R_GIS 05

Introduction aux traitements d'images avec OTB

Octobre 2022

LIÈGE université
Gembloux
Agro-Bio Tech

TABLE DES MATIERES

1. Préambule

- Le présent document a été développé par l'Axe de Gestion des Ressources forestières de Gembloux Agro-Bio Tech – Université de Liège.
- Ce document a été écrit et vérifié par les auteurs. Cependant, il est possible que des erreurs subsistent et les éventuelles remarques et corrections sont toujours les bienvenues.
- La responsabilité de l'ULiège-GxABT et des auteurs ne peut, en aucune manière, être engagée en cas de litige ou dommage lié à l'utilisation de ce document.

Auteur

- Philippe Lejeune (p.lejeune@uliege.be)

2. Introduction

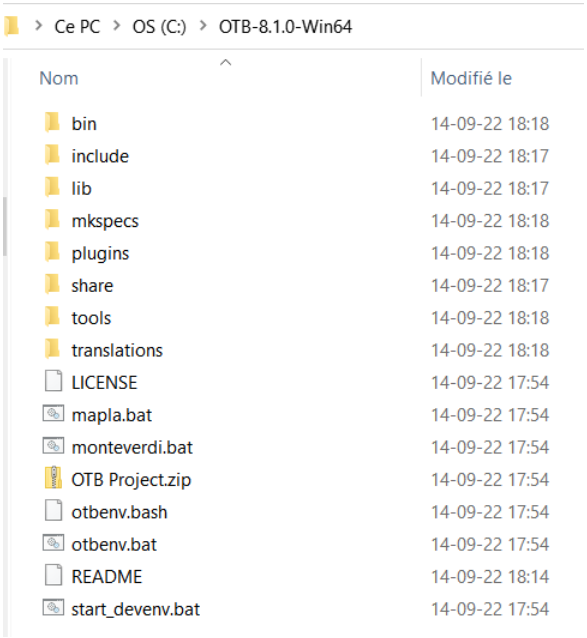
2.1 Objectif

- L'objectif de cet exercice est d'initier à l'utilisation conjointe des librairies GDAL (https://gdal.org/), OTB (https://www.orfeo-toolbox.org/) et de l'environnement R pour réaliser des traitements d'images aériennes ou satellitaires.
- Les traitements qui sont abordés concernent respectivement :
  - le calcul d'indices de végétation ;
  - la réalisation de classifications supervisées par pixels ;
  - la segmentation d'images ;
  - la réalisation de classifications orientées « objet ».
- Certaines parties de ce tutoriel sont largement inspirées des supports de formation proposés sur le site https://www.orfeo-toolbox.org. De même une partie des données utilisées sont également issues de ce site.

2.2 Librairie OTB

- Orfeo ToolBox (OTB) est un projet open-source rassemblant une collection très complète de d'outils pour le traitement et l'analyse de données de télédétection. Ces outils sont appelables depuis différentes plateformes, dont QGIS. Dans cet exercice, les fonctions OTB sont appelées en mode « ligne de commande » dans des scritps R avec la fonction ***system()***.

- OTB doit au préalable être installé. Pour cela, il suffit de télécharger le fichier accessible depuis le lien suivant : https://www.orfeo-toolbox.org/download/
- Le fichier zip téléchargé doit ensuite être décompacté sur un des disques de l'ordinateur. Dans l'exemple ci-dessous, la version 8.1.0 a été installée sur le disque C:.

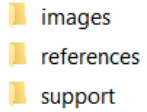


- Par la suite, une variable contenant le chemin vers les exécutables sera utilisée pour exécuter les différentes fonctions d'OTB.

```
# chemins vers les exécutables OTB -----------------
# ces chemins sont utilisés dans les lignes de commande

path_otb = "C:/OTB-8.1.0-Win64/bin/"
```

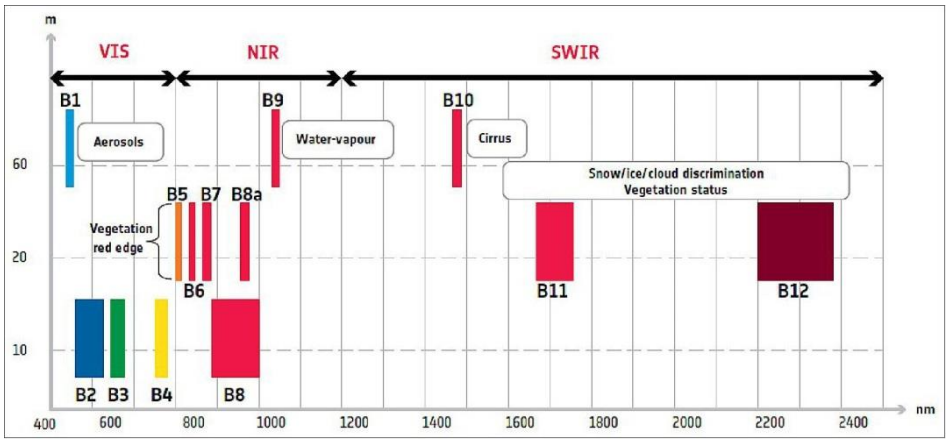
2.3 Présentation des données

- Les données utilisées dans cet exercice sont contenues dans le répertoire \input\classif_pixel. Elles sont issues du site https://www.orfeo-toolbox.org.
- Les données sont organisées en 3 sous-répertoires contenant respectivement les images, les données de référence (entraînement et validation) et différents fichiers de support.



- Les images utilisées sont des extraits d'images Sentinel-2 rééchantillonnées à 20 m de résolution et correspondant à la tuile T31TCJ couvrant la région de Toulouse (France).
- Pour rappel, le tableau et la figure ci-dessous reprennent la liste des bandes spectrales contenues dans les images.

#	Band name	S2 band id	Wavelength	Initial resolution
1	Blue	B2	490 nm	10 m
2	Green	B3	560 nm	10 m
3	Red	B4	665 nm	10 m
4	NIR - Narrow 1	B5	705 nm	20 m
5	NIR - Narrow 2	B6	740 nm	20 m
6	NIR - Narrow 3	B7	783 nm	20 m
7	NIR - Wide	B8	842 nm	10 m
8	NIR - Narrow 4	B8A	865 nm	20 m
9	SWIR 1	B11	1610 nm	20 m
10	SWIR 2	B12	2190 nm	20 m



Source : https://www.eoportal.org

- Les images disponibles correspondent à 3 dates différentes : 2016-06-07, 2016-08-06 et 2016-10-05.
- Les données de références utilisées pour la classification sont constituées de polygones correspondant à différentes classes d'utilisation du sol. Les classes considérées sont reprises dans le tableau ci-dessous.

Code	Nom	#polygones training	#polygones testing
10	Cultures annuelles	3129	3078
31	Forêt feuilles caduques	176	292
32	Forêt feuilles persistantes	23	29
34	Pelouses	2	2
36	Lande ligneuse	63	38
41	Bâti dense	30	33
42	Bâti diffus	326	239
43	Zones industrielles	154	212
44	Routes	162	114
51	Eau	243	332
211	Prairie	320	311
221	Verger	227	254
222	Vigne	129	97

2.4 Liste des images disponibles

- L'automatisation des traitements d'images nécessite de gérer correctement les séries d'images à exploiter.
- Différentes fonctions de R permettent à la fois de gérer les listes de fichiers et d'analyser le contenu des noms de fichier pour en extraire des informations utiles (comme la date d'une image par exemple).
- Dans l'exemple qui suit, les fonctions ***list.files()*** et ***substr()*** sont utilisées respectivement pour dresser la liste des images disponibles et définir les dates correspondantes

```
# Liste des images S2 et des dates correspondantes ----------------

(list_image=list.files(path_s2,pattern="*.tif$",full.names = FALSE))
(list_date=substr(list_image, 1, 8))

> (list_image=list.files(path_s2,pattern="*.tif$",full.names = FALSE))
[1] "20160607_T31TCJ_ROI_20m.tif" "20160806_T31TCJ_ROI_20m.tif"
[3] "20161005_T31TCJ_ROI_20m.tif"
> (list_date=substr(list_image, 1, 8))
[1] "20160607" "20160806" "20161005"
```

3. Calcul d'indices de végétation

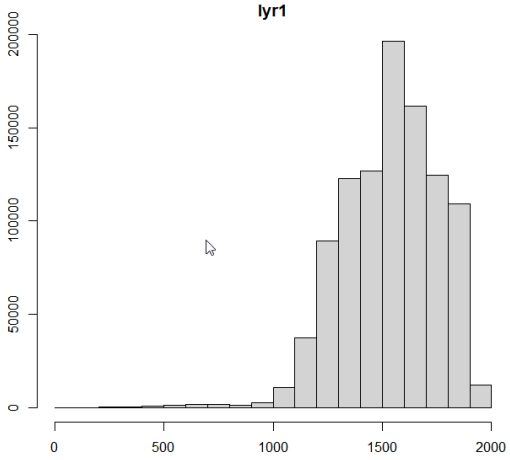
3.1 Calcul de variables dérivées (indices de végétation)

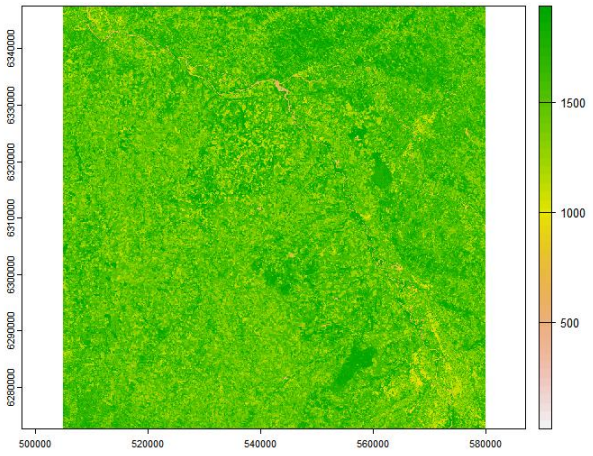
- Le calcul d'indices de végétation s'effectue à l'aide de la fonction ***lapp()***.
- Dans les 2 exemples qui suivent, on calcule le NDVI (Normalized Difference Vegetation Index) et le NDWI (Normalized Difference Water Index).

  NDVI = (NIR - Red) / (NIR + Red)

  NDWI = (Green - NIR) / (Green + NIR)
- Le NDVI est un des indices de végétation les plus utilisés. Les faibles valeurs de NDVI (0,1 et moins) correspondent aux affleurements rocheux, aux sols nus ou zones couvertes de sable ou de neige. L'augmentation de la valeur du NDVI traduit la présence d'une biomasse végétale de plus en plus importante. Le NDWI pour sa part est utilisé pour mettre en évidence le niveau d'humidité des plantes.
- Dans l'exemple qui suit, le NDVI est calculé pour l'image **20160607_T31TCJ_ROI_20m.tif**. La fonction ***ndvi_fun()*** qui est appelée depuis lapp() utilise les bandes B8 (NIR) et B4 (Red) de l'image source pour calculer le NDVI. On remarque que dans la formule utilisée, l'indice subi un rééchelonnement pour faire varier les valeurs de 0 à 2000 (valeurs initiales comprises entre -1 et +1). Cette manière de procéder permet de sauvegarder le résultat en entier 16bit au lieu de valeurs réelles 32bit.

```
# calcul du NDVI : (B8-B4)/(B8+B4)

ndvi_fun = function(B8, B4){
  ((B8 - B4) / (B8 + B4)) * 1000 + 1000
}

s2=rast(list1[1])
names(s2)
f_out=paste0(path_out,"/ndvi.tif")
t1=Sys.time()
ndvi = lapp(c(s2$B8,s2$B4), fun = ndvi_fun,
            filename=f_out,overwrite=T,wopt=opt1)
t2=Sys.time()
(t2-t1)

> (t2-t1)
Time difference of 3.311665 secs
> hist(ndvi)
```





- L'option « wopt » est utilisée pour définir les paramètres de sauvegarde des données dans le fichier de sortie. La variable opt1 contient différents paramètres concernant à la fois le type de données et le niveau de compression du fichier de sortie.

```
> opt1
$gdal
[1] "INTERLEAVE=BAND"       "TILED=YES"             "BIGTIFF=YES"
[4] "COMPRESS=DEFLATE"      "ZLEVEL=9"              "NUM_THREADS=ALL_CPUS"

$datatype
[1] "INT2S"
```

- La fonction ***ndwi_fun()*** est utilisée pour calculer l'indice NDWI pour la même image.

```
# calcul du NDWI : (B3-B8)/(B3+B8)

ndwi_fun = function(B3, B8){
  ((B3 - B8) / (B3 + B8)) * 1000 + 1000
}

f_out=paste0(path_out,"/ndwi.tif")
t1=Sys.time()
ndwi = lapp(c(s2$B3,s2$B8), fun = ndwi_fun,
            filename=f_out,overwrite=T,wopt=opt1)
t2=Sys.time()
(t2-t1)
```

4. Classification supervisée par pixel

4.1 Introduction

- Les étapes d'une classification supervisée par pixel se présentent comme suit :

  1° préparation des images, avec éventuellement le calcul de statistiques descriptives

  2° préparation des données d'entraînement et de validation

  3° entraînement du modèle de classification

  4° application du modèle de classification

  5° post-traitement (application d'un tamisage)

  6° calcul de la matrice de confusion et de la précision

4.2 Exemple 1 : classification *mono-date*

4.2.1 Préparation des images

- La préparation des images consiste à assembler les bandes spectrales utilisées dans 1 seul fichier « multi-bandes ». Dans ce premier exemple, nous utilisons l'image de juin 2016 constituée des 10 bandes issues de l'image Sentinel-2.

