Clustering Algorithms

Data Science and Information Technologies

Fall 2019-20

Semester Project

HELLENIC REPUBLIC
National and Kapodistrian University of Athens

Dimitrios Roussis

Student ID number: DS1190017

E-mail: droussis@outlook.com.gr

Supervisor: Dr. Konstantinos D. Koutroumbas

Table of Contents

INTRODUCTION

In this project, we compare different clustering algorithms by using them for the identification of homogeneous regions in a hyperspectral image.

We make use of the "Salinas_Data.mat" which contains "**Salinas_Image**", a hyperspectral image of the Salinas valley in California, USA and "**Salinas_Labels**", an image with the class labels for each pixel (ground-truth labels). The hyperspectral image depicts the area of the valley in **150x150** resolution with each of its pixels (22500 in total) containing a spectral signature of **204 distinct spectral bands** and can be visualized as a cube (see Fig. 1). The image with the ground truth labels has the same resolution but only one dimension with a categorization of each pixel to one of the eight distinct types of crops in the valley (see Fig. 2).

We implement **4 clustering algorithms** which are able to assign each pixel to a specific cluster -or class- based on the assumption that every type of distinct crop has its own spectral pattern, i.e. it reflects, emits and absorbs very highly correlated values of radiation across the spectral bands.

In the sections that follow, we explain how we preprocessed the data, we describe the framework that we use for our experiments and afterwards, we present each different type of clustering algorithm, as well as all the relevant information that we use in our experiments, such as its cost function J, its tunable hyper-parameters, etc. We also evaluate their performance (using the Rand Index) for all the different configurations that we have tried and comment on the results. Finally, we use a dimensionality reduction technique (Principal Component Analysis) to comment on the differences between the algorithms and the problems that we encountered.

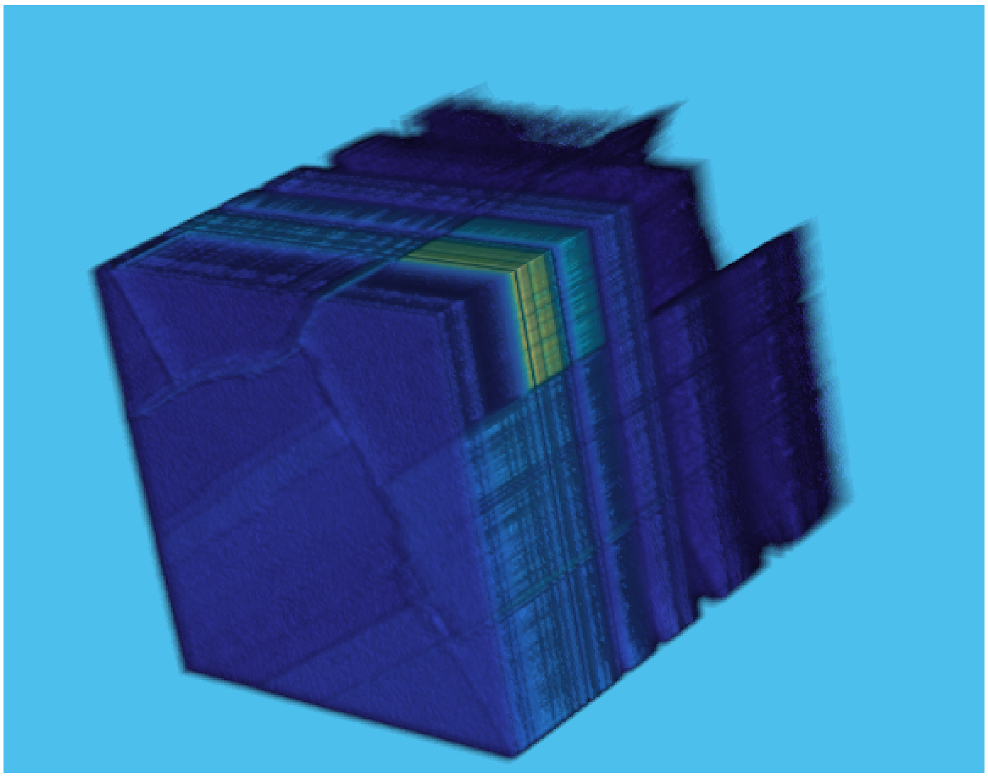

Fig. 1: Hyperspectral cube of the image

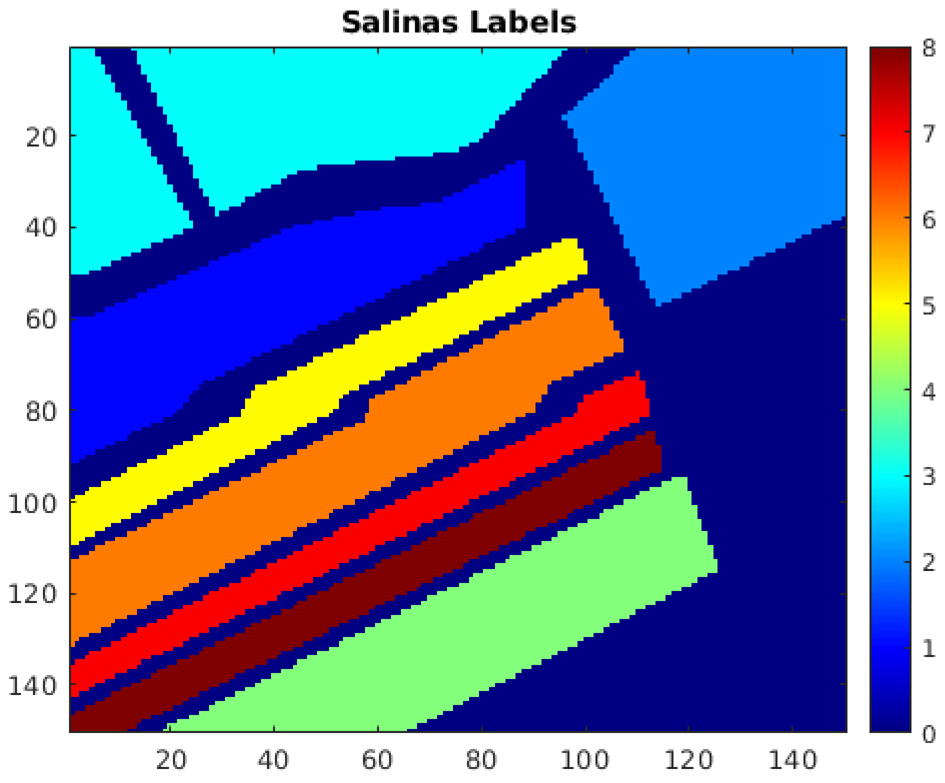

Fig. 2: Ground truth labels of the image

(1) Preprocessing

Before conducting our experiments, we need to apply some transformations to the image cube, i.e. the image with the 204 spectral bands. In particular:

- Reshape the images in order to make them 2-dimensional and **remove the zero class pixels** for which we have no information about. The end results are two matrices (rows: pixels, columns: intensity value per spectral band) with sizes:

  X: [204 x 13908] & **y**: [1 x 13908]

- Apply a **low-pass filter** in order to smooth spatially each spectral band in the hyperspectral data X. After some experimentation, we chose a **2D Gaussian filter** (see Eq. 1) with a 9 x 9 kernel and a standard deviation of **σ** = 3.5, which controls the rate of smoothing [1], i.e. the relevance of the spatial context. The effect of Gaussian filtering is the reduction in intensity of high frequency values if surrounding pixels have lower values and the enhancement of the intensity of low frequency values if surrounded pixels have higher values [2]. In order to choose the values of **σ** and the size of the kernel we performed various experiments by using the k-Means algorithm and measuring the change in its cost function J (e.g. an increase in **σ** corresponds almost to a linear decrease in J, but can also result in very poor clustering results qualitatively).

$$g(x,y)=\frac{1}{2\pi\sigma^2}e^{\frac{-x^2+y^2}{2\sigma^2}} \qquad \textit{(Eq. 1)}$$

- **Normalize** the hyperspectral data **X** by returning the z-score of each point in the data, a method also known as “standardization” (see Eq. 2). We should note, however, that this method changes the mean of the values to 0 and their standard deviation to 1, but does not alter their underlying distribution.

$$z=\frac{x-\mu}{\sigma} \qquad \textit{(Eq. 2)}$$

(2) EXPERIMENTATION FRAMEWORK

For each one of the clustering algorithms that we used in our experiments, we have a common experimentation framework. Below, each sub-process of our framework is listed in ascending order (from lowest-level to higher-level) :

- We run each algorithm for **10 complete iterations**, i.e. we repeat it with new random starting conditions for **Θ** (for some algorithms, e.g. the Possibilistic c-Means, this means repeating 10 ensemble clusterings) and from all those iterations, we choose the one with the **lowest cost function value J_{min}**.
- We repeat the aforementioned procedure for **different numbers of clusters**. More specifically, we test each different setting for integer values of **m** from 3 up to 10 (**$m \in [3, 10]$**). This choice is based on a priori knowledge of the dataset.
- We use different hyper-parameters which are specific to each individual algorithm. For example, we try using **different ways for the initialization of Θ's** (i.e. the starting coordinates of each cluster's representative), or, we implement some of the algorithms using two different distance metrics, the **Squared Euclidean distance** (see Eq. 3) and the **Canberra distance** (see Eq. 4). The Canberra distance is a dimensionless standardized value which can be used to to enhance the contribution of higher frequency spectra [3] and essentially is a weighted version of the Manhattan distance. It is *important to note* that in the cases which we run ensemble clusterings, e.g. when we implement the Possibilistic c-Means algorithm and initialize its **Θ** via the k-Means algorithm, we use the **same distance metric** for both algorithms.
- For each value of **m** (number of clusters) and each different hyper-parameter that we have used, we test and report the clustering performance of the algorithms using the **Rand Index** (see Eq. 5).

