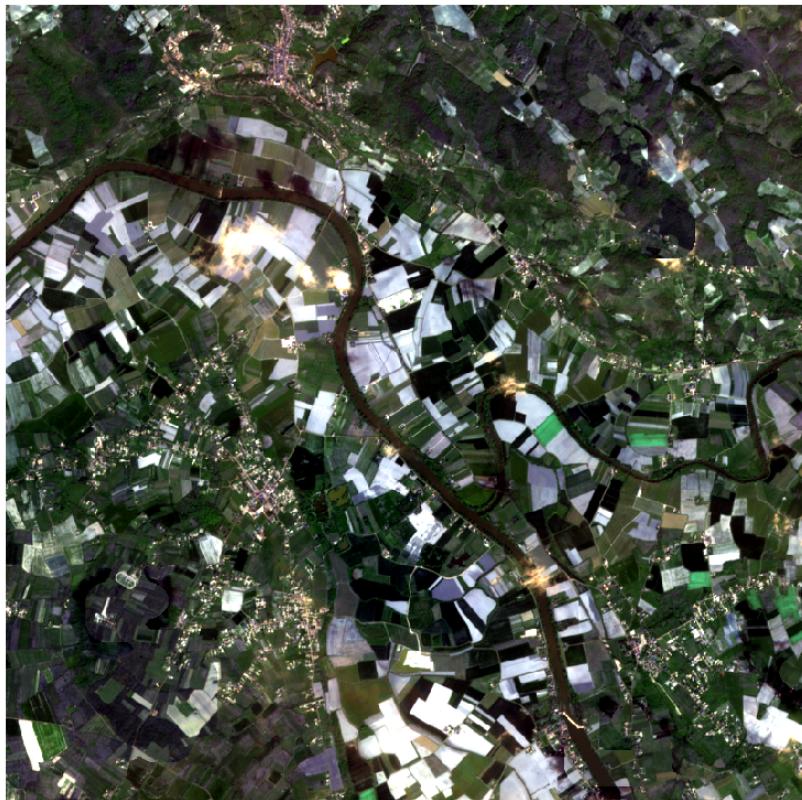


FIGURE 2.4 – Interpolation des images en temps

2.4 Lissage en espace

Le lissage ("smoothing" en anglais) permet de réduire le bruit et d'homogénéiser des zones de l'image tout en gardant les détails comme les lignes, les contours, etc... Le lissage utilisé est dit de diffusion anisotrope, qui correspond à l'évolution d'une équation de diffusion.

Soit $\Omega \subset \mathbb{R}^2$ un sous-espace du plan, et $I(., t) : \Omega \rightarrow \mathbb{R}$ une famille d'image, alors la diffusion anisotrope s'écrit comme suit :

$$\frac{\partial I}{\partial t} = \operatorname{div}(c(x, y, t)\nabla I) = \nabla c \cdot \nabla I + c(x, y, t)\Delta I$$

Avec Δ l'opérateur laplacien, ∇ le gradient, $\operatorname{div}(\dots)$ la divergence et $c(x, y, t)$ le coefficient de diffusion.

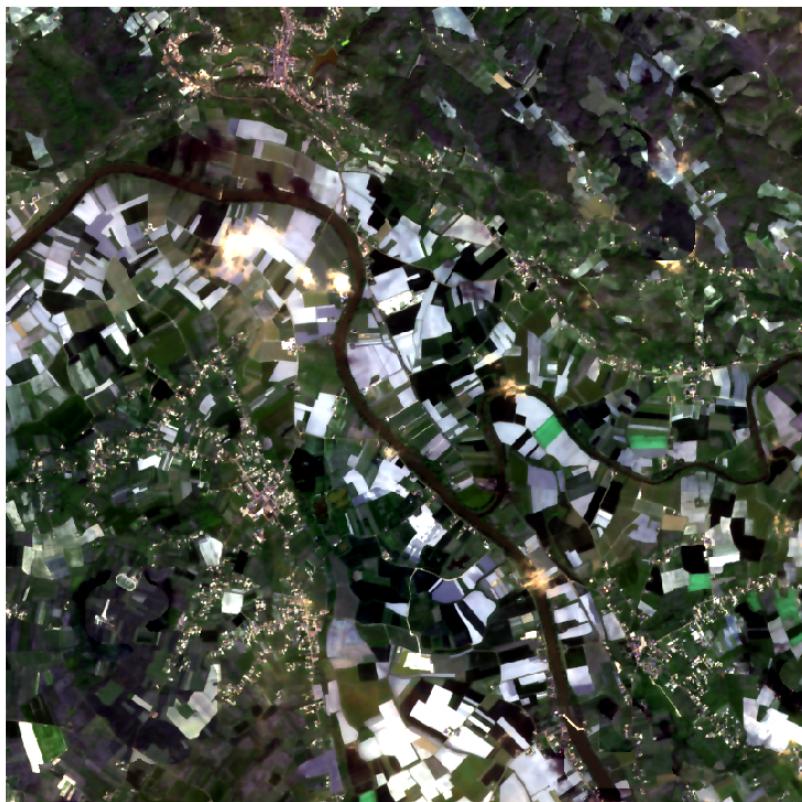


FIGURE 2.5 – Lissage des images en espace

2.5 Concaténation des images date par date

Les images de différentes dates étant dans des dossiers différents, il est temps de les concaténer pour pouvoir traiter l'ensemble des données plus efficacement. Elles sont mises sous la forme d'un bloc (matrice 3D) d'une hauteur correspondante au nombre de bandes multiplié par le nombre de dates.

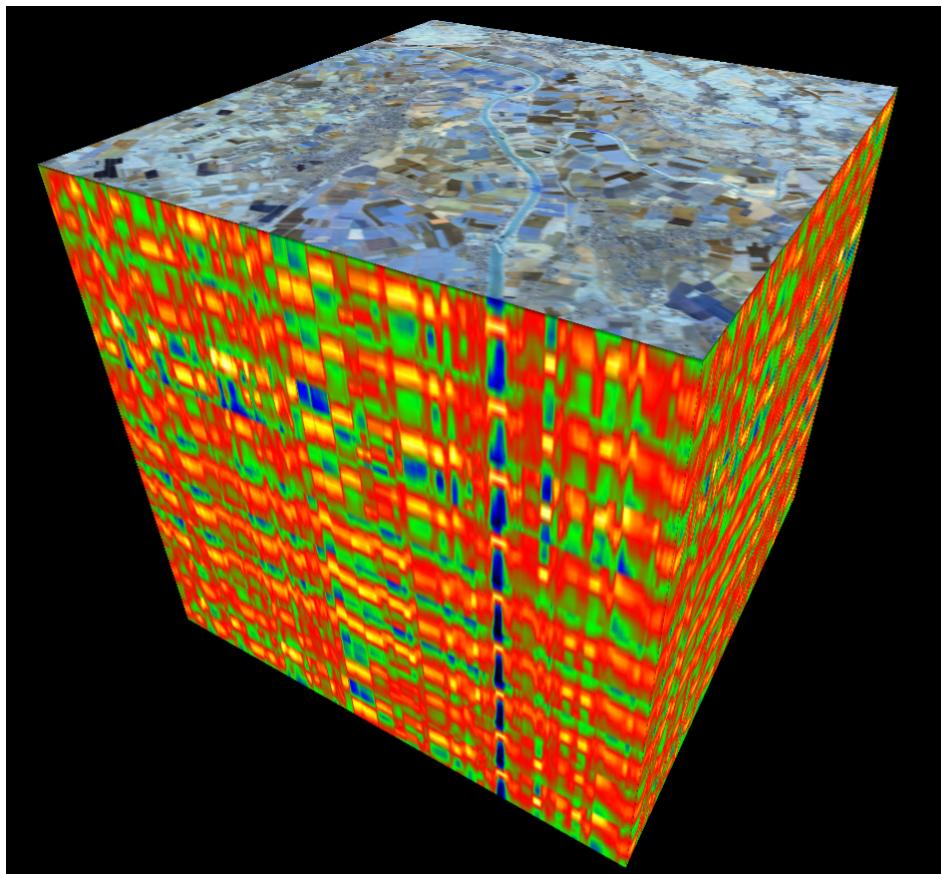


FIGURE 2.6 – Représentation 3D des images après concaténation

2.6 Transformation en indices radiométriques

La transformation des bandes de départ en indices radiométriques permet la création de nouvelles composantes, dédiées à la détection de certaines parties du sol, comme par exemple la végétation, l'eau ou la terre. Pour se faire, la combinaison de plusieurs bandes spécifiques, avec des opérations arithmétiques permet la création d'une nouvelle bande. Il existe une grande variété d'indices radiométriques, on s'est limité à choisir arbitrairement les plus pertinents (7 en l'occurrence) parmi 16 indices radiométriques testés, dont voici la liste :

- Indices de végétation : REP (Red-Edge Position), SAVI (Soil Adjusted Vegetation Index), RENDVI (Red-Edge Normalized Difference Vegetation Index), PSSR (Pigment Specific Simpel Ratio), PSRI (Plant Senescence Reflectance Index), PSRI-NIR (Plant Senescence Reflectance Index), **mNDVI (Modified NDVI)**, MCARI (Modified Chlorophyll Absorption in Reflectance Index), **GRVI (Green Ratio Vegetation Index)**, **NBR (Normalized Burn Ratio)**, BWDRVI (Blue-Wide Dynamic Range Vegetation Index).
- Indices d'eau : **NDWI (Normalized Difference Water Index)**, **RENDWI (Red-Egde Normalized Difference Water Index)**.
- Indices du sol : **BI (Brightness Index)**, **NDSI (Normalized Difference Salinity Index)**.

Les 7 indices radiométriques retenus sont écrits en caractère gras.

Voici trois exemples avec le NDVI, le NDWI et le BI et les résultats obtenus :

$$NDVI = \frac{NIR - Red}{NIR + Red}$$

$$NDWI = \frac{Green - NIR}{Green + NIR}$$

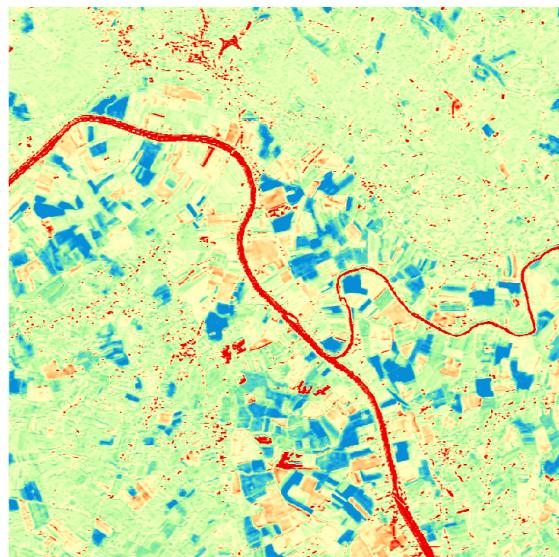


FIGURE 2.7 – NDVI

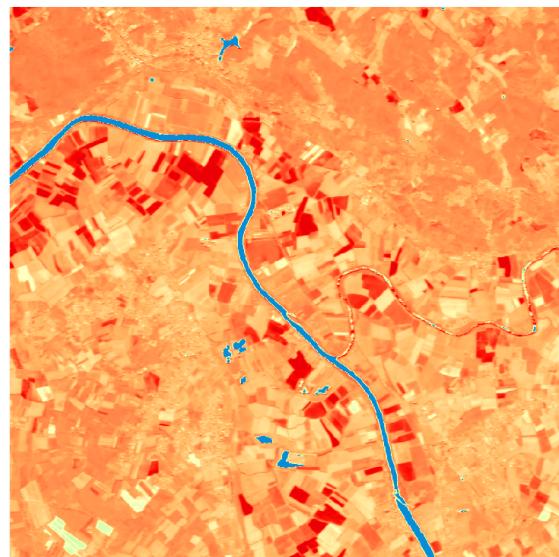


FIGURE 2.8 – NDWI

$$BI = \sqrt{Green^2 + Red^2 + NIR^2}$$

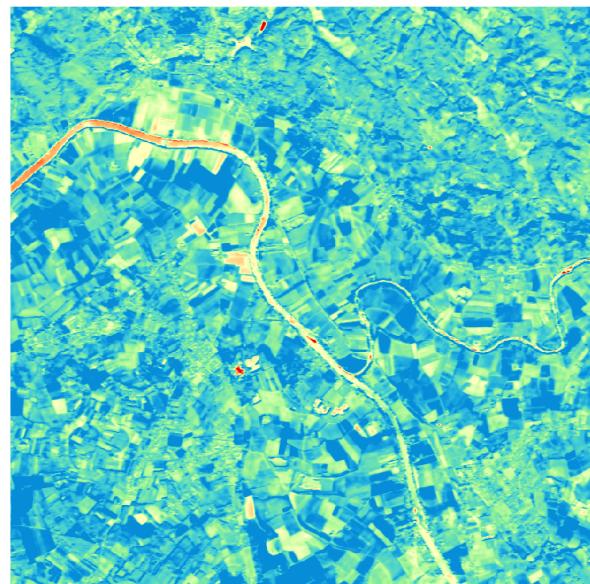


FIGURE 2.9 – BI

2.7 Normalisation des données

La régularisation de nos données se fait bande par bande pour éviter qu'une bande n'interagisse avec une autre bande. Plusieurs méthodes ont été testées pour transformer nos images de façon à les régulariser. Les méthodes sont les suivantes :

- La standardisation : Le fait de centrer et réduire nos données

$$I'(x, y) = \frac{I(x, y) - \mu}{\sigma} \quad \text{avec } \mu \text{ la moyenne et } \sigma \text{ l'écart type}$$

- **La normalisation** : Le fait de projeter sur un cercle unité. Cette méthode en grande dimension marche bien malgré le problème du fléau de la dimension car même sur un cercle unité, les points restent bien espacés.

$$I'(x, y) = I(x, y) / \| I \|$$

- Mise à l'échelle robuste : Le fait de rentrer les données dans un intervalle ($[0,1]$ dans notre cas), en ignorant un certain pourcentage de données les plus extrêmes, dans le but d'éviter les outliers.

$$I'(x, y) = \frac{I'(x, y) - I_{min}}{I_{max} - I_{min}}$$

- Une transformation non-linéaire : Le fait de projeter les données sur une distribution de probabilité, comme une loi uniforme par exemple.



FIGURE 2.10 – Normalisation des images

2.8 Lissage en temps

Le lissage en temps est l'étape finale du pré-traitement des images. Comme pour le lissage en espace, il permet d'homogénéiser les séries temporelles pour chaque pixel. On applique ici un filtre gaussien, avec comme un noyau gaussien centré en 0 et d'une variance $\sigma^2 = 1$.

