Cahier des charges du contrôle qualité des surfaces brûlées issues de la chaîne des feux de l'OEIL

Thomas Avron

2024-03-28

Table des matières

1 Cahier des charges

1.1 Introduction

Sur la base de la spécification STAC et des outils Python (Pystac et Stackstac) associés, nous pouvons maintenant réaliser le cahier des charges de déploiement de la solution retenue de qualification des surfaces brûlées issues de la chaîne des feux.

Le PoC ayant été concluant, le travail de développement ne sera pas trop important. Il s'agit seulement de retravailler les fonctions et les scripts pour les rendre compatibles à un environnement de production.

1.2 Intégration dans le système de l'OEIL

Les scripts du PoC sont fournis, il sera nécessaire de les intégrer de manière propre dans un script qui puisse être appelé automatiquement afin de réaliser les traitements avec le moins d'intervention humaine possible.

Nous estimons à une demie-journée le temps d'échange avec l'OEIL pour bien comprendre la cible et identifier les possibilités d'intégration d'un script réalisé dans les règles de l'art.

La gestion de l'environnement sera importante mais a bien été dégrossie pendant le PoC. Nous avons spécifié dans le benchmarking l'ensemble des librairies nécessaires au contrôle qualité.

Enfin une partie intégration des bonnes pratiques de l'OEIL (utilisation de dotenv pour gérer les chemins d'accès vers les données stockées en local ou à stocker, les credentials), ainsi qu'un travail de parallélisation du script grâce à dask sera nécessaire. Nous estimons le temps de travail pour cela à 1 jour.

La question du déploiement avec `Docker` sera à étudier.

1.2.1 Librairies

Voici les librairies utilisées dans le script du PoC :

```
from pystac_client import Client as psc
import stackstac
import matplotlib.pyplot as plt
import geopandas as gpd
import pandas as pd
import datetime
import numpy

import rioxarray
import rasterio.features
import xarray
from shapely.geometry import mapping
from pyproj import CRS
```

Outre ces librairies fonctionnelles, les librairies suivantes seront nécessaires :

- `dotenv` pour la partie gestion de l'environnement qui permet de développer en s'appuyant sur un fichier de configuration qui passent des variables d'environnement pour gerer les services externes, les credentials, etc..
- `intake` pour le chargement et l'accès aux données
- `dask` pour la parallélisation.

1.2.2 Parallélisation de certaines fonctions

Il faudra donc parralèliser certaines fonctions, notamment les fonctions qui se sont avérées lourdes comme la fonction de sélection des scènes avec un couvert nuageux non restrictif :

```
def select_scenes_without_cc(gdf_filter, stack):
  """
  On utilise la Scene Classification de Sentinel pour vérifier que
  la zone brulée est visible sur l'image.
  """
  data_times = pd.to_datetime(stack['time']).date
  dates_burnedarea = gdf_filter['date_'].values
  images_to_keep = []

  for i, time in enumerate(data_times):
    if time in dates_burnedarea:
      images_to_keep.append(i)
      print(f"on conserve automatiquement l'image {i}")
      continue

    scl_data = stack.isel(time = i).sel(band = "scl")

    mask = (scl_data>=4) & (scl_data<=7)
    filtered_data = scl_data.where(mask)

    percentage = filtered_data.count() / scl_data.count() *100

    if percentage > 95:
      print(f"on prend l'image sufisamment peu couverte {i}")
      images_to_keep.append(i)

  data_to_keep = stack.isel(time=images_to_keep)

  print(f"Nombre d'images après filtrage :{len(images_to_keep)}")

  return data_to_keep
```

La présente fonction doit être parralèlisée avec `dask`. Pour cela, une connaissance du cluster de machines disponibles à l'Oeil est nécessaire.