Let p=(p_1, p_2, ..., p_n) and q=(q_1, q_2, ..., q_n). We then define the following distance metrics:

Squared Euclidean Distance:

$$d_{Euclidean}(p,q)=\sum_{i=0}^{N}(p_i-q_i)^T(p_i-q_i) \qquad \textit{(Eq. 3)}$$

Canberra Distance:

$$d_{Canberra}(p,q)=\sum_{i=0}^{N}\frac{|p_i-q_j|}{|p_i|+|q_j|} \qquad \textit{(Eq. 4)}$$

Let X=(x_1, x_2, ..., x_n) and Y=(y_1, y_2, ..., y_n), where X is the array of pixel labels that results from a clustering algorithm and Y is the array of pixel ground truth labels. For each pixel pair (i, j), where i, j ∈{1, 2, ..., n}, we define:

a=The number of pixel pairs for which $x_i = x_j$ and $y_i = y_j$

b=The number of pixel pairs for which $x_i = x_j$ and $y_i \neq y_j$

c=The number of pixel pairs for which $x_i \neq x_j$ and $y_i = y_j$

d=The number of pixel pairs for which $x_i \neq x_j$ and $y_i \neq y_j$

Finally, we define the **Rand Index** of similarity between X and Y as:

$$RI(X,Y)=\frac{a+d}{a+b+c+d} \qquad \textit{(Eq. 5)}$$

NOTE: We calculate the Rand Index between the clustering results (bel) and the ground truth labels (y), **after** we have removed the zero-class pixels from both; i.e. the dimensions of both are [1 x 13908]. We also calculated the **Jaccard Coefficient** but it did not provide any useful results and thus, *will not be presented in the results*.

(3) K-Means Clustering

First, we use the **k-Means clustering algorithm** in order to detect the homogeneous regions in the image. In each experiment, we run 10 replicates of the algorithm (initializing **Θ** randomly in each one) and report the results of the one that **minimizes the cost function J of k-Means** (see Eq. 6), for different values of **m**.

$$J(U,\Theta)=\sum_{i=1}^{N}\sum_{j=1}^{m}u_{ij}\,d(x_i,\theta_j) \qquad \textit{(Eq. 6)}$$

Number of Clusters (m)	Rand Index (Squared Euclidean distance)	Rand Index (Canberra distance)
3	58.50 %	76.36 %
4	58.50 %	79.03 %
5	76.42 %	79.03 %
6	76.42 %	84.44 %
7	76.42 %	84.95 %
8	76.42 %	87.23 %
9	76.42 %	86.76 %
10	76.42 %	87.36 %

Table 1: Rand Index for the different configurations of the k-Means algorithm

As we can see in **Table 1**, when we use the **Squared Euclidean distance** the performance of k-Means is very stable for the different values of **m**. For low values of

m, the algorithm performs poorly (RI = 58.50%) and we can see that it can discern only 2 different classes (see Fig. 3). After a certain point (**m = 5**), the performance increases by almost **20%** (RI = 76.42%), but stays completely unchanged afterwards, as the algorithm mixes up many classes and ends up discerning only 5 of them; we can see that only 4 of them are the most prominent (see Fig. 4).

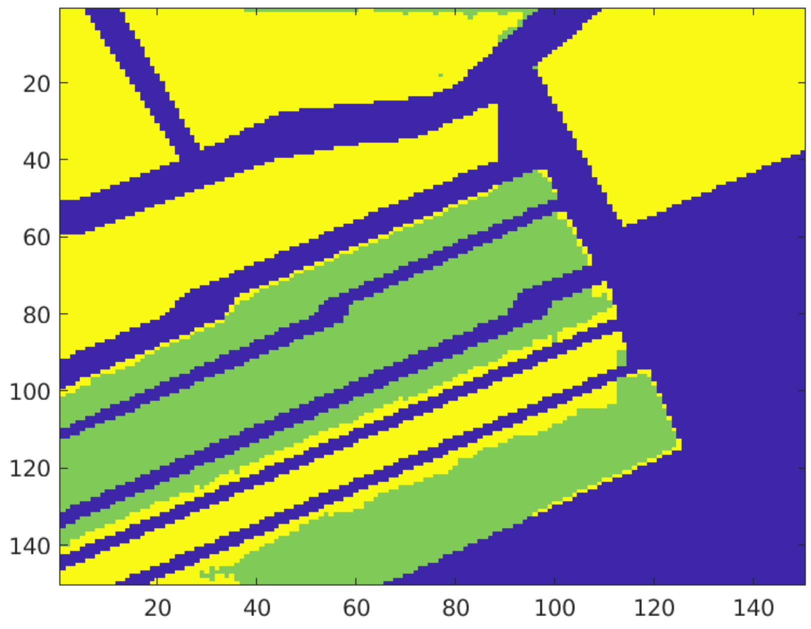

Fig. 3: k-Means clustering results (m=4, Squared Euclidean)

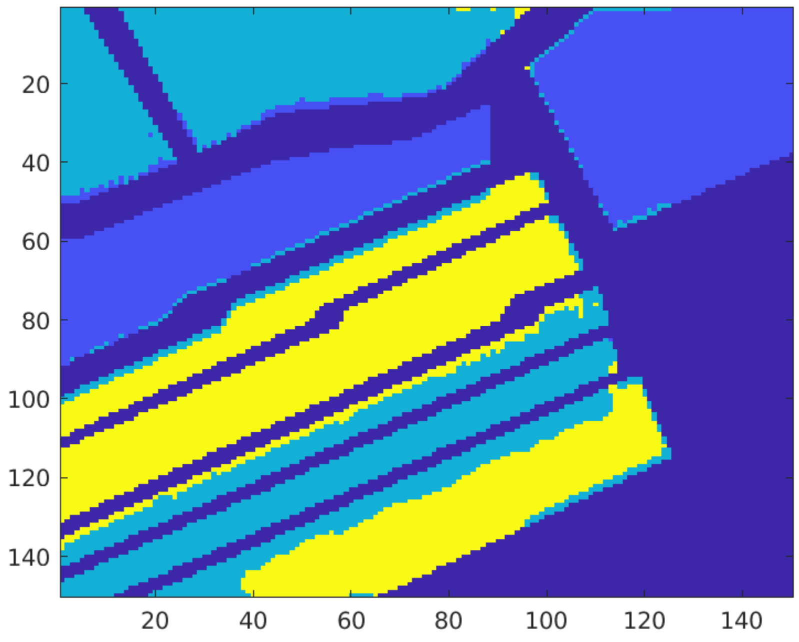

Fig. 4: k-Means clustering results (m=8, Squared Euclidean)

The results are completely different if we use the **Canberra distance** instead. The performance shows an upward trend as the value of **m** increases and drops slightly for **m=9**, perhaps as an indicator that the correct number of classes in the image is 8. For **m=8**, the performance is significantly better (RI = 87.23%) than that of the algorithm with the Squared Euclidean distance and we can also see that some classes are classified correctly, although the fact that some borders are categorized as distinct classes may be problematic (see Fig. 5). There seems to be a slight improvement for **m=10** (RI = 87.36%), but we can see that the problem with the borders persists and that the class in the top is divided into two parts in a worse fashion (see Fig. 6).

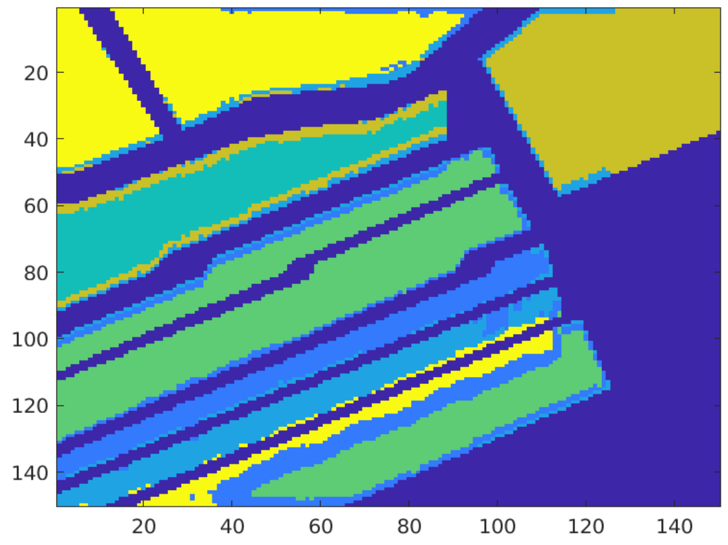

Fig. 5: k-Means clustering results (m=8, Canberra)

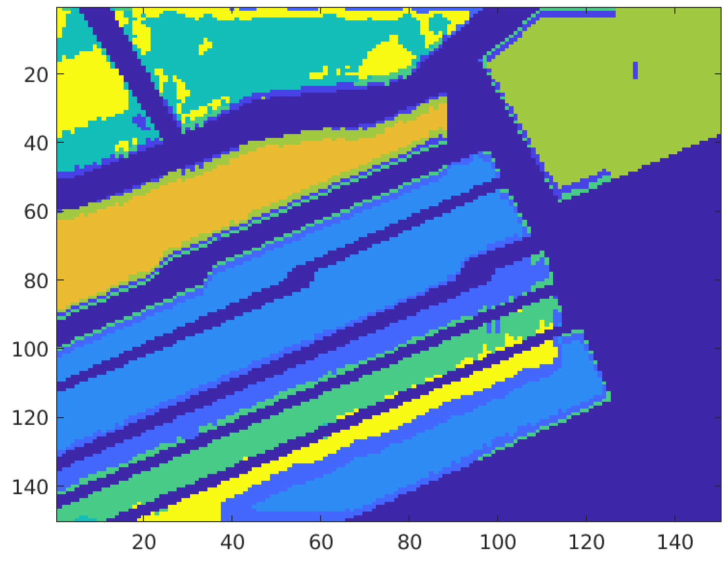

Fig. 6: k-Means clustering results (m=10, Canberra)

(4) Possibilistic C-Means Clustering

In this section, we try different configurations of the **Possibilistic c-Means clustering algorithm**. Once more, we run 10 replicates of the algorithm and report the results of the one that **minimizes the cost function J of Possibilistic c-Means** (see Eq. 7), for different values of **m** (number of clusters). The parameter **η_j** corresponds to the "zone of influence" of cluster **j** [4] and in our implementation, is updated in every iteration of the algorithm (see Eq. 8).

$$J(U,\Theta)=\sum_{i=1}^{N}\sum_{j=1}^{m}u_{ij}d(x_i,\theta_j)+\sum_{j=1}^{m}\eta_j\sum_{i=1}^{N}(u_{ij}\log u_{ij}-u_{ij}) \qquad \textit{(Eq. 7)}$$

$$\eta_j=\frac{\sum_{i=1}^{N}u_{ij}d(x_i,\theta_j)}{\sum_{i=1}^{N}u_{ij}} \qquad \textit{(Eq. 8)}$$

After conducting various experiments, we should report that random initialization of the Possibilistic c-Means works poorly (all points belong to one single super-cluster) and produces some results only if we use Canberra distance which are not good either. Thus, we decided to try two ensemble clusterings, using different types of initialization; **k-Means** and **Fuzzy c-Means** with q = 2 (see next section).

	k-Means Initialization		Fuzzy c-Means (q = 2) Initialization	
Number of Clusters (m)	Rand Index (Squared Euclidean distance)	Rand Index (Canberra distance)	Rand Index (Squared Euclidean distance)	Rand Index (Canberra distance)
3	53.55 %	58.43 %	73.30 %	58.43 %
4	47.40 %	70.21 %	81.20 %	58.44 %
5	47.42 %	65.80 %	80.16 %	58.43 %
6	54.26 %	69.49 %	81.18 %	58.43 %
7	54.39 %	77.17 %	81.76 %	58.49 %
8	54.39 %	77.87 %	81.58 %	58.46 %
9	54.34 %	79.82 %	83.07 %	66.70 %
10	47.45 %	81.73 %	81.64 %	71.38 %

Table 2: Rand Index for the different configurations of the Possibilistic c-Means algorithm

Above, in **Table 2** we can see the results for the different configurations of Possibilistic c-Means; however they may be quite **misleading**. For some configurations, we can see that the Rand Index is quite high. Nonetheless, the resulting classes are **quite noisy** and non-compact. For example, the configuration that uses **m = 8**, k-Means initialization and Canberra distance has a good score (RI = 77.87%), but as we can see in Fig. 8, results in clusters which are filled or constituted by complete noise. The noisy clusters problem is also evident when we use **m = 9**, Fuzzy c-Means initialization and Canberra distance (see Fig. 10), but with lower intensity and a smaller score (RI = 66.70%). We can safely conclude that *the Canberra distance is not appropriate for the Possibilistic c-Means clustering algorithm*, as it produces either mediocre scores and/or noisy clusters.

