8. Clustering: Pattern Classification by Distance Functions

The premise in clustering is that pixels that are close to each other in feature space are likely to belong to the same class. Thus, the distance between pixels in feature space is the measure of similarity. This is an intuitively satisfying approach that is useful if the patterns actually have convenient clustering characteristics, as in the figure below (Figure 8.1).

Because clustering methods rely on the inherent grouping (clustering) of pixels within a class, they generally require the least amount of prior information to operate. Very little detail about the data structure in measurement space is required; ideally this should be "discovered" during the clustering operation.

8.1 Minimum distance pattern classification

Single prototypes: Each class (pattern) is represented by a single prototype vector, \( z \).

Single prototypes will be most effective when the data form circular (or spherical) structures in measurement space (Figure 8.1). A circular pattern suggests that there is no correlation among the measurements \( (x_1, x_2, \ldots) \), and that the variation within the class is the same for all measurements. Assume that there are \( m \) classes and that these classes may be represented by the prototype vectors, \( z_1, z_2, z_3, \ldots, z_m \). A discriminant function for the \( i \)th class may be derived directly from the expression for the Euclidean distance, \( D_i(x) \), of a measurement vector, \( x \), from the prototype vector, \( z_i \):

\[
D_i(x) = |x-z_i| = [(x-z_i)^t (x-z_i)]^{1/2} = [(x-z_i) \cdot (x-z_i)]^{1/2} \quad [8.1]
\]

where:
- \( (x-z_i) \) denotes a column vector
- \( (x-z_i)^t \) denotes the transpose, a row vector
- \( \cdot \) indicates the dot product of two vectors

The obvious discriminant function is simply the negative of the distance:

\[
d_i(x) = -D_i(x) \quad [8.2]
\]

The larger (less negative) \( d_i(x) \), the closer the measurement vector \( x \) lies relative to the prototype vector \( z_i \). The maximum value of \( d_i(x) \) is zero and occurs when \( x \) matches the prototype vector exactly. This expression for \( d_i(x) \) is not computationally efficient. Several computationally convenient modifications can be made without altering the effect of \( d_i(x) \). The most obvious is to simply use the square of the distance, thus eliminating the need for computing the square root. The magnitude of the discriminant function changes, but the value of the discriminant function is equal for any two points that are equidistant from the prototype. The computation may be further simplified by expanding the vector product and recognizing that not all the terms are class dependent:
\( d_i(x) = -(x-z_i)'(x-z_i) = -[xx - 2x'z_i + z_i'z_i] \) \[8.3\]

Note that in Equation [8.3] the term, \( x'x \), is independent of class. This term contributes equally to the discriminant functions for all classes and, if it is eliminated, will not alter the assignment of the measurement vector to a particular class. Note further that the term, \( z_i'z_i \), is dependent on the class but not on the measurement vector. In other words, it is constant for a particular class and need only be computed once for each class -- not for each measurement vector. The crucial term is the product, \( x'z_i \).

"Distance" in feature space is the primary measure of similarity in all clustering algorithms. Even when it is impossible to reduce the number of distance computations, it may be possible to simplify the distance computation itself. For example, instead of computing the Euclidean distance one might compute the crude, but simple "taxicab" distance,

\[
D_i(x) = \sum_{k=1}^{N}|x_k - z_{ki}| \quad [8.4]
\]

where \( N = \# \) of variables (spectral bands)

which is computed as a simple sum, but overestimates diagonal distances.

---

1) Euclidean distance: \( d_i(x) = -D_i(x) = -[(x-z_i)'(x-z_i)]^{1/2} \)
2) Square of the Euclidean distance: \( d_i(x) = -D_i^2(x) = -[(x-z_i)'(x-z_i)] \)
3) Square of Euclidean distance after eliminating components that are independent of class: \( d_i(x) = -[x'z_i + (1/2)(z_i'z_i)] \)
4) taxicab distance \( d_i(x) = -\sum_{k=1}^{N} |x_k - z_{ki}| \)

Table 8.1 Alternate definitions of the discriminant function

The decision boundary

The class decision boundary defined by the single prototype, simple distance discriminant function is simply the set of planar surfaces perpendicular to and bisecting the lines connecting each of the prototypes (Figure 8.2). This type of classifier is called a minimum-distance classifier. If the prototype is chosen to be the mean value of the training pixels for each class, it is called a minimum-distance-to-mean classifier, appropriately enough.
Cluster Seeking

Results of a clustering procedure will depend rather heavily on the prototype choices of the user. Indeed, requiring that the user select the prototypes for each class tends to reduce the utility of a clustering procedure. It would be better if the user-selected prototypes were used only as a first approximation and better still if the user were not required to select the prototypes at all. To this end we will consider techniques which begin with an arbitrary starting point in measurement space and explore the adjacent data distribution to locate an optimal cluster center.

