ECE 8527: Exam 3

Andrew Powell

May 2, 2014

1. **Problem No. 1:** Consider 5 data points: (0, 1), (-1, 0), which belong to class 1 (i.e. ω_1), and (1, 0), (0, -1), and (-$\frac{1}{2}$, $\frac{1}{2}$), which belong to class 2 (i.e. ω_2). In this problem we are going to walk through the k-means clustering process.

   (a) Assume your initial guesses for the two cluster centers are (0, 0) and ($\frac{1}{2}$, $\frac{1}{2}$). Execute an iteration of k-means by computing the new cluster and assigning the data points to the correct cluster. Use averaging to computer the new cluster center.

      i. **Theoretical Solution:** k is equal to 2 since there are two clusters. To distinguish the clusters, the clusters are referred to as ω_a and ω_b. The initial center coordinates of the two clusters are the following.

$$\begin{aligned} C_{\omega_a} &= \{(0,0)\} \\ C_{\omega_b} &= \{(0.5,0.5)\} \end{aligned} \tag{1}$$

      The letters a and b in the clusters ω_a and ω_b are used to reflect the fact the class identities of the clusters are unknown; that is, it is unknown how the true classes (i.e. ω_1 and ω_2) uniquely correspond to the clusters. C_{ω_a} and C_{ω_b} each refer to a set of center coordinates. $C_{\omega_a}(t)$ and $C_{\omega_b}(t)$, where t is a nonnegative integer, will refer to a particular coordinate in the set. For instance, $C_{\omega_a}(0)$ refers to $\{(0,0)\}$.

      In order to calculate the new center coordinates, the 2-dimensional Euclidean distances between each point and the center coordinates are calculated. Points are assigned to each cluster by selecting the cluster whose center coordinate's distance to the points is the smallest.

      To help summarize the results, the following notation is also used. All the points are placed into a set P; that is, $P = \{(0,1),(-1,0),(1,0),(0,-1),(-0.5,0.5)\}$. $P(n)$, where n is a nonnegative integer, will refer to a particular point within the set P. The function $d(\overrightarrow{v_1}, \overrightarrow{v_2})$ is the 2-dimensional Euclidean distance for the arguments $\overrightarrow{v_1}$ and $\overrightarrow{v_2}$, both of which represent coordinates. Each point is assigned to ω_a if $d(C_{\omega_a}(0), P(n)) \leq d(C_{\omega_b}(0), P(n))$ is true, otherwise ω_b is assigned. Below is a table that summarizes the cluster assignments for the iteration of the k-means algorithm. The results of the table are generated from the MATLAB© script presented in the next section.

n	$P(n)$	$d(C_{\omega_a}(0), P(n))$	$d(C_{\omega_b}(0), P(n))$	Assigned Cluster
0	$(0,1)$	1	0.7071	ω_b
1	$(-1,0)$	1	1.5811	ω_a
2	$(1,0)$	1	0.7071	ω_b
3	$(0,-1)$	1	1.5811	ω_a
4	$(-0.5,0.5)$	0.7071	1	ω_a

(2)

Finally, the sets C_{ω_a} and C_{ω_b}—each of which are the center coordinates that correspond to the clusters ω_a and ω_b, respectively—are each updated based on the mean of the assigned points. For instance, the x element of the new center coordinate for ω_a is simply the mean of all the x elements from points assigned to ω_a. The same calculation is applied to the center coordinate's y coordinate and for both x and y elements of ω_b's new center coordinate.

$$\begin{aligned} C_{\omega_a} &= \{(0,0),(-0.5,-0.1667)\} \\ C_{\omega_b} &= \{(0.5,0.5),(0.5,0.5)\} \end{aligned} \tag{3}$$

ii. **MATLAB© Script:**

```
P = [0 -1 1 0 -.5
    1 0 0 -1 .5];
C_omega_a = [0; 0];
C_omega_b = [.5; .5];

iterations = 1;

for i=1:iterations

    % determine the Euclidean distances
    D = dist([P C_omega_a(:,end) C_omega_b(:,end)]);
    D = D(1:(end-2),(end-1):end);   % D refers to the distances

    % determine the cluster assignments
    cA = D(:,1) <= D(:,2);          % cA refers to cluster assignments
    cA = char(uint8(cA));
    cA(cA==1) = 'a'
    cA(cA==0) = 'b'

    % determine new cluster centers based on the cluster assignments
    C_omega_a(:,end+1) = [mean(P(1,cA=='a'),2); mean(P(2,cA=='a'),2)];
    C_omega_b(:,end+1) = [mean(P(1,cA=='b'),2); mean(P(2,cA=='b'),2)];

end
```