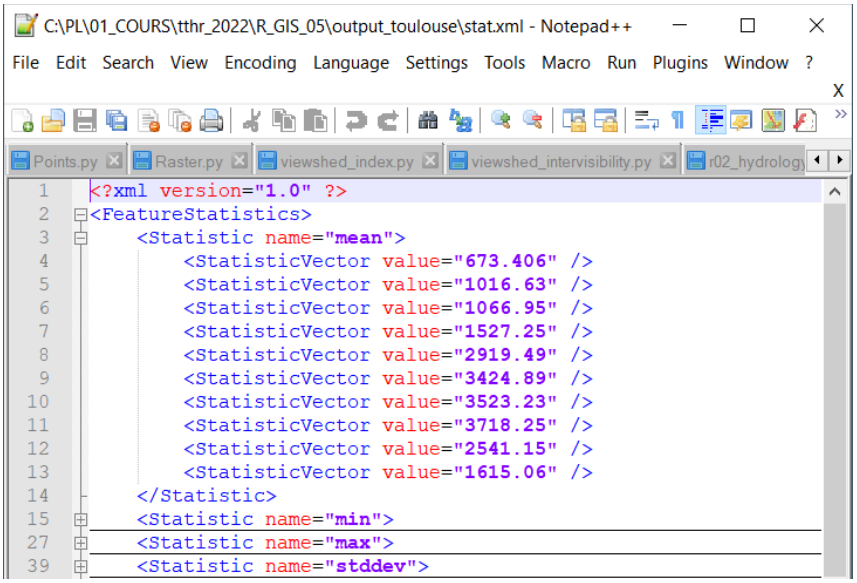
```
# 4.2.1 Préparation des images : calcul des statistiques

f_in = list1[1] # image de 20160607
im=rast(f_in)
names(im)
```

```
> names(im)
 [1] "B2"  "B3"  "B4"  "B5"  "B6"  "B7"  "B8"  "B8A" "B11" "B12"
```

- Lorsque les différentes bandes utilisées par le modèle de classification présentent des propriétés statistiques (moyenne, écart-type) très différentes, il peut s'avérer utile de normaliser les variables lors de la phase d'entrainement du modèle. Il faut pour cela, au préalable, calculer les statistiques de base des images, avec la fonction ***otbcli_ComputeImagesStatistics***.

```
# Calcul des statistiques globales de l'image
f_stat= paste0(path_out,"/stat.xml")
cmdline = paste0(path_otb,"otbcli_ComputeImagesStatistics ",
                 " -il ", f_in,
                 " -out ", f_stat)
system(cmdline)
```

- Les paramètres statistiques (moyenne, min, max et écart-type) des différentes bandes de l'image sont stockés dans un fichier .xml.



4.2.2 Données d'entrainement et de validation

- Les données d'entraînement et de validation se présentent sous la forme de polygones dont un des attributs correspond à la légende considérée dans la classification. Dans cet exemple, c'est le champ [CODE] qui est utilisé. Les codes numériques correspondent au tableau présenté dans le § 2.3.

```
# 4.2.2 Données d'entrainement et de validation

f_train = paste0(path_ref,"/training.shp") # données d'entrainement
f_valid = paste0(path_ref,"/testing.shp") # données de validation
train=st_read(f_train)
valid=st_read(f_valid)

names(train)
table(train$CODE)
```

```
> table(train$CODE)

  10   31   32   34   36   41   42   43   44   51  211  221  222
3129  176   23    2   63   30  326  154  162  243  320  227  129
```

- Les échantillons contenus dans ces 2 fichiers ne sont pas équilibrés. Certaines classes sont surreprésentées, alors que d'autres sont moins fréquentes. Dans le tableau présenté ci-dessous, les surfaces sont converties en nombres de pixels en considérant la résolution des images utilisées (20x20m).

```
# Surface des polygones d'entrainement et de validation
train$surf=as.numeric(st_area(train))
valid$surf=as.numeric(st_area(valid))
tab1= as.data.frame(st_drop_geometry(train)) %>% group_by(CODE) %>%
      summarize(nbpix=sum(surf)/400)
tab1
```

```
> tab1
# A tibble: 13 x 2
    CODE   nbpix
   <db1>   <db1>
 1    10 152024.
 2    31  13394.
 3    32   1206.
 4    34   2277.
 5    36    963.
 6    41   6619.
```

4.2.3 Entrainement du modèle de classification

- Le création du modèle de classification est prise en charge par la fonction ***otbcli_TrainImagesClassifier***.
- Parmi les paramètres à renseigner, on trouve les noms des différents fichiers d'entrée et de sortie, le nom du champ contenant le code de classification, ainsi que l'identification de l'algorithme utilisé pour construire le modèle de classification. Dans cet exercice, nous utiliserons exclusivement l'algorithme Random Forest (***-classifier rf***). Dans les exemples qui sont présentés, nous définissons également 2 paramètres relatifs à l'algorithme rf : le nombre d'arbres de la forêt aléatoire qui est construite (***classifier.rf.nbtrees = 200***) ainsi que la profondeur maximum des arbres (***classifier.rf.max*** = 12).

```
# 4.2.3 Entrainement du modèle ------------------

# Modalité 1 : on n'utilise pas de données  de validation
f_model= paste0(path_out,"/model.xml") # modèle de classification
f_matrix= paste0(path_out,"/matrix.txt") # matrice de confusion
cmdline = paste0(path_otb,"otbcli_TrainImagesClassifier ",
                 " -io.il ", f_in,
                 " -io.vd ", f_train,
                 " -io.out ", f_model,
                 " -io.confmatout ", f_matrix ,
                 " -sample.vfn CODE ",
                 " -classifier rf ",
                 " -classifier.rf.nbtrees 200 ",
                 " -classifier.rf.max 15 ")
system(cmdline)
```

- La fonction ***otbcli_TrainImagesClassifier*** génère une série de résultats qui sont affichés dans la console RStudio.
  - La stratégie d'échantillonnage :

```
2022-10-11 14:34:14 (INFO) TrainImagesClassifier: Sampling rates...
2022-10-11 14:34:14 (INFO) TrainImagesClassifier: Sampling strategy : fit the
 number of samples based on the smallest class
2022-10-11 14:34:14 (INFO) TrainImagesClassifier: Sampling rates for image 1 :
  className  requiredSamples  totalSamples  rate
10       959      150592  0.0063682
211      959      9773    0.0981275
221      959      12638   0.0758823
222      959      4504    0.212922
31       959      13366   0.0717492
32       959      1219    0.78671
34       959      2275    0.421538
36       959      959     1
41       959      5908    0.162322
42       959      51637   0.018572
43       959      38446   0.0249441
44       959      3604    0.266093
51       959      25551   0.0375328
```

- **Remarque importante** : c'est la classe la moins représentée dans l'échantillon qui conditionne le nombre de pixels utilisés pour entraîner le modèle. Dans l'exemple présent, c'est la classe 36, qui ne comporte que 959 pixels, qui fixe le nombre de pixels sélectionnés dans chaque classe.

- La matrice de confusion (lignes : références, colonne : estimations)

```
2022-10-12 05:15:41 (INFO) TrainImagesClassifier: Confusion matrix (rows = reference
 labels, columns = produced labels):
      [10] [31] [32] [34] [36] [41] [42] [43] [44] [51] [211] [221] [222]
[ 10]   368    1    5   27    5    4   15    7    1    0    10   14   21
[ 31]     0  433   10    6   12    0    0    0    0    0     9    8    0
[ 32]     1   11  418    8   17    0    0    0    0    0    16    4    3
[ 34]    24   17   14  255   40    0   14    4    0    0    21   16   73
[ 36]     8    8   14   42  324    0    5    0    0    0    40    4   33
[ 41]     1    0    3    1    0  387   50   16   18    0     0    0    2
[ 42]    16    0    1   16    2   38  291   29   27    0     2    6   50
[ 43]     8    0    1   12    0   39   84  199  116    1     1    4   13
[ 44]     0    0    1    1    0    6   19   71  372    0     0    6    2
[ 51]     0    0    2    0    0    0    0    0    2  474     0    0    0
[211]    14   10    9   59   59    1    4    0    0    0   257    9   56
[221]    13   13    7   23   16    0    5    2    6    0    50  318   25
[222]    23    0    1   27    8    1   18    0    0    0    16    2  382
```

- Les niveaux de précision pour les différentes classes, ainsi que la performance globale.

```
2022-10-12 05:15:42 (INFO) TrainImagesClassifier: Precision of class [10] vs all: 0.773109
2022-10-12 05:15:42 (INFO) TrainImagesClassifier: Recall of class    [10] vs all: 0.769874
2022-10-12 05:15:42 (INFO) TrainImagesClassifier: F-score of class   [10] vs all: 0.771488

2022-10-12 05:15:42 (INFO) TrainImagesClassifier: Precision of class [31] vs all: 0.878296
2022-10-12 05:15:42 (INFO) TrainImagesClassifier: Recall of class    [31] vs all: 0.905858
2022-10-12 05:15:42 (INFO) TrainImagesClassifier: F-score of class   [31] vs all: 0.891864

2022-10-12 05:15:42 (INFO) TrainImagesClassifier: Precision of class [32] vs all: 0.860082
2022-10-12 05:15:42 (INFO) TrainImagesClassifier: Recall of class    [32] vs all: 0.874477
2022-10-12 05:15:42 (INFO) TrainImagesClassifier: F-score of class   [32] vs all: 0.86722

2022-10-12 05:15:42 (INFO) TrainImagesClassifier: Precision of class [34] vs all: 0.534591
2022-10-12 05:15:42 (INFO) TrainImagesClassifier: Recall of class    [34] vs all: 0.533473
2022-10-12 05:15:42 (INFO) TrainImagesClassifier: F-score of class   [34] vs all: 0.534031

2022-10-12 05:15:42 (INFO) TrainImagesClassifier: Global performance, Kappa index: 0.6973
```

La signification des termes employés est la suivante :

  - Precision of class : correspond à la précision de l'utilisateur
  - Recall : correspond à la précision du producteur
  - F-score : 2 * (Precision * Recall) / (Precision + Recall)
- La précision globale exprimée sous la forme de la variable kappa.
- Les niveaux de précision pour les différentes classes.

- En l'absence de données spécifiquement dédiées à la validation, OTB a divisé le jeu de données d'entrainement en 2 parties égales : la première a servi à entrainer le modèle et la seconde à l'évaluer.
- Les niveaux d'erreur présentés dans cette matrice sont très optimistes car il n'y a pas de réelle indépendance entre les données d'entrainement et de validation qui sont prélevés dans les mêmes polygones.
- Dans la seconde modalité, un jeu de données indépendant est utilisé pour valider le modèle et calculer la matrice de confusion. Il apparaît dans les paramètres de la commande avec l'option « –io.valid ».

```
# Modalité 2 : utilisation de données de validation indépendantes
cmdline = paste0(path_otb,"otbcli_TrainImagesClassifier ",
                 " -io.il ", f_in,
                 " -io.vd ", f_train,
                 " -io.valid ", f_valid,
                 " -io.out ", f_model,
                 " -io.confmatout ", f_matrix ,
                 " -sample.vfn CODE ",
                 " -classifier rf ",
                 " -classifier.rf.nbtrees 200 ",
                 " -classifier.rf.max 15 ")
system(cmdline)

2022-10-12 05:19:56 (INFO) TrainImagesClassifier: Global performance, Kappa index: 0.603445
```

- L'index kappa est passé de 0,70 à 0,603 !
- Dans l'exemple qui suit, on utilise les statistiques globales calculées sur les différentes bandes de l'image pour standardiser celles-ci. Cette modalité est susceptible d'améliorer les performances du modèle lorsque les bandes de l'image présentent des niveaux de variabilité très différents les uns des autres. On remarque que dans le cas présent, la précision ne s'améliore pas, puisque l'indice kappa passe de 0,603 à 0,602.

```
# Modalité 3 : utilisation de données de validation indépendantes
#              utilisation de données normalisées

cmdline = paste0(path_otb,"otbcli_TrainImagesClassifier ",
                 " -io.il ", f_in,
                 " -io.imstat ", f_stat,
                 " -io.vd ", f_train,
                 " -io.valid ", f_valid,
                 " -io.out ", f_model,
                 " -io.confmatout ", f_matrix ,
                 " -sample.vfn CODE ",
                 " -classifier rf ",
                 " -classifier.rf.nbtrees 200 ",
                 " -classifier.rf.max 15 ")
t1=Sys.time()
system(cmdline)
t2=Sys.time()
(t2-t1)
2022-10-12 05:21:50 (INFO) TrainImagesClassifier: Global performance, Kappa index: 0.602483
> t2=Sys.time()
> (t2-t1)
Time difference of 38.05708 secs
```

- La fonction ***Sys.time()*** est utilisée pour calculer le temps nécessaire à l'exécution de la commande. Il est de 38 secondes dans le cas présent.
- L'augmentation de la mémoire allouée au traitement réalisé par OTB permet de raccourcir le temps de traitement. Cette allocation de mémoire est réalisée avec l'option « -ram ». Dans l'exemple qui suit, une mémoire de 4000 Mo est allouée à OTB.

```
# Augmenter l'allocation de mémoire vive
# -> réduction du temps de calcul

cmdline = paste0(path_otb,"otbcli_TrainImagesClassifier ",
                 " -io.il ", f_in,
                 " -io.imstat ", f_stat,
                 " -io.vd ", f_train,
                 " -io.valid ", f_valid,
                 " -io.out ", f_model,
                 " -io.confmatout ", f_matrix ,
                 " -sample.vfn CODE ",
                 " -classifier rf ",
                 " -classifier.rf.nbtrees 200 ",
                 " -classifier.rf.max 15 ",
                 " -ram 4000 ")
t1=Sys.time()
system(cmdline)
t2=Sys.time()
(t2-t1)
> t2=Sys.time()
> (t2-t1)
Time difference of 28.64844 secs
```

- Le temps de calcul est passé de 38 à 28,6 secondes.

