Chapter 5. Fuzzy Clustering and Merging

5.1 Fuzzy Weights

Let \{x^{(q)}: q = 1,\ldots,Q\} be a sample of feature vectors. During clustering we assign the Q feature vectors to K clusters according to how close they are to the prototype for each cluster. As was discussed previously, feature vectors that are farther away from the prototypical vector for a cluster should not have as much weight as those that are centrally and densely located. These more distant feature vectors are outliers caused by errors in one or more measurements or a deviation in the processes that formed the object.

To give deviant feature vectors the same weight in the averaging of a cluster as the more centrally and densely situation ones causes the prototype obtained by weighted averaging to no be as representative of its cluster as is should be. The previous discussion suggested that median vectors are more immune to outliers and are are thus more likely to be more typical and representative for a cluster.

While medians are better than averages, there are other ways to obtain weights so that a weighted average will be immune to outliers and yet provide appropriate weighting for more centrally and densely located vectors. These weights are usually called fuzzy weights. One way to generate them is to used the reciprocal of distances. Consider the distance \(D_{qk}\) between \(x^{(q)}\) and \(z^{(k)}\). If the distance is relatively large, then \(x^{(q)}\) should be weighted much less than if the distance is small. Thus

\[
w_{qk} = \frac{1}{D_{qk}}\tag{5.1}
\]

is a weighting that was used in the 1970s. It is relatively large when the distance is small, so that close vectors count much more in the averaging. On the other hand, it is relatively small when the distance is large.

There is one problem with using the reciprocal of the distance. \(D_{qk}\) may be zero when \(x^{(q)} = z^{(k)}\). We can use

\[
w_{qk} = \frac{1}{D_{qk} + 1}\tag{5.2}
\]

instead so that the maximum weight is 1 and the minimum approaches 0. In practice, if a weight computed from Equation (5.1) ever exceeds 1 then we may round it to 1.
Considering the k-means loop of assigning all feature vectors to clusters and then averaging those in each cluster, we can improve the obtained prototypes for each cluster by using the weighted average

\[ z_n^{(k)} = \sum_{\{q: \text{clust}[q]=k\}} w_{qk} x_n^{(q)} \] .................................(5.3)

for each \(n\)th component of the prototype. Upon doing this for each component, \(n = 1, \ldots, N\), we obtain the prototype

\[ z^{(k)} = (z_1^{(k)}, \ldots, z_N^{(k)}) \] .................................(5.4)

Upon doing this for each cluster \(k\), we arrive at new prototypes that are better located inside of their clusters. When we next assign the feature vectors to clusters by nearest distance, some of the assignments may be different.

5.2 Fuzzy Weights

The fuzzy \(c\)-means algorithm was invented by Bezdek [1a, 1b] and Dunn [2] in 1973 (it was Bezdek's dissertation at Cornell, but Dunn's fuzzy ISODATA appeared that year also). Bezdek's idea was to express the total weighted mean-square error

\[ J(w_{qk}, z^{(k)}) = \sum_{(q=1,Q)} \sum_{(k=1,K)} (w_{qk})^p |x^{(q)} - z^{(k)}|^2 \] .............(5.5)

subject to the constraint that

\[ \sum_{(k=1,K)} w_{qk} = 1 \text{ for each } q \] .................................(5.6)

Upon employing Lagrange multipliers and letting \(p > 1\), he obtained

\[ H(w_{qk}, z^{(k)}) = \sum_{(q=1,Q)} \sum_{(k=1,K)} (w_{qk})^p |x^{(q)} - z^{(k)}|^2 + \]

\[ \sum_{(q=1,Q)} \lambda_q (\sum_{(k=1,K)} w_{qk} - 1) \] .................................(5.7)

