Computation in Cluster Analysis

K-means

- **Cluster analysis:** Suppose we observe $X_1, X_2, \ldots, X_n \in \mathcal{R}^d$, and we wish to partition the $X_i$ into $k$ groups ("clusters") such that a pair of elements in the same cluster tends to be more similar than a pair of elements belonging to distinct clusters.

- **Similarity measures:** The first step is to define a similarity measure $d : \mathcal{R}^d \times \mathcal{R}^d \rightarrow \mathcal{R}$. We will consider squared $L_2$ distance $d(X, Y) = \sum_j (X_j - Y_j)^2$, and $L_1$ distance $d(X, Y) = \sum_j |X_j - Y_j|$.

- **Loss function:** Represent a clustering by a map
  \[ \phi : \{1, \ldots, n\} \rightarrow \{1, \ldots, k\}. \]
  Let
  \[ \mu_j(\phi) = \arg\min_{\mu \in \mathcal{R}^d} \sum_i d(X_i, \mu) \mathbb{1}(\phi(i) = j) \]
be the optimal representative of the objects assigned to class $j$ by $\phi$. If $d$ is squared $L_2$ distance, $\mu_j(\phi)$ is the centroid of all objects assigned to class $j$ by $\phi$. If $d$ is $L_1$ distance, $\mu_j(\phi)$ is the sample median.

The goal of K-means is to minimize the loss function
\[ \mathcal{L}(\phi) = \sum_i d(X_i, \mu_{\phi(i)}(\phi)). \]

To be concrete, when $d$ is squared $L_2$ distance, the loss function becomes
\[ \mathcal{L}(\phi) = \sum_i \|X_i - \operatorname{Avg}\{X_\ell : \phi(X_\ell) = \phi(X_i)\}\|^2. \]

- **Optimization:** The global minimum of $\mathcal{L}$ can be found by enumerating the $k^n$ possible assignments of the $X_i$ into the $k$ clusters. In most applications this is not practical. An alternative is to use the following greedy algorithm:

1. Initialize $\phi$ in some way.
2. Set $\mu_j(\phi) = \operatorname{Avg}\{X_\ell : \phi(X_\ell) = j\}$.
3. For $i = 1, \ldots, n$, set $\phi^*(i) = \arg\min_{1 \leq \ell \leq k} \|X_i - \mu_{\phi(i)}(\phi)\|^2$. If $\phi^* \equiv \phi$ stop. Else set $\phi = \phi^*$ and return to 2.
Like all greedy algorithms, the K-means algorithm will reach a local, but not necessarily a global minimum. For this particular loss function, there are a very large number of local minima. Small changes in the initial assignment (step 1) can lead to very different solutions.

- **Implementation**: The following Octave code implements the greedy K-means algorithm using squared $L_2$ distance.

```octave
## Apply the greedy K-means algorithm to identify K clusters among
## the rows of X.
function [M,Q] = KM(X, K)

    ## Number of observation vectors.
    n = size(X,1);

    ## Number of variables.
    q = size(X,2);

    ## The initial map.
    Q = ceil(K*rand(n,1));

    for it=1:100

        ## Update the class centers.
        for k=1:K
            ii = find(Q == k);
            M(:,k) = mean(X(ii,:))';
        endfor

        ## Distance from each object to each class center.
        for k=1:K
            D(k,:) = sum((X' - M(:,k)*ones(1,n)).^2);
        endfor

        ## Reassign the objects to the closest class.
        [u,ix] = sort(D);  
        if (all(Q == ix(1,:)'))  
            break;
        endif
        Q = ix(1,:)';
    endfor

endfunction
```
• **Bottom-up clustering** Rather than attempting to directly cluster the data into \( K \) clusters, as in K-means, we can initially put the data into many small clusters, then sequentially merge clusters until we are left with only \( K \) of them.

• **Partitions:** The final result of a bottom-up clustering procedure is a sequence of partitions of the data, where a *partition* is a set of disjoint subsets \( S_1, \ldots, S_q \subset \{X_1, \ldots, X_n\} \) whose union is the set of all observed values. The \( S_i \) are the *components* of the partition.

• **Hierarchy of partitions:** A *hierarchy* of partitions is a sequence of partitions \( Q_1, \ldots, Q_m \) such that each component of \( Q_{i+1} \) is the union of components in \( Q_i \). Generally \( Q_1 \) places every object into its own component and \( Q_m \) places all objects into a single component.

For example:

\[
\begin{align*}
Q_5 &: 1 \quad 2 \quad 3 \quad 4 \quad 5 \\
Q_4 &: 1 \quad 2 \quad 3 \mid 4 \quad 5 \\
Q_3 &: 1 \mid 2 \quad 3 \mid 4 \quad 5 \\
Q_2 &: 1 \mid 2 \quad 3 \mid 4 \mid 5 \\
Q_1 &: 1 \mid 2 \mid 3 \mid 4 \mid 5
\end{align*}
\]

For any pair of objects there is a greatest index \( k \) such that the two objects are in different components of \( Q_k \). In the above example, the index for objects 1 and 3 is 3. For clustering, the sequence of partitions should have the property that a pair of similar objects should have a small value for this index.

• **Construction of the partition sequence:** The partitions \( Q_k \) are constructed in sequence starting with \( Q_1 \). To form \( Q_{k+1} \) from \( Q_k \) the two components that are most similar are merged.