4.2.4 Comparaison de plusieurs modèles de classification

- L'exemple qui suit exploite pleinement les possibilités liées à l'utilisation de script pour automatiser les traitements de données. Un même modèle de classification va être créé pour chacune des 3 images disponibles. Le critère de précision de chaque modèle sera conservé dans 1 tableau pour comparer les performances de ces 3 modèles. Cette opération est réalisée de manière relativement simple en intégrant la fonction ***otbcli_TrainImagesClassifier*** dans une boucle.

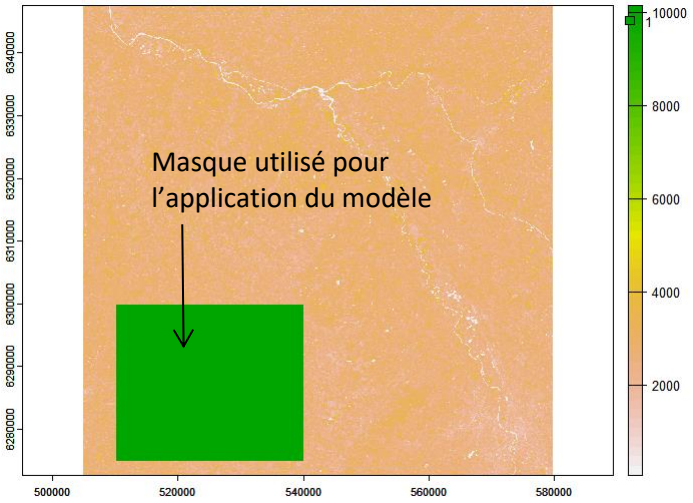
```
# 4.2.4 Comparer les précisions des classifications monodates

result=NULL # df pour stocker la date et le kappa

for (i in 1:length(list1)){

  date=list_date[i]
  print(date)
  f_in = list1[i]
  shp_train = paste0(path_ref,"/training.shp")
  shp_valid = paste0(path_ref,"/testing.shp")
  f_model= paste0(path_out,"/model_",date,".xml")
  f_matrix= paste0(path_out,"/matrix_",date,".txt")
  f_stat= paste0(path_out,"/stat_",date,".xml")

  # Entraîner le modèle
  cmdline = paste0(path_otb,"otbcli_TrainImagesClassifier ",
                   " -io.il ", f_in,
                   " -io.vd ", f_train,
                   " -io.valid ", f_valid,
                   " -io.out ", f_model,
                   " -io.confmatout ", f_matrix ,
                   " -sample.vfn CODE ",
                   " -classifier rf ",
                   " -classifier.rf.nbtrees 200 ",
                   " -classifier.rf.max 15 ",
                   " -ram 4000 ")
  # Exécuter la ligne de commande avec l'option intern=TRUE
  # pour récupérer les messages envoyés par OTB dans 1 variable "texte"
  txt=system(cmdline,intern=TRUE)

  # on extrait de la variable txt la ligne avec le Kappa
  txt1=txt[grepl("Kappa", txt)]
  # transformer la ligne de texte en vecteur
  txt2=unlist((strsplit(txt1," ")))
  kappa=txt2[length(txt2)]
  print(c(date,kappa))
  k=nrow(result)+1
  df=data.frame(date=date,kappa=kappa)
  result=rbind(result,df)

}
print(result)
```

```
> print(result)
      date    kappa
1 20160607 0.603008
2 20160806 0.591378
3 20161005 0.622071
```

- L'image qui permet la classification la plus précise est celle d'octobre 2016.

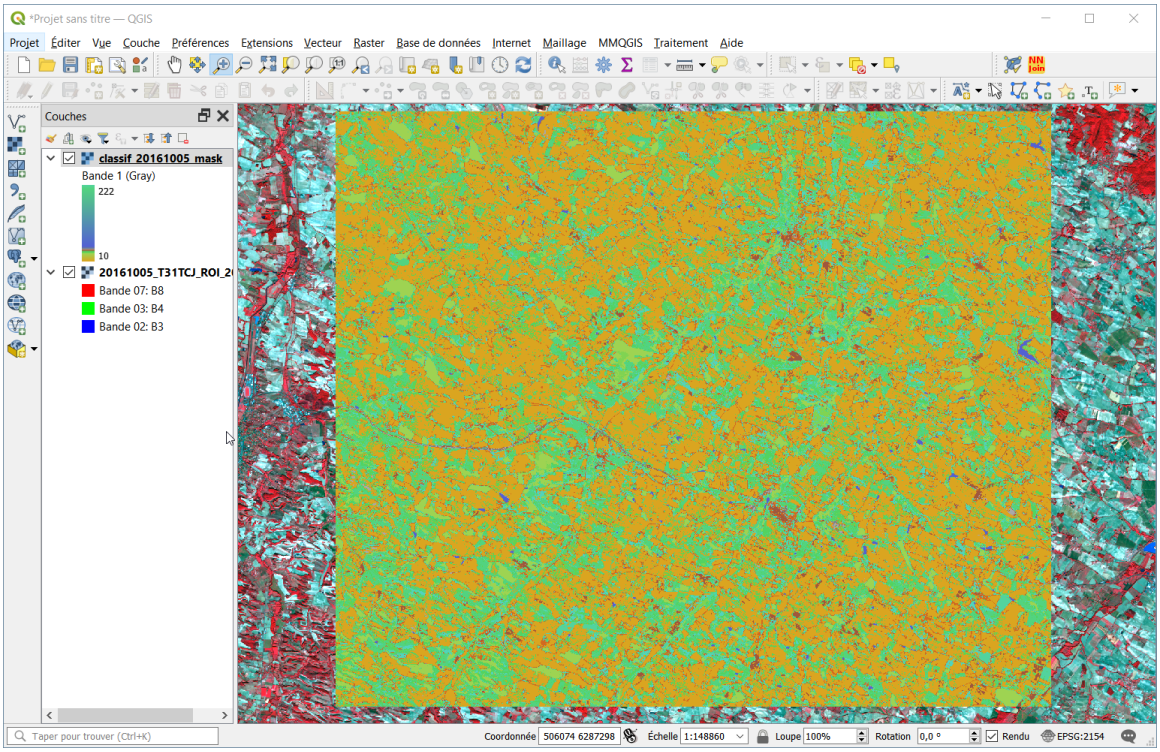
4.2.5 Appliquer un modèle de classification à une image

- L'étape suivante consiste à appliquer le modèle produit à l'image afin d'obtenir une classification. Cette étape est réalisée avec la fonction ***otbcli_ImageClassifier***.
- Cette fonction produit 2 fichiers de sortie :
  - un raster contenant la classification (« -out ») ;
  - un raster contenant le niveau de confiance de la classification (« -confmap »). Dans le cas d'une forêt aléatoire, ce niveau de confiance correspond à la proportion des votes en faveur de la classe retenue.
- Dans le cas présent, le résultat de la classification est codée en entier 8bit (« uint8 »).
- L'exécution de cette fonction peut prendre un temps important en fonction de la taille de la zone d'intérêt, et des paramètres du modèle, notamment le nombre d'arbres de la forêt aléatoire. Dans le cas présent, la durée du traitement est de l'ordre de 20 minutes.
- Pour les besoins de l'exercice, le modèle sera appliqué en ajoutant l'option « -mask » qui permet de n'appliquer le modèle que sur une partie de la zone d'étude. Le masque utilisé correspond à une zone rectangulaire située dans la partie inférieure gauche de la zone d'étude.

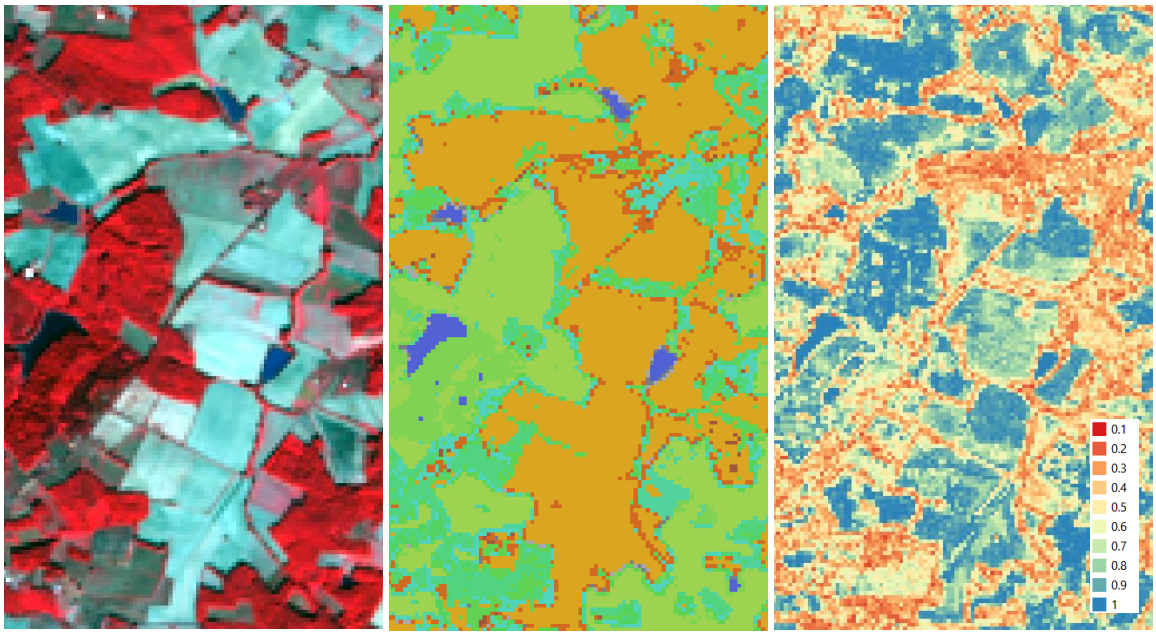


```
# 4.2.5 Appliquer le modèle le + précis (20161005) ------------

kdate=3
date=list_date[kdate]
f_in = list1[kdate]
f_model= paste0(path_out,"/model_",date,".xml")
f_stat= paste0(path_out,"/stat_",date,".xml")
f_out= paste0(path_out,"/classif_", date, "_mask.tif")
f_conf= paste0(path_out,"/conf_model_",date,"_mask.tif")
f_mask=paste0(path_in,"/mask.tif")

cmdline = paste0(path_otb,"otbcli_ImageClassifier ",
           " -in ", f_in,
           " -out ", f_out,
           " uint8 ",
           " -confmap ", f_conf,
           " -mask ",f_mask,
           " -ram 8000 ",
           " -model ", f_model)

t1=Sys.time()
system(cmdline)
t2=Sys.time()
(t2-t1)

> (t2-t1)
Time difference of 4.724999 mins
```

- Les fichiers de sortie peuvent ensuite être visualisés dans QGIS. Le fichier **classif.qml** contenu dans le répertoire **\classif_pixel\support** peut être utilisé pour colorier l'image **classif_20161005.tif**.



- La figure suivante présente respectivement un extrait de l'image originale (composition colorée infra-rouge fausses couleurs), le résultat de la classification et la carte de niveau de confiance de la classification (probabilité comprise entre 0 et 1).