Upon setting the partial derivatives with respect to \(w_{qk}\) equal to zero and solving, the optimizing weights were found to be (where \(D_{qk} = |x^{(q)} - z^{(k)}|\))

\[ w_{qk} = (1/(D_{qk})^{2/(p-1)}) / \sum_{(r=1,K)} (1/(D_{rk})^{2/(p-1)}) \] .............(5.8)
For each cluster $k$, the weights are standardized so that they sum to unity over the cluster so the new prototype for that cluster can be computed as a weighted average. The standardizing is done by dividing the weights for the $k^{th}$ cluster by the sum of all weights for that cluster according to

$$ W_{qk} = \frac{w_{qk}}{\sum_{r=1,Q} w_{rk}} ...........................................(5.9) $$

The fuzzy c-means algorithm allows each feature vector to belong to every cluster with a fuzzy value (between 0 and 1), which is interpreted as its fuzzy truth of belonging. Rather than let a feature vector belong to a single cluster with a truth value of 0 (false, which means it does not belong) or 1 (true, it does belong), the fuzzy weights permit an in-between truth value of belonging.

The fuzzy truths are the weights. Thus $W_{qk}$ gives the fuzzy truth that feature vector $q$ belongs to cluster $k$ when $0 \leq w_{qk} \leq 1$. Given a feature vector $q$, we can take the maximum $w_{qk_{\text{max}}}$ of all $w_{qk}$ over all clusters $k = 1,\ldots,K$ and say that the feature vector belongs to cluster $k_{\text{max}}$ ($w_{qk_{\text{max}}} > w_{qk}$ for all $k \neq k_{\text{max}}$). But if another truth value is very near (or equal) for the same $q$ but a different $k$, then we may conclude that feature vector $q$ belongs to both clusters.

The fuzzy prototypes are updated by the following (fuzzy) weighted averaging.

$$ z^{(k)} = \sum_{q=1,Q} W_{qk} x^{(q)} ...........................................(5.10) $$

### 5.3 The Fuzzy c-means Algorithm

The complete fuzzy c-means algorithm is given below. It requires that a value for $K$ be given. Thus we need to compute clusterings for several values of $K$ and use a validity measure to see which is best. In the next chapter we will describe the Xie-Beni clustering validity measure that can be used for fuzzy clusterings. In the algorithm given below the function $\text{random()}$ is a uniform(0,1) random number generator.

We first draw initial weights $w_{qk}$ randomly and then standardize them so that for each fixed $q$, we have that $w_{q1} + \ldots + w_{qK} = 1$ (all fuzzy truths for it belonging to all clusters sum to 1). We next standardize the weights for each fixed cluster $k$ by summing over $q = 1,\ldots,Q$. This is necessary to find the weighted average for the prototype of each cluster $k$. Then we loop by i) computing new prototypes as weighted averages of all $Q$ feature vectors (rather than just the ones in the $k^{th}$ cluster; ii) computing new weights (and standardizing); and iii) checking the stopping criterion.

-----------------randomly initialize the weights----------------------

**Step 1:**
input K; input p; //p is used in the weight exponent
input I_{max}; I = 0;
for k = 1 to K do
for q = 1 to Q do
w[q,k] = random(); //uniform random numbers
------------------standardize the initial weights-----------------------

Step 2:
for q = 1 to Q do //for each feature vector
Sum = 0.0;
for k = 1 to K do //sum over all k prototypes
Sum = Sum + w[q,k];
for k = 1 to K do
w[q,k] = w[q,k]/Sum; //standardize over K
------------------standardize the initial weights-----------------------

Step 3:
for k = 1 to K do //for each cluster k
Sum = 0.0;
for q = 1 to Q do //sum over all Q
Sum = Sum + w[q,k];
for q = 1 to Q do
W[q,k] = w[q,k]/Sum; //standardize over Q
------------------standardize the initial weights-----------------------

Step 4:
for k = 1 to K do //for each cluster k
for n = 1 to N do
Sum = 0.0;
for q = 1 to Q do //compute weighted sum over Q
Sum = Sum + W[q,k]*x[n,q];
z[n,k] = Sum; //prototype equals weighted sum
------------------compute new prototype vectors---------------------