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}}$$



FIGURE 2.11 – Lissage des images en temps

2.9 Image de référence

Notons qu'on peut voir les images comme un jeu de données avec en feature les différentes bandes des images à chaque date, et en individus les différents pixels. Notre image est alors de taille $800 \times 800 \times 11 \times 7$, correspondant à la taille de l'image, avec les 11 dates, et en plus les indices radiométriques créés. On peut imaginer cela comme une composition de 11 images multi-bandes. Les données étant trop importantes pour le besoin de notre étude, on réduit donc la zone à une taille $200 \times 200 \times 11 \times 7$, ce qui réduit le nombre de données d'un facteur 16.



FIGURE 2.12 – Image de test

De plus, on a une vérité terrain prise sur le site du CESBIO pour labelliser nos pixels, on peut donc s'appuyer dessus pour mesurer l'efficacité des clusterings ou simplement mieux visualiser nos données :

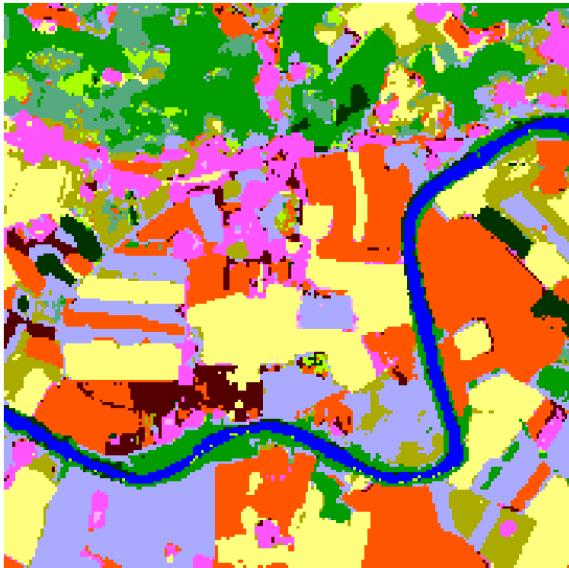


FIGURE 2.13 – Vérité Terrain avant fusion des classes (15 classes)

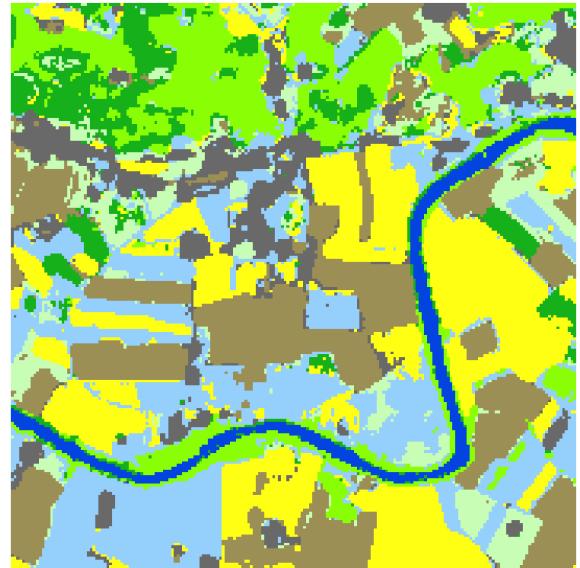


FIGURE 2.14 – Vérité Terrain après fusion des classes (8 classes)

La fusion des clusters dans la vérité de terrain est faite de façon à grouper les classes très proches, les pelouses avec la prairie ou encore les lacs avec les rivières par exemple, pour ne garder que 8 classes de référence :

- Les cultures d'été, en jaune.
- Les cultures d'hiver, en khaki foncé.
- La forêt à feuille caduque, en vert clair.
- La forêt à feuillage sempervirent ou persistant, en vert foncé.
- La ville, en gris.
- L'eau, en bleu.
- Les prairies, en vert pale.
- Les vergers, en bleu ciel.

Chapitre 3

Clustering des images multi-temporelles sous contraintes

Après un pré-traitement des images, on arrive à la phase de clustering sous contraintes. Un clustering sous contraintes correspond à l'apport d'informations à priori sur notre jeu de données, ici des images.

3.1 Clustering sous contraintes

3.1.1 Différents types de contraintes

- Le nombre de clusters souhaités.
- Le nombre minimal ou maximal de points par cluster.
- Le diamètre des clusters, correspondant à une distance maximale entre les points les plus éloignés au sein d'un même cluster.
- La séparation entre clusters, correspondant à une distance entre les différents clusters.
- La densité des individus dans un cluster, correspondant à une distance maximale à respecter entre un individu et son plus proche voisin.
- Les liaisons Must-link et Cannot-link, correspondant à une information à priori sur certains individus selon si on sait qu'ils appartiennent au même cluster ou qu'ils ne peuvent jamais être dans le même cluster.

3.1.2 Différentes familles d'algorithmes

- K-Means
- Metric Learning
- Spectral Graph Theory
- Ensemble Clustering
- Collaborative Clustering
- Declarative Approaches

3.1.3 Méthodes de clustering sous contraintes

Category	Method		
<i>k</i> -Means	COP-COBWEB (Wagstaff and Cardie, 2000) COP-KMeans (Wagstaff et al, 2001) Seed-KMeans (Basu et al, 2002) Constrained-KMeans (Basu et al, 2002) ICOP-KMeans (Tan et al, 2010) Sequenced Assignment COP-KMeans (Rutayisire et al, 2011) MLC-KMeans (Huang et al, 2008) SCREEN (Tang et al, 2007) GA Dispersion & Impurity (Demiriz et al, 1999) CVQE (Davidson and Ravi, 2005) LCVQE (Pelleg and Baras, 2007) PCK-Means (Basu et al, 2004b) Lagrangian Relaxation (Ganji et al, 2016) Tabu Search (Hiem et al, 2016) Fuzzy CMeans (Grira et al, 2006) Non-Negative Matrix Factorisation (Li et al, 2007) Mathematical Program (Ng, 2000) Minimal Capacity Constraints (Bradley et al, 2000) Balanced Clustering (Banerjee and Ghosh, 2006) Minimal Size (Demiriz et al, 2008) Minimal Size & Balanced Clustering (Ge et al, 2007)	Spectral Graph Theory	Adjacency Matrix Modification (Kamvar et al, 2003) Out-Of-Sample Adjacency Matrix Modification (Alzate and Suykens, 2009) CSP (Wang and Davidson, 2010a; Wang et al, 2014) Constraint Satisfaction Lower Bound (Wang et al, 2010) Inconsistent Constraints (Rangapuram and Hein, 2012) Logical Constraint Combinations (Zhi et al, 2013) Distance Modification (Anand and Reddy, 2011) Constraint Propagation Binary Class (Lu and Carreira-Perpiñán, 2008) Constraint Propagation Multi-Class (Lu and Ip, 2010; Chen and Feng, 2012; Ding et al, 2013) Kernel Matrix Learning (Zhang and Ando, 2006; Hoi et al, 2007; Li and Ding, 2008; Li and Liu, 2009) Guaranteed Quality Clustering (Cucuringu et al, 2016)
		Ensemble Clustering	SCEV (Iqbal et al, 2012) Consensus Function (Al-Razgan and Domeniconi, 2009; Xiao et al, 2016; Dimitriadou et al, 2002)
		Collaborative Clustering	SAMARAH (Forestier et al, 2010a) Penta-Training (Domeniconi and Al-Razgan, 2008)
		Declarative Approaches	SAT (Davidson et al, 2010) CP (Dao et al, 2013, 2016, 2017; Guns et al, 2016) ILP Column Generation (Merle et al, 1999; Aloise et al, 2012; Babaki et al, 2014) Restricted Cluster Candidates (Mueller and Kramer, 2010; Ouali et al, 2016)
Metric Learning	Euclidean (Klein et al, 2002) Mahalanobis (Bar-Hillel et al, 2003, 2005; Xing et al, 2002) Kullback-Leibler Divergence (Cohn et al, 2003) String-Edit Distance (Bilenko and Mooney, 2003) LRML (Hoi et al, 2008, 2010) Partially Observed Constraints (Yi et al, 2012)	Miscellaneous	Constrained EM (Shental et al, 2013) Evolutionary Algorithm (Handl and Knowles, 2006) Random Forest (Zhu et al, 2016)
<i>k</i> -Means & Metric Learning	MPC-KMeans (Bilenko et al, 2004) HMRF-KMeans (Basu et al, 2004b) Semi-Supervised Kernel <i>k</i> -Means (Kulis et al, 2005, 2009) CLWC (Cheng et al, 2008)		

3.1.4 Méthodes retenues

Les méthodes retenues pour le stage sont :

- COP-KMeans (Constrained KMeans)
- LCVQE (Linear-time Constrained Vector Quantization Error)

Tabu-search et MPC-KMeans ont aussi été testées, mais exclues à cause des problèmes de mémoire ou des temps de calcul trop importants. La méthode Samarah, utilisant du clustering collaboratif, semblait la plus prometteuse, mais son implémentation reste difficile et a donc été laissée de côté aussi.

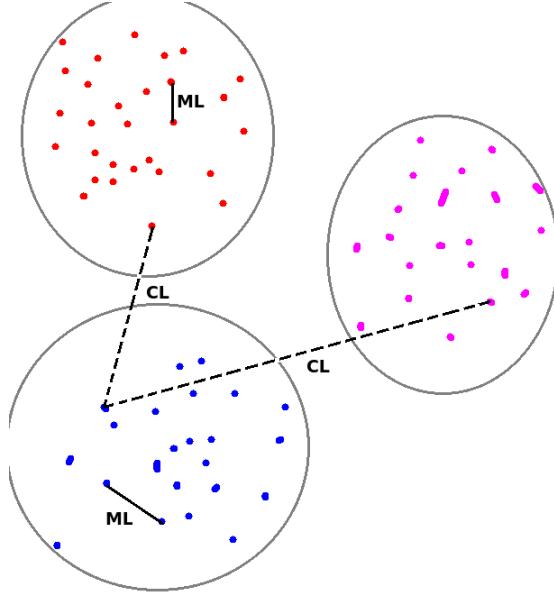
Nos deux méthodes étant des dérivées de l'algorithme KMeans, rappelons succinctement son fonctionnement. KMeans est une méthode d'agrégation autour de centres mobiles. On l'initialise en tirant k centroïdes de façon aléatoire dans l'espace des individus. On alloue chaque individu à son centroïde le plus proche, puis on calcule le centre de gravité de chaque cluster. On répète ces deux étapes jusqu'à ce que le critère de variance interclasses ne croisse plus de manière significative, c'est-à-dire jusqu'à la stabilisation des clusters.

COP-Kmeans est l'algorithme de clustering sous contraintes se rapprochant le plus de KMeans, l'ajout vient d'une pénalité au moment d'allouer un point à son centroïde le plus proche. Si on viole une contrainte, elle permet d'empêcher ou de forcer des points à appartenir à un cluster ou non, en sautant l'étape de mise à jour des centroïdes.

LCVQE est aussi un algorithme dérivé des KMeans. L'avantage au moment des contraintes est qu'il ne regarde les clusters que par paire. Une contrainte Must-Link violée (c'est à dire deux individus dans des clusters différents mais qui devraient appartenir au même) oblige les deux centroïdes des clusters de ces individus à se rapprocher. À contrario, une contrainte Cannot-Link violée (c'est à dire deux individus alloués au même cluster mais qui ne le devraient pas) force à sélectionner l'individu le plus éloigné du centroïde de ce cluster, et de chercher un autre cluster, différent de celui auquel il est actuellement attribué, pour rapprocher ce centroïde dans la direction de cet individu.

3.1.5 Creation des contraintes

On va s'axer sur les contraintes de type Must-links (ML) et Cannot-links (CL), qui sont les plus proches a notre sens de contraintes amenant de l'information venant de l'utilisateur. Les autres contraintes sont plus des contraintes sur les paramtres des algorithmes pour creer les clusters plutt que sur les individus directement.



On a plusieurs faons de creer des contraintes ML et CL :

- Creer les contraintes a la main en cliquant sur l'image et recuprant les individus soit en ML, soit en CL.
- Creer des contraintes en utilisant une matrice de similarit ou dissimilit et en prenant un certain quantile des individus les plus eloigns et les plus proches.
- Utiliser une vrit terrain et ne prendre qu'un nombre limit d'individus dont on sait qu'ils sont ou non dans le mme cluster.

On peut avoir les contraintes ML et CL sous deux formes :

- Deux listes distinctes correspondant a une liste de contraintes ML, et une liste de contraintes CL.
- Une matrice creuse, avec en ML des points de la matrice initialiss a 1, et en CL des points de la matrice initialiss a -1.

Propagation des contraintes entre paires :

Contrainte 1	Contrainte 2	Contrainte combine
$ML(x_i, x_j)$	$ML(x_j, x_k)$	$ML(x_i, x_k)$
$ML(x_i, x_j)$	$CL(x_j, x_k)$	$CL(x_i, x_k)$
$CL(x_i, x_j)$	$ML(x_j, x_k)$	$CL(x_i, x_k)$
$CL(x_i, x_j)$	$CL(x_j, x_k)$	Indfini

Les contraintes ML dfinissant une relation d'quivalence entre objets, elles sont transitives, alors que les contraintes CL sont symtriques mais non transitives.

3.1.6 Algorithmes

Algorithme COP-Kmeans :

Algorithme 1 : COP-KMEANS

Entrée : Jeu de données D, Nombre de clusters k, Contraintes Must-Link

$Con_=\subseteq D \times D$, Contraintes Cannot-Link $Con_!\subseteq D \times D$

Sortie : Le résultat du clustering

1. Soit $C_1 \dots C_k$ les centroïdes initiaux des clusters
 2. **pour chaque** $d_i \in D$ **faire**
 - | Assigner d_i au plus proche centre C_j **tel que** $\text{VIOLATE-CONSTRAINTS}(d_i, C_j, Con_-, Con_!)$ soit Faux. **Si aucune assignation n'est possible, retourner l'ensemble vide**
 - fin**
 3. **pour chaque** *Centroïdes* C_j **faire**
 - | Mettre à jour son centre en moyennant tous les individus d_i appartenant à celui-ci.
 - fin**
 4. Itérer jusqu'à une pseudo-convergence entre l'étape 2. et 3..
 5. Retourner le résultat du clustering.
-

Algorithme 2 : VIOLATE-CONSTRAINTS

Entrée : Jeu de données D, Cluster C, Contraintes Must-Link $Con_=\subseteq D \times D$,

Contraintes Cannot-Link $Con_!\subseteq D \times D$

Sortie : Un booléen

1. **pour chaque** $(d, d_=) \in Con_=\$ **faire**
 - | **si** $d_= \notin C$ **alors**
 - | Retourner Vrai.
 - fin**
 2. **pour chaque** $(d, d_!) \in Con_!$ **faire**
 - | **si** $d_! \in C$ **alors**
 - | Retourner Vrai.
 - fin**
 3. Sinon retourner Faux.
-

Algorithme LCVQE :

Algorithme 3 : LCVQE

Entrée : Jeu de données D, Nombre de clusters k, Contraintes Must-Link

$Con_=\subseteq D \times D$, Contraintes Cannot-Link $Con_-\subseteq D \times D$

Sortie : Le résultat du clustering

On pose $r_1(l), r_2(l)$ les Must-Link, et $s_1(l), s_2(l)$ les Cannot-Link.