Listing 1: MATLAB© Script

As shown in the script, the algorithm can be set to run for however many iterations. As it turns out, though, only a single iteration is necessary to obtain converged results.

(b) Assign an identity to each cluster based on a majority-voting scheme and draw the maximum likelihood decision surface.

The class identities of ω_a and ω_b are determined by selecting the class that has the most number of points from the set of the k points. The set of k points are the k closest points to the center coordinate whose cluster is the cluster to which a class is being assigned. The center coordinates used to obtain the sets of k points are the newly created center coordinates as shown in Equation 3, which are $C_{\omega_a}(1)$ and $C_{\omega_b}(1)$. The table below is an updated version of the table shown in Equation 2. The distances are updated based on the new center coordinates, $C_{\omega_a}(1)$ and $C_{\omega_b}(1)$. The

class identities, ω_1 and ω_2, are also included for each point.

n	$P(n)$	$d(C_{\omega_a}(1), P(n))$	$d(C_{\omega_b}(1), P(n))$	Assigned Cluster	True Class
0	$(0, 1)$	1.2693	0.7071	ω_b	ω_1
1	$(-1, 0)$	0.5270	1.5811	ω_a	ω_1
2	$(1, 0)$	1.5092	0.7071	ω_b	ω_2
3	$(0, -1)$	0.9718	1.5811	ω_a	ω_2
4	$(-0.5, 0.5)$	0.6667	1	ω_a	ω_2

(4)

The table below shows the sets of k-closest points to the center coordinates, where $k = 3$, and the class winners of the majority vote.

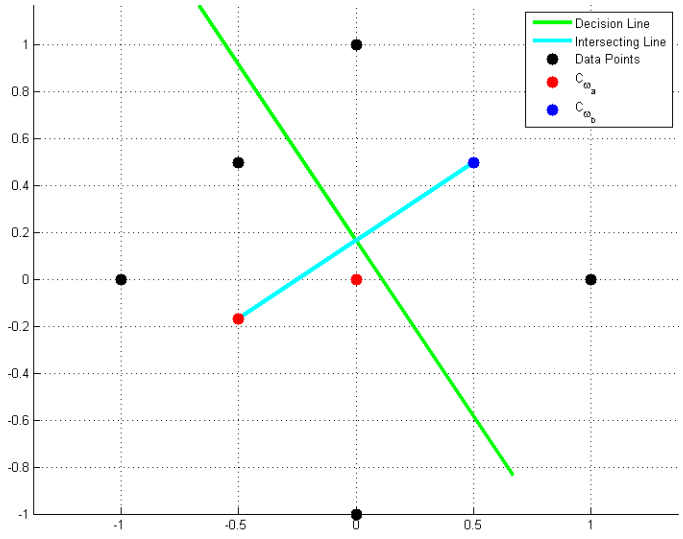
Center Coordinate	Set of k-closest Points Where $k = 3$	Winner
$C_{\omega_a}(1)$	$\{P(1), P(4), P(2)\}$	ω_2
$C_{\omega_b}(1)$	$\{P(0), P(2), P(4)\}$	ω_2

(5)

As it turns out, the winner for both center coordinates is ω_2. Other values of k, all values from the set {1, 3, 5}, were chosen. However, for every value of k from the aforementioned set, the class winners for both clusters always result in ω_2. The possible even values of k always result in ties.

Below is a plot of the decision line and the points from P, C_{ω_a}, and C_{ω_b}.

Figure 1: Feature Plot and Decision Line



Considering both clusters are assigned to ω_2, the decision line seems rather pointless.