Figure 8.3: Diagram illustrating parameters for the simple cluster-seeking algorithm.

A simple cluster-seeking algorithm

The crude and very simple cluster-seeking algorithm is presented here as an introduction to the clustering procedures. This is a first step toward automating the classification process.

1. Select a threshold, \( T \)
   \( T \) is a representative distance in measurement space. The choice of \( T \) in this algorithm is entirely arbitrary; it is also the only input required of the user.

2. Select a pixel with measurement vector, \( x \).
   The selection scheme is arbitrary. Pixels could be selected at random, however, it is operationally more convenient to select pixels in an orderly fashion. For this example assume that the first pixel is selected from the upper left corner of the image, and that subsequent pixels are selected in order from left to right and from top to bottom of the image.

3. Let the first pixel be taken as the first cluster center, \( z_1 \).

4. Select the next pixel from the image.

5. Compute the distance functions, \( D_i(x) \).
   Compute the distance function for each of the classes established at this point, i.e., compute \( D_i(x) \), for \( i=1,N \) where \( N = \) the number of classes. (\( N=1 \) initially.) One would normally assign the pixel to the first class which satisfies the criterion since to assign it to the class for which \( D_i(x) \) is a minimum would require that all pixels be retested whenever a new prototype is established.

6. Compare the \( D_i(x) \) with \( T \).
   For each of the established classes, if \( D_i(x) < T \), then \( x \in \Sigma_i \). On the other hand, if \( D_i(x) > T \), for all \( i \), then let \( x \) become a new prototype vector: \( x \rightarrow z_{N+1} \).
7. Return to step #4 until all pixels have been assigned to a class.

This clustering algorithm is extremely sensitive to the value of T and the order in which pixels are selected. For a given T, two different selection patterns can yield very different results. Alternatively, for the same selection pattern, a different value of T will lead to a different result. These flaws are typical of clustering algorithms and are illustrated in Figure 8.5. All are sensitive to the starting selection of cluster centers and to the particular specification of the clustering criterion. The better algorithms handle the problems cleverly and without the severe problems that would be apparent with the above algorithm.

![Figure 8.4](image-url)

**Figure 8.4** Typical problems in clustering with the simple clustering algorithm.
Clustering results are very sensitive to the threshold size, T.

**Maximin distance algorithm**

The simple clustering algorithm described above is extremely sensitive to both the threshold distance, T, and the order in which pixels are selected from the image. The maximin (maximum-minimum) algorithm represents an attempt to define a less arbitrary and more repeatable distribution of cluster groups. It begins by identifying the cluster regions which are farthest apart, defines an initial threshold distance based on the separation of these cluster centers, and continues selecting cluster centers and readjusting T until all possible cluster centers are established. The procedure is as follows:

1. Select a pixel, \( x \), from the image at random.
2. Let \( x \) be the first cluster center, \( x \rightarrow z_1 \)
3. Sort through the remaining pixels to find the pixel, \( x \), which is farthest from \( z_1 \).
4. Let the most distant pixel be the second cluster center, \( x \rightarrow z_2 \).
5. Find the distance, \( T = ||z_2 - z_1|| \), between the two cluster centers. \( T \) will be an initial scaling distance used to determine the existence of the next cluster center.
6. Compute \( D_{\text{min}}(x_i) = \min[D(x_j)] \) for \( i=1,2 \), for all remaining pixels in the image, i.e., find the distance to the closest cluster center for every pixel in the image.
7. Find \( D_{\text{max}}(x_m) = \max[D_{\text{min}}(x_j)] \) for all \( j \). In other words, sort through all the distances determined in step 6 and select the maximum distance (select the maximum of the minimum distances). This procedure will find the pixel that is farthest (in measurement space) from either of the two cluster centers.
8. If $D_{\text{max}} > T/2$, then let $x_m \rightarrow z_2$, otherwise, 
   if $D_{\text{max}} < T/2$, then terminate the procedure.