On pose aussi Q_j les individus appartenant au cluster j.

1. Soit $C_1 \dots C_k$ les centroïdes initiaux des clusters.

2. Soit $GMLV_j = \{\}$ et $GCLV_j = \{\}$, $\forall j \in 1..k$.

3. pour chaque $d_i \in D$ faire

| Assigner d_i au plus proche centre C_j

fin

4. pour chaque Must-Link $r_1(l), r_2(l)$ tel que C_{j_1} est le plus proche centroïde de $r_1(l)$ et C_{j_2} est le plus proche centroïde de $r_2(l)$ faire

| (a) $\frac{1}{2}[(r_1(l) - C_{j_1})^2 + (r_2(l) - C_{j_2})^2] + \frac{1}{4}[(r_1(l) - C_{j_2})^2 + (r_2(l) - C_{j_1})^2]$.

| (b) $\frac{1}{2}[(r_1(l) - C_{j_1})^2 + (r_2(l) - C_{j_1})^2]$.

| (c) $\frac{1}{2}[(r_1(l) - C_{j_2})^2 + (r_2(l) - C_{j_2})^2]$.

| si (a) est le minimum alors

| | $GMLV_{j_1} = GMLV_{j_1} \cup r_2(l)$ et $GMLV_{j_2} = GMLV_{j_2} \cup r_1(l)$.

| | Assigner r_1 au cluster j_1 et r_2 au cluster j_2 .

| si (b) est le minimum alors

| | Assigner r_1 et r_2 au cluster j_1 .

| si (c) est le minimum alors

| | Assigner r_1 et r_2 au cluster j_2 .

fin

5. pour chaque Cannot-Link $s_1(l), s_2(l)$ tel que C_{j_1} est le plus proche centroïde de $s_1(l)$ et C_{j_2} est le plus proche centroïde de $s_2(l)$ faire

| Fixer $R_n(l) = argmax_{s_i(l), i=1,2} (s_i(l) - C_{j_2})^2$.

| Fixer j_1 comme l'indice du centroïde le plus proche de $R_n(l)$ mais autre que n.

| (a) $\frac{1}{2}[(s_1(l) - C_{j_1})^2 + (s_2(l) - C_{j_1})^2] + \frac{1}{2}(R_{j_1}(l) - C_{j_1})^2$.

| (b) $\frac{1}{2}[(s_1(l) - C_{j_1})^2 + (s_2(l) - C_{j_2})^2]$.

| si (a) est le minimum alors

| | $GCLV_{j_1} = GCLV_{j_1} \cup R_{j_1}(l)$.

| | Assigner s_1 et s_2 au cluster j_1 .

| si (b) est le minimum alors

| | Assigner s_1 au cluster j_1 et s_2 au cluster j_2 .

fin

6. Mettre à jour les C_j comme suit :

$$C_j = \frac{1}{N_j} \sum_{s_i \in Q_j} s_i + \frac{1}{2} \sum_{s_i \in GMLV_j} s_i + \sum_{s_i \in GCLV_j} s_i$$

7. Répéter les étapes 3. à 6. jusqu'à une pseudo-convergence.

8. Retourner le résultat du clustering.

3.2 Réduction de dimension non linéaire pour visualiser les données

Les jeux de données comme des images multi-temporelles et multi-bandes qui sont donc de grande dimension sont difficiles à visualiser. Pour nous aider à visualiser si il y a des structures inhérentes dans notre jeu de données, on va réduire les dimensions à 2 grâce à des algorithmes de Manifold learning (réduction de dimension non linéaire) et les afficher sur un plan. Les algorithmes testés sont "Multi-dimensional Scaling" (MDS), "Isomap" et "t-distributed Stochastic Neighbor Embedding" (t-SNE).

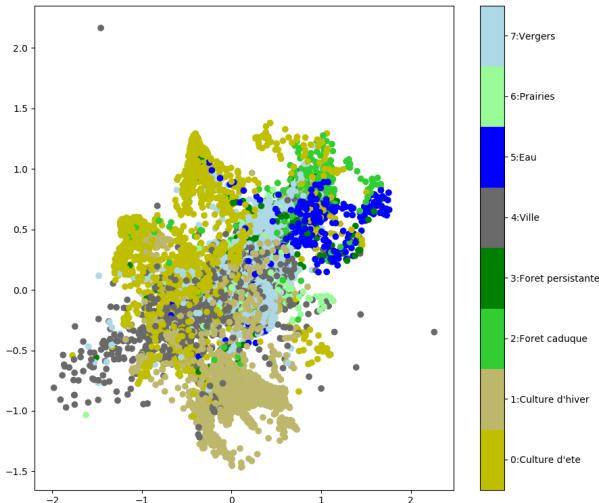


FIGURE 3.1 – Multi-dimensional Scaling

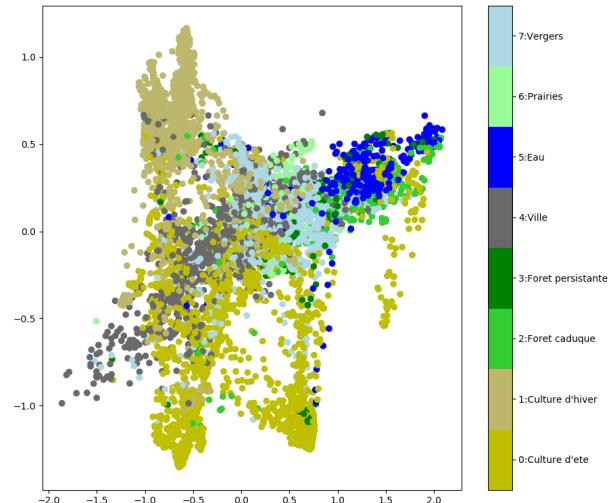


FIGURE 3.2 – Isomap

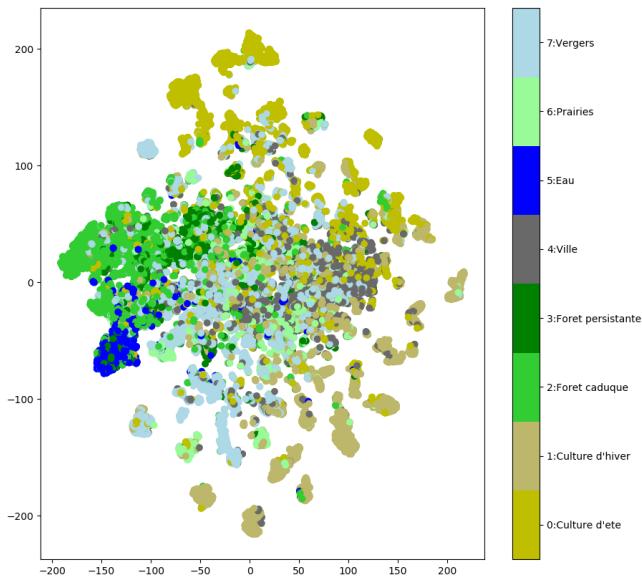


FIGURE 3.3 – t-distributed Stochastic Neighbor Embedding

La représentation sur un espace en deux dimensions de notre réduction de dimensions non linéaire nous montre qu'aucune structure ne ressort et que les données sont assez homogènes, et donc le fait de les séparer avec un algorithme de clustering ne sera pas aisément.

3.3 Application des algorithmes COP-Kmeans et LCVQE

Pour réaliser notre comparatif, on va regarder trois critères, deux utilisant une vérité terrain, l'autre utilisant la dispersion des clusters :

- **L'indice de rang ajusté (ARI)**, mesure la similarité entre deux clusterings (vérité terrain et résultat d'un clustering) en considérant toutes les paires d'individus et compte lesquels sont assignés au même cluster ou à un différent cluster. Plus la valeur est proche de 0, plus les clusterings sont aléatoires, au contraire, un score proche de 1 montre que les clustering sont identiques, alors que -1 signifie aucune correspondance.

$$RI = \frac{a + b}{C_2^{n_{samples}}}$$

Avec a le nombre de paires d'individus appartenant au même cluster dans la vérité terrain ET le clustering, b le nombre de paires d'individus appartenant à des clusters différents dans la vérité terrain ET la clustering et $C_2^{n_{samples}}$ le nombre total de paires possibles

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}$$

- **L'information mutuelle ajustée (AMI)**, mesure la similarité entre les clusters de deux clusterings (vérité terrain et résultat d'un clustering). Plus la valeur est proche de 0, plus les clusterings sont aléatoires, au contraire, un score proche de 1 montre que les clustering sont identiques, alors que -1 signifie aucune correspondance.

$$MI(U, V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} \frac{|U_i \cap V_j|}{N} \log \left(\frac{N |U_i \cap V_j|}{|U_i| |V_j|} \right)$$

Avec U et V la vérité terrain et le clustering et N le nombre de classes.

$$AMI = \frac{MI - E[MI]}{\max(H(U), H(V)) - E[MI]}$$

$$\text{Avec } H(U) = - \sum_{i=1}^{|U|} \frac{|U_i|}{N} \log \left(\frac{|U_i|}{N} \right)$$

- **Le score de Calinski & Harabaz (CH)**, mesure un ratio entre la dispersion intra-cluster et la dispersion inter-clusters. Plus le score est élevé, plus les clusters sont denses et bien séparés.

$$CH(k) = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N - k}{k - 1}$$

Avec Tr() la trace, N le nombre total d'individus, k le nombre de clusters, B_k est la matrice de dispersion intra-cluster et W_k est la matrice de dispersion inter-cluster, définis par :

$$W_k = \sum_{q=1}^k \sum_{x \in C_q} (x - c_q)(x - c_q)^T$$

$$B_k = \sum_q n_q (c_q - c)(c_q - c)^T$$

Avec C_q les individus appartenant au cluster q, c_q le centre du cluster q, n_q le nombre d'individus appartenant au cluster q et c le centre de l'ensemble.

Pour réaliser notre étude sur les critères mesurant la qualité du clustering, on a tout d'abord créé 5 jeux de contraintes de tailles différentes : 10, 50, 100, 200 et 500 puis on a fait tourner 10 fois le même algorithme pour au final faire la moyenne des résultats sur les 3 critères. Un paramètre de 8 classes a été fixé et un repère est réalisé avec l'algorithme Kmeans.

		Nombre de contraintes									
		10		50		100		250		500	
Score		<i>ARI</i>	<i>AMI</i>	<i>ARI</i>	<i>AMI</i>	<i>ARI</i>	<i>AMI</i>	<i>ARI</i>	<i>AMI</i>	<i>ARI</i>	<i>AMI</i>
Algorithmes	<i>COP-Kmeans</i>	0.243	0.325	0.251	0.337	0.250	0.336	0.248	0.335	0.250	0.337
	<i>LCVQE</i>	0.241	0.323	0.250	0.335	0.253	0.336	0.252	0.337	0.233	0.319
	<i>Kmeans (ref)</i>	$ARI = 0.251 / AMI = 0.337$									
Score		HC		HC		HC		HC		HC	
Algorithmes	<i>COP-Kmeans</i>	12508		12894		12822		12658		12414	
	<i>LCVQE</i>	12486		12881		12716		12707		12292	
	<i>Kmeans(ref)</i>	$HC = 12953$									

Quatre possibles "explications" à la perte/stagnation de qualité :

- Cohérence : le clustering sans contrainte respecte déjà les contraintes.
- Silhouette : les classes initiales ne sont pas "clusterisables".
- Contraintes : les contraintes sont erronées ou rentrent en conflit.
- Algorithme : sensibilité des algorithmes aux contraintes

Rappelons notre vérité terrain pour des comparatifs visuels :

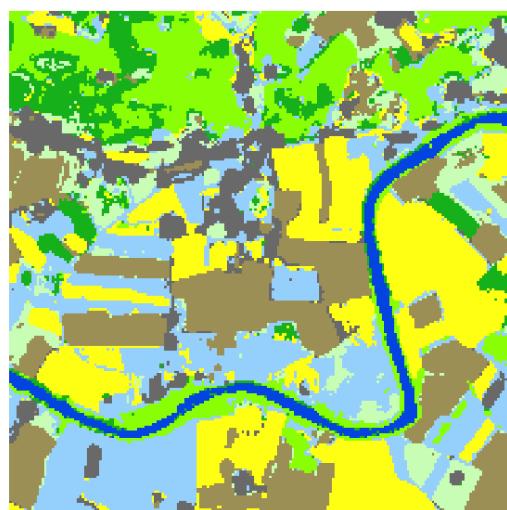


FIGURE 3.4 – Vérité terrain

Les couleurs des classes sont choisis à la main sur une palette de couleur avec le code hexadécimal de celles-ci.