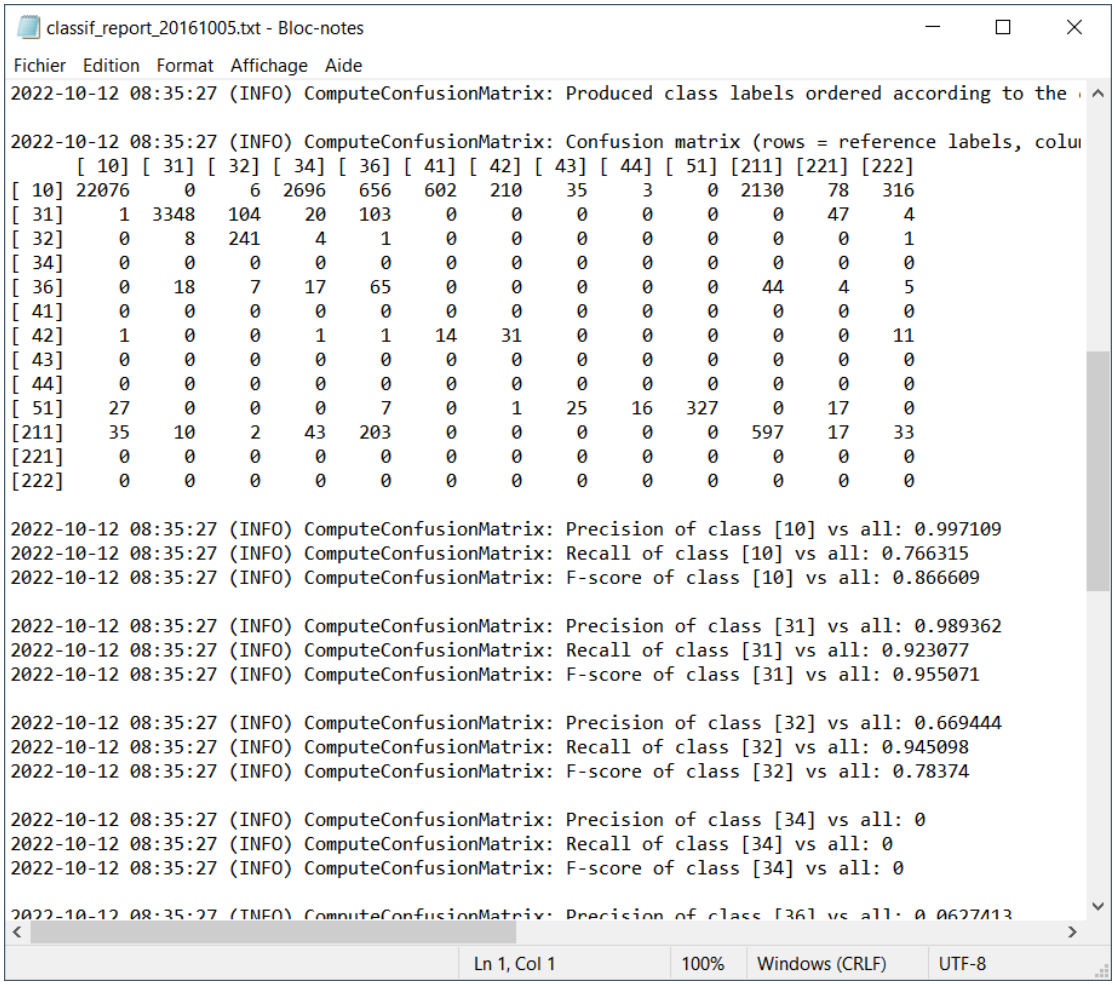
- On remarque que les niveaux de confiance les plus faibles correspondent aux pixels situés à cheval sur plusieurs occupations du sol, et notamment sur les alignements d'arbres en bordures de parcelles agricoles.

4.2.6 Calcul de la matrice de confusion

- La matrice de confusion peut être calculée avec la fonction ***otbcli_ComputeConfusionMatrix***.

```
# 4.2.6. calcul de la matrice de confusion ------------------

f_in= paste0(path_out,"/classif_", date, "_mask.tif")
f_out= paste0(path_out,"/conf_matrix_", date, ".txt")
f_report= paste0(path_out,"/classif_report_", date, ".txt")

cmdline = paste0(path_otb,"otbcli_ComputeConfusionMatrix ",
           " -in " , f_in,
           " -ref ", "vector ",
           " -ref.vector.in ", f_valid,
           " -ref.vector.field ", "CODE",
           " -out ", f_out)
report=system(cmdline,intern=TRUE)
report

# écrire le rapport dans un fichier txt
write(report, f_report, sep=" ")


[110] "2022-10-12 08:35:27 (INFO) ComputeConfusionMatrix: Kappa index: 0.501415"

[111] "2022-10-12 08:35:27 (INFO) ComputeConfusionMatrix: Overall accuracy index: 0.778692"
```

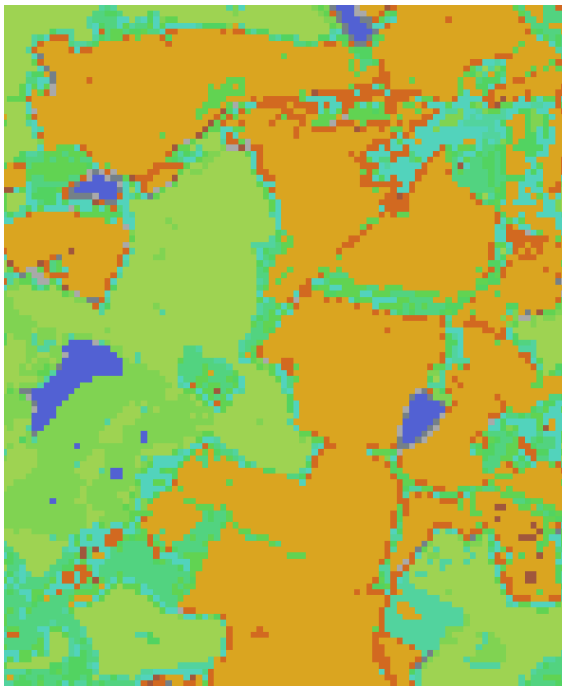
- La figure suivante présente un extrait du fichier classif_report_20161005.txt contenant le rapport généré lors de l'exécution de la fonction ***otbcli_ComputeConfusionMatrix.***

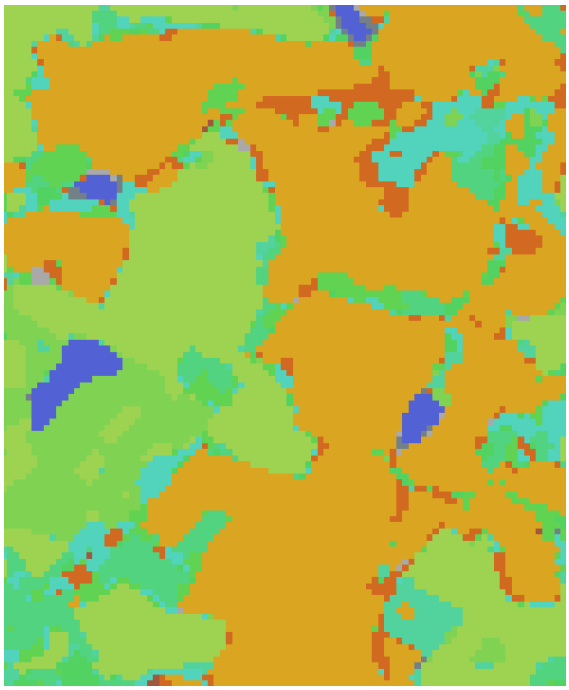


4.2.7 Application d'un filtre majorité à la classification (Régularisation)

- Le résultat de la classification est très bruité : de nombreux pixels appartenant à une classe sont isolés au milieu de pixels d'une autre classe. Ce phénomène est qualifié d'effet « poivre et sel ».
- La fonction ***otbcli_ClassificationMapRegularization*** applique un filtre majorité au résultat de la classification. Cette étape permet de diminuer l'effet poivre et sel.

```
# 4.2.7 Régulariser le résultat de la classification (filtre majorité) ----
f_in= paste0(path_out,"/classif_", date, "_mask.tif")
f_out= paste0(path_out,"/classif_", date, "_reg.tif")

cmdline = paste0(path_otb,"otbcli_ClassificationMapRegularization ",
                 " -ip.radius 1 ",
                 " -io.in ", f_in,
                 " -io.out ", f_out,
                 " uint8 ")
system(cmdline)
```





Fichier classif_20161005.tif Fichier classif_20161005_reg.tif

- A l'issue de cette étape, l'indice kappa est passé de 0,605 à 0,646.

```
# matrice de confusion de l'image régularisée

file_in= paste0(path_output,"/classif_", date, "_reg.tif")
file_out= paste0(path_output,"/conf_matrix_", date, "_reg.txt")
file_report= paste0(path_output,"/classif_report_", date, "_reg.txt")

cmdline = paste0(path_otb,"otbcli_ComputeConfusionMatrix ",
                 " -in " , file_in,
                 " -ref ", "vector ",
                 " -ref.vector.in ", shp_valid,
                 " -ref.vector.field ", "CODE",
                 " -out ", file_out)
report=system(cmdline,intern=TRUE)
report
```

```
[110] "2022-10-12 08:44:59 (INFO) ComputeConfusionMatrix: Kappa index: 0.551025"

[111] "2022-10-12 08:44:59 (INFO) ComputeConfusionMatrix: Overall accuracy index: 0.812192"
```

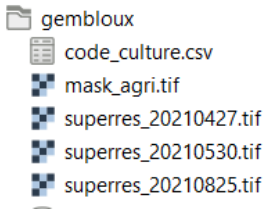
5. Segmentation d'images

5.1 Introduction

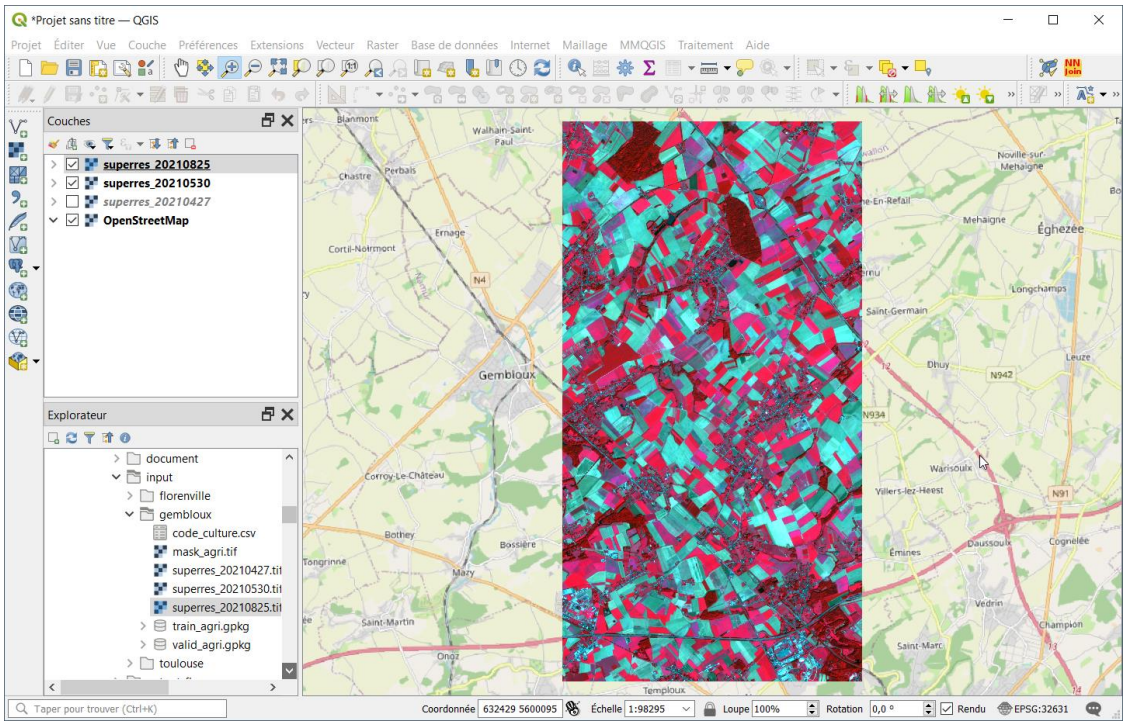
- Le principe de la segmentation d'images est de rassembler des pixels en groupes homogènes appelés segments. Les critères de regroupement des pixels varient en fonction des méthodes utilisées.
- Cette technique est utilisée en amont de méthodes de classification qui vont fonctionner sur les segments ou objets ainsi créés. Ces techniques de classification sont qualifiées de classification orientées « objet » ou OBIA pour « Object Based Image Analysis ».
- Ces approches OBIA sont surtout utilisées pour des images à haute résolution (pixels < 10 m), voire très haute résolution (pixels < 1 m).

