LELEC2870: Exercise Session 1
Vector Quantization & Clustering

1 Objectives

In this first exercise session, you will implement several vector quantization algorithms: competitive learning, frequency sensitive learning and neural gas. Here, quantization consists in reducing the size of a dataset while minimizing the loss of information, by summarising it with a set of centroids. An archive containing all necessary data and and a python script for visualization are on the Moodle website.

2 Vector Quantization

Consider a set of P vectors $\{\mathbf{x}_p | p = 1...P\}$ in a D-dimensional space. A vector quantization consists of a set of Q D-dimensional vectors $\{\mathbf{y}_q | q = 1...Q\}$ minimizing an error criterion linked to the loss of information. Here, Q is typically chosen to be much smaller than P, since the goal of vector quantization is to summarise a large number of instances using only a few vectors. These vectors are called *centroids* and the set of the centroids forms the *codebook*.

The error criterion is generally defined in terms of the distance between the data vectors and the centroids. Usually, it corresponds to the mean squared Euclidean distance between the data points and the centroids

$$\mathbf{E} = \frac{1}{P} \sum_{p=1}^{P} \|\mathbf{x}_p - \mathbf{y}_{q^*}\|^2 \tag{1}$$

where $\mathbf{y}_{q^*}$ is the closest centroid to the vector $\mathbf{x}_p$. During this exercise session, it may be interesting to **monitor** how this objective function changes over iterations and to compare quantization results in terms of this quantity.

Start this session by **plotting** the first dataset. You will see that groups of points are clearly visible, and your objective is now to learn these groups automatically.

3 Competitive Learning

Competitive learning is a *winner-take-all* algorithm, which means that at each presentation of a data vector $\mathbf{x}_p$, only the winner centroid is adapted. The

centroid $\mathbf{y}_q$ is selected as a winner $\mathbf{y}_{q^*}$ if it is the closest to $\mathbf{x}_p$, with respect to the distance measure d. In other words,

$$\mathbf{y}_{q^*} \mid \forall r \in \{1..Q\} : \; d(\mathbf{x}_p, \mathbf{y}_{q^*}) \leq d(\mathbf{x}_p, \mathbf{y}_r). \tag{2}$$

The *adaptation rule* of the codebook is defined as

$$\mathbf{y}_{q^*} \leftarrow \mathbf{y}_{q^*} + \alpha_k (\mathbf{x}_p - \mathbf{y}_{q^*}) \tag{3}$$

where $\alpha_k > 0$ is a factor that controls the speed of the adaptation. In order to obtain convergence, the α_k factor has to decrease over time to end its run close to zero. Time is often described in number of *epochs* when using iterative algorithms. One *epoch* is one *pass* over every observation in the database.
A common method to adapt α after every epoch, is the hyperbolic decrease:

$$\alpha_{k+1} \leftarrow \frac{\alpha_k \beta}{\alpha_k + \beta} \tag{4}$$

where β is a constant. Generally, it is better to **randomize** the order of the observations in the database after each epoch.

Implement competitive learning with 2 codebook initialisations:

- Select the centroids randomly (tip: remain inside the borders of the data)
- Sample the centroids randomly from the observations (tip: avoid using the same point more than once, what do you think would happen?)

Visualize the vector quantizations using the *show_quantization* function provided in the archive. What is the problem with the first initialisation ? Does it occur with the second one ?

4 Frequency Sensitive Learning

Frequency sensitive learning (FSL) is also a *winner-take-all* algorithm. However, by introducing a penalisation on the centroids that often win, the problem of lost units is avoided. One centroid is *lost* when it is never chosen as the winner.

The selection rule for the winner centroid is replaced by

$$\mathbf{y}_{q^*} \mid \forall r \in \{1..Q\} : \; u_{q^*} d(\mathbf{x}_p, \mathbf{y}_{q^*}) \leq u_r d(\mathbf{x}_p, \mathbf{y}_r) \tag{5}$$

where u_q is related to the frequency of selection of $\mathbf{y}_q$. This factor u_q is initially equal to 1 and is incremented each time that the centroid $\mathbf{y}_q$ is selected as the winner. The adaptation rule remains

$$\mathbf{y}_{q^*} \leftarrow \mathbf{y}_{q^*} + \alpha_k (\mathbf{x}_p - \mathbf{y}_{q^*}) \tag{6}$$

Implement FSL. **Compare** the two types of codebook initialisation again. **Check** that the lost centroids problem is solved. Why is it so ?

5 Neural Gas

Unlike the other algorithms, neural gas is a *winner-take-most* algorithm, which means that all the centroids are adapted at each presentation of a vector $\mathbf{x}_p$. The adaptation depends of the distance between the centroids and $\mathbf{x}_p$.

The first step of the algorithm consists in ranking the centroids in function of their distance (euclidean or other) to $\mathbf{x}_p$. Thus, one can define the neighbouring function h such that, if $\mathbf{y}_q$ is the c_{th} closest centroid to $\mathbf{x}_p$, then

$$h\left(\mathbf{x}_p, \mathbf{y}_q\right) = c - 1. \tag{7}$$

For example, this quantity becomes zero for the closest centroid ($c = 1$).

The second step consists in adapting each centroid using the adaptation rule

$$\forall q \in \{1..Q\} : \ \mathbf{y}_q \leftarrow \mathbf{y}_q + \alpha_k \exp\left(-\frac{h\left(\mathbf{x}_p, \mathbf{y}_q\right)}{\lambda}\right)\left(\mathbf{x}_p - \mathbf{y}_q\right) \tag{8}$$

where λ regulates the amount of neighbours which are affected by the update. It can decrease with the number of epochs.

Implement the neural gas algorithm. **Explain** the influence of the λ parameter. What happens when $\lambda = 0$? What happens when $\lambda \to +\infty$?

Compare a *winner-take-most* approach to a *winner-take-all* approach ? What are the advantages of the former?

6 K-means

Finally we will address K-means, an algorithm with the same properties as Lloyd's (see lectures). It processes the P vectors all at once at *each* iteration, contrarely to the 3 previously seen quantization algorithms that iterate over the input vectors individually.

The algorithm goes as follows:

1. Select Q items of the dataset as the initial centroids.
2. For each item in the dataset, find the nearest centroid (i.e. assign a label to each item) by using the Euclidean Distance.
3. Replace each centroid with the center of mass of the group labelled to it (the mean coordinates of all items assigned to this centroid).

Repeat steps 2 and 3 until the centroids converge (i.e. they barely move between iterations) or a maximum number of iterations is reached.
Implement K-means. What is the problem with the first initialisation (run your program several times to discover it)? How does the K-means++ algorithm address this problem and why does it enhance the results?

7 Conclusion

How would you choose a method among the previous ones? Think about some criteria that would allow to evaluate them.