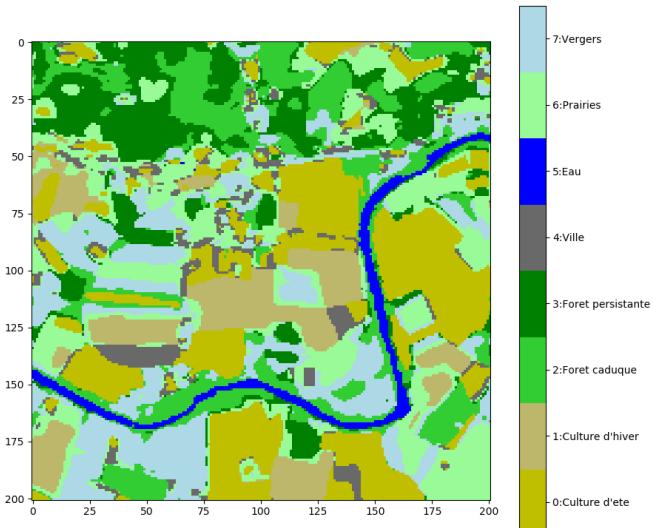


FIGURE 3.5 – Résultat du clustering avec COP-Kmean

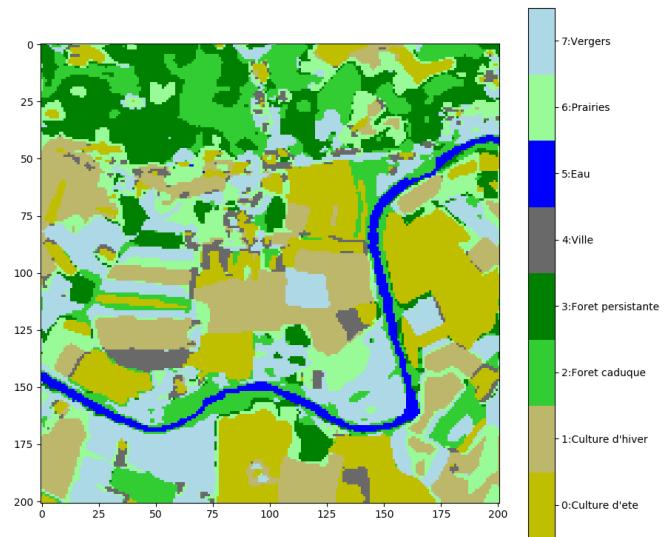


FIGURE 3.6 – Résultat du clustering avec LCVQE

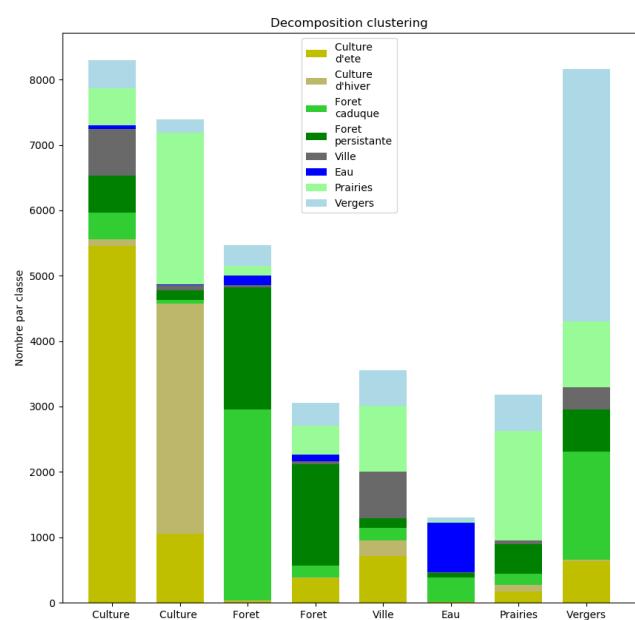


FIGURE 3.7 – Erreur des attributions des clusters pour COP-Kmeans

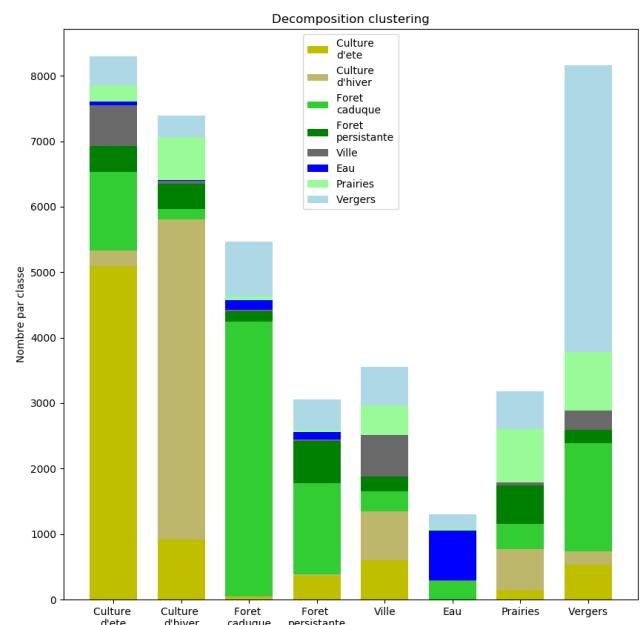


FIGURE 3.8 – Erreur des attributions des clusters pour LCVQE

On constate qu'il est très difficile de classer les villes. Quant aux autres individus, plus de 50% sont bien classés, ce qui est raisonnable sachant que le clustering allait être difficile.

3.4 Clustering en utilisant une mesure de dissimilarité : Dynamic Time Warping

Dynamic Time Warping (DTW), ou Déformation Temporelle Dynamique en français, est un algorithme spécialisé dans la mesure de dissimilarité entre deux séquences temporelles. Dans notre cas, ces séquences temporelles sont nos profils temporels des pixels. Ce n'est pas une distance du fait de la non symétrie, même si ça s'en rapproche. La DTW cherche à harmoniser une série temporelle sur une autre en appliquant une déformation sous certaines restrictions, idéal quand les séries ne se superposent pas totalement. L'algorithme de base étant en complexité $O(N^2)$ puisque il parcourt une matrice des permutations possibles, une méthode pour l'améliorer est de donner des transitions admissibles pendant la recherche du chemin minimum, obligeant le chemin minimum à rester au plus proche de la diagonale. De ce fait, on a plus besoin de parcourir la matrice de tous les alignements possibles mais seulement une partie.

Il est préférable pour comparer des séries temporelles d'utiliser DTW par rapport à des distances comme les distances de minkowski, cosinus et autres, car elle gère mieux les séries temporelles.

Pour visualiser la mesure de similarité effectuée par la DTW, trois façons existent, par comparaison point à point, par inspection du vis-à-vis, ou par affichage de densité.

Les deux figures ci-dessous sont deux exemples : celle de gauche représente deux séries très proches, et celle de droite deux séries ne correspondant pas.

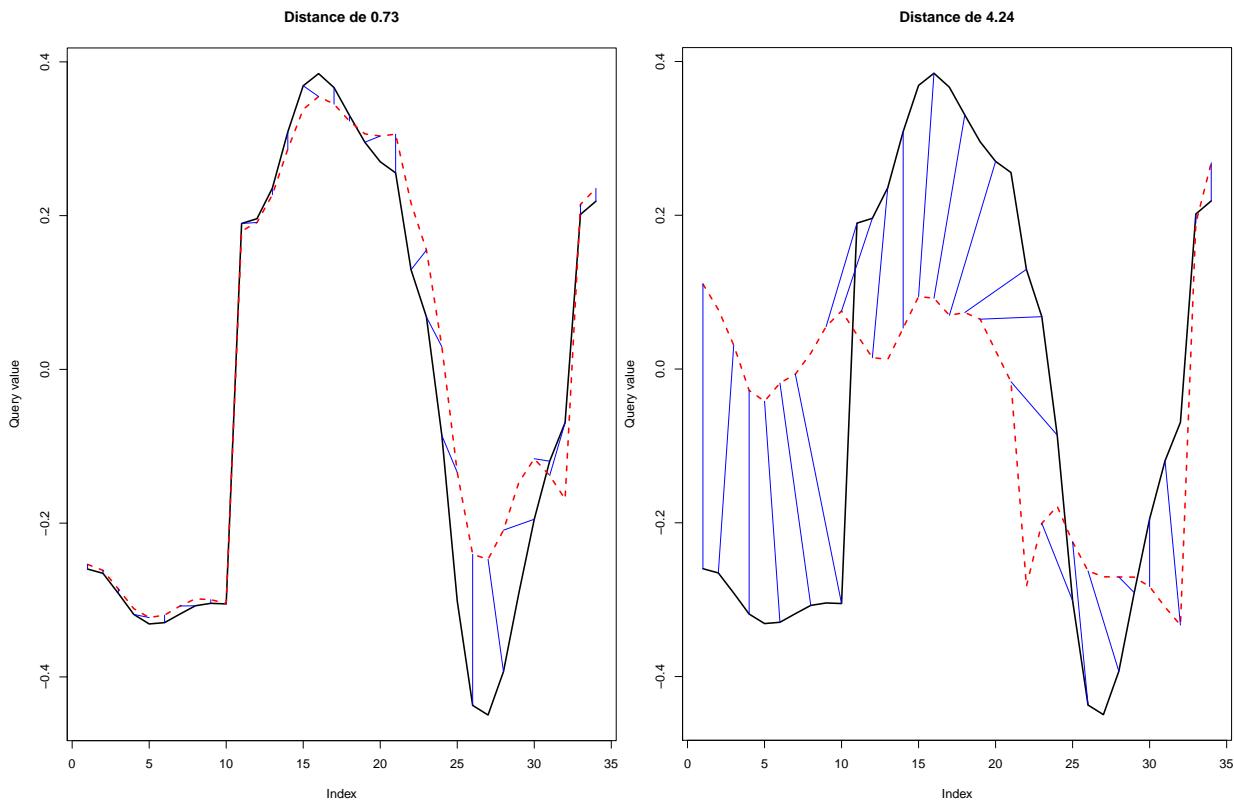


FIGURE 3.9 – Alignement des séries temporelles

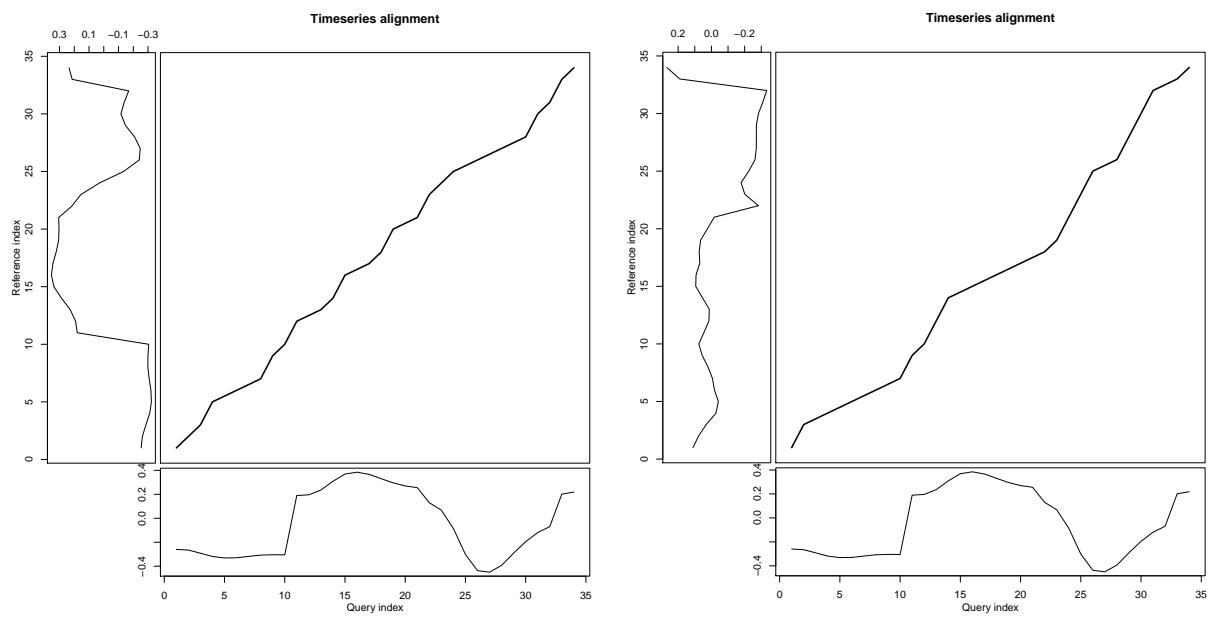


FIGURE 3.10 – Vis-à-vis des séries temporelles et courbe de déformation

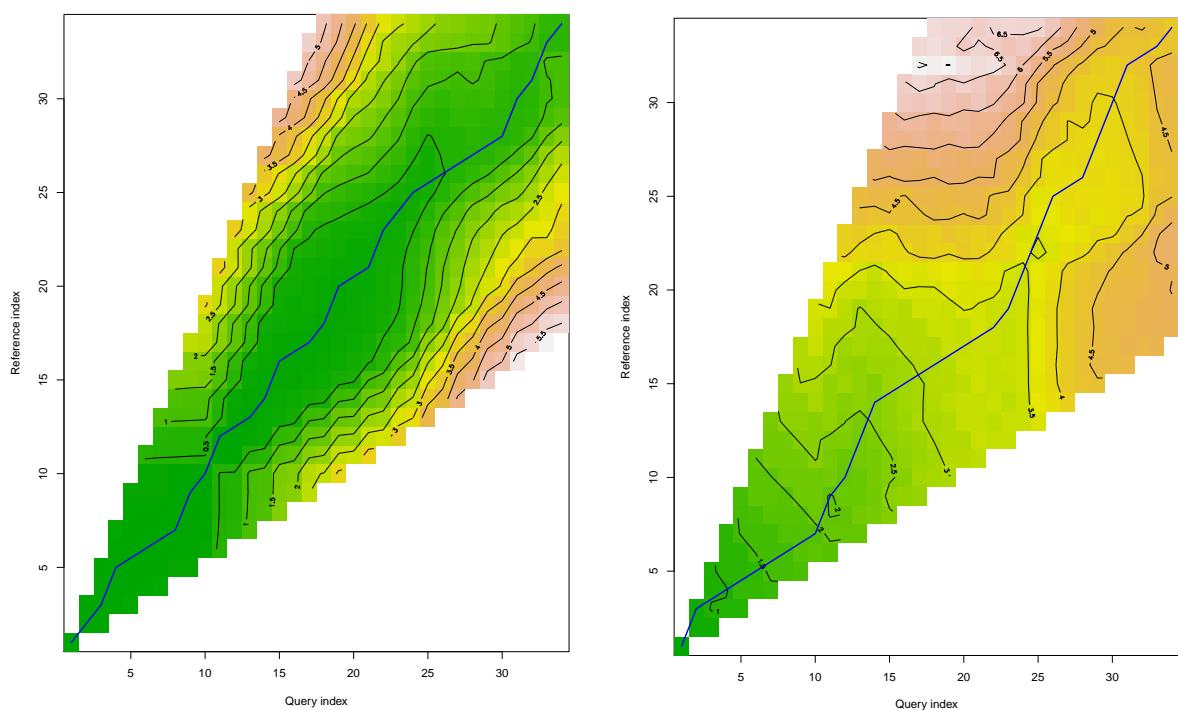


FIGURE 3.11 – Carte du coût cumulatif et chemin imposé

La DTW ressort une mesure de dissimilarité. Pour passer d'une mesure de dissimilarité à une mesure de similarité, on peut utiliser un noyau affine ou un noyau gaussien pour transformer nos données dans l'intervalle $[0, 1]$ avec 1 signifiant similaire et 0 non similaire, puis normaliser notre résultat pour que la somme fasse 1.

