INF264 - Homework 6

Pierre Gillot   Natacha Galmiche[*]

Week 40 - 2021

1 Hierarchical clustering

In this first exercise, we will use hierarchical clustering in order to visualize clusters hierarchy produced on a small dataset representing the 3D coordinates of 30 international cities. We will see how different linkages can affect clusters hierarchy. Visualization will be achieved via dendrograms, which are trees showing the order and distances of clusters merges during hierarchical clustering. Answer the following questions. **Note that this is advised that you help yourself with the provided template**.

1. Load the dataset contained in the file 'cities_coordinates.txt'. Store the features (the xyz coordinates of the cities) in a matrix X and the names of the cities in a vector y.

2. Use the 'scipy.cluster.hierarchy.linkage' method to perform hierarchical clustering on X, for different types of linkage: you will try the 'ward', 'average', 'complete' and 'single' linkage types.

3. Use the 'scipy.cluster.hierarchy.dendrogram' method to display dendrogram representations of your different hierarchical clusterings.

4. Set 'color_threshold=50' and 'above_threshold_color='grey'' in the dendrogram method. This will force every edge in the dendrogram beyond distance 50 to have the same grey color. Other colors then represent your different clusters.

5. For each linkage type, display a 3D scatterplot of the cities' positions, where each city has its 3D point colored like its cluster in the dendrogram. From the observation of your 3D scatterplots, which linkage type seems to be most/least realistic ?

[*]Not a TA for this course this year.

2 An application of clustering to image segmentation

In this second exercise, we will apply a clustering algorithm to solve the problem of image segmentation, which makes for an intuitive and visually satisfying real-world application of unsupervised machine learning. Image segmentation aims at partitioning an image into sets of similar pixels, in order to obtain a meaningful simplified representation of the image.

In the context of this work, we consider an image represented by a three-dimentional matrix in $\mathbb{R}^{m \times n \times d}$, where m is the number of rows, n is the number of columns and d is the number of channels (1 for grayscale images, 3 for RGB images). This means that an image has $m \times n$ pixels, where each pixel is a vector in $\mathbb{R}^d$.

An easy and straightforward way to measure the similarity between two pixels p_1 and p_2 is simply to compute their Euclidian distance in $\mathbb{R}^d$, that is $\|p_1 - p_2\|^2_{L_2}$. We thus can reduce the image segmentation problem to K-means: all we have to do is reshaping the image matrix into a two-dimentional matrix in $\mathbb{R}^{(m \cdot n) \times d}$, meaning the image is considered as a dataset and each pixel in the image is seen as an observation in $\mathbb{R}^d$.

We will consider the following image represented by a matrix in $\mathbb{R}^{184 \times 233 \times 3}$:

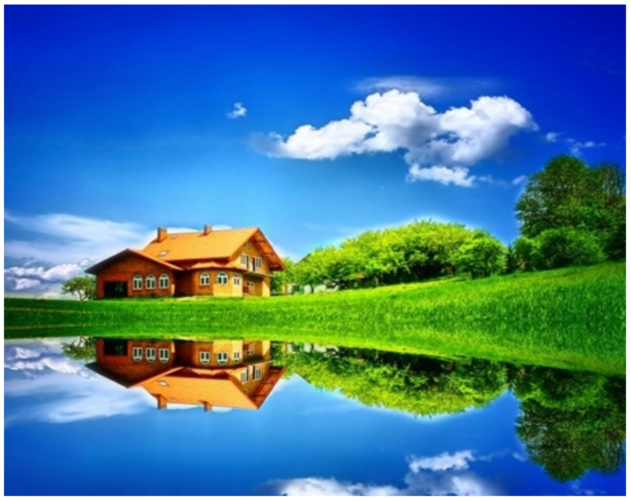

Figure 1: Scenery.jpg

In order to do image segmentation on this image, answer the questions below:

1. The Scenery.jpg image was vectorized into a $\mathbb{R}^{(184 \cdot 233) \times 3}$ matrix, then converted into a .txt file where each line gives the RGB values of a pixel.

Load the dataset contained in the file 'scenery_184_233.txt' in a matrix X.

2. Make a copy of X (this is only for visualization), reshape this copy into a $\mathbb{R}^{184\times233\times3}$ matrix, then convert the obtained matrix to the uint8 format using 'numpy.uint8' and finally display the image using 'matplotlib.pyplot.imshow'.

3. Perform clustering on X (not the copy), for different values of k. You are free to use your implementation or sklearn's which is accessible via the class 'sklearn.cluster.KMeans'. Indicate which hyper-parameters you chose if any.

4. Plot the elbow graph. Which value of k would you choose ?

5. Perform once more clustering on X with the value of k that you selected in the previous question. Also return a 1-dimensional predictions array containing the clusters' index for every pixel. If you are using sklearn's implementation of K-means, you can do all that at the same time using the 'fit_predict' method.

6. Reshape the predictions array from shape $(184 \cdot 233,)$ to shape $(184, 233)$. This corresponds to a predictions mask which partitions the Scenery.jpg image. Display this predictions mask (you can for instance use the 'seaborn.heatmap' method for nice visualization).

7. Separate the predictions mask into k sub-masks (with the same shape), where every pixel in the j-th sub-mask has value 1 if the same pixel is equal to j in the predictions mask, and 0 otherwise. Multiply the $\mathbb{R}^{184\times233\times3}$ matrix representing the Scenery.jpg image by those sub-masks and display the corresponding masked images.