(c) Consider two test data points: $(-\frac{3}{4}, \frac{3}{4})$, which belongs to ω_1, and $(\frac{1}{2}, \frac{1}{2})$, which belongs to ω_2. Compute the probability of error based on your k-means clustering.

Seeing as the classifier created from the k-mean clustering algorithm always assigns each new point to ω_2, the probability of error $P(e)$ is simply the probability of getting a point belonging to ω_1, which is the prior $P(\omega_1)$. Based on the prior information—including the new points mentioned in (c)—the total number of points

for ω_1 and ω_2 are 3 and 4, respectively. Thus, it is known from prior information 3 out of 7 points are classified as ω_1, making $P(\omega_1) = \frac{3}{7} \approx 0.4286$. Since $P(e) = P(\omega_1)$, $P(e) \approx 0.4286$.

I heard from a few students that the prior was already specified as .5. If this is indeed the case, $P(e) = .5$

(d) Compute the probability of error based on a k-nearest neighbor rule. How different should this result be from (c) for large k?

The error rate $P(e)$ determined from (c) is also Bayesian error rate P^*, considering $P^* = \min(P(\omega_1|\overrightarrow{x}), P(\omega_2|\overrightarrow{x})) = P(e) = P(\omega_1)$. If k is always a positive and odd integer, then the probability of error for the k-nearest neighbor rule with only two possible classes is bounded by the following equation.

$$P_{bound}(e) = \sum_{i=0}^{(k-1)/2} \binom{k}{i} \left[(P^*)^{i+1}(1-P^*)^{k-i} + (P^*)^{k-i}(1-P^*)^{i+1} \right] \tag{6}$$

$P_{bound}(e)$ is interpreted as the actual probability of error for the k-nearest neighbor rule, seeing as the bound states the error cannot be any worse (*and I would like to finish this rework exam within the available amount of time*).

If the $P(e)$ is in fact .5, P_{bound} only results in 0.5, seemingly for all nonnegative, odd values of k—which is obvious since it's impossible to get an error greater than 0.5.

However, the following table is generated if $P^* \approx 0.4286$, as determined in (b). As k tends to infinity, P_{bound} appears to tend to P^*, the "ideal" error.

$$\begin{bmatrix} k & P_{bound}, \text{ where } P^* \approx 0.4286 \\ 1 & 0.4898 \\ 3 & 0.4848 \\ 5 & 0.4811 \\ 7 & 0.4781 \\ 9 & 0.4756 \\ 11 & 0.4733 \\ 13 & 0.4713 \\ ... & \end{bmatrix} \tag{7}$$

In retrospect, I realize a line plot probably would have worked better, here.

2. **Problem No. 2:** Consider the same 5 data points above.

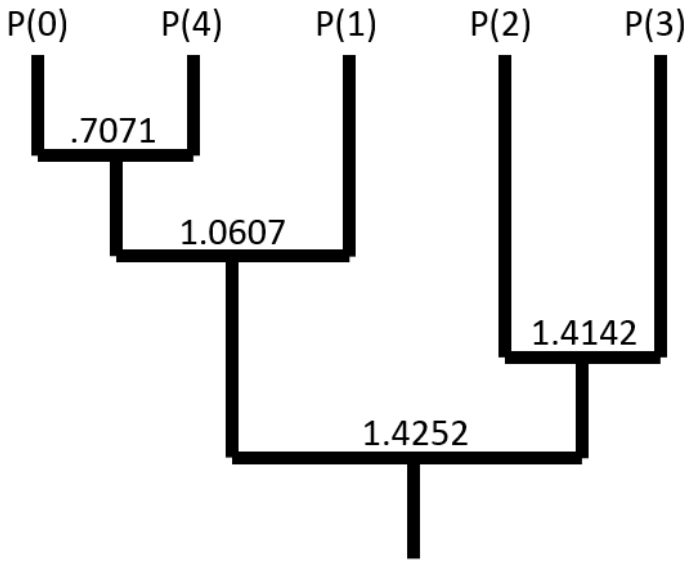
(a) Construct a dendrogram for the data.

