T-61.231 Principles of Pattern Recognition

Answers to exercise 9: 26.11.2001

1. The two Matrix Updating Algorithmic Scheme algorithms, the single and complete link algorithm, are presented in *Theodoridis, p 408-409*. In brief, the algorithms are based on updating the proximity matrix P in a way that:

   1. find the matrix P_t values C_i, C_j such that $d(C_i, C_j) = \min_{r,s=1,\dots,N, r \neq s} d(C_r, C_s)$
   2. merge C_i, C_j into a cluster and form the new P_{t+1}. This is done by deleting the two rows and columns that correspond to the merged clusters and adding a new row and a new column that contain the distances between the newly formed cluster and the old clusters according to the selected distance measure $d(\cdots)$.
   3. repeat if more rows (or columns) still exist

   The distance functions are

   $$\begin{array}{ll} d(C_q, C_s) = \min\{d(C_i, C_s), d(C_j, C_s)\} & \text{(single link algorithm)} \\ d(C_q, C_s) = \max\{d(C_i, C_s), d(C_j, C_s)\} & \text{(complete link algorithm)} \end{array}$$

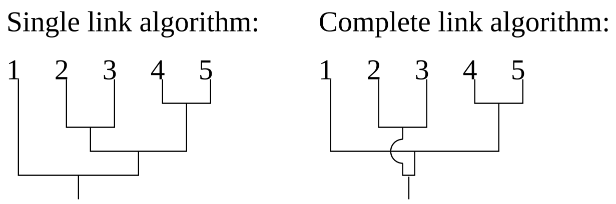
   Thus, for the single link algorithm we have

   $$\begin{bmatrix} 0 & 4 & 9 & 6 & 5 \\ 4 & 0 & 1 & 8 & 7 \\ 9 & 1 & 0 & 3 & 2 \\ 6 & 8 & 3 & 0 & 1 \\ 5 & 7 & 2 & 1 & 0 \end{bmatrix} \begin{array}{c} \text{join 4,5} \\ \rightarrow \end{array} \begin{bmatrix} 0 & 4 & 9 & 5 \\ 4 & 0 & 1 & 7 \\ 9 & 1 & 0 & 2 \\ 5 & 7 & 2 & 0 \end{bmatrix} \begin{array}{c} \text{join 2,3} \\ \rightarrow \end{array} \begin{bmatrix} 0 & 4 & 5 \\ 4 & 0 & 2 \\ 5 & 2 & 0 \end{bmatrix} \begin{array}{c} \text{join (2,3) and (4,5)} \\ \rightarrow \end{array} \begin{bmatrix} 0 & 4 \\ 4 & 0 \end{bmatrix}$$

   And for the complete link algorithm,

   $$\begin{bmatrix} 0 & 4 & 9 & 6 & 5 \\ 4 & 0 & 1 & 8 & 7 \\ 9 & 1 & 0 & 3 & 2 \\ 6 & 8 & 3 & 0 & 1 \\ 5 & 7 & 2 & 1 & 0 \end{bmatrix} \begin{array}{c} \text{join 4,5} \\ \rightarrow \end{array} \begin{bmatrix} 0 & 4 & 9 & 6 \\ 4 & 0 & 1 & 8 \\ 9 & 1 & 0 & 3 \\ 6 & 8 & 3 & 0 \end{bmatrix} \begin{array}{c} \text{join 2,3} \\ \rightarrow \end{array} \begin{bmatrix} 0 & 9 & 6 \\ 9 & 0 & 8 \\ 6 & 8 & 0 \end{bmatrix} \begin{array}{c} \text{join 1 and (4,5)} \\ \rightarrow \end{array} \begin{bmatrix} 0 & 9 \\ 9 & 0 \end{bmatrix}$$

   The same results are also obtained, if the pair (2,3) is merged first. The dendreograms for the operations are thus

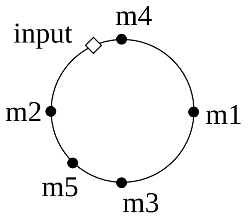
   

   This example illustrates the common trend for the single link algorithm to produce first an elongated cluster, whereas the complete link algorithm proceeds by recovering compact clusters.

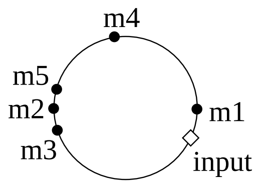
2. If the image consisting of $k \times k$ pixels is coded with 3 8-bit colors, the total amount of bits per image is naturally $24k^2$. When using LBG coding, we code a $k \times k$ window with respect to code vectors, and with n bits we have naturally 2^n code vectors. (Or 2^n code vectors can be obtained with n bits).