5.2 Présentation des données

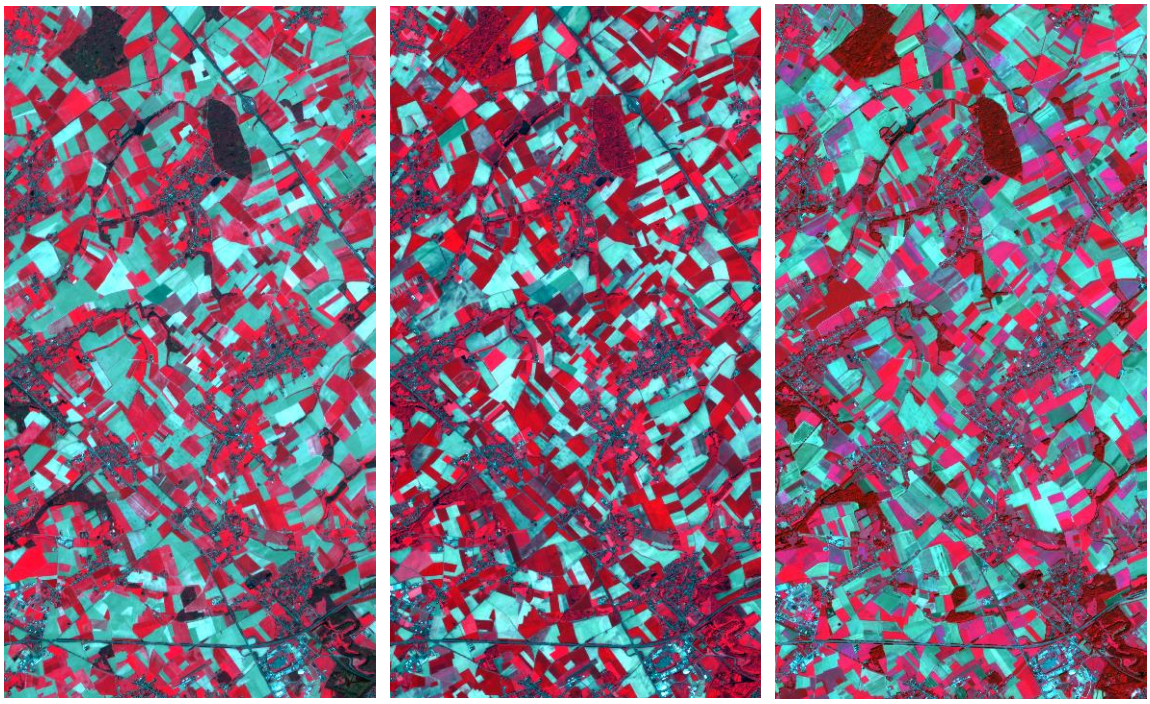
- Afficher les images **superres_20210427.tif**, **superres_20210427.tif** et **superres_20210427.tif** contenues dans le répertoire /input/gembloux**.**



- Ces 3 images sont issues d'une fusion d'images Sentinel-2 et Planetscope avec une approche par apprentissage profond (https://www.mdpi.com/2072-4292/12/15/2366). Les produits ainsi fusionnés présentent une résolution spatiale de 2,5 m et rassemblent 10 bandes spectrales correspondant aux bandes « 10 m » et « 20 m » de Sentinel-2.
- Les images utilisées dans cet exercice contiennent 7 des 10 bandes des images originales : B2, B3, B4, B8, B5, B11 et B12.
- Les images proposées couvre une zone de 8 x 15 km située à l'est de la ville de Gembloux.

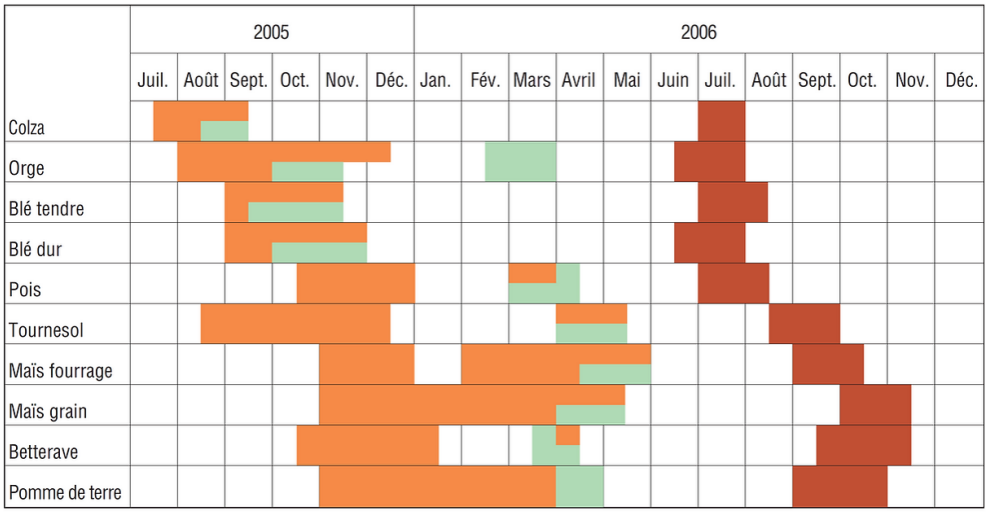


- Les noms des fichiers contiennent les dates d'acquisition des 3 images : 27 avril 2021, 30 mai 2021 et 25 août 2021.



27 avril 2021 30 mai 2021 25 août 2021

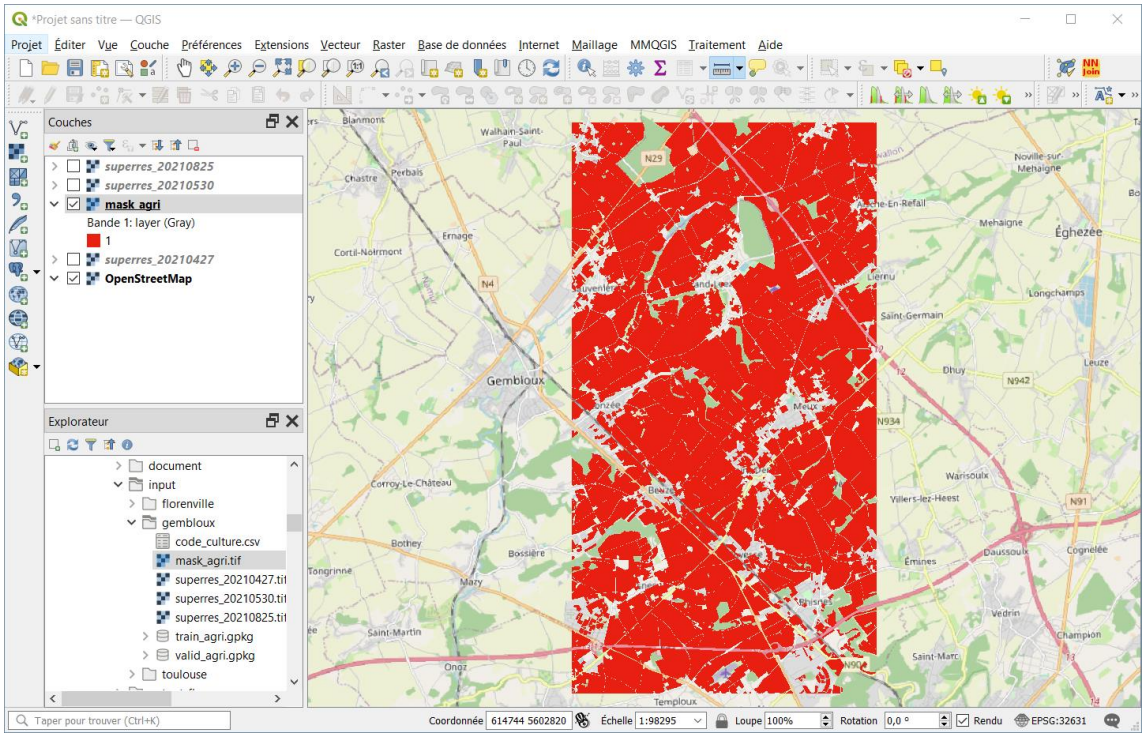
- Ces séries d'images multi dates sont particulièrement intéressantes pour discriminer les grandes cultures qui présentent des calendriers culturaux différents comme le montre l'exemple de la figure suivante.



Source : Agreste - Enquête Pratiques culturales 2006

Source : https://www.paprec-agro.com

- Dans la suite de l'exercice, nous allons procéder à une segmentation des ces images, avant de procéder à une classification des segments produits pour prédire les cultures présentes dans ces segments.
- Cette classification des types de cultures sera réalisée au sein d'un masque délimitant les zones agricoles. Le masque est contenu dans l'image **mask_agri.tif**.



5.3 Préparation des données

- Les 3 images totalisent 21 bandes. Pour éviter que le processus de segmentation des images ne prennent trop de temps de calcul, il est recommandé de réduire le volume de données utilisées.
- On peut opérer cette réduction en calculant par exemple un indice de végétation (comme le NDVI) pour chaque date en agrégeant les indices des 3 dates.
- Une autre solution consiste à procéder à une Analyse en Composante Principale (ACP) sur l'ensemble du jeu de données et conserver un nombre restreint de composantes.
- Avant de réaliser cette ACP, il convient d'empiler les bandes spectrales des 3 images dans 1 seul fichier. Cet empilement est construit à l'aide d'un raster virtuel rassemblant 3 x 7 bandes.

```
# 5.3. Préparation des données ------------------

# Générer la liste d'images
list1=list.files(path_in,pattern=".tif$",full.names = T)
list1
# Conserver uniquement les 3 images dans la liste
list1=list1[grepl("superres",list1)]
list1

# Empiler les bandes des 3 images dans 1 vrt
list_band=c("B2","B3","B4","B8","B5","B11","B12")
list2=list()
for(i in 1:3){
  print(i)
  im=rast(list1[i])
  for(j in 1:7){
    f_vrt=paste0(path_out,"/im_",i,"_",list_band[j],".vrt")
    gdalbuildvrt(gdalfile=list1[i], b=j,
                 output.vrt= f_vrt,overwrite=TRUE)
    list2=rbind(list2,f_vrt)
  }
}

# Créer le vrt
f_vrt=paste0(path_out,"/stack_3dates.vrt")
gdalbuildvrt(gdalfile=list2, separate=TRUE,
             output.vrt= f_vrt,overwrite=TRUE)
```

- L'ACP est réalisée sur ce raster virtuel à l'aide de la fonction **otbcli_DimensionalityReduction**. Le nombre de composantes conservées est fixé à 6.

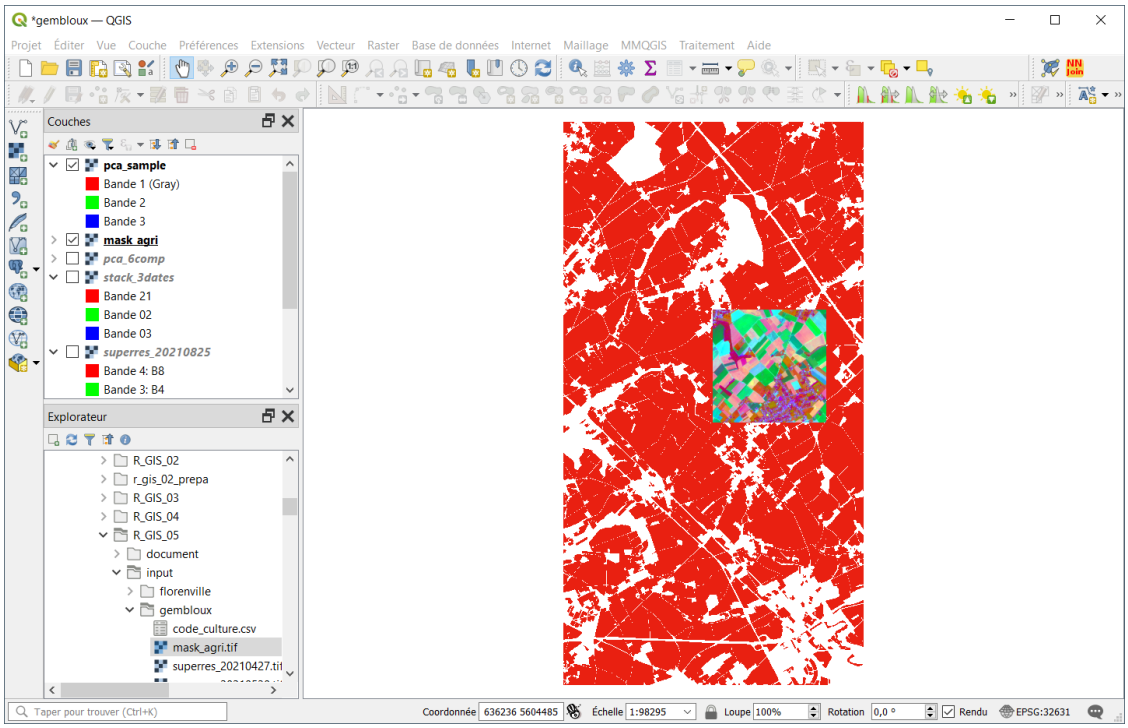
```
# ACP sur l'ensemble des bandes du vrt
f_pca=paste0(path_out,"/pca6.tif")
cmdline = paste0(path_otb, "otbcli_DimensionalityReduction ",
                 ' -in ' , f_vrt,
                 ' -out ', f_pca,
                 ' -method pca',
                 ' -ram 4000 ',
                 ' -nbcomp 6')

t1=Sys.time()
system(cmdline)
t2=Sys.time()
(t2-t1)
```

```
> t2=Sys.time()
> (t2-t1)
Time difference of 1.385118 mins
```

- Les temps de traitement liés à l'étape de segmentation peuvent être longs, voire très longs. Pour mettre au point les paramètres de la segmentation, il est recommandé de travailler sur une zone test de petite taille, représentative de la zone d'étude, ce qui permet de réduire les temps de calcul.