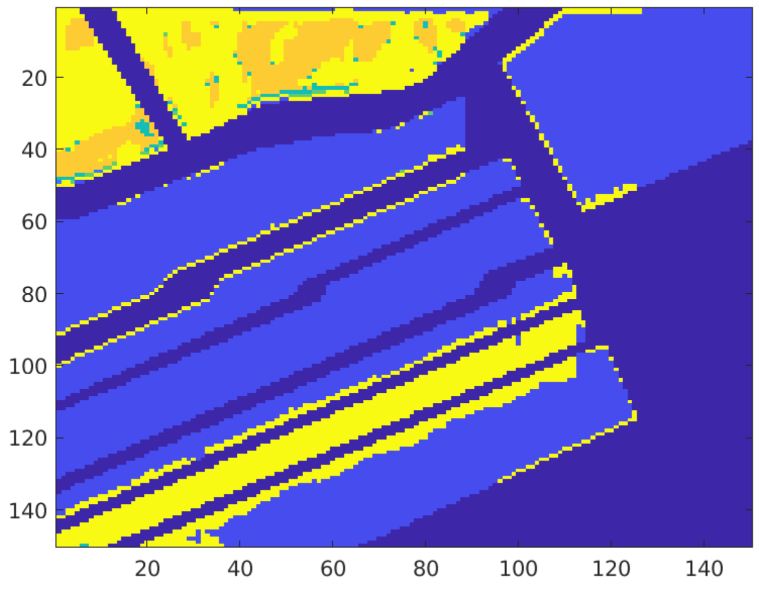

Fig. 7: Possibilistic c-Means clustering results (m=8, k-Means Initialization, Squared Euclidean)

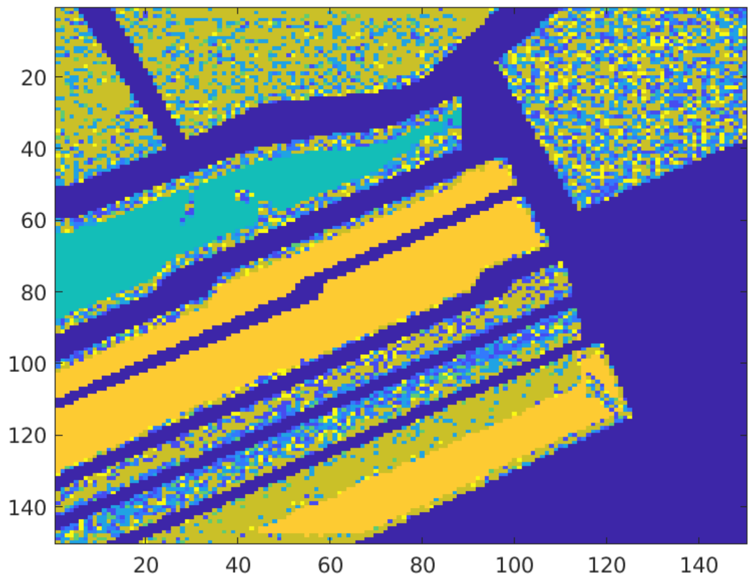

Fig. 8: Possibilistic c-Means clustering results (m=8, k-Means Initialization, Canberra)

The worst scores, however, are noticeable for the configurations of **m** that use **k-Means initialization** and the **Squared Euclidean distance**, as they range from 47.40% to 54.39%. However, using the Squared Euclidean distance and in contrast with the utilization of the Canberra distance, does not result in noisy clusters (see Fig. 7)

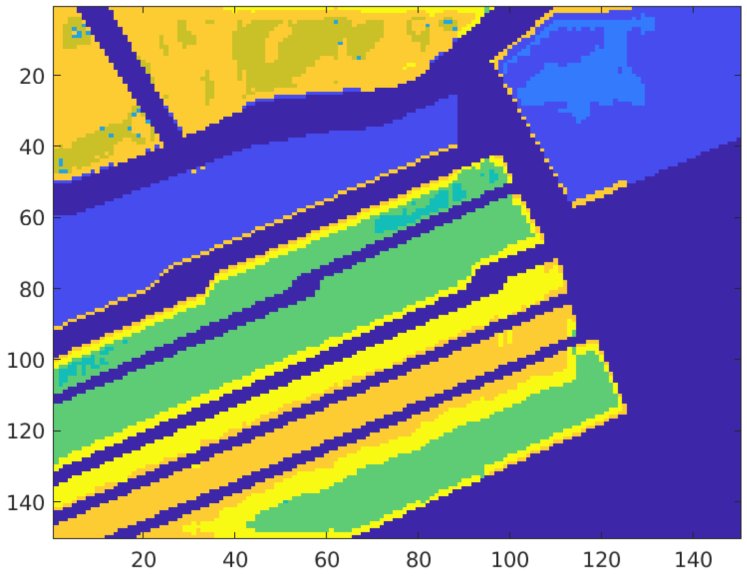

Fig. 9: Possibilistic c-Means clustering results (m=8, Fuzzy c-Means Initialization, Squared Euclidean)

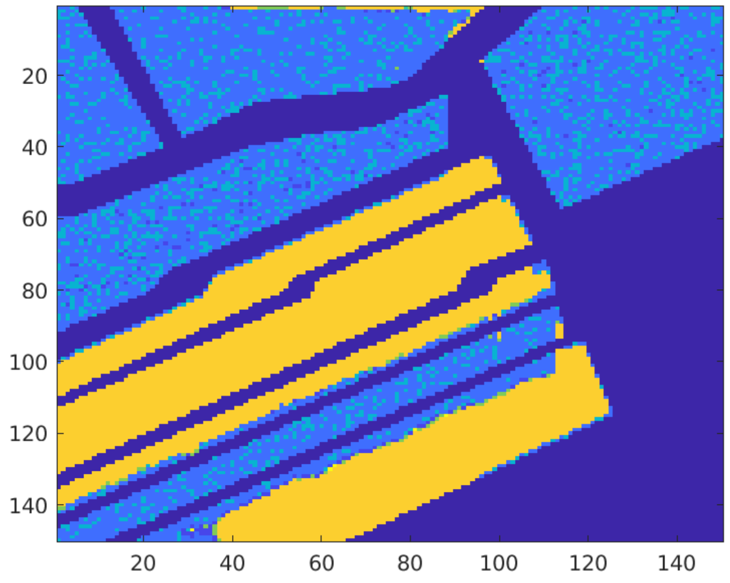

Fig. 10: Possibilistic c-Means clustering results (m=9, Fuzzy c-Means Initialization, Canberra)

In general, we can see that the Possibilistic c-Means algorithm *requires many experiments and different configurations in order to work properly* and -at least in the dataset of the project- does not work at all alone. The configuration which had the best RI scores and was more reliable than the others, is the Fuzzy c-Means initialization (q=2) using the Squared Euclidean distance. As we can see for **m=8** in Fig. 9 (RI = 81.58%), the algorithm gives good results, but tends to divide -ground truth- classes into two or even three different clusters in seemingly random regions; not just border regions.

(5) Fuzzy C-Means Clustering

In this section, we make use of the **Fuzzy c-Means clustering algorithm**. Yet again, we run 10 replicates of the algorithm and report the results of the one that **minimizes the cost function J of Fuzzy c-Means** (see Eq. 9), for different values of **m** (number of clusters). The parameter **q**, which is known as the "fuzzifier", controls the bias towards fuzzy and hard -e.g. k-Means- clustering (for q > 1, there are cases in which fuzzy clustering outperforms the best hard clustering in terms of the cost function J).

$$J(U,\Theta)=\sum_{i=1}^{N}\sum_{j=1}^{m}u_{ij}^{q}\,d(x_i,\theta_j) \qquad \textit{(Eq. 9)}$$

	Fuzzifier q = 2		Fuzzifier q = 3	
Number of Clusters (m)	Rand Index (Squared Euclidean distance)	Rand Index (Canberra distance)	Rand Index (Squared Euclidean distance)	Rand Index (Canberra distance)
3	76.39 %	71.05 %	76.43 %	60.44 %
4	81.11 %	71.26 %	81.29 %	60.45 %
5	83.41 %	70.90 %	83.21 %	60.45 %
6	84.85 %	70.61 %	85.51 %	60.44 %
7	87.56 %	70.46 %	85.52 %	60.44 %
8	87.99 %	70.47 %	86.79 %	60.44 %
9	88.03 %	71.82 %	87.24 %	60.44 %
10	88.22 %	73.57 %	87.79 %	60.44 %

Table 3: Rand Index for the different configurations of the Fuzzy c-Means algorithm

As we can see in Table 3, the algorithm generally works a lot better (5% up to 25% increase in RI) when we use the Squared Euclidean distance and its performance often peaks at values of **m** close to 8 (i.e. the correct number of classes in the image). As we can see in Fig. 12 and Fig. 14, when we use the **Canberra distance**, the algorithm tends to combine 2-4 ground truth classes to a supercluster with some limited sporadic noise. However, when we use the Canberra distance, we can see that the choice of **q** (fuzzifier) is very important, leading to a 10% decrease In RI (when changing **q = 2** to **q = 3**), as well as a very stable performance for all the different values of **m**.

