SAE 4.03

—

INGENIEUR LOGICIEL 1

—

DETECTION DE BIOMES SUR DES EXOPLANETES

Bastien JALLAIS, Titouan LETONDAL, Alann MONNIER, Fabien TOUBHANS

20/06/2024

Table of Contents

SAE 4.03-IL1 : Détection de biomes sur des Exoplanètes

1. Introduction

1.1 Contexte et objectifs du projet

L'objectif de la SAE est à partir d'une banque d'images données de détecter sur chacune d'entre elles les différents biomes présent sur l'exoplanète puis de définir les différents écosystèmes présents.

2.Prétraitement de l'image

Pour la gestion du flou nous avons tester les deux méthodes, gaussiennes et par moyenne avec différé t'es valeurs. Dans un premier temps nous n'avons pas relevé d'intérêts notoires pour l'un des deux, c'est alors pas analyse de l'approche mathématiques que nous avons compris que l'approche gaussienne était mieux adaptée à nos besoin, accordant une plus grande importances aux pixels centraux plutôt qu'à ceux en périphérie.

3.Détection et visualisation des biomes

Pour cette partie, nous avons commencé tout d'abord par sélectionné le clustering le plus adapté pour trouvé les différents biomes sur l'image de l'exoplanète.

Nous avons choisi DBScan car il va nous permettre de déterminer un nombre de clusters sans que l'on est besoin de connaître le nombre au lancement de l'algorithme et est plus efficace que le clustering avec KMeans.

Le problème que nous avons rencontré est le problème de mémoire car le programme n'était pas assez optimisez malgré l'utilisation de Set. Également, ce clustering demande un temps de génération important.

C'est pourquoi nous nous sommes dirigés vers le second clustering : « KMeans ».

Celui-ci est plus rapide, plus simple à mettre en place et nous évite les problèmes de mémoires.

Au niveau de la norme, nous avons choisi la Norme Readman car nous avons trouvé sur internet qu'elle était adapté pour la segmentation et regroupement de biomes sur des cartes.

Au niveau des choix des paramètres, nous avons tout d'abord décider de créer une interface de clustering commune à chacun des algorithmes. Cette interface, contient une méthode avec comme paramètre un tableau à double entrée. Ce tableau représente un Objet avec ces propres caractéristiques. Pour DBScan , on a des objets de couleurs avec en caractéristique RGB. Pour KMeans, on a commencé par prendre pour chaque pixel des coordonnées en x et y pour les représenter.

Pour chacun de ces algorithmes, nous avons une méthode de calcul de distance. Le premier algo va calculer des distances entre couleurs pendant que le second calcul des distances entre des coordonnées.

4.Détection et affichage des écosystèmes pour chaque biome

Pour détecter les écosystèmes pour chacun des biomes trouvés, nous avons décidé d'utiliser l'algorithme de clustering K Means. L'intérêt d'utiliser cet algorithme est qu'il permet de créer différents cluster à l'aide de position en abscisse et ordonnée dans une image donnée. On va donc pouvoir définir une distance pour créer des clusters à l'aide d'une distance euclidienne entre des coordonnées.

En entrée de l'algorithme, on récupère un ensemble de coordonnées d'un biome sélectionné.

L'algorithme s'appuie sur l'utilisation de centroïde qui va représenter un groupe de coordonnées.

Le principe consiste à choisir un nombre donnée de centroïde qui sera à une des positions de notre liste de données.

```
Random random = new Random();

// Initialisation des centroides avec des données aléatoires
for (int i = 0; i < ng; i++) {
    centroids[i] = Arrays.copyOf(donnees[random.nextInt(donnees.length)], donnees[0].length);
}
```

Puis on construit des groupes à partir de ces centroïdes.

```java
boolean fini = false;
int iteration = 0;
while (!fini && iteration < 1000) {
    fini = true;

    // Réinitialiser les nouveaux centroides et les tailles de clusters
    for (int i = 0; i < ng; i++) {
        Arrays.fill(newCentroids[i], val: 0.0);
        clusterSizes[i] = 0;
    }

    // Attribution des points de données aux centroides les plus proches
    for (int i = 0; i < donnees.length; i++) {
        int nearestCentroid = indiceCentroidePlusProche(donnees[i]);
        clusterAssignments[i] = nearestCentroid;
```

Pour créer ces groupes on commence tout d'abord à chercher l'indice du centroïde le plus proche de la donnée passé en paramètre.

```java
private int indiceCentroidePlusProche(double[] donnee) {  1 usage  Alann +1
    int indice = 0;
    double distanceMin = Double.MAX_VALUE;

    for (int i = 0; i < centroids.length; i++) {
        double distance = calculerDistance(donnee, centroids[i]);
        if (distance < distanceMin) {
            distanceMin = distance;
            indice = i;
        }
    }
    return indice;
}
```

Pour récupérer l'indice du centroïde le plus proche de notre donnée, on a besoin d'utiliser une méthode qui compare une distance entre la donnée et un centroïde sélectionné.

Cette méthode va renvoyer la distance en calculant la distance euclidienne entre les coordonnées du centroïde et de la donnée.

```java
private double calculerDistance(double[] a, double[] b) {  1 usage  Bastien Jallais +1
    double sum = 0;
    for (int i = 0; i < a.length; i++) {
        double diff = a[i] - b[i];
        sum += diff * diff;
    }
    return Math.sqrt(sum);
}
```

Une fois qu'on a ajouté une nouvelle donnée à un groupe, on met à jour les centroïdes avec les nouvelles données présent dans les groupes.

```java
// Mise à jour des centroïdes
for(int l=0; l < ng; l++){
    // Si un groupe n'a pas été déjà créé on ne le mets pas à jour
    if(groupes.get(l).isEmpty()) continue;

    // Calcul le nouveau centroïde
    double[] centroid = barycentre(groupes.get(l));
    // Condition pour savoir si on reboucle pour réutiliser l'algorithme
    boolean isEqualCentroidX = (this.centroids[l][0] == centroid[0]);
    boolean isEqualCentroidY = (this.centroids[l][1] == centroid[1]);
    int long1 = this.centroids[l].length;
    int long2 = centroid.length;

    if(!(isEqualCentroidX || isEqualCentroidY || (long1 != long2)  )){
        // Change la valeur du centroide avec la nouvelle valeur
        this.centroids[l] = centroid;
        fini = false;
    }
}
```

On calcule le barycentre à l'aide de la méthode suivante en réalisant une moyenne des différents points dans un groupe donnée.

```
    // Calculer les nouveaux centroides
    for (int i = 0; i < ng; i++) {
        if (clusterSizes[i] > 0) {
            for (int j = 0; j < newCentroids[i].length; j++) {
                newCentroids[i][j] /= clusterSizes[i];
            }
        }
    }

    // Vérifier si les centroides ont changé
    for (int i = 0; i < ng; i++) {
        if (!estApproxEgal(centroids[i], newCentroids[i])) {
            centroids[i] = Arrays.copyOf(newCentroids[i], newCentroids[i].length);
            fini = false;
        }
    }
    iteration++;
}
```

5.Compte rendu du travail effectué

5.1 Résultats obtenus