The dendrogram presented as the answer to **Problem No. 2** is created with the *Agglomerative Hierarchical Clustering (AHC)* algorithm. The algorithm is implemented as follows.

Every unlabeled point $P(n)$ from the set $P = \{(0,1), (-1,0), (1,0), (0,-1), (-0.5,0.5)\}$ is first labeled as its own cluster. Let's call this set of cluster centers C. Initially, C is set equal to P. In every iteration of the AHC algorithm, the center coordinates in C are merged together based on the two center coordinates closest to each other. The algorithm finally finishes once there is only a single center coordinate remaining in C.

i. **Theoretical Solution:**

Figure 2: Dendrogram created from the AHC algorithm



The dendrogram in Figure 2 starts from the top and then proceeds downward for each iteration. $P(n)$ refers to each point in $P = \{(0,1),(-1,0),(1,0),(0,-1),(-0.5,0.5)\}$. The values shown above each new cluster in the dendrogram are the minimum distances determined from each iteration of the AHC algorithm. The minimum distances are also selected as the *similarity values* for the dendrogram. The similarity values are used in (c) for establishing what the "unsupervised" clusters should be.

ii. **MATLAB© Script:** Listing 2 shows the MATLAB© script that implements the AHC algorithm.

```
% initial the cluster set with P
C = P;

% Cset is a set of structures
createCset = @(C, minD)struct('C', C, 'minD', minD);
Cset = createCset(C, Inf);

% this algorithm must be run until there is only one cluster left
while numel(C(1,:)) ~= 1

    % find the vector(s) the smallest distance
    l = numel(C(1,:));
    D = dist(C) + tril(Inf*ones(l),0);
    [columnD, indicesD] = columnize(D);
    [minD, i] = min(columnD);
    equalSizedV = indicesD(i,:);

    % determine the cluster centers that make up those vectors
    newC = unique(reshape(equalSizedV, ...
        numel(equalSizedV),1));

    % combine the cluster centers to form a new cluster
    newCc = [mean(C(1,newC)) ; mean(C(2,newC))];
```

```
24
    % replace cluster centers with the combined cluster center
26  C(:,newC) = [];
    C(:,end+1) = newCc;
28
    % add data to structure
30  Cset(end+1) = createCset(C, minD);
end
32
```

Listing 2: MATLAB© Script

The results of the script are organized in the table shown in Equation 8 (and from the table, the dendrogram in Figure 2 is constructed). i refers to each iteration of the algorithm.

$$\begin{bmatrix} i & C & \text{Minimum Distance} \\ 1 & \{(0,1),(-1,0),(1,0),(0,-1),(-0.5,0.5)\} & \text{N/A} \\ 2 & \{(-1,0),(1,0),(0,-1),(-0.25,0.75)\} & 0.7071 \\ 3 & \{(1,0),(0,-1),(-0.625,0.375)\} & 1.067 \\ 4 & \{(-0.625,0.375),(0.5,-0.5)\} & 1.4142 \\ 5 & \{(-0.625,-0.625)\} & 1.4252 \end{bmatrix} \tag{8}$$

(b) construct a top-down clustering (e.g., LBG) clustering (you can also think of this as a crude decision tree).

As suggested, the top-down clustering is done through a variation of the *Linde-Buzo-Gray (LBG)* algorithm. Similar to creating a dendrogram with the AHC algorithm, the LBG algorithm produces a set of clusters for each iteration of the algorithm. The difference of course is the LBG algorithm initializes with a single cluster in its cluster set C. For every iteration, clusters associated with at least with two points are divided into two clusters.

The following explains how each cluster is divided. The center coordinate of the cluster is first separated into two center points. Let's say $C(n)$, where n is simply a nonnegative integer used to refer to a particular cluster in the set C, is the center coordinate of interest. $C_1(n)$ and $C_2(n)$ refer to the two center points created from $C(n)$. The following equation demonstrates how $C_1(n)$ and $C_2(n)$ are determined.

