INF 264 - Exercice 7

Instructions

- Deadline: 25/10/2019, 23.59
- Submission place: `https://mitt.uib.no/courses/19532/assignments`
- Format: Your answers are to be returned in a single pdf report. You can also return scanned pages for your calculations.

The present document is the seventh of a series of eight weekly exercises. The deadline for weekly exercises is on Friday evening. You will be awarded points for solving weekly exercises. Each week you can earn at most 1.5 raw points (depending on how many tasks you solve). Your solutions do not need to be perfect; we award points for reasonable effort. Weekly exercises give at most 10 points towards the final grade; if you get more than 10 raw points, it will count as 10 points. To pass the course, you will need to get at least 5/10 points from the weekly exercises. Note that you do not have to get points every week, just at least 5 points in total.

In this exercise we will introduce clustering. In a first section, we will write our own implementation of Lloyd's algorithm, a simple heuristic for K-means, and apply it on a small dataset. Then in a second section, we will use hierarchical clustering on a dataset representing the 3D coordinates of international cities and visualize the clusters using dendrograms. Finally, in a bonus section we will attempt to perform image segmentation using clustering.

1 Lloyd's algorithm

In this first section, we will implement the famous Lloyd's algorithm, a well known heuristic solving K-means. The purpose of the algorithm is to partition a set of observations into K clusters, where an observation should belong to a cluster if this cluster is the closest from the observation. To evaluate how far an observation is from a cluster, the distance between the observation and the centroid of the cluster is measured using a Euclidian norm.

It is known that this partitioning problem is difficult (in fact it is NP-hard even with only 2 clusters and even in the plane), but there exist heuristics that converge quickly to a local minimum. Lloyd's algorithm is one such heuristic which, while a bit "naive", has been widely used and successfully modified into efficient algorithms achieving state-of-the-art performance in clustering problems.

The pseudo-code for Lloyd's algorithm is as follows:

Lloyd's algorithm

Input:

- A dataset $\mathcal{D}$ represented by a matrix X of size $n \times d$, consisting of n observations in $\mathbb{R}^d$
- An integer $k \geq 1$, representing the number of clusters
- An integer $r \geq 1$, representing the number of restarts
- A float $p \geq 0$, determining the precision of the optimization (defaults to 0)

Output:

- A partition of $\mathcal{D}$ into k clusters $\mathcal{D}_1, \ldots, \mathcal{D}_k$ represented by their centroids $\mu_1, \ldots, \mu_k$, where an observation $x_i = X[i,:]$ belongs to the cluster $\mathcal{D}_l$ if μ_l is the closest centroid to x_i
- The cost of the partition: $\frac{1}{n}\sum_{j=1}^{k}\sum_{x \in \mathcal{D}_j} \|\mu_j - x\|_{L_2}^2$

1. **For** restart $= 1 : r$
   2. Initialize k clusters $\mathcal{D}_1, \ldots, \mathcal{D}_k$, i.e. create k centroids $\mu_1, \ldots, \mu_k$ and set them to k distinct observations in X picked randomly
   3. **While** $|\text{previous\_cost} - \text{new\_cost}| > p$
      4. Find the new partition, i.e for each observation $x_i = X[i,:]$ find the cluster $\mathcal{D}_l$ whose centroid μ_l is the closest to x_i, that is such that $l = \underset{j}{\text{argmin}} \|\mu_j - x_i\|_{L_2}^2$, then place x_i inside $\mathcal{D}_l$
      5. Compute the cost of the new partition: $\frac{1}{n}\sum_{j=1}^{k}\sum_{x \in \mathcal{D}_j} \|\mu_j - x\|_{L_2}^2$
      6. If $|\text{previous\_cost} - \text{new\_cost}| > p$ update the clusters, i.e. for each cluster $\mathcal{D}_j$, update its centroid μ_j to become the mean of the observations assigned to this cluster, that is $\mu_j \leftarrow \underset{i}{\text{mean}}\{x_i \in \mathcal{D}_j\}$

      End

   7. If the cost is smaller than all the costs obtained in previous restarts, store the clusters and the cost

   End

8. Return the stored clusters and cost

Once you have carefully read this pseudo-code, do the following:

1. Implement Lloyd's algorithm. **You can follow exactly the above pseudo-code, but this is not mandatory. You will find many other pseudo-codes on the internet**.
2. Load the Iris dataset using the 'load_iris' function from sklearn. You can alternatively download the dataset here: `https://archive.ics.uci.edu/ml/datasets/iris`. Store the three features in a matrix X. Discard the target labels: we are doing unsupervised machine learning thus we don't need them.
3. Perform clustering on X, for different values of k. Note that for this unsupervised learning task, we do not split X into train/validation/test sets, nor do we do cross-validation. Moreover, we do not use the target labels y during clustering.
4. Plot the "elbow graph", that is the curve of the cost as a function of the number of clusters k. You can set the number of restarts to 5 and the precision to 0.
5. The obtained curve should have an elbow shape. A common heuristic is to select the value of k at the angle of the elbow. Based on your curve, which value of k would you select ? Is this value of k close to the number of distinct classes in the target labels y ?
6. Perform once more clustering on X with $k = 3$ (i.e. create as many clusters as there are classes in the Iris dataset). How well does the obtained partition $\mathcal{D}_1, \ldots, \mathcal{D}_3$ matches the target labels y ?

2 Hierarchical clustering

In this second section, 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. Install the scipy library for python.
2. Download the dataset 'cities_coordinates.txt' following this link: `https://mitt.uib.no/courses/19532/assignments/22280`. Store the features (the xyz coordinates of the cities) in a matrix X and the names of the cities in a vector y.
3. 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.
4. Use the 'scipy.cluster.hierarchy.dendrogram' method to display dendrogram representations of your different hierarchical clusterings.
5. 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.
6. 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 ?

3 Bonus: An application of clustering to image segmentation

In this third and final section, 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. **Disclaimer: this section is not mandatory, you can skip it !**

In the context of this work, we consider an image represented by a tridimentional 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 bidimentional 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. **Image processing is not trivial and takes some practice, so do not hesitate to follow the template provided in order to save time, and to ask for help whenever you're stuck**.

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. Download the dataset 'scenery_184_233.txt' following this link: `https://mitt.uib.no/courses/19532/assignments/22280`.

2. Store the dataset into a matrix X. Make a copy of X (this is only for visualization), reshape this copy into a $\mathbb{R}^{184\times 233\times 3}$ 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 masks (with the same shape), where every pixel in the j-th mask has value 1 if the same pixel is equal to j in the predictions mask, and 0 otherwise. Multiply the $\mathbb{R}^{184\times 233\times 3}$ matrix representing the Scenery.jpg image by those masks and display the corresponding masked images.