Step 5:
for q = 1 to Q do //for each feature vector q
Sum = 0.0;
for k = 1 to K do //for each prototype k
D[q,k] = 0.0;
for n = 1 to N do //compute distance components
D[q,k] = D[q,k] + (x[n,q] - z[n,k])^2;
Sum = Sum + (1/D[q,k])^{1/(p-1)}; //compute sum of distance reciprocals
for k = 1 to K do
w[q,k] = (1/D[q,k])^{1/(p-1)}/Sum; //compute standardized weights
------------------compute new weights---------------------

Step 6:
I = I + 1; //update iteration number
if I > I_{max} then stop;
else goto Step 3;
------------------compute new weights---------------------
Bezdek and others have proved certain convergence theorems under various conditions. But convergence is not assured to a global minimum because of local minima and saddle points of the function $J(-)$. Thus we should run the fuzzy c-means algorithm multiple times with different initial weight sets and then choose the results corresponding to the lowest functional value $J$ in Equation (5.5).

5.3 A Variation of the Fuzzy c-means Algorithm

Smolensky [4] reverses the sums in the constraints in Equation (5.7) for the purpose of forcing the weights to be large for representative feature vectors and forcing the nonrepresentative ones to be low. He uses

$$G(w_{qk}, z^{(k)}) = \sum_{q=1}^{Q} \sum_{k=1}^{K} (w_{qk})^p |x^{(q)} - z^{(k)}|^2 +$$

$$\Sigma_{k=1, K} \lambda_k \left( \Sigma_{q=1, Q} (1 - w_{qk})^p \right)$$

This is not the same as Equation (5.7). Upon putting the partial derivatives to zero and solving, the result is

$$w_{qk} = 1 / \left\{ 1 + \left( D_{qk}^2 / \lambda_k \right)^{1/(p-1)} \right\}$$

The value of $p$ determines the weights of the final partition into $K$ clusters. The values of $\lambda_k$ determine the distance at which the fuzzy weight becomes 0.5. Some practical values found to work well are

$$\lambda_k = A \Sigma_{q=1, Q} (w_{qk})^p |x^{(q)} - z^{(k)}|^2 / \Sigma_{q=1, Q} (w_{qk})^p$$

$A = 1$ can be used with good results. The algorithm is the same as that of the fuzzy c-mean except that the weights are computed from Equation (5.12) and $\lambda_k$ is computed from Equation (5.13) for use in Equation (5.12).

5.4 A Simple Effective Fuzzy Clustering Algorithm

**Fuzzy Set Membership Functions.** Let $\{x^{(q)}: q = 1,...,Q\}$ be a sample of feature vectors in $N$-dimensional Euclidean space. The Gaussian function
\[ x = f(x; z^{(k)}) = \exp[-|x - z^{(k)}|^2/(2 \sigma^2)] \] .......................................(5.14)

is called a fuzzy set membership function [3] for the linguistic variable \textit{SIMILAR TO} \( z^{(k)} \). The closer \( x \) is to \( z^{(k)} \), the higher is the fuzzy truth \( f(x; z^{(k)}) \) of \textit{SIMILAR TO} \( z^{(k)} \). If \( x = z^{(k)} \) then the fuzzy truth of \( x \) is \textit{SIMILAR TO} \( z^{(k)} \) is \( f(x; z^{(k)}) = 1 \). The fuzzy truth values go from 0 (but never attain 0) to 1. Figure 1 below shows the function on 2-dimensional feature vectors.

Figure 1. A Gaussian Fuzzy Set Membership Function

Thus fuzzy logic is not like binary logic where a switch is off (0) or on (1) and nothing in between. Fuzzy logic is like a water faucett that can be off (0), partially on (value \( f \) where \( 0 < f < 1 \) ) or on all the way (1). It doesn't make much difference whether it is on all the way (1) or almost all the way on (say, 0.987).

Gaussian functions are the most natural but there are other types of fuzzy set membership functions and an unlimited number of linguistic variables. In the case given above, if \( x \) is
very close to \( z^{(k)} \), then the fuzzy truth of the given linguistic variable is very close to 1, but as \( x \) moves farther away, the fuzzy truth starts decreasing increasingly rapidly. They are also defined on vector spaces of any dimension \( N \) without difficult contrivances.