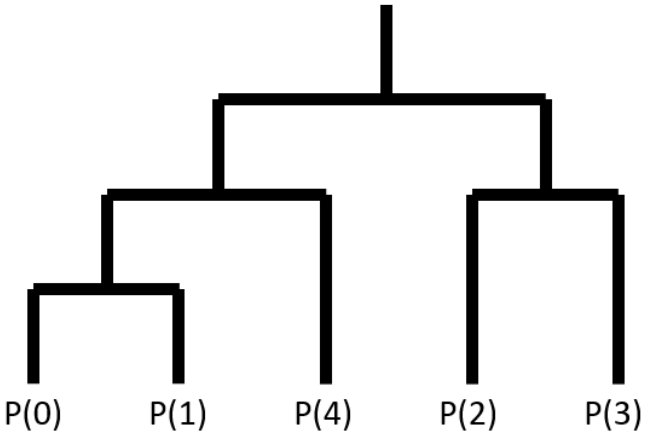
$$\begin{aligned} C_1(n) &= C(n) \times (1+\epsilon) \\ C_2(n) &= C(n) \times (1-\epsilon) \end{aligned} \tag{9}$$

The set of points associated with the cluster $C(n)$ are distributed between $C_1(n)$ and $C_2(n)$ through the k-means algorithm. $C_1(n)$ and $C_2(n)$ are also updated as part of the k-means algorithm and then replace $C(n)$ in the set of clusters C.

It is worth mentioning there is an issue wherein a cluster $C(n)$ with two or more points is not properly distributed with the k-means algorithm and either $C_1(n)$ or $C_2(n)$ is assigned all the points of $C(n)$. In order to overcome the issue, the LBG algorithm is implemented such that one of the points associated with $C(n)$ is automatically assigned to the cluster with zero assigned points.

i. **Theoretical Solution:** The tree shown in Figure 3 is created from the MATLAB© script that implements the LGB algorithm. It is important to note the tree starts from the top and then travels downward for each iteration.

Figure 3: Top-Down Clustering Tree created from the LGB algorithm



The table shown below contains the center points of each cluster in C and for each iteration.

$$\begin{bmatrix} i & C \\ 1 & \{(-0.1, 0.1)\} \\ 2 & \{(-0.5, 0.5), (0.5, -0.5)\} \\ 3 & \{(-0.5, 0.5), (-0.5, 0.5), (1, 0), (0, -1)\} \\ 4 & \{(0, 1), (-1, 0), (-0.5, 0.5), (1, 0), (0, -1)\} \end{bmatrix} \tag{10}$$

ii. **MATLAB© Script:**

```
Pcolumns = numel(P(1,:));

% initialize set of cluster centers as a single cluster
createCstruct = @(points)struct('P', points, 'c', mean(points,2));
Cstructs = createCstruct(P);

% needed for the separation of the data
epsilon = .3;

% keep iterating the algorithm until the number of cluster centers
% equal that of the original set of points
while numel(Cstructs) ~= Pcolumns
    newCstructs = {};
    for n=1:numel(Cstructs)
        if numel(Cstructs(n).P(1,:)) > 1
            Cdivide = [Cstructs(n).c*(1+epsilon) ...
                Cstructs(n).c*(1-epsilon)];
            D = dist([Cstructs(n).P Cdivide]);
            D = D(1:(end-2),(end-1):end);
            decider = D(:,1) <= D(:,2);
            if sum(decider) == numel(decider)
                decider(end) = 0;
            end
            newCstructs{end+1}  = ...
                [createCstruct(Cstructs(n).P(:, decider==1)) ...
                createCstruct(Cstructs(n).P(:, decider==0))];
        else
            newCstructs{end+1} = Cstructs(n);
        end
    end
    Cstructs = [newCstructs{:}];
```

```
end
```

Listing 3: MATLAB© Script

(c) If you were to use your dendrogram to do unsupervised clustering of the data, what clusters would you create (specify them by the mean and the elements associated with the cluster).

Based on the result determined in (a), the clusters C would include the cluster $C(0) = (-0.625, 0.375)$ whose associated points are $P_{C(0)} = \{P(0), P(4), P(1)\} = \{(0,1), (-0.5, 0.5), (-1,0)\}$, and the cluster $C(1) = (0.5, -0.5)$ whose associated points are $P_{C(1)} = \{P(2), P(3)\} = \{(-1,0), (1,0)\}$.