   In words, if the maximum distance is greater than half the distance between the two 
closest cluster centers, then $x_m$ becomes a new cluster center, otherwise terminate the 
procedure.

9. Increment the number of cluster centers by 1: $N_c = N_c + 1$.

10. Reset the scaling distance:

    $$T = \sum_{i=1}^{n} \sum_{j=1}^{n} \|z_i - z_j\| / \sum_{k=1}^{n} k(k+1)/2.$$ 


---

**Figure 8.5** Examples of partitioning of feature space by the minimax algorithm where the 
cluster centers are labeled according to the order of selection: a) clearly separable, 
circular clusters of about the same size can often be located by the maximin algorithm, 
but b) more realistic distributions can easily improperly partitioned.

The partitioning of feature space by the maximin algorithm is a significant improvement over 
the simple clustering algorithm discussed previously. The essential idea is to isolate those points 
that are farthest apart assuming that the greater the distance between points, the more likely they 
are to belong to different classes (Figure 8.6a). The algorithm generally works well with roughly 
circular clusters whose size (radius) is smaller than the separation between clusters. However, 
the algorithm is very sensitive to details of the data structure. Consider, for instance, the two 
class problem illustrated in Figure 8.6b. The "true" data clusters are illustrated by the dark and 
open circles while the partitioning of the space is indicated by the numbered cluster centers and 
the decision boundaries.

**Comment on the necessity for programming efficiency.**

Computational complexity is an inherent problem in image processing. Because of the large 
volume of data and the need to repeat complex computations many times, it is crucial to 
streamline computations as much as possible. Clever, efficient programming is essential. The 
maximin algorithm, for example, is computationally demanding. Each time a new cluster center 
is selected, the feature space distance must be computed for every point from every cluster 
center. On the $i^{th}$ pass through an $m \times n$ pixel image there will be $i+1$ clusters requiring $m*n*i$ 
distance computations.
K-means Algorithm

In the previous clustering examples, once a point has been selected as a clustering center, it remains a clustering center, even if it is a relatively poor representative of its cluster. The K-means algorithm is an improvement in the sense that the clustering is optimized by shifting the cluster centers in order to optimize a performance index. Many variations of the K-means algorithm have been developed, but the steps of a basic are given below.

1. Choose \( K \) initial cluster centers, \( z_1, z_2, z_3, \ldots, z_k \).

   The only real requirement of the user in the K-means algorithm is the specification of the number of desired clusters. (In practice the number of desired cluster centers should be greater than the number of expected classes.) The actual positions of the cluster centers is sometimes specified by the user and is sometimes assigned by the particular routine. If selected by the routine they may be chosen in a variety of ways:
   a) by random selection from the image data,
   b) by applying some a priori information (e.g., training data),
   c) by selecting points based on preliminary calculations (minimum/maximum grey values, variance in each band, localized data density, etc.), or
   d) by applying a theoretical principal independent of actual data.

2. Distribute the samples among the K means

   Samples should be assigned to the class represented by the nearest cluster center, i.e.:
   \[
   x \in S_i(n) \text{ if } \| x - z_i(n) \| \leq \| x - z_j(n) \| \text{ for all } j = 1, 2, 3, \ldots, K; \text{ where } i \neq j
   \]

   \( S_i(n) \) is the set of samples whose cluster center is \( z_i(n) \), where \( n \) indicates that this is the \( n^{th} \) iteration of this procedure.

3. Compute new cluster centers for each set \( S_i(n) \)

   Find a new value for each \( z_i \). The new cluster center, \( z_i(n+1) \) will be the mean of all the points in \( S_i(n) \) such that:
   \[
   z_i(n+1) = \frac{1}{N_i} \sum_{x \in S_i(n)} x
   \]

4. Compare \( z_i(n) \) and \( z_i(n+1) \) for all \( i \)

   Compute the distance between each pair of points for the consecutive iterations. If there is no substantial change, terminate the procedure, otherwise return to step 2. for the next iteration. Some possible criteria for termination are:
   a) if \( \| z_i(n+1) - z_i(n) \| < T \) for all \( i \)
   b) if \( \sum_{i=1}^{K} \| z_i(n+1) - z_i(n) \| < T \) for all \( i \)
   c) if \( z_i(n+1) = z_i(n) < T \) for all \( i \)
The procedure is illustrated in Figure 8.6. Results of K-means algorithms will depend on the number of means chosen, the initial distribution and the distribution of the image data in feature space.