- Les lignes de code qui suivent génèrent un fichier vrt qui reprend les 21 couches du vrt initial mais qui ne couvre qu'une petite partie de la scène étudiée dans cet exercice (3 km x 3 km).

```
# Créer un vrt sur une petite partie de la zone d'étude
bbox ="625500 5601000 628500 5604000" # xul, yul, xlr, ylr
f_out = paste0(path_out,"/pca_sample.vrt")
gdalbuildvrt(gdalfile=f_pca,te=bbox,
             output.vrt= f_out,overwrite=TRUE)
```



5.4 Segmentation avec l'algorithme meanshift

5.4.1 Introduction

- La segmentation proprement dite est prise en charge par la commande ***otbcli_Segmentation***. L'algorithme utilisé est « **mea**n**shift »**.
- **Il s'agit d'une méthode non-paramétrique qui réalise, dans un premier temps, un lissage de** l'image en combinant un critère de distance (ou spatial radius correspondant au paramètre -filter.meanshift.spatialr) et un critère de similarité spectrale (ou range radius défini par le paramètre -filter.meanshift.ranger). Pour simplifier, chaque pixel reçoit la valeur moyenne d'un groupe de pixels homogènes (défini par le range radius) présent dans son voisinage (défini par le spatial radius). La recherche de cette valeur s'opère de manière itérative, le centre de la fenêtre étant repositionné au centre du groupe de pixels identifié à l'itération précédente. La création des segments s'opère ensuite au départ de l'image lissée en regroupant les pixels dont les valeurs spectrales sont proches les unes des autres (différence inférieure au range radius).

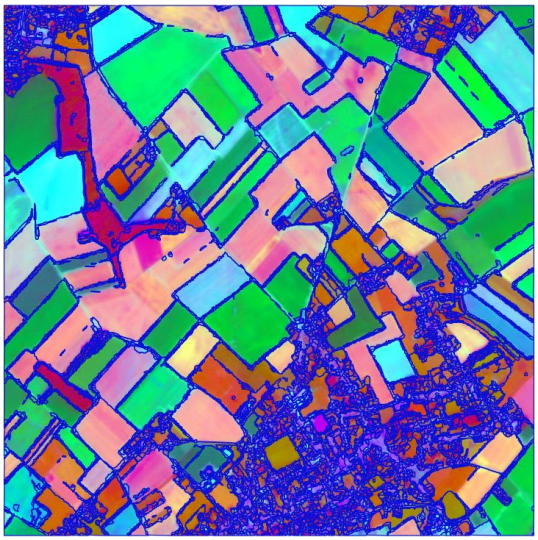
5.4.2 Choix des paramètres de segmentation

- L'impact des 2 paramètres sur le résultat de la segmentation peut être apprécié au travers d'une analyse de sensibilité en faisant varier leurs valeurs et en observant le résultat.
- La documentation OTB relative à cette commande renseigne que les valeurs par défaut de ces 2 paramètres sont respectivement de 5 pour le *spatial radius* et 15 pour le *range radius*.
- **Remarque importante** : le range radius doit être fixé en lien avec les gammes de valeurs dans les images. Ainsi la valeur de 15 par défaut est prévue pour des images codées en 8bit (0-255). Dans le cas présent, les bandes relatives aux composantes générées par l'ACP présentent des gammes de valeurs allant de 5 pour la composante 1 à 16 pour la composante 6. Le paramètre *range radius* doit donc être adapté à cette gamme de valeurs.
- Dans le script présenté ci-dessous, la commande ***otbcli_Segmentation*** est insérée dans une double boucle qui teste plusieurs combinaisons des 2 paramètres. Le résultat de chaque traitement est sauvegardé dans un fichier séparé. Les valeurs de 2 paramètres, ainsi que le temps de calcul et le nombre de segments générés sont sauvegardés dans un dataframe pour comparaison.

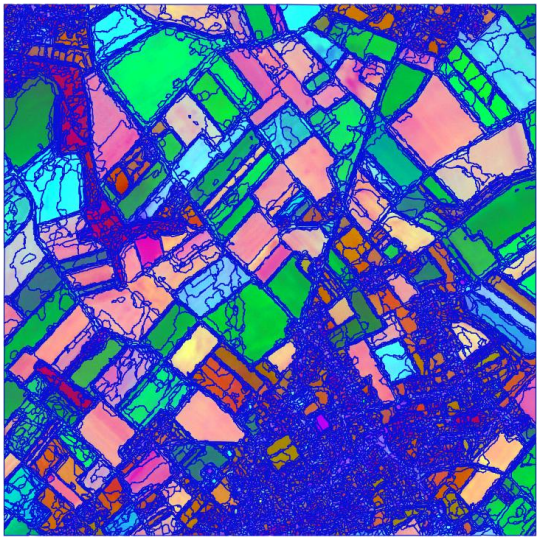
```
# Boucle pour tester différentes valeurs des paramètres de segmentation

result=NULL
for (k_spatial in c(5,10,15)){
  print(k_spatial)
  for (k_range in c(0.3,0.4,0.5)){
    txt_range=k_range*100
    print(k_range)
    f_out = paste0(path_out,"/segm_",k_spatial,
                   "_",txt_range,"_pca6.sqlite")

    # avant d'exécuter, supprimer le fichier de sortie s'il existe
    if(file.exists(f_out)){
      file.remove(f_out)
    }
    cmdline = paste0(path_otb, "otbcli_Segmentation ",
                     ' -in ' , f_in,
                     ' -filter "meanshift" ' ,
                     ' -filter.meanshift.spatialr ',k_spatial,' ' ,
                     ' -filter.meanshift.ranger ',k_range,' ' ,
                     ' -filter.meanshift.maxiter 10' ,
                     ' -filter.meanshift.minsize 20' ,
                     ' -mode.vector.out ',  f_out,
                     ' -mode.vector.outmode "ovw" ' ,
                     ' -mode.vector.tilesize 4000 ',
                     ' uint32',
                     ' -mode.vector.simplify 0 ')
    t1=Sys.time()
    system(cmdline)
    t2=Sys.time()
    (t2-t1)

    sf=st_read(f_out)
    df=data.frame(spatial = k_spatial, range= k_range,
                  dt=t2-t1, nbpol=nrow(sf))
    result=rbind(result,df)
  }
}
result
```

```
> result
  spatial range            dt nbpol
1       5   0.3 15.87669 secs 13345
2       5   0.4 13.24635 secs 10220
3       5   0.5 13.09548 secs  7377
4      10   0.3 23.52739 secs 14554
5      10   0.4 20.82742 secs 11418
6      10   0.5 21.47286 secs  8646
7      15   0.3 38.39682 secs 14063
8      15   0.4 40.37232 secs 10717
9      15   0.5 33.56786 secs  7954
```

- Il convient ensuite de visualiser les différents résultats dans QGIS.



5-0,5 (7377 polygones)



15-0,3 (14063 polygones)

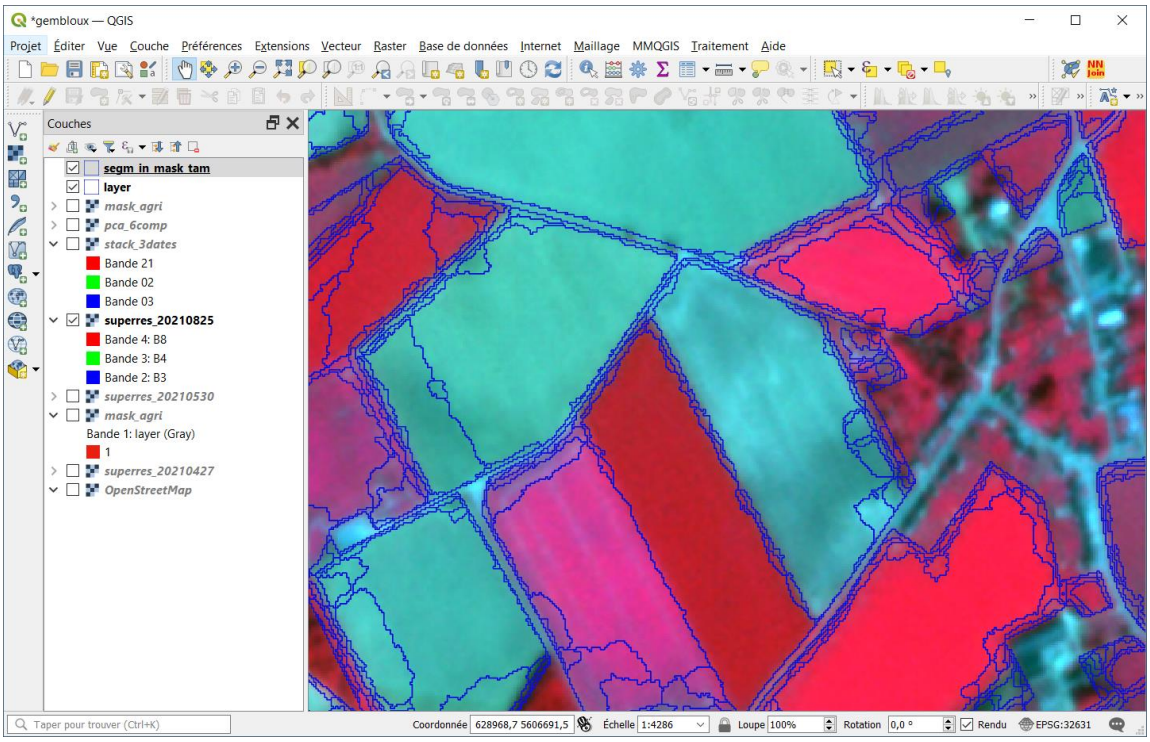
5.4.3 Segmentation de la scène complète

- Les valeurs des paramètres qui sont finalement retenues sont « spatialr = 12 » et « ranger = 0.4 ».
- Par ailleurs, l'option « –mode.vector.inmask » est utilisée pour limiter la création de segments aux surfaces couvertes par la couche **mask_agri** présentée au § 5.3 et correspondant aux zones agricoles.
- Le traitement de la scène complète à une durée d'environ 6 minutes. Le résultat de ce traitement se trouve dans le répertoire /solution.

```
# 5.4.3 Segmentation de la scène complète ---------------

if(file.exists(f_out)){
  file.remove(f_out)
}

f_pca=paste0(path_out,"/pca_6comp.tif")
f_out = paste0(path_out,"/segm_in_mask.gpkg")
f_mask=paste0(path_in,"/mask_agri.tif")
file.exists(f_mask)
cmdline = paste0(path_otb, "otbcli_Segmentation ",
                 ' -in ' , f_pca,
                 ' -filter "meanshift" ' ,
                 ' -filter.meanshift.spatialr 12 ' ,
                 ' -filter.meanshift.ranger 0.4 ' ,
                 ' -filter.meanshift.maxiter 10' ,
                 ' -filter.meanshift.minsize 20' ,
                 ' -mode.vector.out ',  f_out,
                 ' -mode.vector.inmask ',f_mask,
                 ' -mode.vector.outmode "ovw" ' ,
                 ' -mode.vector.tilesize 8000 ',
                 ' uint32',
                 ' -mode.vector.simplify 0 ')

t1=Sys.time()
### system(cmdline)
### ne pasw exécuter (durée : 6 minutes)
### le fichier se trouve dans le répertoire \solution
t2=Sys.time()
(t2-t1)

> (t2-t1)
Time difference of 6.171408 mins
```

5.4.4 Post-traitement

- La figure qui suit présente un zoom sur la couche de segments qui vient d'être produite. Celle-ci comporte un très grand nombre de petits polygones situés aux limites des parcelles agricoles.