The reason for selecting the clusters C is specifically due to the results shown in the dendrogram presented in Figure 2 and the table shown in Equation 8. The similarity value changes from 1.0607 to 1.4142 on iterations i 3 to 4, respectively. The difference between the two similarity values is larger than the difference calculated from any other pair of similarity values from adjacent iterations.

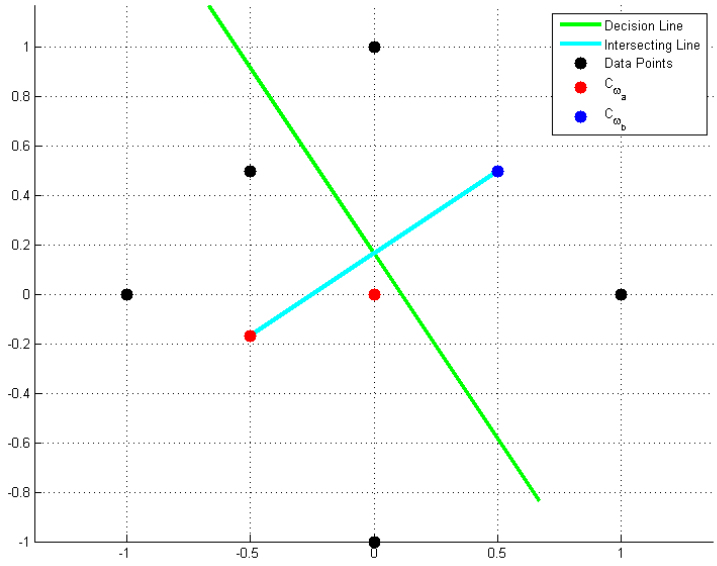
This observation is significant because a larger change in distance between clusters indicates the clusters being merged are relatively far from each other. Thus, it is assumed the iteration prior to the largest change in distance—that is, the largest change in similarity value—contains the most "reasonable" set of clusters in terms of the distance between the points.

Tried my best to explain my reasoning, Dr. Picone.

(d) Suppose (0, 1) and (1, 0) occur 5 times more often than the rest of the data points. How would you adjust your strategy for clustering the data? How would that impact your decision regions?

The strategy for improving the initial guesses of the clusters' center coordinates with k-means would stay the same; however, the strategy for determining the number of clusters and the initial guesses of each cluster would need modification.

Figure 4: Altered Feature Plot and Decision Line



The MATLAB© scripts shown in (1a) and (2a) are altered such that the points $P(0) = (0, 1)$ and $P(2) = (1, 0)$ occur 5 times more often and are assigned the same classes. The resultant decision line is shown in Figure 4. The decision line seems to look exactly like the decision line created from the original set of points (see Figure 1).

What changes, however, are the class assignments, depending on the value of k.

$$\begin{bmatrix} k & \omega_a \text{ Assigned Class} & \omega_b \text{ Assigned Class} \\ 1 & \omega_1 & \omega_1 \\ 3 & \omega_2 & \omega_1 \\ 5, 7, 9 & \omega_1 & \omega_1 \\ 11 & \omega_1 & \omega_2 \\ 13 & \omega_2 & \omega_2 \end{bmatrix} \tag{11}$$

Running the AHC algorithm generates results similar to the table shown in Equation 8. However, the AHC algorithm takes the first several iterations to combine the repetitive points. The first several iterations all have similarity values of 0 since the repetitive points have no distance between each other. The next highest similarity value is 0.7071, and the iteration previous to getting the next highest similarity value is the same iteration whose clusters are equivalent to the original set of points P.

With the aforementioned thoughts in mind, it might be a good idea to somehow "scatter the redundant data" (or keep drawing more points so that the set of points becomes very large and more dense). Scattering in this context implies shift the redundant points such that they are not overlapping each other, but keep them close enough that the data doesn't become "too distorted". The fact that so many of the same data points may very well be important information, suggesting there may be a peak in the data's underlining model.