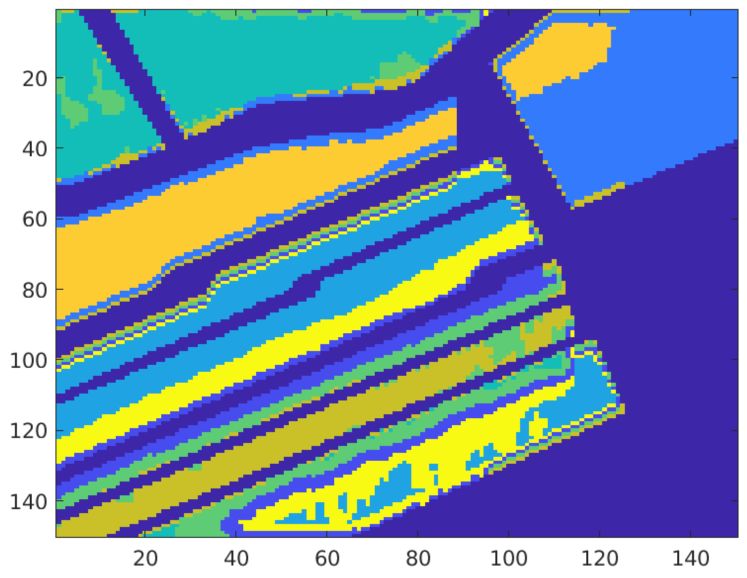

Fig. 11: Fuzzy c-Means clustering results (m=8, q=2, Squared Euclidean)

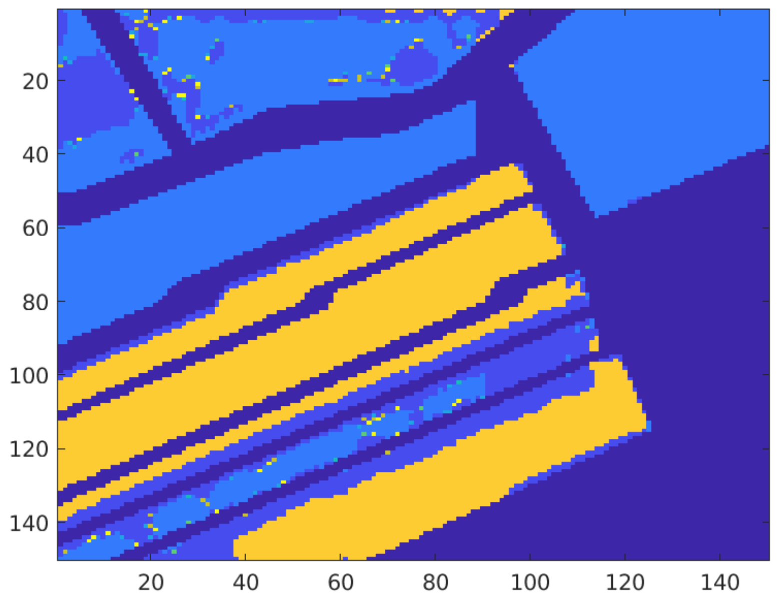

Fig. 12: Fuzzy c-Means clustering results (m=8, q=2, Canberra)

On the contrary, when we use the **Squared Euclidean distance**, the choice of **q** does not seem to play a huge role as both configurations give very good results with the performance peaking for **m = 10** (RI = 88.22% for **q = 2** and RI = 87.79% for **q = 3**). The same problems appear in both configurations: difficulties in border regions, occasional noise within the clusters (non-compact clusters) and merging/division of ground truth classes. These problems can be observed in Fig. 11 (RI = 87.99% for **m = 8** ; **q = 2**) and Fig. 13 (RI = 86.79% for **m = 8** ; **q = 3**).

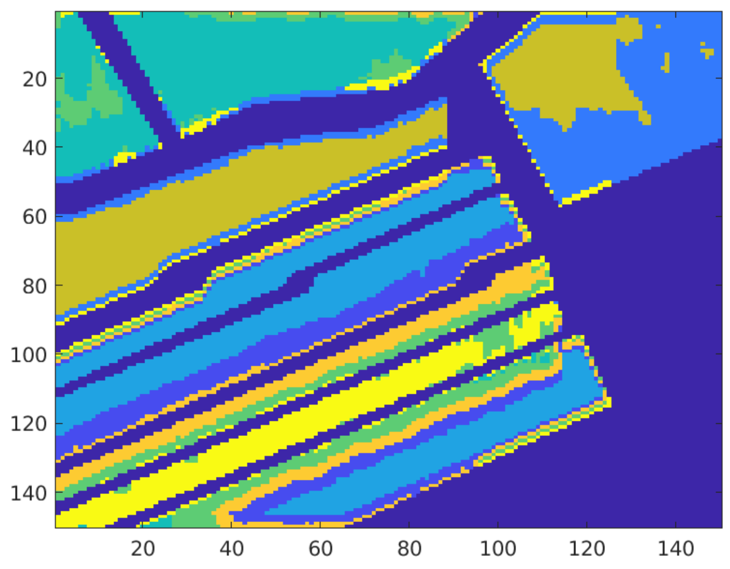

Fig. 13: Fuzzy c-Means clustering results (m=8, q=3, Squared Euclidean)

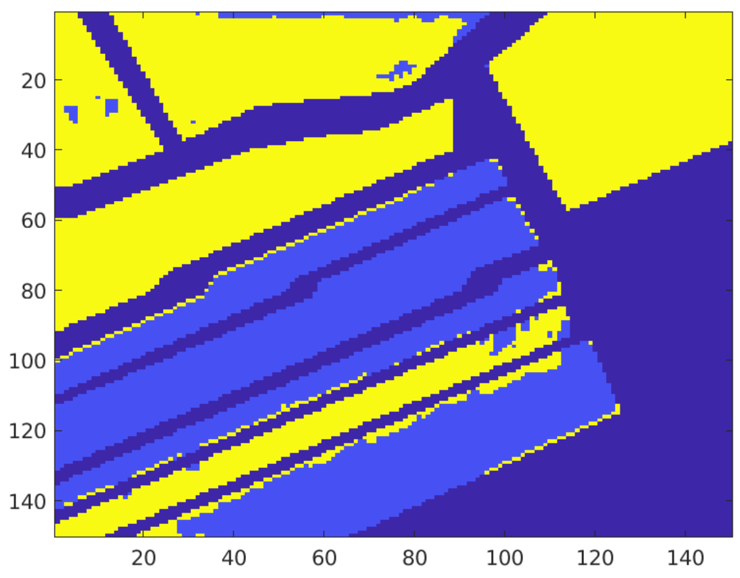

Fig. 14: Fuzzy c-Means clustering results (m=8, q=3, Canberra)

(6) Probabilistic (Gaussian Mixture) Clustering

In this section, we conduct our experiments using a **Probabilistic clustering algorithm**, which models each class as a Gaussian distribution (Gaussian Mixture), and belongs to the more general Expectation-Maximization framework. The built-in function in MATLAB (fitgmdist) provides functionality for running 10 replicates and choosing the one which maximizes the log-likelihood (see Eq. 10). We test the algorithm for two different configurations (for each value of **m**); initializing the means (μ's) **randomly** and initializing them using the **k-means++** algorithm [5]. The covariance matrices (Σ's) are initialized as diagonal matrices, where element j is the variance of the j-th dimension.

$$L(X;\Theta,P)=\sum_{i=1}^{N}\sum_{j=1}^{m}P(j|x_i;\theta_j)\ln(P_j\,p(x_i|j;\theta_j)) \qquad \textit{(Eq. 10)}$$

where $\boldsymbol{\theta}_J = \{\mu_j, \Sigma_j\}$ as we need both for each cluster

Number of Clusters (m)	Rand Index (Random Initialization)	Rand Index (k-Means++ Initialization)
3	75.01 %	74.93 %
4	82.69 %	81.44 %
5	84.00 %	85.42 %
6	87.31 %	85.58 %
7	87.57 %	88.10 %
8	88.31 %	88.54 %
9	89.22 %	88.93 %
10	91.49 %	91.58 %

Table 4: Rand Index for the different configurations of the Gaussian Mixture algorithm

As we can see in **Table 4**, the performance increases as the number of clusters increases and is actually very good for both configurations. When we use **random initialization**, the RI is 88.31% for **m = 8** and the algorithm manages to discern large parts of individual classes, but cannot classify even one correctly, as it faces problems in borders regions and the smaller classes in the middle and the bottom (see Fig. 15). For **m=10**, the performance increases to 91.49%, but the problem persists (see Fig. 16).

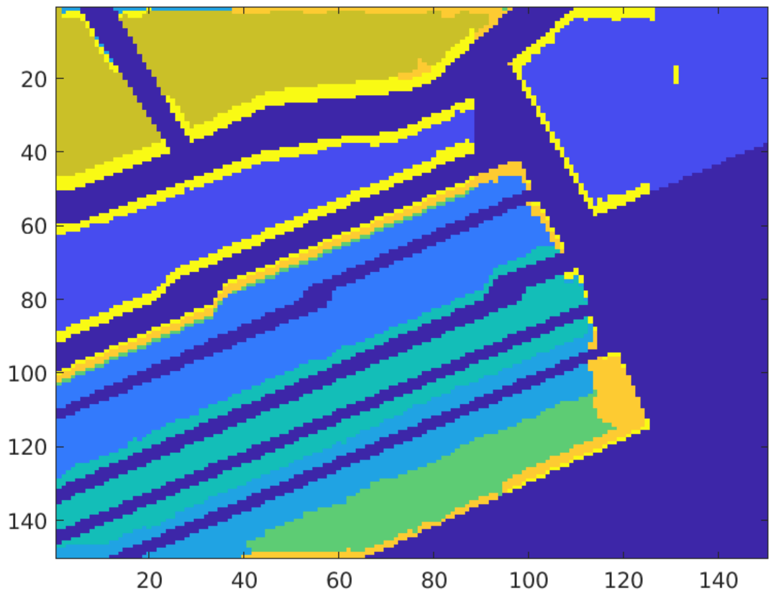

Fig. 15: Probabilistic clustering results (m=8, Random Initialization)

