Math 444: Homework 2

Iris Zhang

February 13, 2020

Problem 1

The Matlab code for K-means are attached in the appendix I and for K-medoids are in the appendix II.

Problem 2

For **WineData.mat**, first we look at the original data set of the first few attributes with their labels, see Figure 1. From the graph, we can see that even with the different color there are not exactly clear clusters.

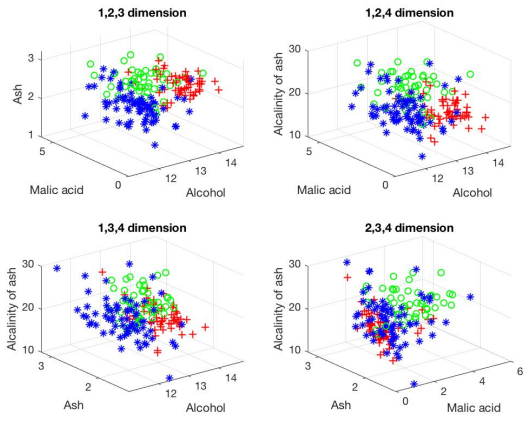

Figure 1: original data set with their true annotation

So, we then try to use the k-means algorithm that we developed in Problem 1 to separate the data set into 3 clusters. We plot the results of the same set of attributes in Figure 2.

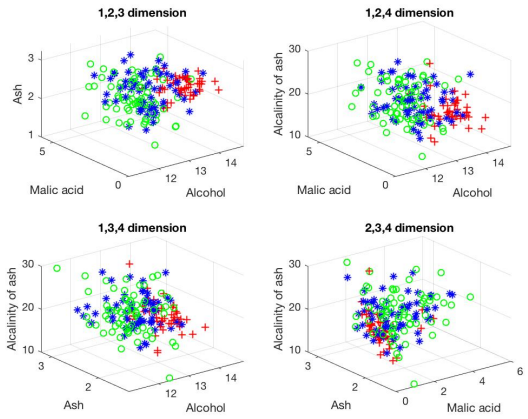

Figure 2: new clusters after doing k-means

Compared Figure 2 with Figure 1 we can see that the clusters we obtained using k-means are almost the same, indicating that the k-means works really well in separating this set of data.
Now, we try to use the k-medoids algorithm written in problem 1 to divide the data into 3 clusters. We plot the results of the same set of attributes in Figure 3.

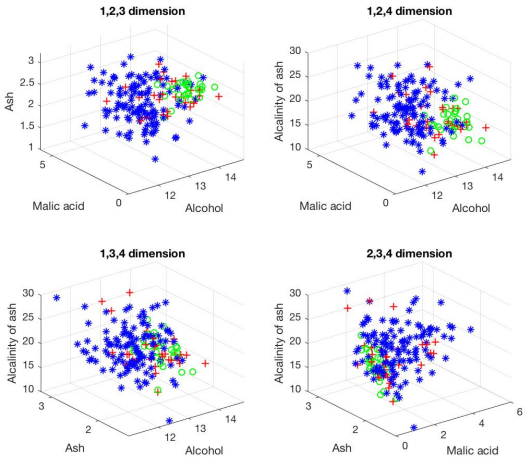

Figure 3: new clusters after doing k-medoids

Compared Figure 3 with Figure 1, we can see that the k-medoids gives us a relatively clear one cluster but the other two are kind of mixed with each other and have differences between the original assigned clusters. This shows that the k-medoids does not well so well in this situation.

Then we do a PCA of the raw data. The graph of all the singular values are in Figure 4. We can see that the first one is dominant.

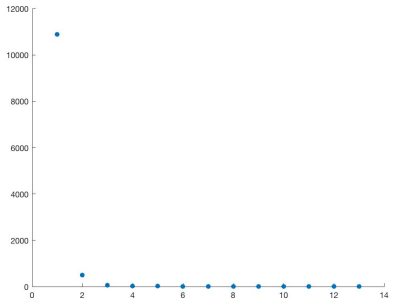

Figure 4: singular values

So, we continue to do a plot of the first three principle components, see Figure 5. We can see that the purple one is more separated from the other two, but the yellow and green are mixed.

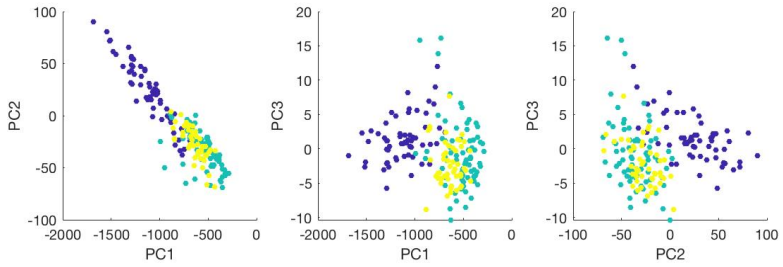

Figure 5: singular values

The color for each type of wine is randomly since it depends on the initialization of the characteristic vectors, which is randomly assigned.
In conclusion, I think that the three wine types were NOT EASY to cluster. As I mentioned above, there are not exactly separated three clusters in Figure 1 even with the color difference.