1.2.3 Adaptation de certaines fonctions aux données d'entrées

En input nous aurons un geodataframe qui contient toutes les surfaces brulées à traiter. Voici les fonctions qui réalisent les prétraitements :

```
local_path_to_BA = r"C:\Users\Administrateur\OneDrive\Documents" + \
r"\APID\CLIENTS\OEIL" + \
r"SISSPEO\datas\sentinel_surfaces_detectees_sept_oct_2023.gpkg"
BurnedArea_data = gpd.read_file(local_path_to_BA)


def preprocess_geometries_df(gdf):
  """
  On rajoute une colonne date_ au format datetime et on passe
  de multipolygones à polygon
  On récupère la liste de dates des geométries de surfaces brûlées
  avec un intervalle de temps
  de 120 jours avant la première détection de surface brûlée et
  le geodataframe corrigé (passé de multipolygones à polygones)
  """
  gdf['date_']= pd.to_datetime(gdf['date'], format='%Y-%m-%d').dt.date
  gdf = gdf.explode()
  datemin = (min(gdf['date_']) \
    - datetime.timedelta(days=120)).strftime('%Y-%m-%d')
  datemax = (max(gdf['date_']) \
    + datetime.timedelta(days=60)).strftime('%Y-%m-%d')
  dates = f"{datemin}/{datemax}"
  print(f"Interval temporel {dates}")
  return (gdf, dates)
```

Cette fonction part du geodataframe pour en récupérer la date. Il faudra réaliser une fonction englobante qui parcourt les lignes du geodataframe pour effectuer les traitements sur chaque surface brûlée du gpkg en input. Cette fonction englobante qui appellera toutes les autres fonctions devra être parallélisée au maximum etant donné qu'elle fera appel à une boucle pour réaliser le parcours du geodataframe.

```
def construct_stack(gdf_filtrer, cc, URL, collection, dates):
  """
  Fonction de construction de la stack à partir
  d'un geodataframe filtré sur une surface brûlée.
  """
  bbox = gdf_filtrer["geometry"].to_crs(4326).total_bounds
```

```
print(f'Emprise spatiale de la géométrie sélectionnée : {bbox}')
client = psc.open(URL)
search = client.search(

  collections=[collection],
  bbox=bbox,
  datetime=dates,
  query={"eo:cloud_cover": {"lt": cc}}

  )

print(f"{search.matched()} scenes Sentinel-2 L2A trouvées dans
l'intervalle temporel ayant - de {cc}% de couverture nuageuse")
items = search.item_collection()
stack = stackstac.stack(
  items,
  bounds_latlon=[bbox[0], bbox[1],  bbox[2],  bbox[3]],

  gdal_env=stackstac.DEFAULT_GDAL_ENV.updated(
    {'GDAL_HTTP_MAX_RETRY': 3,
    'GDAL_HTTP_RETRY_DELAY': 5,
    }),
  epsg=4326
  ).rename({'x': 'lon', 'y': 'lat'})

print("stack.crs", stack.crs)

return stack
```

A l'issue de l'appel de ces fonctions, nous récupérons une `stack` qui contient toutes les scènes associées à une bounding box d'une géométrie et sur un intervalle de date que nous avons pris par défaut à 120 jours avant la date de détection de la surface brûlée par la chaîne des feux, et 60 jours après. Il ne faudra surtout pas filtrer sur le couvert nuageux et donc supprimer l'option `query` dans l'appel `client.search`. L'intervalle temporel sera à préciser à l'issue de l'étude statistique.

1.2.4 Calcul des indices à la volée

Il est à prévoir une correction de l'offset selon la formule L2A_SRi = (L2A_DNi + BOA_ADD_OFFSETi) / QUANTIFICATION_VALUEi. Nous n'avons pas trouvé comment accéder à l'offset dans les métadonnées des DataArray. Une investigation est à prévoir pour corriger cet offset et trouver la quantification associée :

Une piste est d'utiliser les snippets de code suivants :

```
def snippet_indication():
  data_indices = sentinel_stack.to_dataset(dim='band')
  for i in range(len(data_indices.coords['band'].values[()])):
    # On boucle sur les 32 bandes de la stack
    name = data_indices.coords["raster:bands"][i].common_name.values[()]
    # On récupère le nom de la bande pour corriger la bande par la suite
    offset = data_indices.coords["raster:bands"][i].values[()][0]["offset"]
    # On récupère l'offset
    # définir si la quantification_value est bien la coordonnée Scale

    # Faire le calcul
    # data_indices[name] = data_indices[name] - offset / scale ???
```

```
def correction_offset (data_indices):

  data_indices["red"] = data_indices["red"]+0.1
  data_indices["nir"] = data_indices["nir"]+0.1
  data_indices["swir22"] = data_indices["swir22"]+0.1
  data_indices["swir16"] = data_indices["swir16"]+0.1
  data_indices["green"] = data_indices["green"]+0.1
  data_indices["blue"] = data_indices["blue"]+0.1
  data_indices["nir08"] = data_indices["nir08"]+0.1
  data_indices["rededge2"] = data_indices["rededge2"]+0.1
  data_indices["rededge3"] = data_indices["rededge3"]+0.1

  return data_indices
```