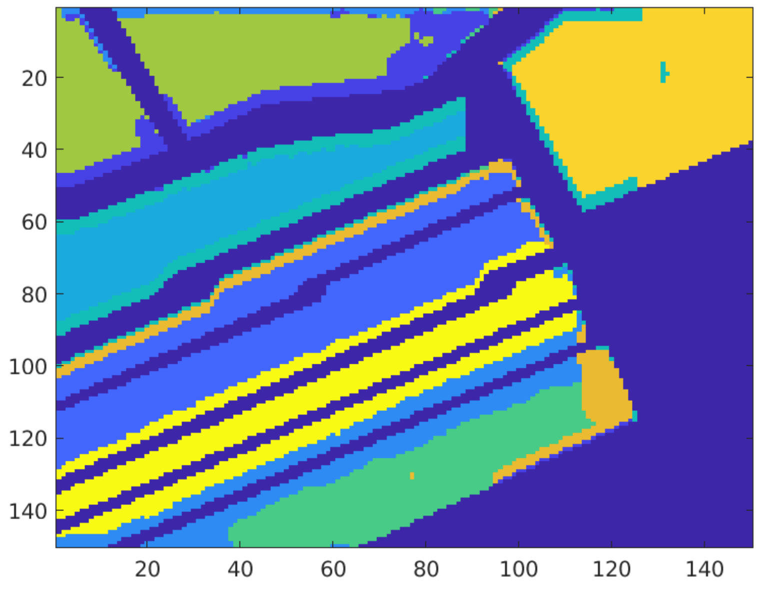

Fig. 16: Probabilistic clustering results (m=10, Random Initialization)

When we initialize the Gaussian Mixture algorithm using the **k-Means++**, we can see quite similar results. For **m = 8**, the increase in performance in contrast with the first configuration (random initialization) is 0.23% and the same problems as before are prominent (see Fig. 17). For m=10, the increase in performance is 0.09% and we reach the **top performance** of all the algorithms that we have used so far in the project (RI = 91.58%). While the problem with the borders seems less nuanced in the middle and bottom classes, it is more pronounced in the top class (see Fig. 18).

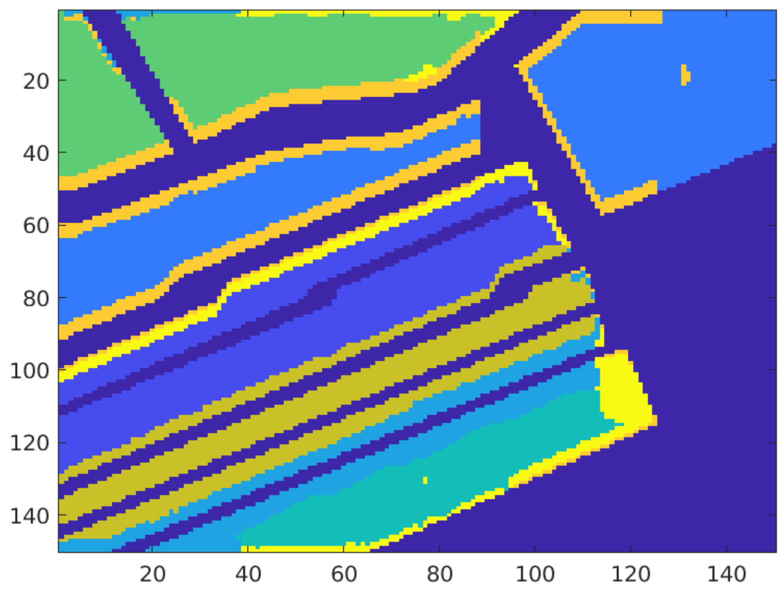

Fig. 17: Probabilistic clustering results (m=8, k-Means++ Initialization)

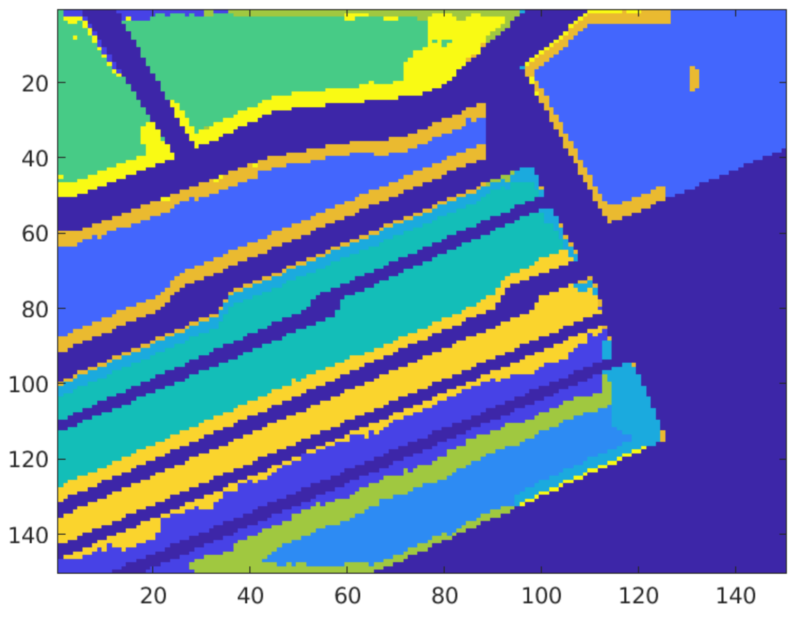

Fig. 18: Probabilistic clustering results (m=10, k-Means++ Initialization)

(7) Principal Component Analysis

In this section, we conduct **Principal Component Analysis** (PCA) on the dataset and particularly on X, **without filtering or normalizing it** (see section 1). PCA is a simple technique of dimensionality reduction, i.e. the "compression" of matrix X to another one with fewer features which manage to preserve as much information as possible.

After running the PCA algorithm provided by the instructor, we found out that just the **first principal component** explains 95.66% of the variance in the dataset. Below, we can see how the dataset looks like after its pixels have been projected on the space spanned by the first principal component (see Fig. 19).

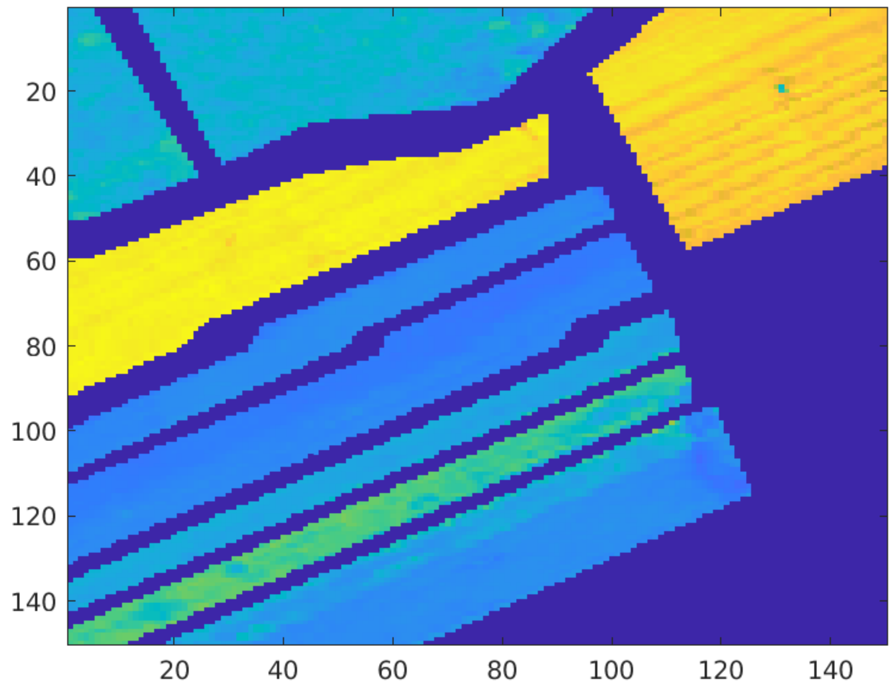

Fig. 19: 1st Principal Component of the dataset

In what follows, we can see the clustering results for some configurations of the algorithms that we have used throughout the project, using a transformed X with only the **first 4 principal components which explain 99.84% of the variance** in total. We should note that we test **only** for a *fixed number of clusters* (**m** = **8**).

First, we try running the **k-Means** algorithm using the **Squared Euclidean** distance. The results are somewhat better (**RI = 78.21%**) than those in section 3 (see Table 1). Furthermore, we can see that the resulting clusters with PCA (see Fig. 20) closely resemble those that we have seen in the earlier implementation of k-Means (see Fig. 4). We should not that we also tried implementing k-Means using the Canberra distance, which was the best configuration in the earlier experiments, but we only got very poor results (RI = 13.95%). Later on, we will see if we can alleviate this problematic situation.

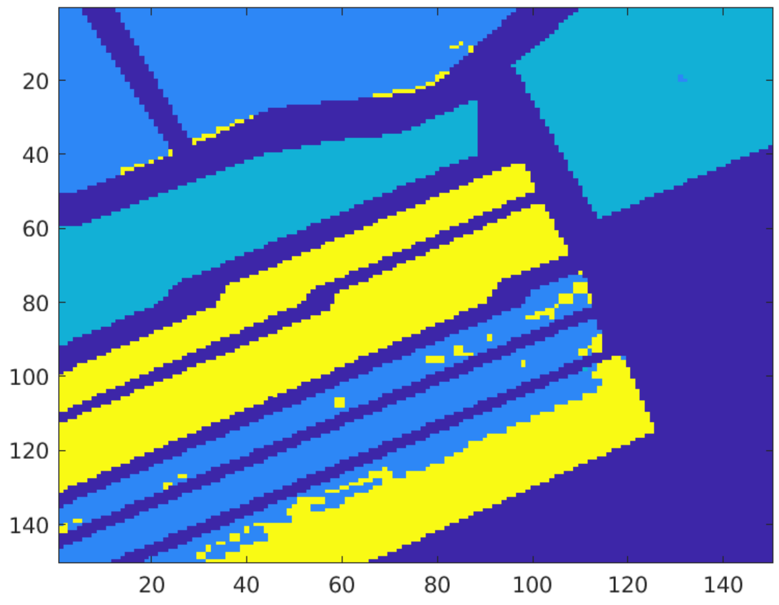

Fig. 20: k-Means clustering results (m=8, Squared Euclidean, PCA)

Next, we run the **Possibilistic c-Means** with **k-Means initialization** and with the use of the **Canberra** distance. Although the performance (RI = 68.53% - see Fig. 21) is lower than that of the implementation without PCA (RI = 77.87%), we do not observe the extremely noisy clusters that we had seen in the earlier experiments (see Fig. 8).