We must also keep track of a similarity matrix defined on the set of components of the active partition. For \( Q_1 \) this is just the similarity matrix of the data (a \( n \times n \) symmetric matrix). Since \( Q_2 \) has only \( n-1 \) components, the similarity matrix must be reduced to an \( n-1 \times n-1 \) matrix. The size of the similarity reduces by 1 at each level. The final similarity matrix is just a scalar.

• **Updating the similarity matrix:** If neither \( \ell_1 \) nor \( \ell_2 \) are among the partitions to be joined, then the similarity between \( \ell_1 \) and \( \ell_2 \) is unchanged.

If \( \ell_1 \) and \( \ell_2 \) are the two components being joined, and \( r \) is some third component, then we have three ways to define the similarity between \( r \) and the new component:

- **Single linkage:** \( \min\{D(\ell_1, r), D(\ell_2, r)\} \).
- **Complete linkage:** \( \max\{D(\ell_1, r), D(\ell_2, r)\} \).
- **Average linkage:** \( (V_{\ell_1}D(\ell_1, r) + V_{\ell_2}D(\ell_2, r))/(V_{\ell_1} + V_{\ell_2}) \), where \( V_{\ell} \) is the number of objects in component \( \ell \).
• **Example (single linkage):** Suppose we are clustering vectors in $\mathbb{R}^2$, and $Q_1$ is:

$$\{(5, 9)\} \quad \{(6, 4)\} \quad \{(9, 4)\} \quad \{(1, 6)\}$$

If the similarity is $L^1$ distance, we have the following similarity matrix:

$$
\begin{pmatrix}
0 & 6 & 9 & 7 \\
0 & 3 & 7 \\
0 & 10 \\
0 \\
\end{pmatrix}
$$

The second partition is

$$\{(5, 9)\} \quad \{(6, 4), (9, 4)\} \quad \{(1, 6)\}$$

and the new similarity matrix under single linkage is

$$
\begin{pmatrix}
0 & \min\{6, 9\} & 7 \\
0 & \min\{7, 10\} \\
0 \\
0 \\
\end{pmatrix}
= 
\begin{pmatrix}
0 & 6 & 7 \\
0 & 7 \\
0 \\
0 \\
\end{pmatrix}.
$$

The third partition is

$$\{(5, 9), (6, 4), (9, 4)\} \quad \{(1, 6)\}$$

with similarity matrix

$$
\begin{pmatrix}
0 & \min\{7, 7\} \\
0 \\
\end{pmatrix}
= 
\begin{pmatrix}
0 & 7 \\
0 \\
\end{pmatrix}.
$$

The fourth partition is

$$\{(5, 9), (6, 4), (9, 4), (1, 6)\}.$$

• **Example (complete linkage):** Using the same data and similarity as the previous example, the second partition is

$$\{(5, 9)\} \quad \{(6, 4), (9, 4)\} \quad \{(1, 6)\}$$

and the new similarity matrix under single linkage is

$$
\begin{pmatrix}
0 & \max\{6, 9\} & 7 \\
0 & \max\{7, 10\} \\
0 \\
0 \\
\end{pmatrix}
= 
\begin{pmatrix}
0 & 9 & 7 \\
0 & 10 \\
0 \\
0 \\
\end{pmatrix}.
$$
The third partition is

\{ (5, 9), (1, 6) \} \{ (6, 4), (9, 4) \}

with similarity matrix

\[
\begin{pmatrix}
0 & \max\{9, 10\} \\
0 & 0 \\
\end{pmatrix} = \begin{pmatrix}
0 & 10 \\
0 & 0 \\
\end{pmatrix}.
\]

The fourth partition is

\{ (5, 9), (6, 4), (9, 4), (1, 6) \}.

**Implementation:** Here is Octave code for the three hierarchical clustering methods:

```octave
## Hierarchical clustering using: single linkage (M=1), complete linkage
## (M=2), or average linkage (M=3). The matrix D contains dissimilarities.
## The rows of 'cluster' are the partitions.
function cluster = hclust(D, M)

[R, C] = size(D);

for i=1:size(D,1)
    D(i,i) = inf;
endfor

## Put every point is in its own cluster.
cluster(1,:) = [1:R];

for i=2:size(D,1)
    cluster(i,:) = cluster(i-1,:);

    ## Find closest clusters
    [MR, ID] = min(D);
    [T, J] = min(MR);
    I = ID(J);

    ## Swap so that I <= J.
    U = sort([I, J]);
    I = U(1);
    J = U(2);

    ## Merge cluster j with cluster i, then delete cluster j.
```

5
ii = find(cluster(i,:)==J);
cluster(i,ii) = I;
ii = find(cluster(i,:)>J);
cluster(i,ii) = cluster(i,ii) - 1;

## Calculate new distance matrix
if (M == 1)
    D(:,I) = min(D(:,I), D(:,J));
    D(I,:) = min(D(I,:), D(J,:));
elseif (M == 2)
    D(:,I) = max(D(:,I), D(:,J));
    D(I,:) = max(D(I,:), D(J,:));
elseif (M == 3)
    D(:,I) = (D(:,I) + D(:,J)) / 2;
    D(I,:) = (D(I,:) + D(J,:)) / 2;
endif

ii = complement(J, [1:size(D,2)]);
D(I,I) = Inf;
D = D(ii,ii);

endfor
endfunction