Un noyau affine prend la forme suivante :

Soit $X \in \mathbb{R}^n$, $\forall i \in \{1..n\}$ on a :

$$X'_i = \frac{X_i}{\sum_{j=0}^n \left(\frac{X_j}{\sum X} \right)}$$

Le noyau affine a tendance à conserver les proportions d'écart entre chaque valeur. Par exemple si on a deux scores de valeurs 2 et 4, alors le noyau linéaire va transformer ce résultat en $\frac{2}{3}$ et $\frac{1}{3}$ qui garde la proportion d'un rapport 2.

Un noyau gaussien prend la forme suivante :

Soit $X \in \mathbb{R}^n$, $\forall i \in \{1..n\}$ on a :

$$X'_i = \frac{\exp\left(\frac{-X_i^2}{2\sigma^2}\right)}{\sum_{j=0}^n \exp\left(\frac{-X_j^2}{2\sigma^2}\right)}$$

Le noyau gaussien quand à lui, à tendance à mettre plus de poids aux points proches de 0, ce qui permet en général de faire ressortir un score plus qu'un autre. On le priviliege par rapport au noyau linéaire car il sépare mieux les scores similaires des scores dissimilaires.

Plusieurs méthodologies s'offrent à nous pour utiliser la DTW couplée au clustering :

1. Soit partir d'un résultat de clustering, puis raffiner les clusters en les fusionnant par similarité entre eux jusqu'à avoir k clusters.
2. Soit partir d'un résultat de clustering avec un plus grand nombre de classes que celui souhaité, puis raffiner en fusionnant les clusters par similarité à des profils temporels de référence.
3. Soit utiliser des profils temporels de référence, puis attribuer les pixels au profil temporel le plus similaire.

Les profils temporels de référence se présentent sous la forme suivante, avec en trait épais représentant la médiane qui est plus robuste aux outliers que la moyenne (nos profils temporels de référence), et en fond semi-transparent la variance (rappel : 8 profils temporels de référence, 7 bandes dérivées des indices radiométriques et 11 dates par bande) :

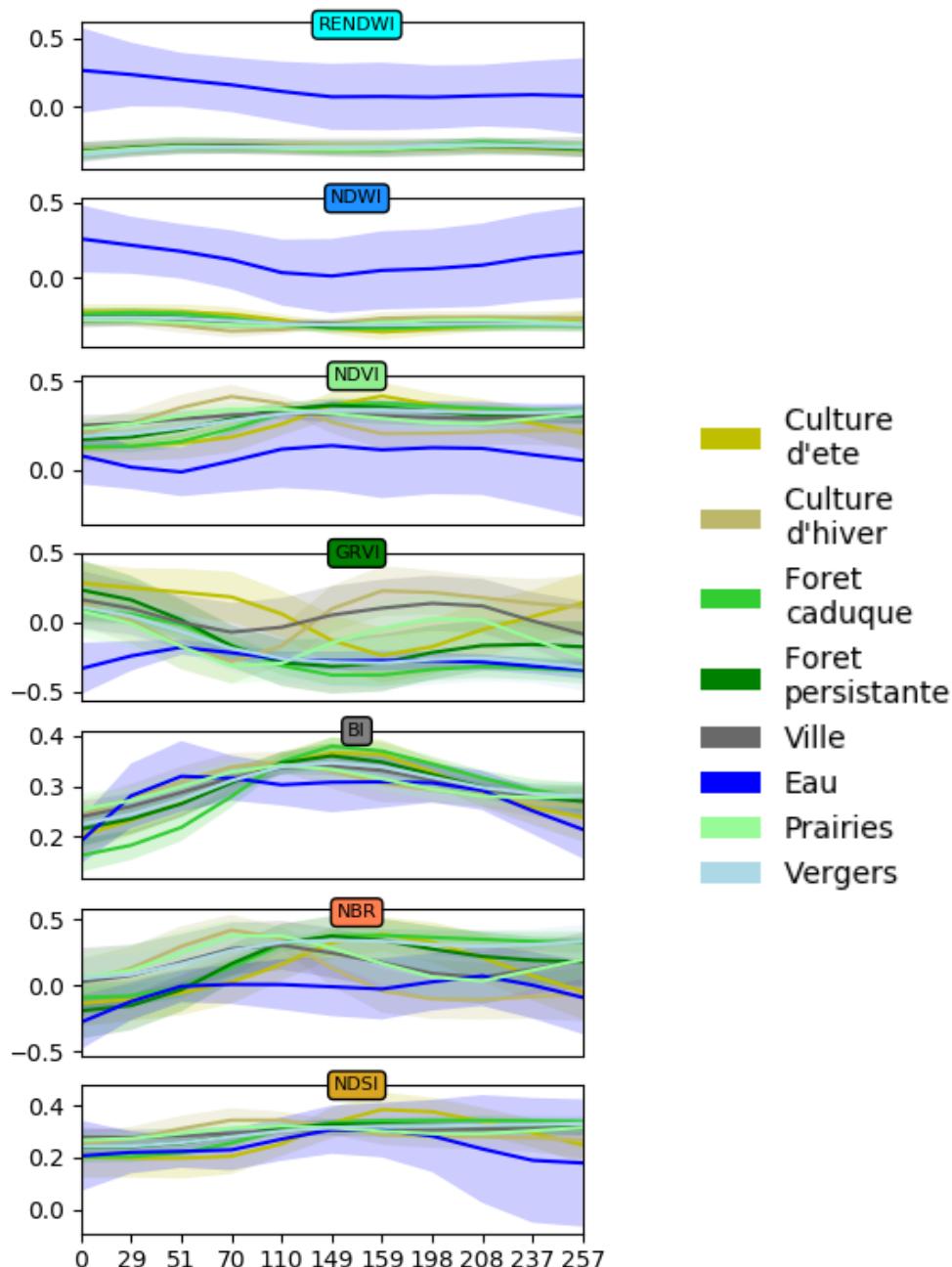


FIGURE 3.12 – Profils temporels de référence

3.4.1 Raffinement par similarité inter-classes

Prenons en exemple nos 8 classes (*part. 2.9*).

En créant une matrice de similarité entre chaque pixel, on pourrait, en théorie, utiliser un algorithme de classification hiérarchique ascendante (CAH) pour grouper les individus entre eux. Dans l'exemple ci-dessous qui a été réalisé sur seulement nos 8 profils temporels, on constate qu'on a deux voire trois classes qui ressortent. Cependant, cette méthode est extrêmement gourmande en mémoire sur des images, car une matrice de similarité sur une image 200x200 (qui n'est pas très grande) revient à calculer pour chaque pixel sa similarité aux autres pixels, et donc d'avoir une matrice de similarité de taille 200x200x200x200, ce qui revient à stocker 1 600 000 000 points.

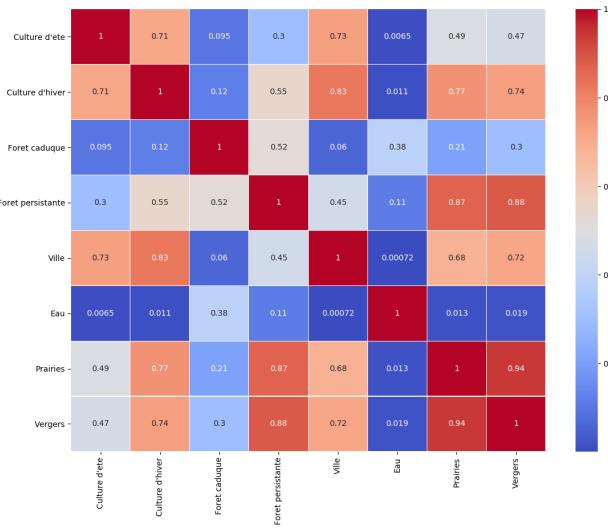


FIGURE 3.13 – Matrice de similarité

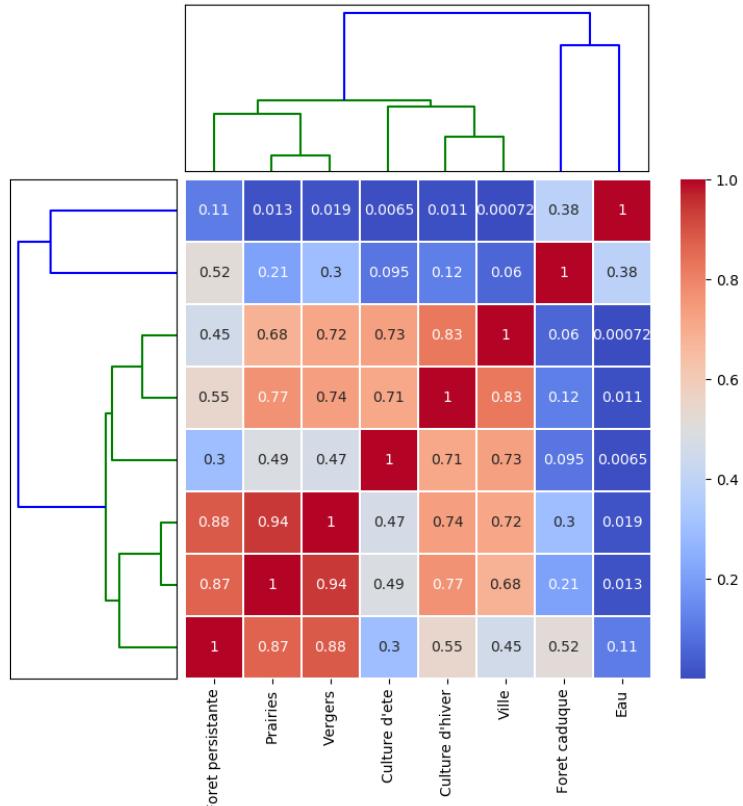


FIGURE 3.14 – CAH sur la matrice de distance

3.4.2 Raffinement par similarité à des références temporelles

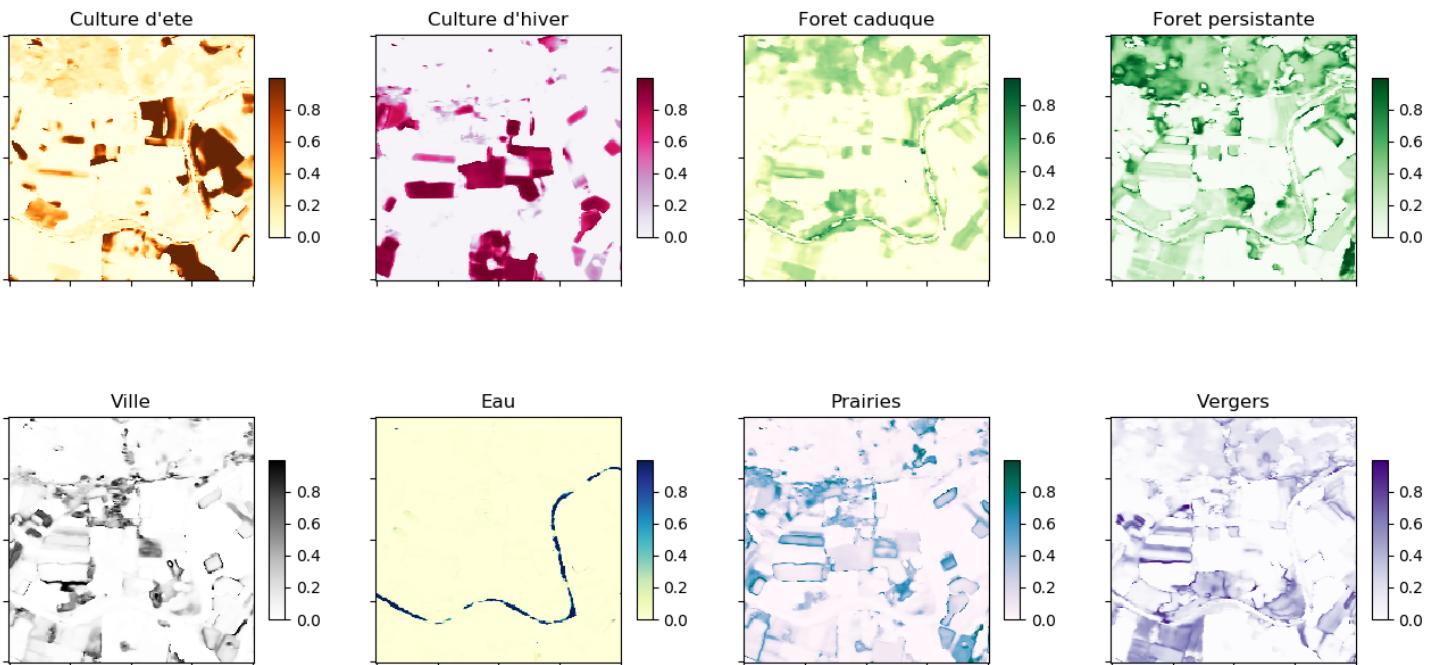
Appliquons tout d'abord l'algorithme KMeans sur notre jeu de données, avec un nombre k de classes supérieur aux 8 classes que l'on cherche à trouver, par exemple k=16. À partir de ces 16 clusters, trouver la médiane de chaque cluster et la comparer aux profils de références en utilisant la mesure de dissimilarité DTW. On projette alors nos 16 clusters sur les 8 classes de référence en prenant la distance minimale obtenue par cluster. Les résultats par rapport aux algorithmes sous contraintes sont les suivants :

<i>Ari</i> avant raffinement	<i>Ami</i> avant raffinement	<i>Ari</i> après raffinement	<i>Ami</i> après raffinement
0.24	0.32	0.32	0.38

On constate que les résultats sont bien meilleurs qu'un clustering directement sur 8 classes, mais il faut avoir cette information sur les profils temporels au départ.

3.4.3 Clustering par similarité à des références temporelles

Le principe de cette méthode est de réaliser un clustering en regardant les DTW des profils temporels de chaque pixel et de les comparer aux différentes références temporelles, puis de regarder le score réalisé et de prendre le score minimal correspondant à la similarité la plus forte. On a 8 classes de référence, et donc 8 scores à comparer par pixel.



Cette méthode peut s'apparenter à du clustering en un seul pas autour de médoïdes fixes qui sont nos profils de référence, et une mesure de distance différente de la distance euclidienne usuelle, puisque ici on utilise la DTW.

Chapitre 4

Détection de changements

La détection de changement dans des images multi-temporelles est un problème mal posé, plusieurs directions peuvent être prises pour différents types de données.

Est-ce que l'image est mono-bande ou multi-bandes ? Est-ce que le capteur prend les mesures de manière homogène ? Est-ce qu'on fait une détection bi-date ou multi-temporelle ?