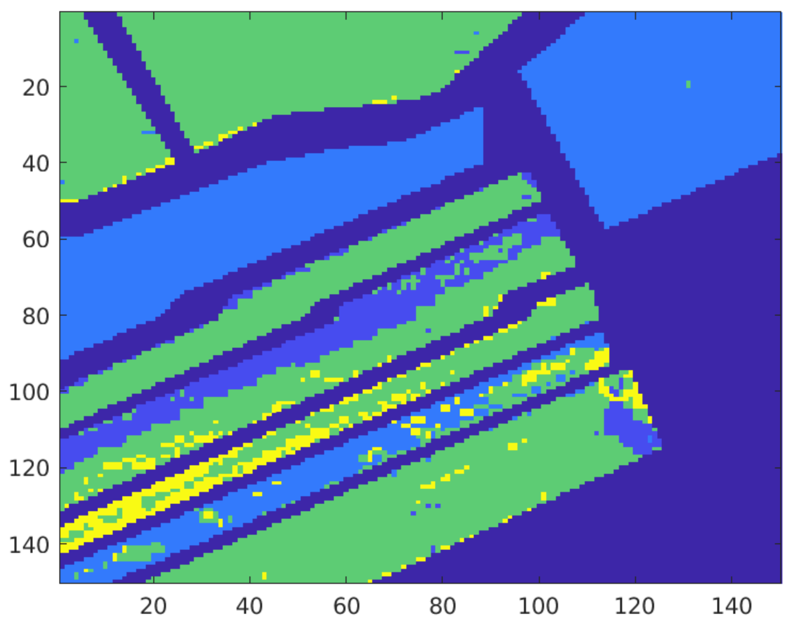

Fig. 21: Possibilistic c-Means clustering results (m=8, k-Means Initialization, Canberra, PCA)

Another configuration that we tried after reducing the dimensionality of our data with PCA, is the **Fuzzy c-Means** clustering, with **q = 2** and the **Squared Euclidean** distance. Its performance (RI = 91.49%) is higher than all those reported in Table 3 and as we can see in Fig. 22, some of the problems we had without PCA persist; the occasional noise within the clusters and the merging/division of ground truth classes. However, the algorithm now does not seem to face problems in border regions.

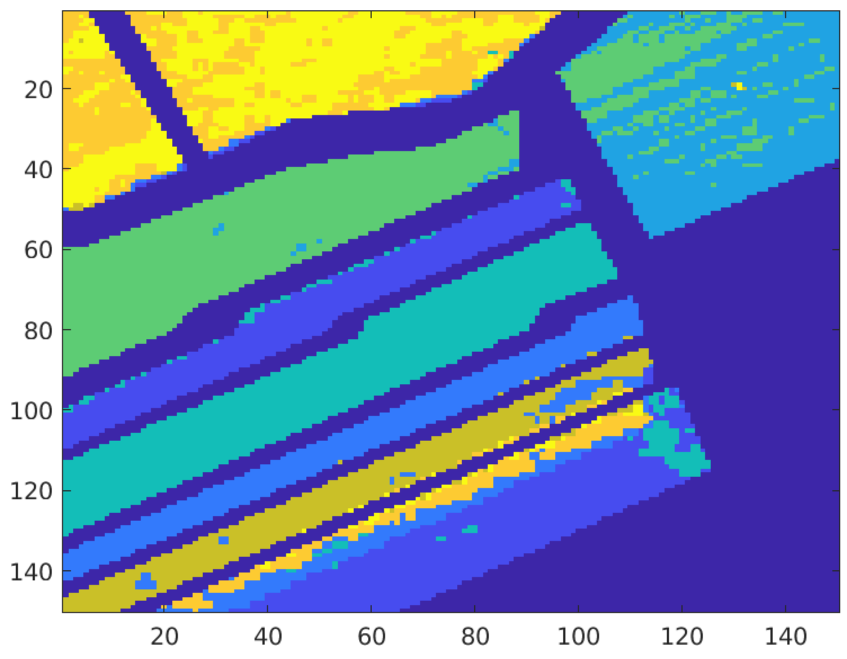

Fig. 22: Fuzzy c-Means clustering results (m=8, q=3, Squared Euclidean, PCA)

Finally, we use the **Probabilistic** (Gaussian Mixture) clustering algorithm with initialization of the **μ**'s with the **k-Means++** algorithm. Its performance is by far the best from all the configurations that we have experimented upon, a spectacular **RI=96.51%**. As we can see in Fig. 23, the problems associated with the -thick- border regions do not appear at all now (see Fig. 17 and Fig. 18 for contrast) and the algorithm has produced clusters which resemble closely the correct classes (see Fig. 2). The only problem that we can observe concerns the class at the bottom region of the image, which is divided in -mainly- 2 different clusters (as well as the the occasional noise).

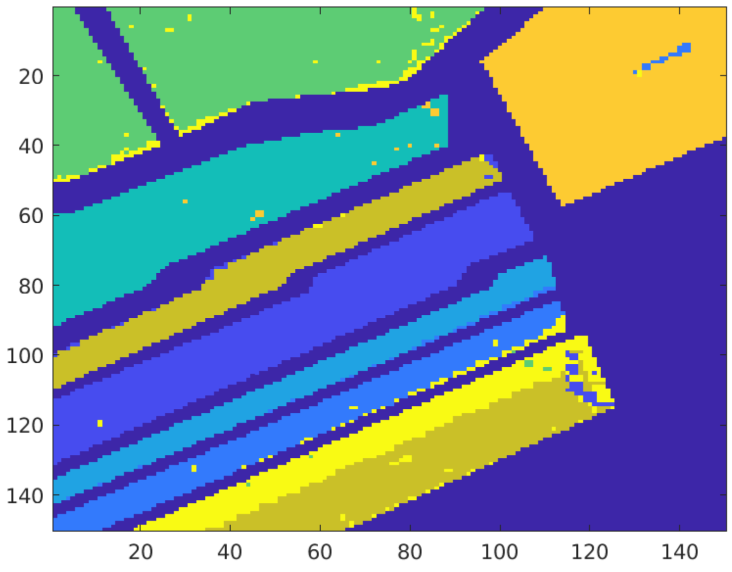

Fig. 23: Probabilistic clustering results (m=8, k-Means++ Initialization, PCA)

After conducting experiments with the use of PCA, we can observe that it enhances the performance of many algorithms, while giving poor results for some of the best configurations that we had before, when we preprocessed the data using normalization and filtering. However, we should emphasize the fact that **all the algorithms had a huge boost in terms of computational speed after utilizing PCA**. This of course should be quite obvious, as there are now 200 less dimensions for each data vector (x_i) to be processed.

While not part of the requirements of the project, we decided to conduct an additional experiment, by performing an **ensemble clustering**. In particular we use the best configurations of the Fuzzy c-Means and the k-Means algorithms. As we saw earlier, the k-Means algorithm using Canberra distance, gave us very poor results when we first use PCA on the dataset in order to reduce its dimensionality. Thus, we would like to see what would happen if we initialize **Θ** differently and in particular, using Fuzzy c-Means with q = 2 and the Squared Euclidean distance (**NOTE:** This is the only experiment in which we combine algorithms using two different distance metrics).

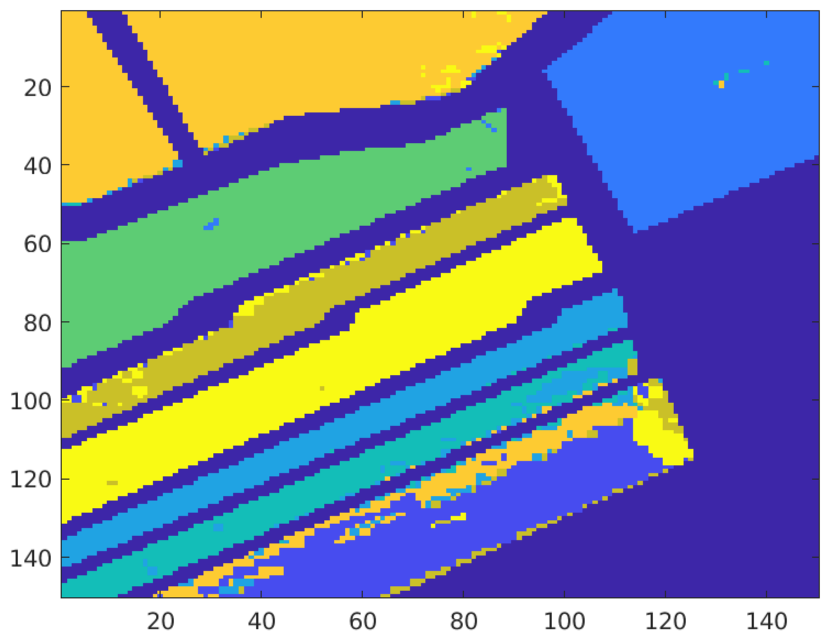

Fig. 24: Fuzzy c-Means (q=2, Squared Euclidean) and k-Means (Canberra) Ensemble results (m=8, PCA)

As we can see above in Fig. 24, an ensemble of Fuzzy c-Means (with q = 2 and Squared Euclidean distance) and k-Means (with Canberra distance) gives us a tremendous performance with **RI = 96.05%**, indicating that combining clustering algorithms -even with different distance metrics- can provide very good results. By comparing it with the results of the Probabilistic clustering (see Fig. 23), we can also observe that the clustering algorithms face many difficulties concerning the class at the bottom region of the picture; this is also the case for implementations where we did not use PCA.

(8) General Remarks

In this section, we discuss the results of the previous sections by comparing the algorithms that we have used and providing remarks on some of their issues and their possible causes:

- Best configurations for each algorithm based on Rand Index:

- K-Means: Using **m = 10** and the **Canberra** distance, the algorithm scores **87.36%**, although its score for **m = 8** is not that different (RI = 87.23%).

- Possibilistic c-Means: Using **m = 9**, initialization of **Θ** with **Fuzzy c-Means** (q = 2) and the **Squared Euclidean** distance metric, the algorithm scores **83.07%**, while its scores for **m = 7**, **m = 8** and **m = 10**, are very similar and range from 81.58% to 81.76%.

- Fuzzy c-Means: Using **m = 10**, **q = 2** and the **Squared Euclidean** distance, the algorithm scores **88.22%**. For m = 10 and q = 3, its score is similar (RI = 87.79%) and in general, when we use the Squared Euclidean distance, we observe a general upward trend for larger values of m.

- Probabilistic (Gaussian Mixture): Using **m = 10** and initialization of the **μ**'s with the **k-Means++** algorithm, the algorithm scores **91.58%**. If we use random initialization, the score is quite similar (RI = 91.49% for m = 10). As with the Fuzzy c-Means, both methods of initialization show a general upward trend for larger values of m.