**Figure 8.6a** K-means algorithm: Three-class data distribution in two-dimensional measurement space. The class means are indicated by solid squares (■).

<table>
<thead>
<tr>
<th>mean vector</th>
<th># of members</th>
</tr>
</thead>
<tbody>
<tr>
<td>( m_1 = (4.7, 4.1) )</td>
<td>66</td>
</tr>
<tr>
<td>( m_2 = (10.0, 2.3) )</td>
<td>66</td>
</tr>
<tr>
<td>( m_3 = (11.4, 5.9) )</td>
<td>168</td>
</tr>
</tbody>
</table>

**Figure 8.6b**: Initial cluster centers (●), the associated partitioning of measurement space (bold solid lines), and the new cluster centers (o) -- computed as the means of the new partitions.
Figure 8.6c  Second iteration of the K-means clustering with the previous cluster centers (•), the 2nd partitions (bold solid lines), and the new cluster centers (o), and the 3rd partitions (narrow solid lines).
ISODATA Algorithm

ISODATA: Iterative Self-Organizing Data Analysis Technique A

The ISODATA algorithm is essentially a refinement of the K-Means algorithm. The specific refinements are:

1) Clusters that have too few members are discarded.
2) Clusters that have too many members are split into two new cluster groups.
3) Clusters that are too large (too disperse) are split into two new cluster groups.
4) If two cluster centers are too close together they are combined.

for making any of the required decisions. The user must select more or less arbitrary values for the various process parameters. This can be done iteratively: choose an initial parameter set; evaluate the clustering procedure; adjust the parameters; etc. Thus, the automated "unsupervised" classifier can be used interactively.

1. Select an initial (seed) set of clustering centers, \( z_1, z_2, \ldots, z_{Nc} \)

A set of clustering centers (prototypes) must be chosen for most clustering algorithms. Most require the user to specify the number of initial cluster centers. Some allow the user to specify the specific locations of the cluster centers; most, however, simply assign the cluster centers (often with an undocumented procedure). Clustering algorithms tend to be sensitive to the number and location of the initial cluster centers. Selecting a good set of seed values, can not only speed up the clustering process (faster convergence) but can also lead to "better" results.

2. Specify process parameters

\( K \) = number of cluster centers desired. If specific initial cluster centers are not passed to the program, \( K \) can also serve as the number of initial cluster centers. The actual number of clusters returned at the end of clustering will be between \( K/2 \) and \( 2K \).

\( N_{\text{min}} \) = the minimum number of samples allowed for a viable cluster.

\( \sigma_{\text{lim}} \) = standard deviation parameter. A measure of the maximum allowable size of a single cluster.

\( D_{\text{lump}} \) = lumping parameter -- a distance in gray values. If two cluster centers are closer than the lumping parameter distance, the clusters will be grouped into a single cluster. Only pairs of clusters will be lumped together.

\( N_{\text{lump}} \) = maximum number of pairs of cluster centers which can be lumped during one iteration.

\( N_{\text{iter}} \) = number of iterations allowed. If the procedure is going to work given the initial parameters, it should converge fairly quickly, i.e., in a few iterations. What "quickly" means in time will depend on the number of clusters being sought, the size, complexity and dimensionality of the data set, and the specific machine on which the process is running. If the routine does not converge "quickly" it could go on indefinitely.
3. Distribute samples among the cluster centers
   \[ x \in S_j \text{ iff } \| x - z_j \| \leq \| x - z_i \| \]
   where: \( S_j \) = the set of samples assigned to cluster center \( z_j \)
   \( i, j = 1, 2, 3, \ldots, N_c; \ i \neq j \)
   \( N_c \) = number of cluster centers

4. Discard sample subsets with fewer than \( N_{\text{min}} \) members
   If \( S_j \) contains \( N_j \) members and \( N_j < N_{\text{min}} \), discard \( z_j \) and reduce \( N_c \) by 1. The members of
   may be dealt with in a number of ways, peculiar to the specific algorithm. They may be:
   a. incorporated into other clusters
   b. ignored for this iteration
   c. assigned to an unclassified group and ignored for the rest of the procedure.