Dans notre cas, on se fixe à la détection des changements dans des images multi-bandes et multi-temporelles corrigées au niveau atmosphérique. Pour se faire, on va utiliser des algorithmes de machine learning, dont le but est de détecter les individus qui ne sont "pas normaux". Ce procédé est connu sous le nom de détection d'anomalies. Une des premières définitions à ce sujet a été dite par Grubbs en 1969 : "Une observation isolée, ou anomalie, est une observation qui dévie sûrement des autres membres d'un échantillon auquel elle appartient". Cette version de l'époque était plus dans l'optique de nettoyer les données d'apprentissage des outliers car certains algorithmes étaient sensibles aux outliers.

Avec le développement d'algorithmes plus robustes, l'intérêt dans la détection d'anomalies a décrue. Cependant, il y a eu un tournant dans les années 2000, quand des chercheurs se sont intéressés plus aux anomalies en elles mêmes, car elles sont souvent associées à des événements intéressants ou des enregistrements suspicieux.

Depuis, de nombreux algorithmes ont vu le jour et la définition de Grubbs a été étendue si bien qu'aujourd'hui, les anomalies sont connues pour avoir deux caractéristiques :

1. Les anomalies diffèrent de la normalité.
2. Elles sont rares dans un jeu de données en comparaison des données normales.

De ce fait, l'image prise devra avoir un terrain homogène, avec un changement ne prenant pas un grand pourcentage de l'image.

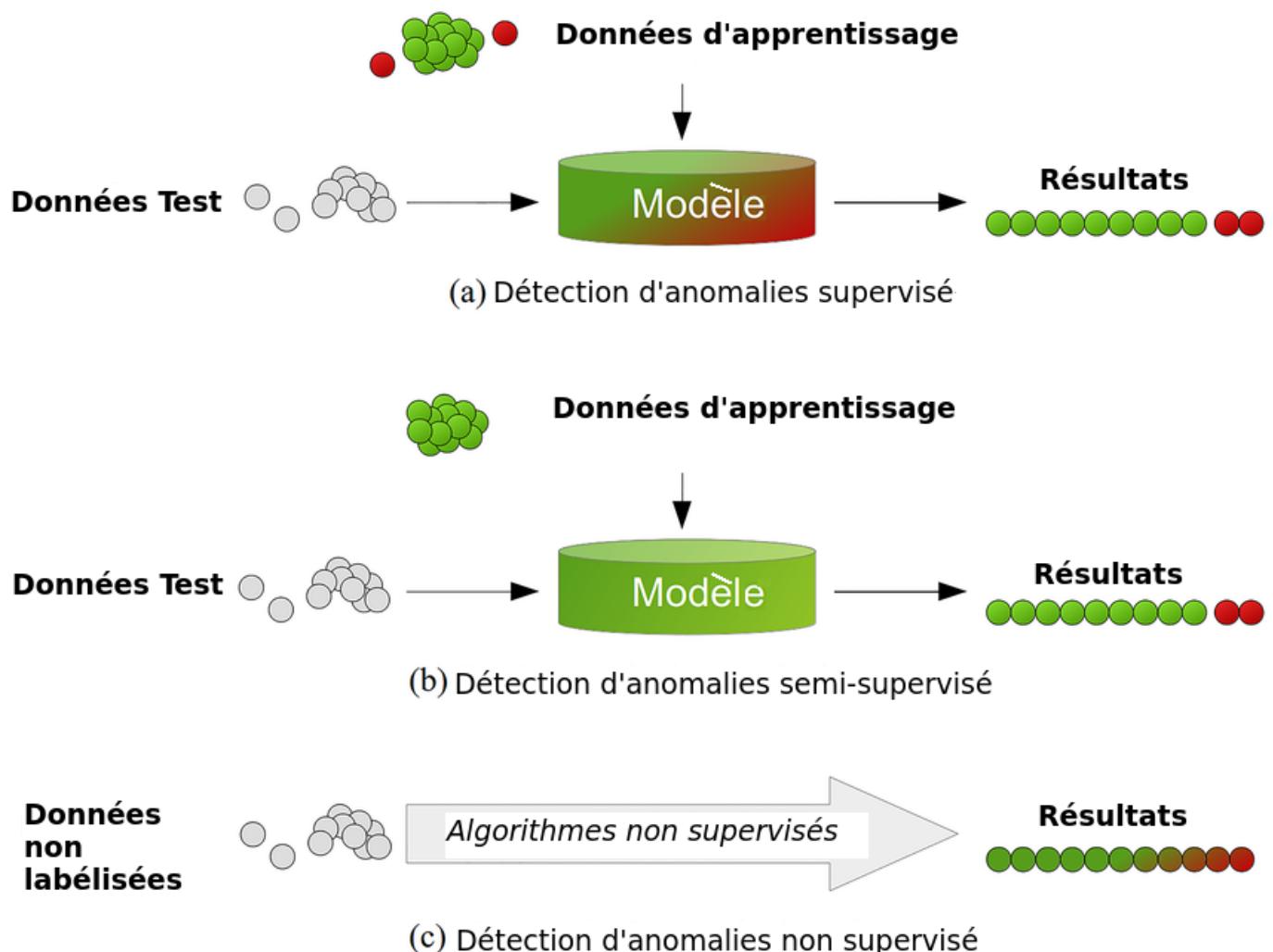


FIGURE 4.1 – Résumé de la détection d'anomalies

Dans le cadre du stage, on va s'occuper des algorithmes non supervisés pour la détection de changements.

L'image sur laquelle on va se baser est une zone de taille 300x300 située sur Toulouse, correspondant au futur "Parc des expositions et Centre de conventions de Toulouse Métropole" (PEx). On le situe au centre de l'image en blanc, avec tout autour des évolutions comme des parkings ou de nouvelles routes.



FIGURE 4.2 – PEx au temps 0



FIGURE 4.3 – PEx au temps 5



FIGURE 4.4 – PEx au temps 11

4.1 Détection d'outliers (anomalies)

4.1.1 Algorithmes utilisés

Les algorithmes de détection d'outliers utilisés ici sont des algorithmes non supervisés, ils prennent en entrée un jeu de données et cherchent directement des individus correspondant à des outliers.

En s'aidant du package spécialisé pour la détection d'outlier sous python "PyOD", plusieurs méthodes ont été testées, dont voici la liste :

- Angle-base Outlier Detection (ABOD), est un algorithme utilisé dans le cas de grandes dimensions. Le principe pour contrer le fléau de dimension est non plus de prendre des distances entre les points, mais plutôt l'angle réalisé entre un point et les autres points. Plus ses angles sont petits, plus il y a de chance que le point soit un outlier car les angles pointent tous vers une densité de distribution. À contrario, un point qui n'est pas un outlier verra ses angles avec les autres points assez disparates, car il sera au milieu d'une densité de distribution et donc les angles partiront dans toutes les directions.
- k-Nearest Neighbors Detector (kNND), est un algorithme dérivé de kNN utilisé comme moyen de mesurer la densité. Le but est de regarder pour un individu, la distance à son k-ième voisin comme un score d'outlier. On peut regarder pour tous les individus le maximum de distance du k-ième voisin, la moyenne des k premiers voisins ou bien la médiane des k premiers voisins.
- Local Outlier Factor (LOF) utilise aussi kNN. On peut le voir comme une mesure de déviation locale de la densité d'un échantillon donné par rapport à ses voisins.
- **Clustering Based Local Outlier Factor (CBLOF)** décompose le résultat d'un clustering (KMeans par exemple) en regardant cluster par cluster des sous clusters de grandes et petites tailles. La score d'outlier se fait alors en deux parties, la taille du cluster auquel appartient l'individu, et sa distance par rapport au plus proche cluster voisin de taille assez grande.
- Histogram-based Outlier Detection (HBOS) est un algorithme assumant que les features en entrée sont indépendantes, et calcule des histogrammes pour détecter les outliers.
- Outlier Detection with Minimum Covariance Determinant (MCD) est utilisé pour détecter des outliers dans un jeu de données avec une distribution gaussienne en utilisant le déterminant de covariance minimum.
- Principal Component Analysis Outlier Detector (PCAOD) utilise un algorithme de réduction de dimension linéaire pour chercher des outliers. Son fonctionnement cherche la distance du projeté d'un individu par rapport aux vecteurs propres.
- **Isolation Forest Outlier Detector (IFOD)**, comme ABOD, est utilisé dans le cas de grandes dimensions. Il utilise l'algorithme des forêts d'arbres aléatoires. Il "isole" des individus en choisissant de manière aléatoire une feature, puis en choisissant une valeur de séparation entre la valeur minimale et la valeur maximale pour les séparer en deux. Comme le partitionnement peut se représenter comme un arbre, le nombre de partitionnement requis pour isoler un échantillon d'individus est équivalent à la longueur du chemin jusqu'au noeud final. Plus la longueur du chemin est longue, plus les points restants ont de chances d'être des outliers.
- **One-class Support Vector Machine (OCSVM)** est un dérivé des SVM, très proche d'un classifieur binaire pour SVM. Son objectif est de trouver l'hyperplan avec la plus grande marge entre les points pour les séparer en deux. Un "Kernel Trick" est utilisé pour transformer ce classifieur linéaire en classifieur non linéaire.

4.2 Résultats des détections d'anomalies

Parmi les algorithmes testés, trois ont été retenus car ils ont un bon compromis temps de calcul et précision, One Class SVM, Isolation Forest et CBLOF.

Ils nous ressortent une carte avec un score de déviation par pixel, et plus le score du pixel est grand, plus il a de chance d'être un outlier.

Pour avoir le masque de changement, on utilise un quantile sur la carte de score correspondant à 15% des valeurs les plus hautes.

Commençons par Isolation Forest :

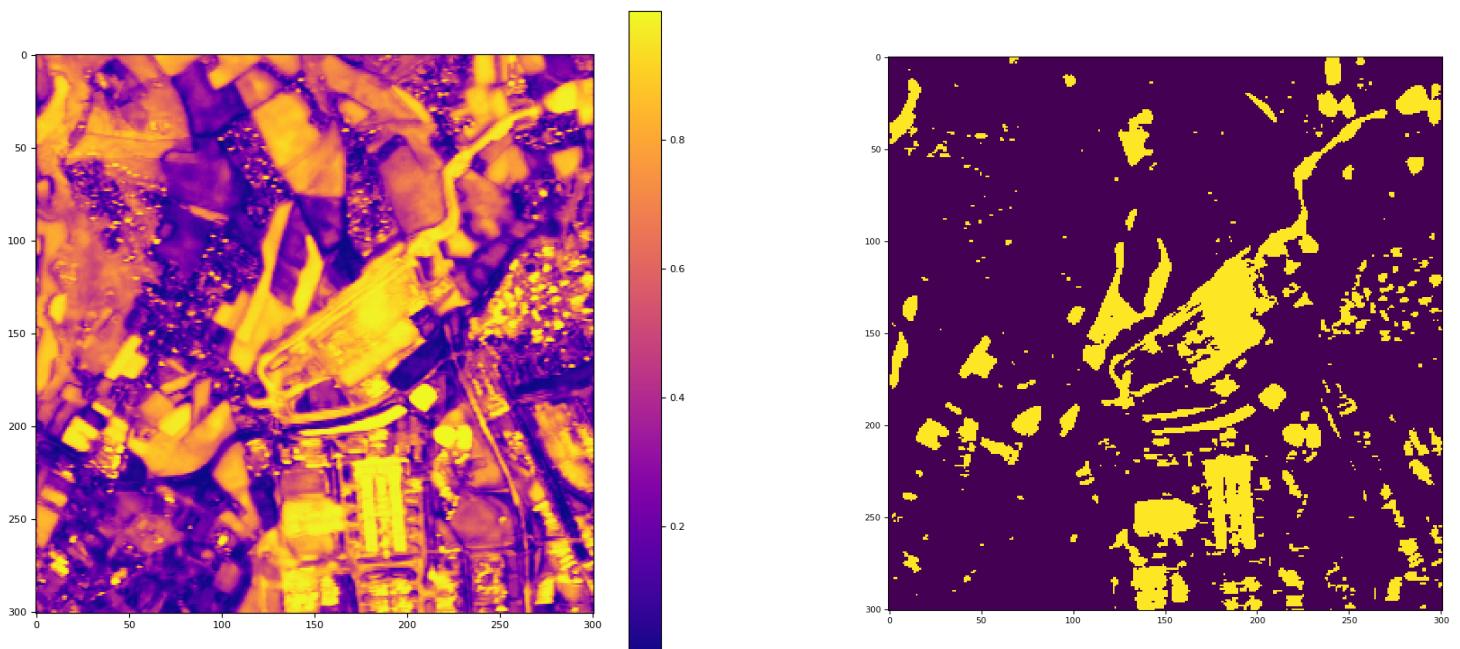


FIGURE 4.5 – Isolation forest
à gauche, le score des outliers, à droite, le masque de changement

Le résultat d'Isolation forest semble avoir réussi à capturer la zone où se déroule les changement, mais il a aussi capturer beaucoup de résidus.