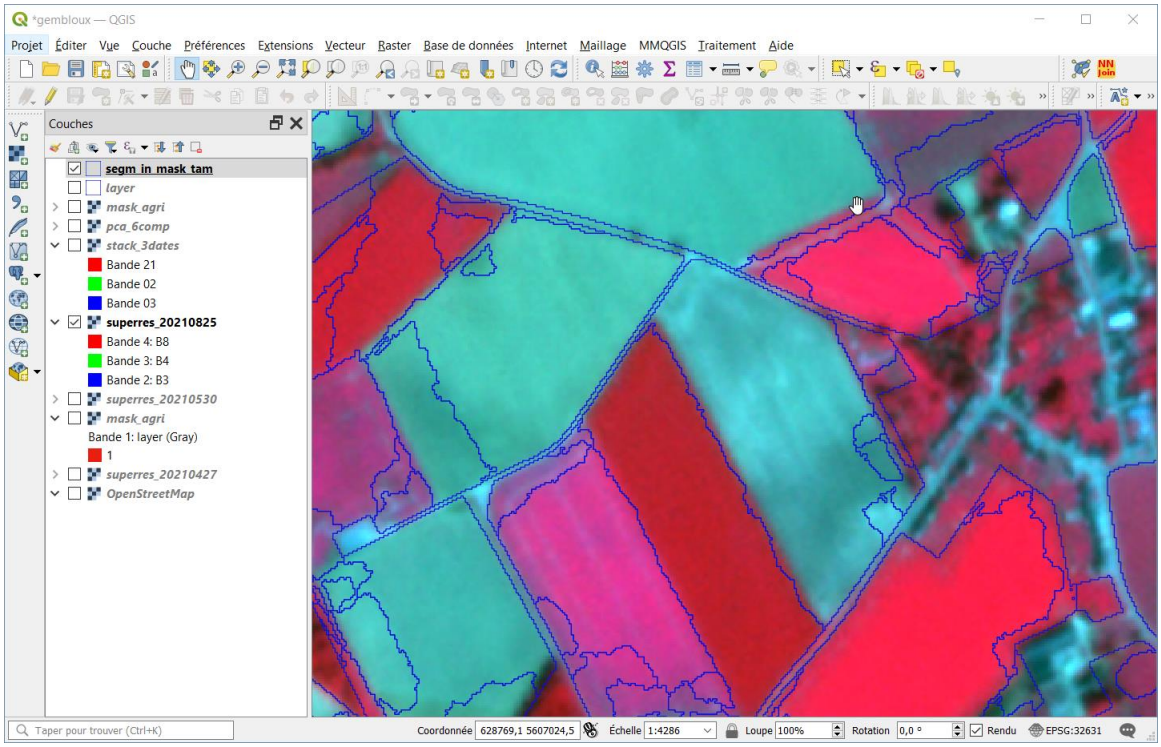


- L'étape de post-traitement qui suit va tenter d'en diminuer le nombre.
- Ce post-traitement est réalisé en 3 étapes : (1) la couche de segments est rasterisée. (2) La couche raster est ensuite « nettoyée » par tamisage. (3) Le résultat de ce dernier est finalement retransformé en couches de polygones.

```
# Elimination des segments "parasites"
f_in = paste0(path_out,"/segm_in_mask.gpkg")
segm=vect(f_in) # lecture des segments (polygones)

# Rasterisation des segments (template : masque)
f_rast = paste0(path_out,"/segm_in_mask.tif")
r0=rast(f_mask)
r=rasterize(segm,r0,field="DN",filename=f_rast,overwrite=T)
```

```
# Application d'1 tamisage sur la version raster des segments
f_tam = paste0(path_out,"/segm_tam.tif")
library(qgisprocess)
algo="gdal:sieve"
result = qgis_run_algorithm(
  algorithm=algo,
  INPUT = f_rast,
  THRESHOLD = 100,
  OUTPUT = f_tam,
  .quiet = TRUE)

# Vectorisation de la couche tamisée
r=rast(f_tam)
mask=rast(f_mask)
r=r*mask # appliquer le masque "zones agricoles"
polyg=as.polygons(r,dissolve=T) # conversion raster -> polygones
f_out=paste0(path_out,"/segm_in_mask_tam.gpkg")
writeVector(polyg,f_out,overwrite=T)

# Diminution du nombre de polygones
nrow(segm)
nrow(polyg)

> nrow(segm)
[1] 123192
> nrow(polyg)
[1] 17548
```

- Ce post-traitement a permis de réduire le nombre de polygones de 123192 à 17548. Le paramètre de tamisage a été fixé à 100 : il s'agit d'un compromis à fixer : supprimer suffisamment de petits polygones, mais éviter de modifier de manière trop importante la forme de certaines parcelles.
- La figure suivante montre un zoom sur la couche finale.



6. Classification orientée objet

6.1 Introduction

- Dans la suite de l'exercice, une classification orientée objet est réalisée sur le résultat de la segmentation produite au paragraphe précédent. L'objectif est d'identifier les principales cultures pratiquées au sein des parcelles agricoles.
- Pour cela nous disposons d'un jeu de données d'entrainement constitué de 1500 points répartis au sein de parcelles représentatives de principales cultures pratiquées dans la région. Un second jeu de données contient 600 points réservés pour la validation (100 points par type de culture).

```
# 6.1. Données de référence --------------
f_train=paste0(path_in,"/train_agri.gpkg")
f_valid=paste0(path_in,"/valid_agri.gpkg")

# données d'entraînement
train=st_read(f_train,quiet = T)
(df_tr=as.data.frame(table(train$nom)))

valid=st_read(f_valid,quiet = T)
(df_val=as.data.frame(table(valid$nom)))
```

```
> # données d'entraînement
> train=st_read(f_train,quiet = T)
> (df_tr=as.data.frame(table(train$nom)))
                Var1 Freq
1             Autres  250
2  Betterave sucrière  250
3     Froment d'hiver  250
4       Maïs ensilage  250
5      Pomme de terre  250
6 Prairie et fourrage  250
>
> valid=st_read(f_valid,quiet = T)
> (df_val=as.data.frame(table(valid$nom)))
                Var1 Freq
1             Autres  100
2  Betterave sucrière  100
3     Froment d'hiver  100
4       Maïs ensilage  100
5      Pomme de terre  100
6 Prairie et fourrage  100
```

6.2 Préparation des données

6.2.1 Calcul des statistiques spectrales pour les segments

- Avant de construire le modèle de prédiction, il convient de préparer les données. Cette préparation concerne notamment le calcul des attributs spectraux des segments : cela consiste à décrire les segments en regards des différentes bandes spectrales disponibles. Les segments étant constitués de groupe de pixels, plusieurs paramètres statistiques peuvent être calculés pour chaque segment et pour chaque bande spectrale : moyenne, écart-type…

- Ces données seront utilisées par l'algorithme de classification pour prédire l'appartenance d'un segment à une de 6 classes de culture.
- Ce traitement est pris en charge par la fonction ***otbcli_ObjectsRadiometricStatistics***.
- Il est demandé de ne pas exécuter ce traitement qui demande un temps de calcul de plus d'une heure ! Le résultat se trouve dans le répertoire /solution.

```
# 6.2.1 Calcul des statistiques spectrales par segment --------------

f_vrt=paste0(path_out,"/stack_3dates.vrt")
f_segm=paste0(path_out,"/segm_in_mask_tam.gpkg")
segm=st_read(f_segm)
segm$id_segm=seq.int(nrow(segm))
segm=dplyr::select(segm,id_segm)
st_write(segm,f_segm,delete_layer = T)
cmdline = paste0(path_otb,'otbcli_ObjectsRadiometricStatistics ',
           ' -in ', f_segm,
           ' -im ', f_vrt,
           ' -field "id_segm" ')
t1=Sys.time()
#####system(cmdline) # ne pas exécuter (durée > 1 heure)
t2=Sys.time()
(t2-t1)

> t2=Sys.time()
> (t2-t1)
Time difference of 1.430041 hours

# Lire les segments
segm=st_read(f_segm)
names(segm)
> names(segm)
  [1] "id_segm"  "NbPixels" "Flat"     "Round"    "Elong"    "Perim"    "meanB1"
  [8] "meanB2"   "meanB3"   "meanB4"   "meanB5"   "meanB6"   "meanB7"   "meanB8"
 [15] "meanB9"   "meanB10"  "meanB11"  "meanB12"  "meanB13"  "meanB14"  "meanB15"
 [22] "meanB16"  "meanB17"  "meanB18"  "meanB19"  "meanB20"  "meanB21"  "stdB1"
 [29] "stdB2"    "stdB3"    "stdB4"    "stdB5"    "stdB6"    "stdB7"    "stdB8"
 [36] "stdB9"    "stdB10"   "stdB11"   "stdB12"   "stdB13"   "stdB14"   "stdB15"
 [43] "stdB16"   "stdB17"   "stdB18"   "stdB19"   "stdB20"   "stdB21"   "MedB1"
 [50] "MedB2"    "MedB3"    "MedB4"    "MedB5"    "MedB6"    "MedB7"    "MedB8"
 [57] "MedB9"    "MedB10"   "MedB11"   "MedB12"   "MedB13"   "MedB14"   "MedB15"
 [64] "MedB16"   "MedB17"   "MedB18"   "MedB19"   "MedB20"   "MedB21"   "VarB1"
 [71] "VarB2"    "VarB3"    "VarB4"    "VarB5"    "VarB6"    "VarB7"    "VarB8"
 [78] "VarB9"    "VarB10"   "VarB11"   "VarB12"   "VarB13"   "VarB14"   "VarB15"
 [85] "VarB16"   "VarB17"   "VarB18"   "VarB19"   "VarB20"   "VarB21"   "KurtB1"
 [92] "KurtB2"   "KurtB3"   "KurtB4"   "KurtB5"   "KurtB6"   "KurtB7"   "KurtB8"
 [99] "KurtB9"   "KurtB10"  "KurtB11"  "KurtB12"  "KurtB13"  "KurtB14"  "KurtB15"
[106] "KurtB16"  "KurtB17"  "KurtB18"  "KurtB19"  "KurtB20"  "KurtB21"  "SkewB1"
[113] "SkewB2"   "SkewB3"   "SkewB4"   "SkewB5"   "SkewB6"   "SkewB7"   "SkewB8"
[120] "SkewB9"   "SkewB10"  "SkewB11"  "SkewB12"  "SkewB13"  "SkewB14"  "SkewB15"
[127] "SkewB16"  "SkewB17"  "SkewB18"  "SkewB19"  "SkewB20"  "SkewB21"  "geom"
```

- La table d'attributs de la couche **segm_mask_tam** contient les statistiques descriptives relatives aux 21 bandes spectrales contenues dans le fichier **stack_3dates.vrt**. Les noms des champs sont constitués du nom du paramètre (mean :moyenne, std : écart-type, Med : médiane, Var : variance, Kurt : kurtosis, Skew : skewness) et du nom de la bande spectrale (B1 à B21).
- Des paramètres de forme relatifs aux segments sont également calculés : Flat, Round et Elong.

6.2.2 Préparation des données d'entrainement pour le modèle de classification

- Les données d'entraînement et de validation se présentent sous la forme de points. Il convient de transférer le type de culture de chaque point vers le segment qui contient le point. Cette opération est réalisée en combinant une opération d'intersection et une jointure de table.
- Lors de la sauvegarde des segments d'entrainement et de validation, on décide de ne conserver que les attributs correspondant aux moyennes de bandes spectrales par segment (meanB1 à meanB21).

```
# 6.2.2 Préparer les données d'entraînement/validation -----------

# agréger les points d'entrainement et de validation
ref=rbind(train,valid)

# croiser les segments avec la couche de points
int=st_intersection(segm,ref)

# jointure
int=st_drop_geometry(int)
df=left_join(segm,int,by=c("id_segm"="id_segm"))
df$train[is.na(df$train)]=0
df$valid[is.na(df$valid)]=0

segm$culture=df$culture
segm$train=df$train
segm$valid=df$valid
names(segm)

# extraire les segments d'entrainement et de validation
segm_tr=segm[segm$train==1,]
nrow(segm_tr)
segm_val=segm[segm$valid==1,]
nrow(segm_val)

# sauvegarder les segments d'entrainement et de validation
# on conserve uniquement les moyennes spectrales

segm_tr=dplyr::select(segm_tr,culture,meanB1:meanB21)
f_segm_tr=paste0(path_out,"/segm_train.gpkg")
st_write(segm_tr,f_segm_tr,delete_layer = T)

segm_val=dplyr::select(segm_val,culture,meanB1:meanB21)
f_segm_val=paste0(path_out,"/segm_valid.gpkg")
st_write(segm_val,f_segm_val,delete_layer = T)
```

6.3 Entrainement du modèle de classification (forêt aléatoire)

- L'algorithme de classification utilisé dans cet exemple est le même que pour la classification « pixel ». Il s'agit de l'algorithme Randomforest (forêt aléatoire).
- Le modèle est construit avec la fonction ***otbcli_TrainVectorClassifier***. Les paramètres de commandes définissent notamment la couche contenant les segments d'entraînement (paramètre « -io.vd »), les segments utilisés pour la validation (paramètre «-valid.vd »), le nom du champ qui définit les classes à prédire (paramètre « -cfield »), ainsi que la liste des attributs à prendre en compte dans la construction du modèle (paramètre « -feat »).
- Dans l'extrait du script qui suit, on construit 3 modèles intégrant 7, 14 ou 21 bandes spectrales correspondant respectivement à 1, 2 ou 3 dates.

```
# 6.3 Entraînement du modèle --------------

# 6.3.1. définition des attributs à prendre en compte dans les 3 modèles

f_segm=paste0(path_out,"/segm_train.gpkg")

# génération des listes d'attributs pour 3 modèles (1, 2 ou 3 dates)
feat7 = NULL
feat14 = NULL
feat21 = NULL
for(i in 1:21){
  feat21=paste0(feat21," meanB",i)
  if(i<=7){
    feat7=paste0(feat7," meanB",i)
  }
  if(i<=14){
    feat14=paste0(feat14," meanB",i)
  }
}
feat7
feat14
feat21

list_feat=c(feat7,feat14,feat21)
> feat7
[1] " meanB1 meanB2 meanB3 meanB4 meanB5 meanB6 meanB7"
> feat14
[1] " meanB1 meanB2 meanB3 meanB4 meanB5 meanB6 meanB7 meanB8 meanB9
 meanB10 meanB11 meanB12 meanB13 meanB14"
> feat21
[1] " meanB1 meanB2 meanB3 meanB4 meanB5 meanB6 meanB7 meanB8 meanB9
 meanB10 meanB11 meanB12 meanB13 meanB14 meanB15 meanB16 meanB17 mea
nB18 meanB19 meanB20 meanB21"
```

```
# 6.3.2 Construction des modèles
# basés sur les données de 1, 2 ou 3 image(s)/date(s) ---
result=NULL
for (i in 1:3){
  print(i)
  features=list_feat[i]
  f_model= paste0(path_out,"/model_",i,".xml") # stockage du modèle
  cmdline = paste0(path_otb,'otbcli_TrainVectorClassifier ',
             ' -io.vd ', f_segm_tr,
             ' -valid.vd ', f_segm_val,
             ' -io.out ', f_model,
             ' -feat ', features,
             ' -cfield culture ',
             ' -classifier rf ',
             ' -classifier.rf.nbtrees 100')
  txt=system(cmdline,intern=TRUE)

  # on extrait de la variable txt la ligne avec le Kappa
  txt1=txt[grepl("Kappa", txt)]
  # transformer la ligne de texte en vecteur
  txt2=unlist((strsplit(txt1," ")))
  kappa=txt2[length(txt2)]
  df=data.frame(nbdates=i,kappa=kappa)
  result=rbind(result,df)
}
```

```
> print(result)
  nbdates kappa
1       1 0.352
2       2 0.538
3       3 0.754
```

- On constate que la prise en compte de la seule date d'avril 2021 conduit à un indice kappa d'à peine 0,352. L'ajout de la date mai 2021 puis celle de septembre 2021 permet d'atteindre des niveaux de précision de respectivement 0,538 et 0,754.