Une fois que les bandes ont été corrigées de l'offset, il faut maintenant calculer les indices. En fonction du résultat de l'étude statistique, les indicateurs pertinents seront calculés. Nous mettons ici le calcul pour 5 indices : le `ndvi`, le `nbr`, le `nbr+`, le `bais2` et le `badi`.

```
def calcul_indices (data_to_keep):
  """
  On calcule les indices à partir de la stack data_to_keep
  """
  data_indices = data_to_keep.sel(band=["blue","rededge2", "rededge3", "green",
   "red", "nir", "nir08", "rededge1","swir16",
    "swir22", "scl"]).to_dataset(dim='band')

  for elt in data_indices.data_vars :
```

```
    data_indices[elt] = data_indices[elt] + 0.1 #Correction de l'offset
    # à supprimer si la fonction de correction est appliquée avant.

  data_indices['ndvi'] = ((data_indices['nir'] - data_indices['red'])/\
    (data_indices['nir'] + data_indices['red']))

  data_indices['nbr'] = ((data_indices['nir'] - data_indices['swir22'])/\
    (data_indices['nir'] + data_indices['swir22']))

  data_indices['nbr+'] = ((data_indices['swir22'] - \
    data_indices['nir08'] - data_indices['green'] \
    - data_indices['blue'])/(data_indices['swir22'] + \
        data_indices['nir08'] + data_indices['green'] \
        + data_indices['blue']))

  data_indices['bais2'] = (1-(numpy.sqrt((data_indices['rededge2'] \
    * data_indices['rededge3'] \
    * data_indices['nir08'])/data_indices['red']))*\
        ((data_indices['swir22'] -\
         data_indices['nir08'] )/ numpy.sqrt((data_indices['swir22']\
             + data_indices['nir08'] ))+1))

  data_indices['f1'] = ((data_indices['swir22'] + data_indices['swir16']) - \
    (data_indices['nir'] + data_indices['nir08'])) /\
        (numpy.sqrt((data_indices['swir22'] + data_indices['swir16']) +\
             (data_indices['nir'] + data_indices['nir08'])))

  data_indices['f2'] = (2 - numpy.sqrt((data_indices['rededge2']\
    *data_indices['rededge3']*(data_indices['nir'] + data_indices['nir08']))/\
        (data_indices['red']+data_indices['rededge1'])))

  data_indices['badi'] = data_indices['f1']*data_indices['f2']

  return data_indices
```

Les indices ainsi calculés, nous pouvons passer à l'étape suivante et calculer le masque de la surface brûlée.

1.2.5 Calcul du masque de la surface

```
def create_mask(dataset, gdf):
  """
  On utilise le geodataframe filtré sur la géométrie voulue
  pour créer un masque de la geométrie et le rajouter au dataset
  """
  ShapeMask = rasterio.features.geometry_mask(gdf['geometry'].\
    to_crs(4326).apply(mapping),
                                              out_shape =(len(dataset.lat),
                                              len(dataset.lon)),
                                              transform = dataset.transform,
                                              invert = True)
  ShapeMask = xarray.DataArray(ShapeMask, dims = ("lat","lon"))
  dataset['mask'] = ShapeMask

  return dataset
```

Une fois notre dataset complet, nous avons toutes les données à bancariser dans un netCDF.