   Thus, the compression ratio is $\frac{n}{24k^2}$

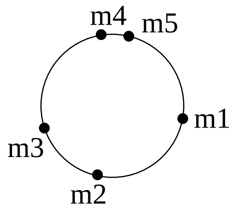
3. An example solution with two inputs:



First, m_4 is the nearest unit (m_4, m_3, m_5 updated)



Second, m_1 is the nearest unit (m_1, m_2, m_5 updated)



The final ordering

4. Using the energy function $C = \sum_q \sum_r h(r-q) \sum_{x \in V_r} ||x - m_q||^2$

   a) First derivate with respect to m_q and set to zero;

$$\frac{\partial C}{\partial m_q} = \sum_r h(r-q) \sum_{x \in V_r} (-2)(x - m_q) = -2 \sum_r h(r-q) \sum_{x \in V_r} (x - m_q) = 0$$

   b) Now we have $c_r = \frac{1}{N} \sum_{x \in V_r} x$. Thus

$$-2 \sum_r h(r-q)(Nc_r - Nm_q) = 0 \Leftrightarrow m_q = \frac{\sum_r h(r-q) c_r}{\sum_r h(r-q)}$$

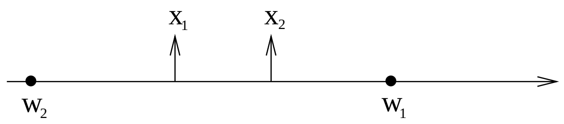
   Now if the sum of the neighborhood function over all the units is 1 (ie. $\sum_r h(r-q) = 1$), the proposed $m_q = \sum_r h(r-q) c_r$ follows.

5. First, a brief introduction to the Learning Vector Quantization - algorithm. There are several variations of LVQ, but a basic Learning Vector Quantization (LVQ) approach may be defined as:

   - Take m_i prototype vectors (also often called codebook vectors). There may be one or more prototype vectors per class, usually more than one is used.
   - Let $c = \arg\min_i(||x - m_i||)$ define the nearest prototype to an input vector x, denoted by m_c. When classifying, x is usually taken to belong to the class of it's nearest prototype vector, but m_c is also used in the learning phase.
   - Now the learning proceeds (with modifying only the nearest prototype $m_c(t)$ for the input $x(t)$ each step t) as in
     $m_c(t+1) = m_c(t) + \alpha(t)[x(t) - m_c(t)]$, if x and m_c belong to the same class and
     $m_c(t+1) = m_c(t) - \alpha(t)[x(t) - m_c(t)]$, if x and m_c belong to different classes.
     $0 < \alpha(t) < 1$ is the learning coefficient.
   - $\alpha(t)$ may be constant or decrease monotonically with time. For example in the lvq_pak LVQ1, an α initially smaller than 0.1 and decreasing linearly with time is used. The iteration is generally stopped either after a predefined number of steps, when $\alpha(t)$ reaches its minumum value or when a desired training set error rate is reached.

   Now to the task at hand. It is very intuitive that the weights would converge at 0 and 1, respectively. Now, to prove this let's first consider the most difficult situation, where the weight vectors w_1 and w_2 are on the wrong sides of the input:

   The input sequence is a stochastic process and in some stage either one of the weights will be the best matching unit for all the inputs:



$$\begin{array}{ccc} |w_2| + 1 < w_1 & \text{or} & w_1 < |w_2| + 1 \\ (w_2 \text{ always wins}) & & (w_1 \text{ always wins}) \end{array}$$

Let us consider the latter case: If the input belongs to class ω_1, $\Delta w_1 = \alpha(0 - w_1)$. If the input belongs to class ω_2, $\Delta w_1 = -\alpha(1 - w_1)$. The prior probabilities of the classes are equal: $E[\Delta w_1] = \frac{1}{2}\alpha(-w_1 - (1 - w_1) = -1\frac{\alpha}{2}$, $E[\Delta w_2] = 0$.

This means that w_1 drifts to the left whereas w_2 remains still. At some point w_1 passes w_2, which then becomes the best matching unit for all the inputs. In this situation $E[\Delta w_1] = 0$ and $E[\Delta w_2] = \frac{\alpha}{2}$. Now w_2 drifts to the right until $|w_1| < w_2$ and $|w_1| + 1 > w_2 - 1$, which means that each of the weight vectors are the best marching unit for their own input cluster.

Eventually the weights will converge to the points $w_1 = 0$ and $w_2 = 1$. Every other initial setting will lead to a situation described at some stage above. Therefore, the weights will always converge to the correct positions.