- Le tableau ci-dessous représente la matrice de confusion pour la classification reposant sur les images des 3 dates. Il apparaît clairement que la classe « Autres » est celle qui fonctionne le moins bien. La précision du producteur associée à cette classe est d'à peine 0,25 !

  Cette faible performance s'explique par le fait que la classe « autres » rassemble une grande diversité de spéculations très hétérogènes et dont la signature spectrale est peu spécifique.

Références \ Prédictions	[6]	[91]	[99]	[201]	[311]	[901]
[6]	91	0	2	2	2	3
[91]	0	84	4	5	0	7
[99]	9	5	25	11	44	6
[201]	2	2	1	92	1	2
[311]	0	0	4	0	96	0
[901]	0	1	8	2	0	89

Code	Nom
6	Prairie et fourrage
311	Froment d'hiver
91	Betterave sucrière
901	Pomme de terre
201	Maïs ensilage
99	Autres

- Pour mesurer l'effet néfaste de cette classe « autres » sur les performances du modèle, il suffit de relancer l'ajustement des modèles en retirant cette classe des données d'entrainement et de validation.

```
# Ajustement du modèle en supprimant la classe "99-Autres"

train=st_read(f_segm_tr)
valid=st_read(f_segm_val)
train=train[train$culture!=99,]
valid=valid[valid$culture!=99,]
f_segm_tr2=paste0(path_out,"/train_sans99.gpkg")
f_segm_val2=paste0(path_out,"/valid_sans99.gpkg")
st_write(train,f_train2,delete_layer = T)
st_write(valid,f_valid2,delete_layer = T)


f_model2= paste0(path_out,"/model_sans99.xml")
cmdline = paste0(path_otb,'otbcli_TrainVectorClassifier ',
                 ' -io.vd ', f_segm_tr2,
                 ' -valid.vd ', f_segm_val2,
                 ' -io.out ', f_model2,
                 ' -feat ', feat21,
                 ' -cfield culture ',
                 ' -classifier rf ',
                 ' -classifier.rf.nbtrees 100')
system(cmdline)
```

```
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Confusion matrix (rows = reference labels,
 columns = produced labels):
      [6] [91] [201] [311] [901]
[  6]    92     0     1     4     3
[ 91]     0    90     4     0     6
[201]     2     2    92     1     3
[311]     0     0     0   100     0
[901]     0     0     3     0    97

2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Precision of class [6] vs all: 0.978723
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Recall of class    [6] vs all: 0.92
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: F-score of class   [6] vs all: 0.948454

2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Precision of class [91] vs all: 0.978261
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Recall of class    [91] vs all: 0.9
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: F-score of class   [91] vs all: 0.9375

2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Precision of class [201] vs all: 0.92
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Recall of class    [201] vs all: 0.92
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: F-score of class   [201] vs all: 0.92

2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Precision of class [311] vs all: 0.952381
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Recall of class    [311] vs all: 1
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: F-score of class   [311] vs all: 0.97561

2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Precision of class [901] vs all: 0.889908
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Recall of class    [901] vs all: 0.97
2022-10-17 17:43:08 (INFO) TrainVectorClassifier: F-score of class   [901] vs all: 0.92823

2022-10-17 17:43:08 (INFO) TrainVectorClassifier: Global performance, Kappa index: 0.9275
[1] 0
```

- L'indice kappa est passé de 0,754 à 0,928. L'amélioration est donc importante. Mais ce nouveau modèle n'est pas capable de détecter correctement les parcelles relevant de la classe « autres ». Dans les lignes de script qui suivent, on ré-entraine le modèle en utilisant les données

d'entrainement sans la classe « Autres », mais cette classe « Autres » est conservée dans les données de validation.

```
# Ajustement du modèle en supprimant la classe "99-Autres"
# Les données de validation contiennent la classe "99-Autres"
f_model2= paste0(path_out,"/model_sans99.xml")
cmdline = paste0(path_otb,'otbcli_TrainVectorClassifier ',
                 ' -io.vd ', f_segm_tr2,
                 ' -valid.vd ', f_segm_val,
                 ' -io.out ', f_model2,
                 ' -feat ', feat21,
                 ' -cfield culture ',
                 ' -classifier rf ',
                 ' -classifier.rf.nbtrees 100')
system(cmdline)
```

```
2022-10-18 08:01:54 (INFO) TrainVectorClassifier: Confusion m
atrix (rows = reference labels, columns = produced labels):
      [6] [91] [99] [201] [311] [901]
[  6]    92     0     0     1     4     3
[ 91]     0    90     0     4     0     6
[ 99]    10    20     0     4    52    14
[201]     2     2     0    92     1     3
[311]     0     0     0     0   100     0
[901]     0     0     0     3     0    97
```

- On constate que les segments de validation appartenant à la classe « Autres » sont majoritairement classés dans la classe « 311 – Froment d'hiver » (52 %) ainsi que dans la classe « 91 - Betterave sucrière » (20 %).
- Cette exemple illustre bien la difficulté à procéder à des classifications mettant en œuvre un (très) grand nombre de classes, avec des fréquences d'occurrence très différentes.
- Le tableau qui suit présente une partie des statistiques issues du parcellaire agricole anonyme pour la zone couverte par cet exercice de classification. Seules les 25 cultures les plus fréquentes (sur un total de 75) sont reprises.
- Pour prendre en compte un plus grand nombre de classes dans la classification et réduire l'impact de la classe « Autres », il faudrait pouvoir travailler sur une zone d'étude plus étendue, afin de disposer d'échantillons d'entrainement/validation suffisants pour un plus grand nombre de cultures.

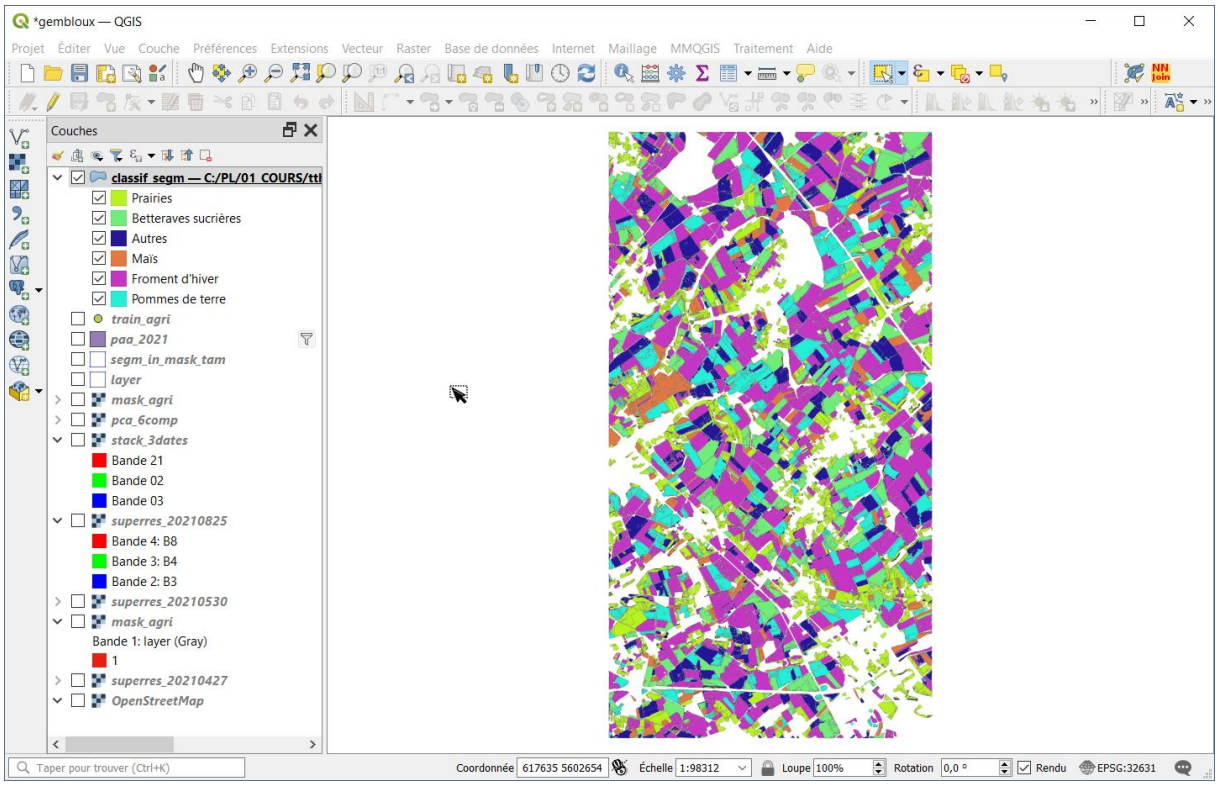
Cultures	Code	Nombre	Fréquence
Prairie et fourrage	6	1264	35.97
Froment d'hiver	311	644	18.33
Betterave sucrière	91	257	7.31
Pomme de terre	901	222	6.32
Maïs ensilage	201	158	4.50
Autres	99	114	3.24
Jachère	80	85	2.42
Chicorée à inuline	9811	73	2.08
Epeautre d'hiver	36	69	1.96
Pois récoltés à l'état frais	931	62	1.76
Orge d'hiver	321	56	1.59
Lin textile	921	53	1.51
Maïs grain	202	52	1.48
Colza d'hiver	4111	41	1.17
Autres fourrages	743	39	1.11
Luzerne	73	35	1.00
Haricots	9410	28	0.80
Autres couverts semés	85	21	0.60
Miscanthus	884	20	0.57
Endives (chicons)	9515	17	0.48
Autres légumes de plein air	951	10	0.28
Orge de brasserie	323	10	0.28
Fraises	9516	9	0.26
Total			95.02

6.4 Application du modèle de classification

- La dernière étape consiste à appliquer le modèle de classification à l'ensemble des segments. Cette opération est prise en charge par la fonction ***otbcli_VectorClassifier***.
- Celle-ci génère une couche vectorielle dans laquelle chaque segment se voit attribuer une prédiction d'appartenance à une des classes (paramètre « -cfield »).

```
file_segm = paste0(path_output,"/segm_valid.shp")
file_out = paste0(path_output,"/classif_objet.shp")
cmdline = paste0(path_otb,'otbcli_VectorClassifier ',
       ' -in ', file_segm,
       ' -model ', file_model,
       ' -cfield predicted ',
       ' -feat ', features,
       ' -out ', file_out)
system(cmdline)
```

- L'affichage de la couche avec le fichier de style **culture.qml** devrait se présenter comme dans la figure qui suit.


*gembloux — QGIS
Projet Éditer Vue Couche Préférences Extensions Vecteur Raster Base de données Internet Maillage MMQGIS Traitement Aide
Couches
classif_segm — C:/PL/01_COURS/ttl
Prairies
Betteraves sucrières
Autres
Maïs
Froment d'hiver
Pommes de terre
train_agri
paa_2021
segm_in_mask_tam
layer
mask_agri
pca_6comp
stack_3dates
Bande 21
Bande 02
Bande 03
superres_20210825
Bande 4: B8
Bande 3: B4
Bande 2: B3
superres_20210530
mask_agri
Bande 1: layer (Gray)
1
superres_20210427
OpenStreetMap
Taper pour trouver (Ctrl+K)
Coordonnée 617635 5602654
Échelle 1:98312
Loupe 100%
Rotation 0,0 °
Rendu
EPSG:32631