1.2.6 Première partie du programme `main`

Le programme `main` aura donc la forme suivante :

```
def main():
  gdf, dates = preprocess_geometries_df(BurnedArea_data)

  # Il faudra remplacer cette étape par
  # la fonction encapsulante qui permettra de boucler
  # sur les surfaces brûlées du géodataframe
  gdf_filter = gdf[gdf["surface_id"]==359594]

  # Ces autres fonctions seront donc appellées dans une boucle.
  sentinel_stack = construct_stack(gdf_filter, 100, api_url,
  collection, dates)
  data_to_keep = select_scenes_without_cc(gdf_filter, sentinel_stack)
  data_indices = calcul_indices (data_to_keep)
  dataset = create_mask(data_indices, gdf_filter)
  return 0
```

Mais il faudra aussi inclure dans chaque tour de boucle la bancarisation en netCDF :

```
def sauvegarder_netcdf (data_indices, gdf_filtered):
  dataset_save = data_indices.drop([c \
    for c in data_indices.coords if not (c in ['time', 'lat', 'lon'])])
  dataset_save = dataset_save.drop_vars([v \
    for v in dataset_save.data_vars if not (v in ['ndvi', 'nbr',
    'nbr+', 'bais2', 'mask'])])
  dataset_save.attrs['spec'] = str(dataset_save.attrs['spec'])
  dataset_save.to_netcdf(f"{str(gdf_filtered["surface_id"].iloc[0])}
  _{str(gdf_filtered["nom"].iloc[0])}_
  {str(gdf_filtered["date"].iloc[0])}.nc")
  return dataset_save
```

Nous commencons par supprimer les coordonnées et les variables du xarray dont nous n'avons pas l'utilité. Ensuite, nous devons caster l'attribut specifications qui est de type RasterSpec en string sinon nous ne pouvons sauvegarder le fichier en netCDF.

Une fois ceci fait nous pouvons faire appel à la méthode `to_netcdf` d'un dataset xarray et ainsi sauvegarder notre dataset xarray en un fichier.

Il faudra vérifier que le cast de l'attribut spécifications `RasterSpec` n'engendre pas d'erreur par la suite (à la réutilisation des fichiers netCDF).

1.3 Gestion de l'information

Des suites du PoC, nous avons schématisé les flux de données comme suit :