Ensuite, pour One Class SVM, deux kernels sont sortis du lot, RBF (radial basis function) et Sigmoïd, voici donc les résultats avec ceux-ci :

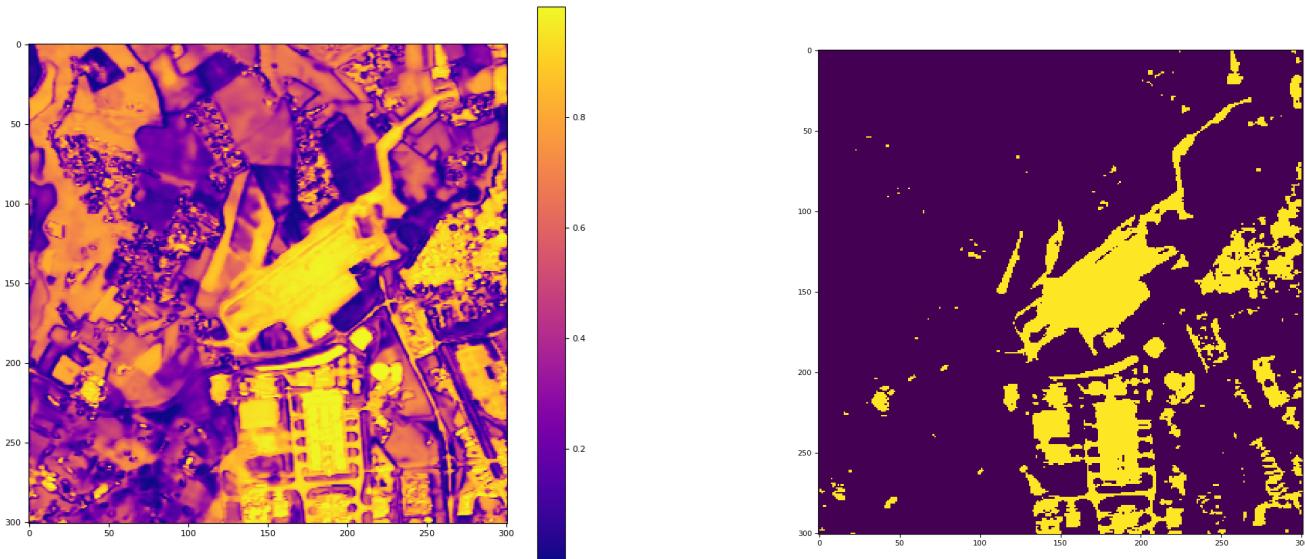


FIGURE 4.6 – One Class SVM, kernel RBF
à gauche, le score des outliers, à droite, le masque de changement

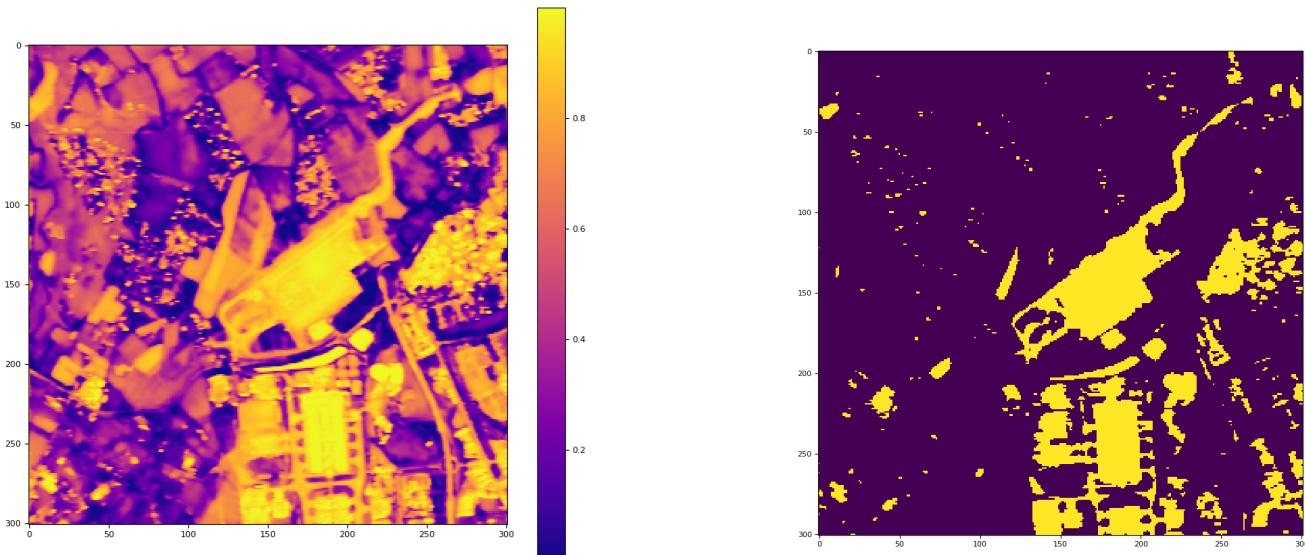


FIGURE 4.7 – One Class SVM, kernel sigmoïd
à gauche, le score des outliers, à droite, le masque de changement

On constate pour One Class SVM que les résultats sont mieux qu'avec Isolation Forest, il y a beaucoup moins de résidus sur le masque de changement. La différence entre ces deux kernels n'est pas flagrante, avec un kernel polynomial par exemple, les résultats seraient bien moindres, et même pires que Isolation forest.

Finalement, les résultats de CBLOF, avec Mini Batch KMeans comme clustering :

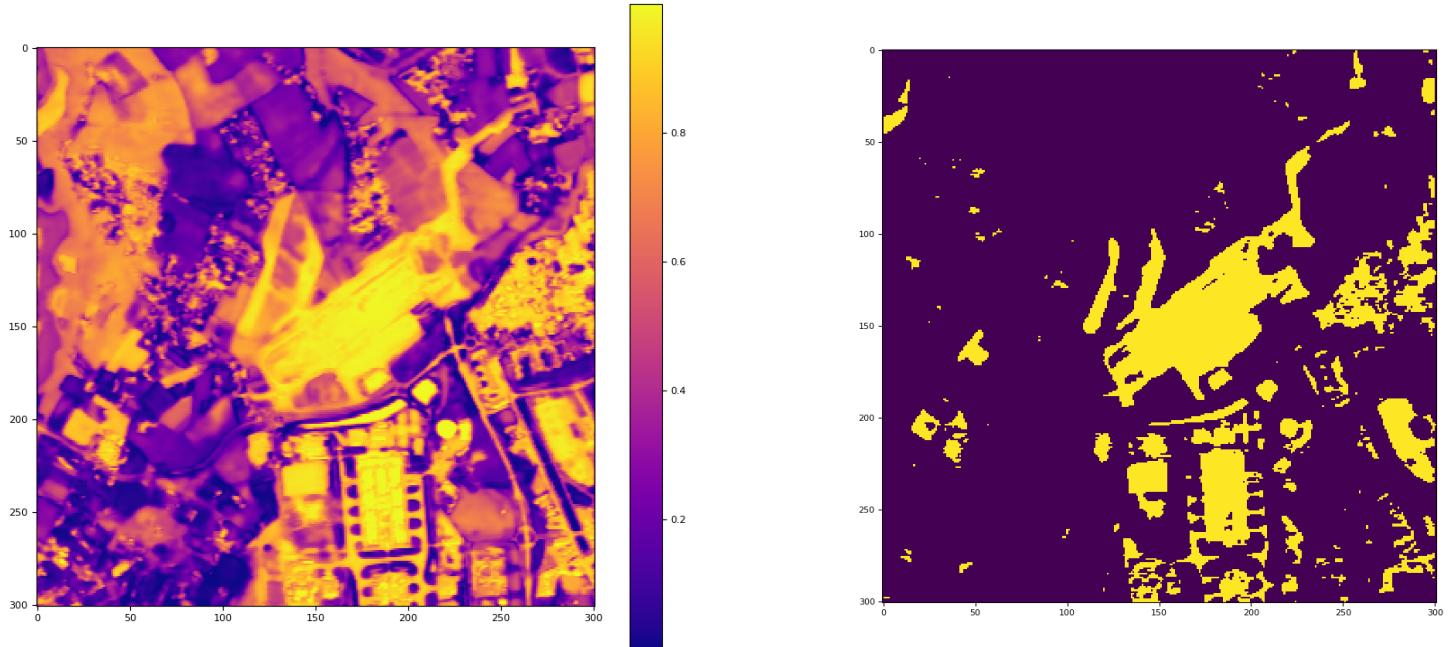


FIGURE 4.8 – CBLOF
à gauche, le score des outliers, à droite, le masque de changement

CBLOF est du même calibre que One Class SVM sur la qualité de résultat. Cependant, il y a un facteur qui rentre en compte et qui n'est pas affiché précédemment qui fait en sorte qu'il est meilleur que son rival : Le temps de calcul.

En effet, l'algorithme est dérivé des méthodes de clustering, n'importe quelle méthode peut être appliquée, en particulier Mini Batch KMeans, très efficace en temps de computation, qui semble idéale pour de grandes images.

C'est donc le meilleur algorithme à mon sens pour ces types de données.

4.3 Détection d'outliers en utilisant une matrice de dissimilarité

Dans la même optique que dans la partie clustering en utilisant la DTW, on va ici aussi l'utiliser mais à fin de détecter des changements. Rappelons que DTW créer une mesure de dissimilarité entre deux profils temporels, c'est à dire que plus la valeur est grande, plus la dissimilarité est importante. Deux méthodes s'offrent à nous pour utiliser la DTW :

- Utiliser un seuillage sur la matrice de dissimilarité, ce qui voudrait dire qu'on connaît l'intensité à partir de laquelle on peut considérer un pixel comme changeant.
- Utiliser un quantile sur la matrice de dissimilarité, ce qui voudrait dire qu'on connaît le taux de pixel changeant dans l'image.

Comme on connaît à peu près le taux de changement dans l'image, on utilise alors la méthode du quantile avec un taux de changement à 15%.

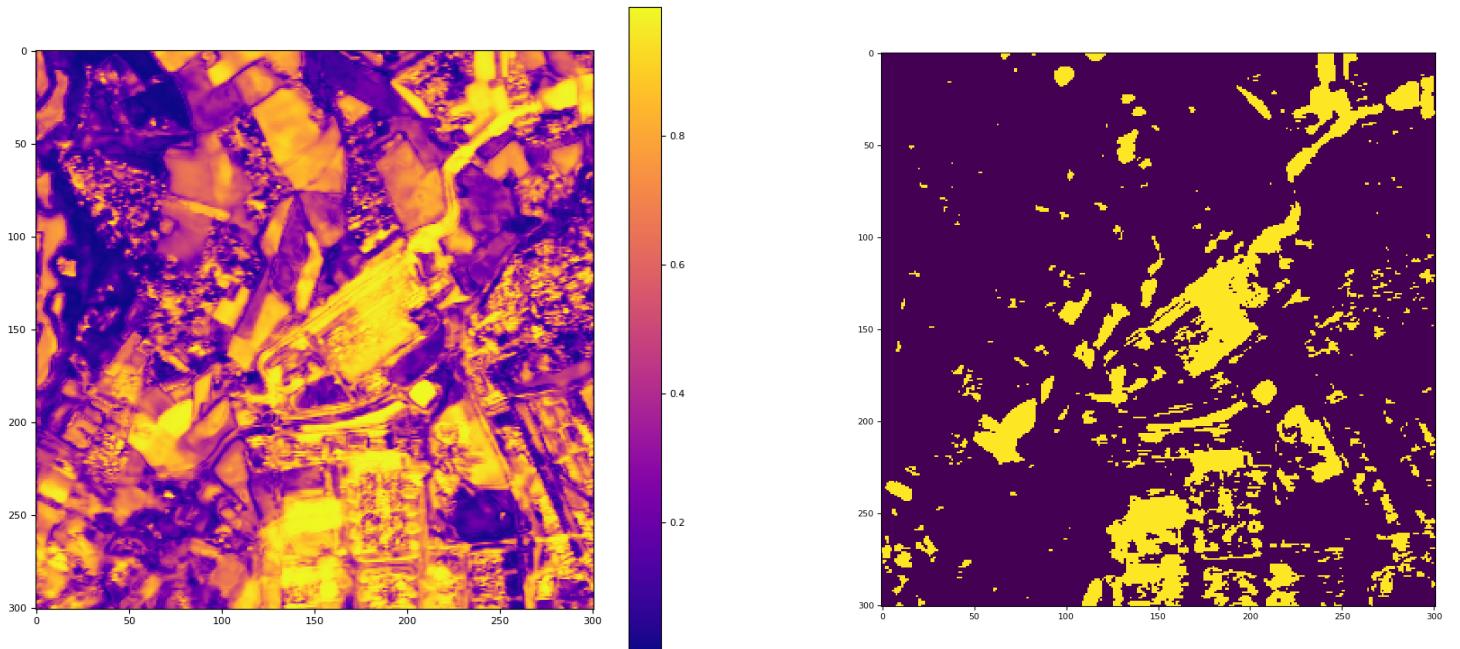


FIGURE 4.9 – DTW
à gauche, le score des outliers, à droite, le masque de changement

Le résultat de la détection de changement avec cette méthode est du même calibre que pour Isolation Forest. On a beaucoup de résidus, même si l'image capturée correspond en grande partie à la zone du changement.

Chapitre 5

Discussions

5.0.1 Qualité de la carte de vérité terrain

Notre carte de vérité terrain est extraite d'une carte d'occupation des sols de 2017 labellisée à partir d'un algorithme de machine learning (Random Forest) sur toute la France. Pour se faire, des images de 2016 ont été utilisées.

Notre carte cumule donc les erreurs, d'une part car Random Forest n'atteint pas une précision 100% selon les classes, et d'autre part des dates antérieur à nos images étudiées (janvier 2017 - novembre 2017 contre tout 2016). Ces deux facteur nous montre une incertitude sur la carte de vérité terrain.

5.0.2 Indices de texture

En plus des indices radiométriques, qui sont utiles pour mieux détecter la végétation, l'eau et les sols, des indices de texture auraient pu être utilisés pour mieux détecter la ville par exemple, qui a été le point faible du clustering. Les indices de Haralik remplissent ce rôle, dont la fonction sous l'OTB est disponible.

Parmi ces indices, on retrouve des formules de moyenne, variance, entropie, corrélation, etc...

L'information texturale des surfaces pourrait donc être exploitée comme un critère supplémentaire de séparabilité.

5.0.3 Séries d'images sur plusieurs années

On a travaillé sur des images dont les dates allaient du janvier 2017 à octobre 2017. Une étude sur plusieurs année pourrait apporter de l'information supplémentaire, d'une part car on aura plus d'éléments de travail, et d'autre part car les séries temporelles des images auraient un effet de périodicité.

5.0.4 Autres méthodes de clustering sous contraintes

Autre que les méthodes COP-KMeans et LCVQE qui sont dérivées des KMeans, dans la même famille, on a aussi testé la méthode "factorisation de matrice non négative pénalisée" qui permet l'utilisation des Must-links et Cannot-links. Cette méthode n'a pas eu les résultats escomptés car elle se base sur du biclustering et non pas sur du clustering.

Les méthodes comprenant du "Metric-Learning" sont à mettre de côté pour traiter des images (de taille raisonnable) car le coût en mémoire est trop élevé.

Dans une étude réalisée par des chercheurs de l'université de Strasbourg, les méthodes de "Spectral Graph Theory", "Declarative approaches" et "Constrained EM" n'ont pas réussi à se démarquer et de la même manière que pour KMeans, les résultats stagnaient.

Seules les méthodes de "Collaborative Clustering" sont sorties du lot en ayant de très bons résultats, mais au prix d'un temps de calcul long. Il reste donc à regarder les méthodes d'"Ensemble Clustering" et dans la catégorie "Miscellaneous", les "Random Forest" et "Evolutionary Algorithm".

Chapitre 6

Conclusion

Le stage était découpé en deux parties, tout d'abord analyser des images multi-temporelles avec du clustering sous contraintes, et ensuite détecter des changements dans des images multi-temporelles.

Dans un premier temps, l'analyse des images multi-temporelles avec du clustering sous contraintes s'est surtout axé sur des méthodes de type KMeans, avec des contraintes Must-link et Cannot-link.