5. Compute new cluster centers for each set \( S_j \)
   \[ z_j = \frac{1}{N_j} \sum_{x \in S_j} x \quad j = 1, 2, 3, \ldots, N_c \]

6. Compute the cluster size parameter, \( D_j \)
   \( D_j \) is the average distance of samples in each cluster, \( S_j \), from the corresponding cluster
   center, \( z_j \):
   \[ D_j = \frac{1}{N_j} \sum_{x \in S_j} | x - z_j | \quad j = 1, 2, 3, \ldots, N_c \]

7. Compute the overall size, \( D_s \)
   \( D_s \) is weighted average of distances of samples from their respective cluster centers:
   \[ D_s = \frac{1}{N} \sum_{j=1}^{N_c} N_j D_j \]
   where \( N \) is the total number of pixels:
   \[ N = \sum_{j=1}^{N_c} N_j \]

8. Branch point
   a) On the last iteration, set \( D_l = 0 \) and go to step 12,
   b) if \( N_c \leq K/2 \) go to step 9,
   c) On even iterations or if \( N_c \geq 2K \), go to step 12,
   d) otherwise go to step 9.
9. Find the standard deviation vector for each class, \( \boldsymbol{\sigma}_j = (\sigma_{1j}, \sigma_{2j}, \ldots, \sigma_{nj}) \)

\[
\sigma_{ij} = \frac{1}{N_j} \left[ \sum_{x \in S_j} (x_i - z_{ij})^2 \right]
\]

where: \( j = 1, 2, 3, \ldots, N_c \)
\( i = 1, 2, 3, \ldots, n \)
\( n = \) number of bands

\( \sigma_{ij} \) represents the standard deviation of all the samples in \( S_j \) along the principal coordinate axes, \( x_j \) in feature space.

10. Find the maximum component of each \( \boldsymbol{\sigma}_j \)

\( \sigma_{i,\text{max}} = \max \{ \sigma_{ij} \} \)

This will be the primary measure of the size of the cluster.

11. Test for splitting

\[ \text{IF} \]

\[ \text{the variance for any cluster, } j, \text{ exceeds the limiting value,} \]
\[ \text{that is, if } \sigma_{i,\text{max}} > \sigma_{\text{lim}} \]

\[ \text{AND EITHER} \]
\[ D_j > D_s \quad \text{and} \quad n_j > 2(K+1) \]

\[ \text{that is, if the cluster is more disperse than the average and if splitting} \]
\[ \text{will not result in exceeding the upper limit of clusters.} \]

\[ \text{OR} \]
\[ N_c < K/2 \]

\[ \text{that is, if there are too few clusters.} \]

\[ \text{THEN} \]
\[ \text{split the cluster, } S_j, \text{ into two new clusters.} \]
\[ \text{let} \]
\[ z_{N_c+1} = (z_1, z_2, \ldots, z_j + k\sigma_{j,\text{max}}, \ldots, z_n) \]
\[ z_j = (z_1, z_2, \ldots, z_j - k\sigma_{j,\text{max}}, \ldots, z_n) \]
\[ N_c = N_c + 1 \]

If splitting occurs then go to step 2; otherwise continue.

12. Compute the pairwise distances, \( D(i,j) \), between all cluster centers

\[ D(i,j) = |z_i - z_j|, \quad i = 2, \ldots, N_c \]
\[ j = 1, 2, \ldots, i-1 \]

13. Find and sort the \( N_{\text{lump}} \) smallest distances, such that, \( D(i,j) < D_{\text{lump}} \)

Find the set of \( D_k(i,j) \leq D_{\text{lump}} \), where:
\[ k = 1, 2, 3, \ldots, N_{\text{lump}} \]
\[ \text{and} \quad D_k(i,j) \leq D_{k+1}(i,j) \]
14. **Lump clusters**

Let $D_k(i,j)$ be the distance between cluster centers at $z_i$ and $z_j$. If neither $z_i$ nor $z_j$ has been used for lumping during this iteration, then merge the two clusters:

$$z_{\text{lump}} = \frac{1}{N_i + N_j} \left[ z_i N_i - z_j N_j \right]$$

delete $z_i$ and $z_j$ and decrement $N_c$: $N_c = N_c - 1$

Note: - only pairwise lumping is allowed
- if any $z_i$ appears in more than one pair, only the first pair will be merged.

15. **Continue or terminate**

**IF** $N_{iter}$ iterations have already been performed.

**OR IF** on the $k$th iteration: $N_c(k) = N_c - 1$ and $|z_i(k) - z_i(k-1)| < \epsilon$

**THEN** TERMINATE

**ELSE** CONTINUE

---

**Suggested Reading**


**References**