To develop an algorithm that is similar to the improved k-means, we initially draw \( K \) prototypes \( \{ z^{(k)} : k = 1,...,K \} \) with a uniform random number generator. We assign each feature vector \( x^{(q)} \) according to nearest distance and put \( \text{clust}[q] = k_{\text{min}} \), where \( k_{\text{min}} \) is the index of the nearest prototype. After all \( Q \) feature vectors have been assigned, we compute the fuzzy weights. A Gaussian fuzzy set membership function is centered on each prototype and we compute

\[
w_{qk} = \tilde{f}(x^{(q)}, z^{(k)}) \tag{5.15}
\]

We next standardize the weights for each cluster \( k \) by the mapping

\[
w_{qk} = w_{qk}/\left\{ \sum_{q: \text{clust}[q]=k} w_{qk} \right\} \tag{5.16}
\]

The new fuzzy prototypes are computed via

\[
z^{(k)} = \sum_{q: \text{clust}[q]=k} w_{qk} x^{(q)} \tag{5.17}
\]

The weighted fuzzy expected value (WFEV) \( z \) of a set of real values \( x_1,...,x_p \) is defined recursively. Starting with the arithmetic mean \( z^{(0)} \) we iterate via

\[
z^{(r+1)} = \sum_{p=1,p} \alpha_p x_p \tag{5.18}
\]

where

\[
\alpha_p = \exp[-(x_p - z^{(r)})^2/(2 \sigma^2)] / \sum_{s=1,p} \exp[-(x_s - z^{(r)})^2/(2 \sigma^2)] \tag{5.19}
\]

For vectors we compute the WFEV of each component. Figure 2 shows the WFEV of 5 vectors compared with the mean and median vectors.
Figure 2. A weighted fuzzy expected vector

The actual computation of the weighted fuzzy expected vector is done componentwise for each $n = 1,...,N$. The higher level simple fuzzy clustering algorithm follows (some details are shown and some functions are used that have been defined previously.

------------------initialize prototypes for large $K$-------------------

Step 1:
input $K$; input $I_{\text{max}}$;
for $k = 1$ to $K$ do
for $n = 1$ to $N$ do
$z[n,k] = \text{random}();$

------------------run functions, improved k-means algorithm------------------
Step 2:

delete(); //deletes prototypes too close to another prototype
kmeans(); //run simple k-means clustering as before
eliminate(p); //eliminate clusters with less than p members
kmeans(); //reassign vectors to remaining clusters
sigmas(); //compute variances of all clusters

-------------compute fuzzy weights, fuzzy cluster averages-------------

Step 3:

fuzIts = 0;
for k = 1 to K do //for each cluster k do fuzzy average recursively
for n = 1 to N do //componentwise fuzzy averaging
  Sum[n,k] = 0.0;
for q = 1 to Q do
  if clust[q] = k then //if feature vector q is in cluster k
    w[q,k] = exp(-(x[n,k] - z[n,q])^2 / (2σ^2)); //fuzzy weights for component n
    Sum[n,k] = Sum[n,k] + w[n,k]; //denominator sum for standardizing
  z[n,k] = 0.0; //zero out for summing up new fuzzy average as prototype;
for q = 1 to Q do
  if clust[q] = k then
    w[q,k] = w[q,k]/Sum[n,k]; //standardize
    z[n,k] = z[n,k] + w[q,k]z[n,k]; //fuzzy averaging

------------------check for finish fuzzy averaging------------------

Step 4:

fuzIts = fuzIts + 1;
if fuzIts < 20 then goto Step 3; //iterate prototype component fuzzy averages
else
  assign(); //this function assigns feature vectors to clusters
  I = I + 1;
  if I < I_max then goto Step 3;
else
  merge(); //merge clusters close together
  stop; //all fuzzy clustering iterations done

-----------------------------------------------------------

-----------------------------------------------------------
References