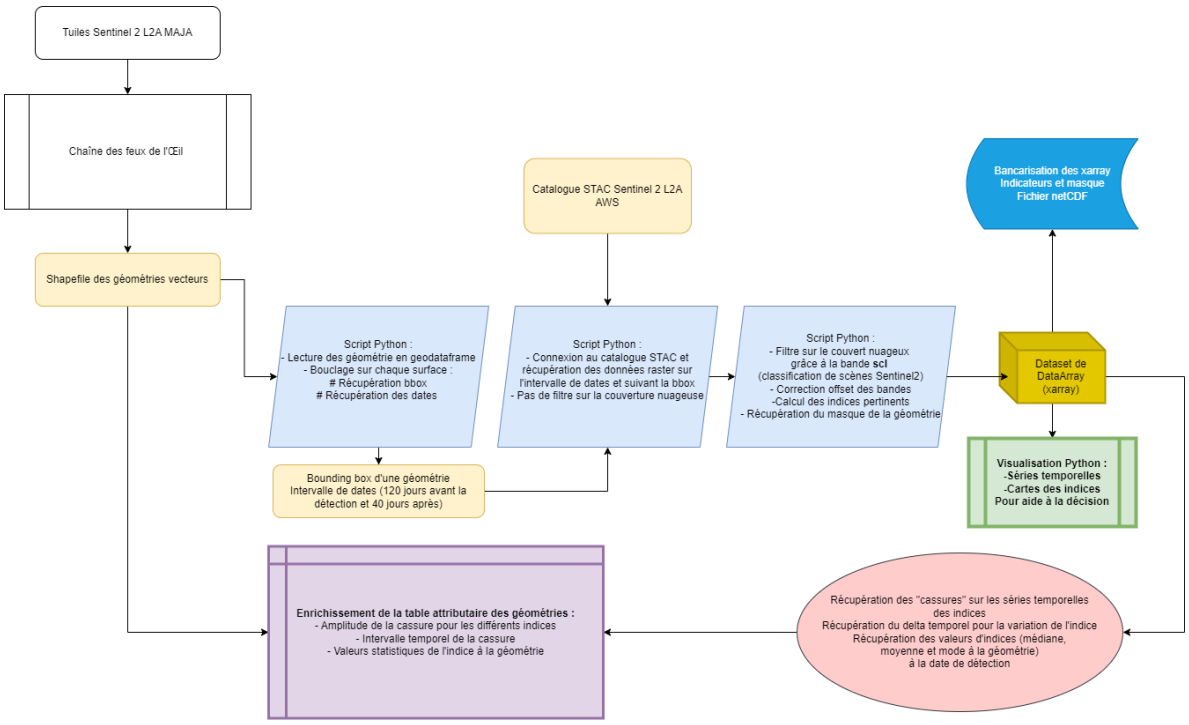

Figure 1: Schéma des flux de données

Pour bancariser correctement les netCDF, il sera intéressant d'établir un catalogue STAC des données. Nous estimons le temps de travail pour réaliser cela à une à deux journées.

Comme indiqué dans le point suivant (identification des méthodes de regroupement des indices), il sera nécessaire d'écrire la partie de script qui enrichit la table attributaire des surfaces brûlées. Nous estimons le temps de travail pour cela à 2 jours.

L'établissement d'un catalogue STAC peut s'appuyer sur la page de tutoriel suivante.

1.4 Identification des méthodes de regroupement des indices à la surface brûlée

Afin de disposer des données sources, la bancarisation du calcul d'indices a été imaginée sous la forme de fichier netCDF, comme indiqué dans la partie Benchmarking des trois outils. Cette bancarisation permet de disposer d'un jeu d'indicateurs calculé à l'échelle de la surface brûlee, sur une certaine plage temporelle (dans le PoC nous avions pris un intervalle de 120 jours amont de la date de détection de surface brûlée et 40 jours aval).

Cette bancarisation permettra de mettre en oeuvre d'autres algorithmes sur la donnée source, si le protocole de photo-interprétation évolue et qu'une partie automatisation s'intègre par la suite. Cette automatisation pourra disposer d'un catalogue de données d'indicateurs.

Pour autant les résultats du PoC ont mis en lumière qu'un enrichissement de la table attributaire des surfaces brûlées est directement possible. En effet lors de l'étude des 10 surfaces photo-interprétées comme brûlées, des 10 surfaces photo-interprétées comme non-brûlées et des 10 surface en doute, nous avons constaté (de manière purement qualitative, l'échantillon étudié n'est pas suffisamment représentatif pour conclure avec certitude de la validité quantitative de cette méthode) que nous nous y prennions, sur la base des indicateurs disponibles, d'une manière toujours similaire pour interpréter les données :

- Nous recherchions toujours un évènement en amont de la date de détection.

Un évènement est visible sur les séries temporelles par une brusque modification (une "cassure") de la valeur de l'indice. Cette variation brusque était toujours significative sur les séries temporelles.

De manière empirique, nous avons constaté :

- Qu'une diminution sur un intervalle de temps de l'ordre de la dizaine de jours ou moins de plus de 0,3 du ndvi est significative
- Qu'une diminution sur un intervalle de temps de l'ordre de la dizaine de jours ou moins de plus de 0,3 du nbr est significative
- Qu'une augmentation sur un intervalle de temps de l'ordre de la dizaine de jours ou moins de plus de 0,3 du nbr+ ou du bais2 est significative

Ces variations "brusques" indiquent toujours qu'il s'est passé quelque chose. Mais cet évènement n'est pas forcément un feu, cela peut être un coup de sécheresse, ou bien une récolte sur un champ. Pour discriminer de manière plus sûre que l'évènement est bien un feu, nous nous appuyons ensuite sur le delta NBR. En effet, sur un intervalle de temps le plus centré autour de la date de la cassure, nous traçons les cartes 2D du deltaNBR et retrouvons la géométrie lorsque l'évènement est un feu.

Puis, dans l'interprétation des résultats du PoC, nous nous appuyons sur des cartes colorées du NDVI. Une évolution confirme qu'un évènement a eu lieu. Si post évènement le NDVI est bas à l'échelle de la géométrie (inférieur à 0,2) alors nous pouvons être quasiment certains que l'évènement était bien un feu.