Les résultats obtenus montrent que les apports de contraintes n'améliorent pas forcément la qualité du clustering pour ces méthodes et ces types de données. Seules des suppositions permettent d'expliquer cette stagnation des résultats.

L'utilisation de la mesure "Dynamic Time Warping" (DTW) nous permet à l'aide de profils temporels de référence de faire basculer la balance en notre faveur et d'améliorer les résultats du clustering en raffinant par similarité.

Dans un second temps, la détection de changement quand à elle s'est montrée plus fructueuse, puisque avec une image bien choisie (changement ne prenant pas une grande partie de l'image et terrain homogène, c'est à dire une densité de distribution bien répartie), on a réussi à capter en grande partie la zone du changement.

Le meilleur algorithme d'après nos résultats, Clustering Based Local Outlier Factor (CBLOF), est en continuité avec le stage, puisqu'il est basé sur du clustering. On peut l'utiliser avec n'importe quel algorithme de clustering ce qui le rend polyvalent, la phase de détection d'outliers se faisant ultérieurement.

Le bonus aurait été de détecter les dates des changements, mais c'est un problème plus complexe.

Bilan

Ce stage a été pour moi très épanouissant, il m'a permis de mettre en pratique des connaissances acquises au cours de mon cursus, et de faire preuve d'imagination et d'innovation pour trouver des solutions aux problèmes posés.

Même si j'ai travaillé sur un projet en parallèle de celui de mes collègues de bureau, j'ai pu observer l'engrenage du travail en équipe et ses bénéfices, et en moindre mesure ses inconvénients.

Le fait de travailler au contact de différentes personnes permet aussi de se comparer, d'apprendre et d'évoluer. L'exemple le plus concret est d'avoir côtoyé des virtuoses du C++, qui a mis en lumière mes lacunes dans ce langage de programmation.

Au final, mon ressenti sur ce stage est très positif, que ce soit sur le domaine scientifique ou sur l'entreprise, tout m'a plu.

Annexes

Delay-Based Reservoir Computing :

Reservoir Computing (RC) est un algorithme qui est en quelque sorte une extension des réseaux de neurones. Delay-Based Reservoir Computing (TDRC) est une variante faite pour traiter des signaux temporels.

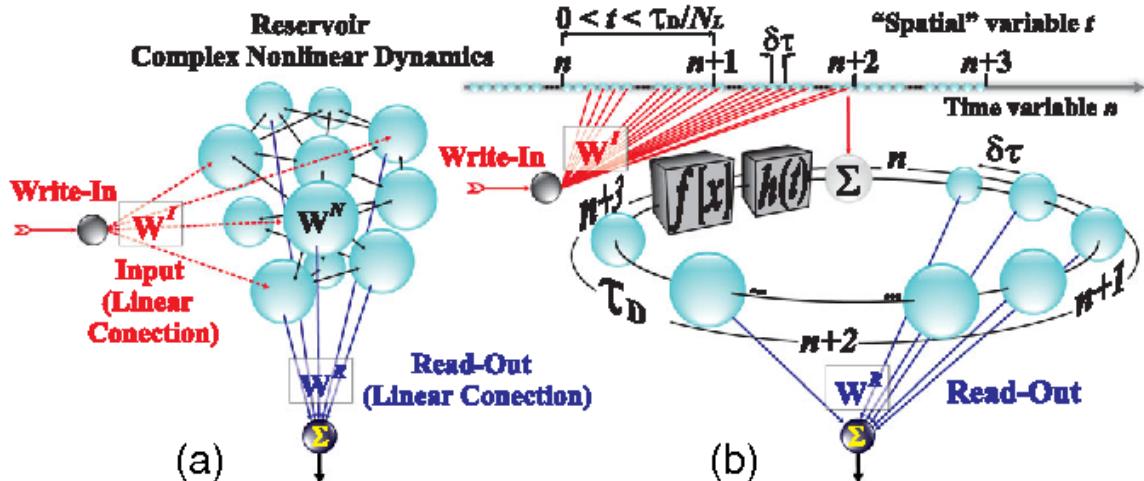


FIGURE 6.1 – Comparaison entre RC et TDRC

Le principe de fonctionnement de TDRC est le suivant :

Soit $z(t) \in \mathbb{R}$ un signal en entrée.

On fixe un réservoir avec un nombre de "neurones virtuels" de taille N .

On multiplexe le signal en appliquant un masque $c \in \mathbb{R}^N$ (le masque doit être le même sur chaque intervalles $[t, t + 1]$, $\forall t \in (0, T - 1)$), et l'on note ce résultat I , donc $I = cz(t)$.

I se décompose en T intervalles de taille N , $I = (I(1), \dots, I(T))$ dont $I(1) = (I_1(1), \dots, I_N(1))$ jusqu'à $I(T) = (I_1(T), \dots, I_N(T))$.

On met en place la boucle qui envoie les $I(1), \dots, I(T)$ dans le réservoir, et on note $x'(t)$ le résultat au temps t du réservoir.

$$x'(t) = -x(t) + f(x(t - \tau), I(t), \theta)$$

Avec f une fonction harmonique non linéaire (noyau non linéaire), θ les paramètres de la fonction f et $\tau > 0$ le délai du réservoir.

Comme pour I , x se décompose en T intervalles de taille N .

En utilisant un schéma d'Euler, on a alors :

$$\frac{x(t) - x(t - d)}{d} = -x(t) + f(x(t - \tau), I(t), \theta)$$

Avec $d = \frac{\tau}{N}$ la distance entre deux neurones successifs.

Une solution du problème est alors sous la forme :

$$x_i(t) = \frac{1}{1+d}x_{i-1}(t) + \frac{d}{d+1}f(x_i(t-1), I_i(t), \theta), \quad \forall i \in 1, \dots, N$$

et

$$x_0(t) = x_N(t-1)$$

Chaque $x(t)$ est envoyé dans une couche de sortie W_{out} (readout layer) réalisant une transformation linéaire. Cette transformation est adaptée à une tâche d'intérêt en utilisant une régression ridge (régression linaire avec pénalité L_1) sur un signal d'apprentissage, pour être moins sensible au bruit.

Plusieurs méthodes pour entraîner la couche de sortie W_{out} sont disponibles, comme utiliser l'état des neurones virtuels à chaque temps et mettre à jour à chaque fois, ou bien entraîner que sur l'état final des neurones virtuels, ce qui est bien moins gourmand en temps de calcul, ou encore regarder les composantes principales à chaque temps.

Notre choix se porte sur l'entraînement de l'état final, qui est plus simple à réaliser, et se déroule comme suit :

Phase d'apprentissage :

On suppose $X^{final} \in \mathbb{R}^{N \times J}$ l'état des neurones virtuels à l'état final, avec J le nombre de signaux à entraîner.

On cherche à trouver $W^{final} \in \mathbb{R}^{K \times N}$ les poids des neurones virtuels tel que $W^{final} X^{final} \simeq y$, $y \in \mathbb{R}^{K \times J}$ est la matrice d'appartenance des classes, avec k le nombre de classe, et dont les valeurs sont 1 à la ligne k et colonne j si le signal numéro j correspondant à la classe k , 0 sinon.

La régression ridge prend la forme :

$$W^{final} = \underset{W \in \mathbb{R}^{K \times N}}{\operatorname{argmin}} \{ \|y - WX^{final}\|_2^2 + \lambda \|W\|_2^2 \}$$

Que l'on peut résoudre explicitement par :

$$W = yX^T(XX^T + \lambda I_N)^{-1}$$

Phase de prédiction :

Il ne reste plus qu'à prédire les classes des données de test en appliquant

$$Z = W^{final} X_{test}^{final}$$

et de chercher l'argmax de chaque colonne

$$y_{pred} = \underset{j \in J}{\operatorname{argmax}} Z(., j)$$

Voici deux schémas permettant d'en comprendre le fonctionnement :

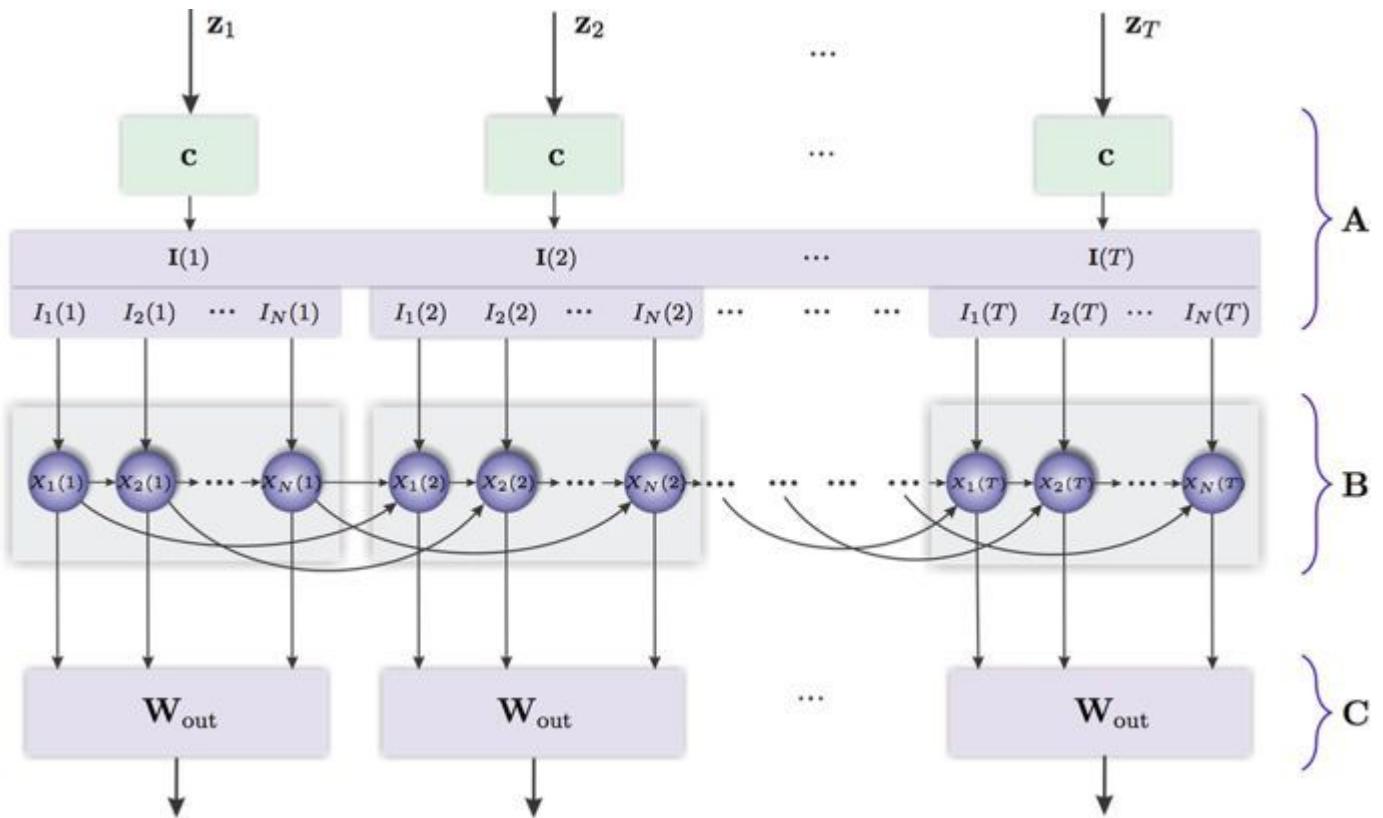


FIGURE 6.2 – Schéma TDRC 1
A) Couche d'entrée, **B)** Time-Delay Reservoir, **C)** Couche readout

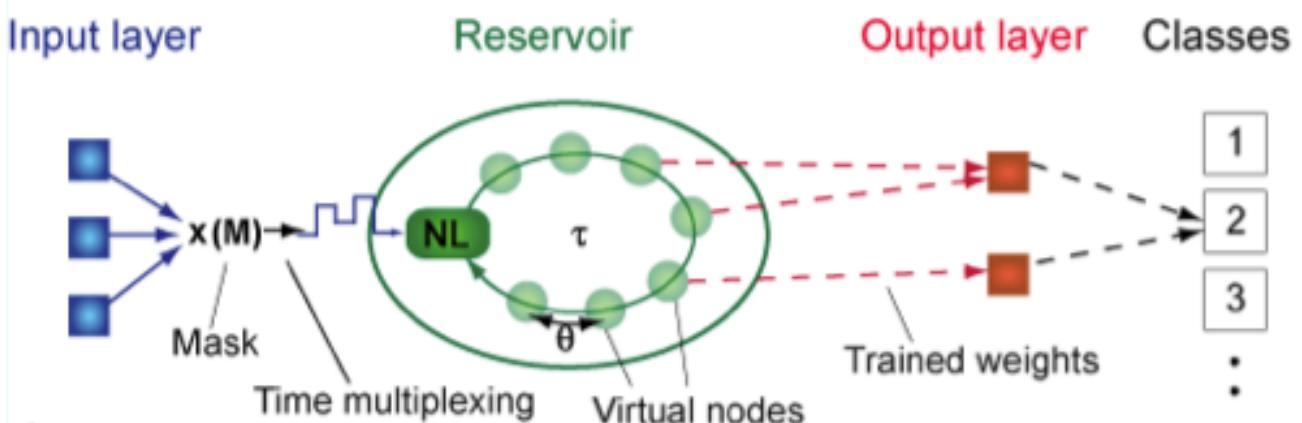


FIGURE 6.3 – Schéma TDRC 2

Bibliographie

- [1] Thomas Lampert & Pierre Gançarski & Baptiste Lafabregue. *Constrained Clustering : Why and How ?*
- [2] Kiri Wagstaff & Claire Cardie & Seth Rogers Stefan Schroedl. *Constrained K-means Clustering with Background Knowledge.*
- [3] Mikhail Bilenko & Sugato Basu & Raymond J. Mooney. *Integrating Constraints and Metric Learning in Semi-Supervised Clustering.*
- [4] Thiago F. Covões & Eduardo R. Hruschka & Joydeep Ghosh. *A Study of K-Means-Based Algorithms for Constrained Clustering.*
- [5] François Petitjean & Jordi Inglada & Pierre Gançarski. *Satellite Image Time Series Analysis under Time Warping.*
- [6] Markus Goldstein & Seiichi Uchida. *A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data.*
- [7] Karanjit Singh & Dr. Shuchita Upadhyaya. *Outlier Detection : Applications And Techniques.*
- [8] Lyudmila Grigoryeva & Julie Henriques & Laurent Larger & Juan-Pablo Ortega. *Optimal nonlinear information processing capacity in delay-based reservoir computers.*
- [9] Ashley A. Prater. *Comparison of echo state network output layer classification methods on noisy data.*