- Preprocessing with Principal Component Analysis: For **m = 8** and **4 principal components**, the **Probabilistic** (Gaussian Mixture) clustering algorithm gives the **best score in the project**, i.e. 96.51% (when initialized with k-Means++).
A similarly good performance is given by an **ensemble** of Fuzzy c-Means (q = 2; Squared Euclidean distance) and k-Means (Canberra distance) which reaches **96.05%**.
Finally, the **Fuzzy c-Means** (with q = 2 and **Squared Euclidean** distance) has a very good score on its own, **91.49%**.

- Qualitative comparison of algorithms based on their clustering results:

The **worst algorithm** in terms of performance is the **Possibilistic c-Means** algorithm, as it is very sensitive to the method of initialization of Θ's and works poorly if it is initialized randomly. Perhaps, if we had implemented it in a different way, by setting fixed η's which are derived from the initialization algorithm -either k-Means or Fuzzy c-Means- we would have seen better results or at least not so much random noise within the clusters.

The **2nd place in terms of worst performance**, is shared by the **k-Means** algorithm and the **Fuzzy c-Means** algorithm. Both algorithms exhibit similar, good performance when we use *certain types of distance*. In particular, the **Canberra distance** works quite well for **k-Means** and the **Squared Euclidean distance** for **Fuzzy c-Means**, while both exhibit mediocre performance for some configurations (Fuzzy c-Means does not work very well with the Canberra distance). Although both algorithms face some difficulties with border regions, their most prevalent problem is the *combination of different classes into a single cluster* and the *division of a single class into two or three different clusters*. The data points in Fuzzy c-Means do not have absolute membership over a particular cluster, but a certain probability of belonging to one and thus, the algorithm may be better suited for datasets with clusters which overlap. Nevertheless, both algorithms, tend to produce compact, hyperspherical clusters. The k-Means algorithm has the additional benefit of being a lot faster in terms of computational speed.

The **best algorithm** in terms of performance is the **Probabilistic clustering algorithm** (Gaussian Mixture model) which -in contrast with the other methods- can detect **diversely sized clusters**. The main assumption, namely that the pixels in each class follow a Gaussian distribution, poses no insurmountable problem and is actually a weaker assumption than -for example- that of the k-Means algorithm, i.e. that the clusters are compact and hyperspherical.

- Comparison of algorithms based on their problems and their behavior:

As we mentioned earlier, the **Possibilistic c-Means** is the only algorithm which produces such noisy clusters and this could be attributed to the way we calculate **η**'s (updating them in each iteration instead of fixing them to certain values). However, it is also possible that this is a consequence of the algorithm working in a fundamentally different way than the others. Since it assigns the possibility -not probability- of a point belonging to a certain cluster, it has many commonalities with *valley-seeking methods* and would perform better if there were very clear "valleys" between clusters, i.e. if the classes had very clear boundaries.

Overall, all the algorithms seem to *have trouble discerning the borders* of the classes correctly and -quite frequently- end up **dividing a class into two** or **combining two distinct classes**. However, there is another, subtler issue, as we can see that many algorithms result in clusters with **borders belonging to a different class**, a problem which is more pronounced (thicker borders) in the implementation of the *Probabilistic clustering algorithm* (see Fig. 15, 16, 17, 18). In other types of datasets this could be attributed to intertwined clusters with borders which overlap. However, in the specific dataset in which we have different crops that are assumed to have a *distinct spectral pattern* and are also separated with zero-class labels (although we do not know what those missing labels actually correspond to), this problem should probably not emerge; or at least not have such a high intensity. Thus, it could also be attributed to the way that we preprocessed the data and more specifically, to the **2D Gaussian filter** which has the effect of *blurring the borders between classes*.

The reason why it is **more prevalent** in the Probabilistic clustering algorithm, is that the specific algorithm models the clusters as Gaussian distributions which can also overlap and assigns each point to the cluster it most likely belongs to. Thus, since the pixels in border regions have been smoothed in such a way that their intensity value gets closer to that of the pixels of their surrounding classes, the Gaussian distributions of the Probabilistic clustering algorithm **will overlap to a much greater degree**. Because of these, it would be a good idea not to combine Gaussian filtering with Probabilistic (Gaussian Mixture) clustering, although another possible solution would be to apply the filter **before** reshaping the original dataset X by removing the zero-class labels, as the changes would not apply to the border regions (the fact that their intensity values equal to zero could be solved with the use of symmetric or replicate padding).

Furthermore, we can observe (see Fig. 22, Fig. 23, Fig. 24) that preprocessing the data with the exclusive utilization of Principal Component Analysis (i.e. not normalizing or filtering the data), *makes the problem associated with the border regions disappear*; a **strong indicator** that the reasons we gave above about the causes of this problem are most likely correct. If this is truly the case, then it also enables us to observe that the best configurations of our algorithms have difficulties in classifying correctly a specific region of the image, i.e. the **bottom region**. Moreover, we would also be in a better position to explain the **irregular results** (noisy clusters) that the **Possibilistic c-Means** algorithm exhibits and attribute them to the effect of the 2D Gaussian filtering too, i.e. the blurring of the borders between classes; it creates even more difficulties -compared to the Probabilistic clustering algorithm- for this particular algorithm (which has certain commonalities with valley-seeking methods, as we explained earlier) and they are partly alleviated only with the initialization of **Θ** with Fuzzy c-Means probably because, in this case, the Fuzzy c-Means has already resulted in a good clustering and the Possibilistic c-Means does not particularly improve on it.

Nevertheless, preprocessing the data with **PCA**, did not provide us with good results either and thus, it may be the case that **Possibilistic c-Means** is not well-suited to the specific task of the project at all, as it is an algorithm which is **very sensitive** to the *shape of the classes*, the *method of initialization of Θ* and the *preprocessing of the data*; a **quite undesirable behavior** if we were to work with data for which we do not know the ground truth labels and could only rely on internal criteria. On the other hand, the **Probabilistic** clustering algorithm is characterized by consistency, robustness, better interpretability and performance and thus, would be a better alternative.

(9) Conclusion

In this project, we used 4 different clustering algorithms in order to detect the ground-truth labels of the pixels in a hyperspectral image, that of the Salinas valley in California, USA. In the **section 1**, we described how we preprocessed the dataset and in the **section 2**, we laid out the general framework with which we proceeded to conduct our experiments. In the **next 4 sections**, we reported the results of the 4 algorithms, for different hyper-parameters and briefly commented on the results. In **section 7**, we observed the effects of using a different preprocessing method (Principal Components Analysis) in order to reduce the dimensionality of the dataset. Finally, in **section 8**, we provided general observations about how the algorithms compare in terms of performance and what specific problems we have encountered.

We can conclude that in this particular case study, the Probabilistic (Gaussian Mixture) clustering algorithm had the best performance -and behavior- overall, while we can also get a very good performance from ensembles (which use different distance metrics). However, there is always the need to perform many experiments with different hyper-parameters, as there are some configurations which provide decent results and some others which work very poorly. Finally, we observed the impact of the preprocessing methods both on the computational speed and the overall performance of our algorithms, as both were enhanced when we made use of Principal Component Analysis. We also noticed how a careful preprocessing could solve recurring problems, such as the one associated with the border regions of the clusters, which was probably caused by the 2D Gaussian filtering and how each algorithm responds to this.

In future research aimed at detecting the homogeneous regions of hyperspectral images, we could utilize different methods of preprocessing found in the literature, such as Singular Value Decomposition (SVD), Spectral Unmixing (SU), Vector Quantization (VQ), Discrete Fourier Transform (DFT), etc. We could also utilize different types of distance metrics, such as the Spectral Angle Measure (SAM) or the Spectral

Correlation Coefficient (SCC), or compare the results of additional types of algorithms, such as graph-based methods (Spectral clustering), hierarchical methods (Agglomerative clustering), density-based methods (DBSCAN), etc. For a more in-depth research project, we could also make use of more ensemble clustering schemes and produce a final consensus clustering with the use of a co-association matrix and afterwards, perform various hypothesis tests (with the utilization of Monte Carlo simulations) in order to ascertain the validity of our clustering results.

REFERENCES

[1] **Ertürk, A.; Çeşmeci, D.; Güllü, M. K.; Gerçek, D. and Ertürk, S.** (**2014**). *Endmember extraction guided by anomalies and homogeneous regions for hyperspectral images*, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 7 : 3630-3639.

[2] **Bilgin, G.; Erturk, S. and Yildirim, T.** (**2008**). *Unsupervised classification of hyperspectral-image data using fuzzy approaches that spatially exploit membership relations*, IEEE Geoscience and Remote Sensing Letters 5 : 673-677.

[3] **Wang, K.; Gu, X.; Yu, T.; Meng, Q.; Zhao, L. and Feng, L.** (**2013**). *Classification of hyperspectral remote sensing images using frequency spectrum similarity*, Science china technological Sciences 56 : 980-988.

[4] **Krishnapuram, R. and Keller, J. M.** (**1996**). *The possibilistic c-means algorithm: insights and recommendations*, IEEE transactions on Fuzzy Systems 4 : 385-393.

[5] **Arthur, D. and Vassilvitskii, S.** (**2006**). *k-means++: The advantages of careful seeding*. Stanford.