Ainsi nous recommandons d'effectuer une réelle étude statistique sur un échantillon plus représentatif des biômes et des saisons de Nouvelle-Calédonie. Cette étude statistique permettra d'identifier de manière précise les seuils de qualification de présence ou non d'un évènement (à partir de la détection d'une "cassure" sur une ou plusieurs séries temporelles) mais aussi de déterminer si cet évènement est bien un feu à partir des valeurs absolues des indices.

Cette étude statistique pourra s'appuyer sur les scripts du PoC réalisé.

Une fois des seuils confirmés par étude statistique, il sera alors possible d'enrichir la table attributaire des surfaces brûlées obtenue par la chaîne des feux de la manière suivante, pour

chaque indice jugé pertinent grâce à la présente étude et grâce à l'étude statistique (à réaliser) :

- Delta maximum de l'indice sur une période de temps inférieure à 15 jours
- Date d'occurence de ce delta
- Intervalle de temps exact du delta
- Delta maximum de l'indice sur une période de temps inférieure à 1 mois (afin de s'assurer que la couverture nuageuse ne bloque pas l'étude)
- Date d'occurence de ce delta
- Intervalle de temps exact du delta
- Valeur médiane de l'indice à la géométrie à la date de détection de la surface brûlée
- Valeur moyenne de l'indice à la géométrie à la date de détection de la surface brûlée
- Valeur modale (après traitement par classe) de l'indice à la géométrie à la date de détection de la surface brûlée

Cela enrichit la table attributaire de 9 colonne par indice.

Nous estimons le développement du script Python réalisant cet enrichissement à environ 2 jours. Tests compris.

1.5 Estimation des coûts de mise en oeuvre

En s'appuyant sur le travail réalisé pendant le PoC, il sera alors possible de réaliser l'intégration des scripts dans le système de l'OEIL en 6 jours au total. Nous pouvons prendre une marge de sécurité d'une journée pour assurer la réalisation. Ainsi nous estimons qu'en 7 jours maximum l'intégration est possible.

Cette estimation s'appuie sur un profil de développeur experimenté. Le langage retenu est bien entendu Python et nous ne prennons pas en considération le temps de développement d'interfaces utilisateurs. Les scripts sont supposés tourner en automatique, avec le moins d'intervention humaine possible.

1.6 Limites

Les limites d'une telle méthode, à vérifier, sont les temps mis pour réaliser les différents traitements. Nous n'avons pas parallélisé le travail et par surface brûlée, sur un poste personnel, nous mettions aux alentours d'une minute à réaliser le traitement de la stack et de deux à trois minutes pour réaliser les plots des séries temporelles et des cartes 2D de couleur des indices.

Il sera important de faire un test sur un système permettant la parallèlisation.

1.7 Conclusion du Cahier des charges

Le Cahier des charges amène à estimer à 7 jours le temps de développement nécessaire pour intégrer les résultats de cette étude dans le système de l'OEIL.

Pour autant, avant toute intégration, une étude statistique est conseillée pour réfléchir à des seuils pertinents pour les différents indicateurs et leurs utilisation. En outre la question de l'algorithme de classification se pose aussi. Le PoC réalisé semble montrer qu'un algorithme de type forêt aléatoire semble plutôt bien indiqué pour réaliser la classification des surfaces détectées par la chaîne des feux. Pour autant, c'est l'algorithme utilisé dans la chaîne des feux même et il serait potentiellement intéressant de réfléchir à un autre type d'algorithme pour cette classification afin de varier de méthode.

La partie étude statistique et reflexion sur un algorithme cible pour la classification n'est pas comptée dans les 7 jours de développement évoqués plus haut.

2 Conclusion générale

Nous pouvons conclure qu'au regard du PoC réalisé, plusieurs pistes intéressantes de réutilisation de divers indicateurs sont apparues, avec une clarification quant à la bancarisation de ces données.

Nous regrettons de n'avoir pas eu le temps de réfléchir à l'intégration de données exogènes (tel que celles d'un MOS à jour). Pour autant l'étude réalisée, bien que qualitative, donne des indications fortes pour la suite.