Code Appendix

- Code for the main experiments (roussis_project.m):

```
clc;
format compact;
clear;
close all;
%rand('seed',1);
rng('default');
rng(2);


%% Loading and Preprocessing %%

load Salinas_Data.mat;
[p1,n1,l]=size(Salinas_Image);
X=reshape(Salinas_Image,p1*n1,l);
X=X';
[p2,n2]=size(Salinas_Labels);
y=reshape(Salinas_Labels,p2*n2,1);
y=y';


%% Remove zero class labels %%

zero_idx=find(~y);
nonzero_idx=find(y);
X(:,zero_idx)=[];
y(:,zero_idx)=[];
[l,N]=size(X);


%% Visualizations %%

im=Salinas_Labels;
figure(1), imagesc(im);
title('Salinas Labels');
hold off

vol=Salinas_Image;
figure(2),volshow(vol,'Colormap',colormap);
hold off


%% NOTE: Use only one of the two preprocessing methods below %%

%% Filtering and Normalization %%

X=imgaussfilt(X,3.5,'FilterSize',9);
X=normalize(X);
```

```
%% Dimensionality Reduction with PCA %%

[eigenval,eigenvec,explain,Y,mean_vec]=pca_fun(X,4);
X=Y;
[l,N]=size(X);


%% k-Means Algorithm %%

rand_k_means_all=[]
jaccard_k_means_all=[]

for m=3:10

   costs=[];
   bels=[];

   for i=1:10
      theta_init=rand(l,m);
      [theta,bel,J]=k_means(X,theta_init);
      bels=[bels;bel];
      costs=[costs;J];
   end

   figure(3),plot(costs)
   [cost_m,bel_index]=min(costs)
   bel=bels(bel_index,:);

   cl_label=bel';
   cl_label_tot=zeros(p1*n1,1);
   cl_label_tot(nonzero_idx)=cl_label;
   im_cl_label=reshape(cl_label_tot,p1,n1);
   figure(4), imagesc(im_cl_label);

   [rand_k_means,jaccard_k_means]=metrics(bel,y)

   rand_k_means_all=[rand_k_means_all;
                  rand_k_means]
   jaccard_k_means_all=[jaccard_k_means_all;
                  jaccard_k_means]

end


%% Possibilistic c-Means Algorithm %%

rand_possibilistic_all=[]
jaccard_possibilistic_all=[]

for m=3:10

   costs=[];
   bels=[];

   for i=1:10
      theta_init=rand(l,m);
      theta_init=k_means(X,theta_init);
      theta_init=fuzzy_c_means(X,theta_init,2);
```

```
        [theta,bel,J]=possibilistic_c_means(X,theta_init);
        bels=[bels;bel];
        costs=[costs;J];
    end

    figure(5),plot(costs)
    [cost_m,bel_index]=min(costs)
    bel=bels(bel_index,:);

    cl_label=bel';
    cl_label_tot=zeros(p1*n1,1);
    cl_label_tot(nonzero_idx)=cl_label;
    im_cl_label=reshape(cl_label_tot,p1,n1);
    figure(6), imagesc(im_cl_label);

    [rand_possibilistic,jaccard_possibilistic]=metrics(bel,y)

    rand_possibilistic_all=[rand_possibilistic_all;
                            rand_possibilistic]
    jaccard_possibilistic_all=[jaccard_possibilistic_all;
                               jaccard_possibilistic]

end


%% Fuzzy c-Means Algorithm %%

rand_fuzzy_all=[]
jaccard_fuzzy_all=[]

for m=3:10

    costs=[];
    bels=[];

    for i=1:10
        theta_init=rand(l,m);
        [theta,bel,J]=fuzzy_c_means(X,theta_init,2);
        bels=[bels;bel];
        costs=[costs;J];
    end

    figure(7),plot(costs)
    [cost_m,bel_index]=min(costs)
    bel=bels(bel_index,:);

    cl_label=bel';
    cl_label_tot=zeros(p1*n1,1);
    cl_label_tot(nonzero_idx)=cl_label;
    im_cl_label=reshape(cl_label_tot,p1,n1);
    figure(8), imagesc(im_cl_label);

    [rand_fuzzy,jaccard_fuzzy]=metrics(bel,y)

    rand_fuzzy_all=[rand_fuzzy_all;
                    rand_fuzzy]
    jaccard_fuzzy_all=[jaccard_fuzzy_all;
                       jaccard_fuzzy]
```

```
end


%% Probabilistic c-Means Algorithm %%

rand_probabilistic_all=[];
jaccard_probabilistic_all=[];

for m=3:10

    options=statset('Display','final','MaxIter',500,'TolFun',5e-7);
    gmfit=fitgmdist(X',m,'Start','plus','CovarianceType', 'full', ...
                'Options',options,'RegularizationValue',1e-4, ...
                'Replicates',10);
    bel=cluster(gmfit,X');

    [rand_probabilistic,jaccard_probabilistic]=metrics(bel',y)

    rand_probabilistic_all=[rand_probabilistic_all;
                            rand_probabilistic]
    jaccard_probabilistic_all=[jaccard_probabilistic_all;
                               jaccard_probabilistic];

    cl_label=bel;
    cl_label_tot=zeros(p1*n1,1);
    cl_label_tot(nonzero_idx)=cl_label;
    im_cl_label=reshape(cl_label_tot,p1,n1);
    figure(9), imagesc(im_cl_label);

end


%% Visualization of the 1st component of PCA %%

[eigenval,eigenvec,explain,Y,mean_vec]=pca_fun(X,1);
Y_tot=zeros(p1*n1,1);
Y_tot(nonzero_idx)=Y(1,:);
Y_reshaped=reshape(Y_tot(:,1),p1,n1);
figure(10), imagesc(Y_reshaped);


%% Ensemble Fuzzy c-Means -> k-Means %%

rand_fuzzy_k_means_all=[]

for m=3:10

    costs=[];
    bels=[];
    for i=1:10
        theta_init=rand(l,m);
        [theta,bel,J]=fuzzy_c_means(X,theta_init,2);
        [theta,bel,J]=k_means(X,theta);
        bels=[bels;bel];
        costs=[costs;J];
    end

    figure(11),plot(costs)
    [cost_m,bel_index]=min(costs)
    bel=bels(bel_index,:);

    [rand_fuzzy_k_means]=metrics(bel,y)

    rand_fuzzy_k_means_all=[rand_fuzzy_k_means_all;
                            rand_fuzzy_k_means]

    cl_label=bel';
    cl_label_tot=zeros(p1*n1,1);
    cl_label_tot(nonzero_idx)=cl_label;
    im_cl_label=reshape(cl_label_tot,p1,n1);
    figure(12), imagesc(im_cl_label);

end
```

- Code for the k-Means algorithm (k_means.m):

```
function [theta,bel,J]=k_means(X,theta)


[l,N]=size(X);
[l,m]=size(theta);
e=1;
iter=0;
e_thres=0.001;
max_iter=100;
while(e>e_thres && iter<max_iter)
    iter=iter+1;
    theta_old=theta;
    dist_all=[];
    for j=1:m
        p=ones(N,1)*theta(:,j)';
        q=X';

        % Squared Euclidean Distance
        %dist=sum(((p-q).^2)');

        % Canberra Distance
        dist=sum((abs(p-q)./(abs(p)+abs(q)))');

        dist_all=[dist_all; dist];
    end

    [q1,bel]=min(dist_all);
    J=sum(min(dist_all));

    for j=1:m
        if(sum(bel==j)~=0)
            theta(:,j)=sum(X'.*((bel==j)'*ones(1,l))) / sum(bel==j);
        end
    end
    e=sum(sum(abs(theta-theta_old)));
end
```

- Code for the Possibilistic c-Means algorithm (possibilistic_c_means.m):

```
function [theta,bel,J]=possibilistic_c_means(X,theta)

[l,N]=size(X);
[l,m]=size(theta);
eta=ones(m,1);
e=1;
iter=0;
e_thres=0.001;
max_iter=100;
U=zeros(N,m);
dist=zeros(N,m);
while(e>e_thres && iter<max_iter)
    iter=iter+1;
    for i=1:N
        for j=1:m
            p=X(:,i);
            q=theta(:,j);

            % Squared Euclidean Distance
            %dist(i,j)=norm(p-q)^2;

            % Canberra Distance
            dist(i,j)=sum((abs(p-q)./(abs(p)+abs(q))));

            U(i,j)=exp(-dist(i,j)/eta(j));
        end
    end

    for i=1:N
        [q1,bel]=max(U');
    end

    theta_old=theta;
    theta=zeros(l,m);
    J=0;

    for j=1:m
        temp=0;
        for i=1:N
            theta(:,j)=theta(:,j)+U(i,j)*X(:,i);
            eta(j)=eta(j)+U(i,j)*dist(i,j);
            temp=temp+U(i,j);
        end
        theta(:,j)=theta(:,j)/temp;
        J=J+eta(j)+(eta(j)/temp)*(temp*log(temp)-temp);
        eta(j)=eta(j)/temp;
    end

    e=sum(sum(abs(theta-theta_old)));

end
```

- Code for the Fuzzy c-Means algorithm (fuzzy_c_means.m):

```
function [theta,bel,J]=fuzzy_c_means(X,theta,fuzz)

[l,N]=size(X);
[l,m]=size(theta);
e=1;
iter=0;
e_thres=0.001;
max_iter=100;
U=zeros(N,m);
dist=zeros(N,m);
while(e>e_thres && iter<max_iter)
    iter=iter+1;
    for i=1:N
        for j=1:m
            temp=0;
            p=X(:,i);
            q=theta(:,j);

            % Squared Euclidean Distance
            dist_j(i,j)=norm(p-q)^2;

            % Canberra Distance
            %dist_j(i,j)=sum((abs(p-q)./(abs(p)+abs(q))));

            for k=1:m
                r=theta(:,k);

                %Squared Euclidean Distance
                dist_k(i,k)=norm(p-r)^2;

                %Canberra Distance
                %dist_k(i,k)=sum((abs(p-r)./(abs(p)+abs(r))));

                if dist_j(i,j)~=0 && dist_k(i,k)~=0
                    temp=temp+(dist_j(i,j)/dist_k(i,k));
                end
            end
            U(i,j)=1/(temp^(1/(fuzz-1)));
        end
    end

    for i=1:N
        [q1,bel]=max(U');
    end

    theta_old=theta;
    theta=zeros(l,m);
    J=0;

    for j=1:m
        temp=0;
        for i=1:N
            theta(:,j)=theta(:,j)+U(i,j)^fuzz*X(:,i);
            temp=temp+U(i,j)^fuzz;
        end
        theta(:,j)=theta(:,j)/temp;
        J=J+temp*sum(dist_j(:,j));
    end

    e=sum(sum(abs(theta-theta_old)));

end
```

- Code for the Principal Component Analysis (pca_fun.m):

```
function [eigenval,eigenvec,explain,Y,mean_vec]=pca_fun(X,m)

[l,N]=size(X);

% Subtracting the mean
mean_vec=mean(X')';
X_zero=X-mean_vec*ones(1,N);

% Computing the covariance matrix and its eigenvalues/eigenvectors
R=cov(X_zero');
[V,D]=eig(R);

eigenval=diag(D); %diatasume analoga me tin idiotimi, prwta t megisto klp klp
[eigenval,ind]=sort(eigenval,1,'descend');
eigenvec=V(:,ind);

explain=eigenval/sum(eigenval); % Î»1/(Î»1+Î»2) ÎºÎ±Î¹ Î³Î¹Î± Ï„Î· 2Î· Î»2/(Î»1+Î»2)
% Keeping the first m eigenvaules/eigenvectors
eigenval=eigenval(1:m);  %diatirisis megaluteru variance stis m diastaseis
eigenvec=eigenvec(:,1:m);

% Computing the transformation matrix
A=eigenvec(:,1:m)';

% Computing the transformed data set
Y=A*X;
```