Problem 3

For the **CardiacData.mat**, we will use the K-medoids to identify the clusters. We first write the distance matrix between the patients using a dissimilarity index, formulated as: $d_{i,j} = \frac{n_{10}+n_{01}}{n_{10}+n_{01}+n_{11}+n_{00}}$ since all the entries are represented in binary numbers and we do not assume that the matching zeros are less important than matching ones. I include a snippet of my MATLAB code.

```
[p,n]= size(X_attributes);
D = zeros(n,n);
for i = 1:n
    for j = 1:n
        xi = X_attributes(:,i);
        xj = X_attributes(:,j);
        n_10 = sum(xi == 1 & xj == 0);
        n_01 = sum(xi == 0 & xj == 1);
        n_00 = sum(xi == 0 & xj == 0);
        n_11 = sum(xi == 1 & xj == 1);
        dij = (n_10 + n_01)/(n_10 + n_01 + n_11 + n_00);
        D(i,j) = dij;
    end
end
```

Then, we run the k-medoids algorithm to cluster the data in two groups. In order to see whether this algorithm works well in this set, we Write a matrix C_{2X2} as described in the problem. I include a snippet of my MATLAB code.

```
normal = find(index == 1)
abnormal = find(index == 2)
C = zeros(2,2);
C_11 = 0;
C_12 = 0;
C_21 = 0;
C_22 = 0;
for i = 1:n
    if (I(1,i) == 1 & ismember(n, abnormal))
        C_11 = C_11+1;
    end
    if (I(1,i) == 1 & ismember(n, normal))
        C_12 = C_12+1;
    end
    if (I(1,i) == 0 & ismember(n, abnormal))
        C_21 = C_21+1;
    end
    if (I(1,i) == 0 & ismember(n, normal))
        C_22 = C_22+1;
    end
end
C = [C_11 C_12; C_21 C_22]
```

The matrix C we get indicates that the algorithm runs well since $C(1,1) + C(2,1) = 187$ or $C(1,2) + C(2,2) = 187$, depending on the initialization, see Figure 6. Here we do not use the "majority" method, so we calculate the sum.

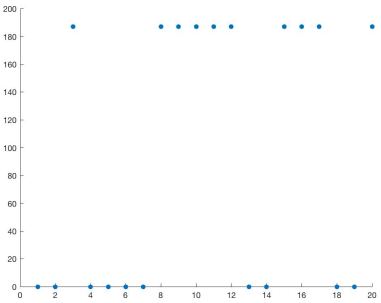

Figure 6: results of C(1,1)+C(2,1)

Now we use PCA to reduce the dimensionality. From Figure 7, we can see that the first singular value is dominant.

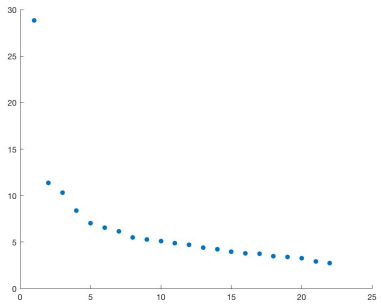

Figure 7: singular values

Then, we plot the first three principal components, see Figure 8.

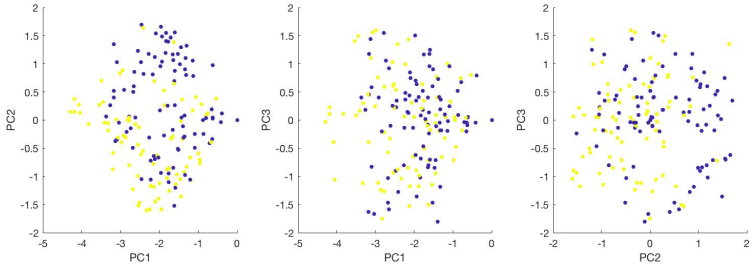

Figure 8: first three principal components

Appendix I

```
% k-means algorithm
function [centroid,partition_index] = k_means(X,k,tau)
% initialization
partition_index = zeros(1,size(X,2));
coherence = 10^16;
  % picking randomly k data vectors to be the initial medoids
  centroid_index = (floor((size(X,2)).*rand(k,1) + 1))';
  centroid = X(:,centroid_index);
  [q, partition_index] = min(centroid,[],2);
  tightness = sum(q.^2);
  % Repeat the initialization 20 times
  % pick the initial clustering that corresponds to the lowest tightness.
  partition_index_t = zeros(1,size(X,2));
  for t = 1:20
    centroid_index_t = (floor((size(X,2)).*rand(k,1) + 1))';
    centroid_t = X(:,centroid_index_t);
    [q_t, partition_index_t] = min(centroid_t,[],2);
    tightness_t = sum(q_t.^2);
    if tightness_t<tightness
       tightness = tightness_t;
       centroid = centroid_t;
       partition_index = partition_index_t;
    end
  end
partition_index = partition_index';
A = zeros(k+2,size(X,2));

% iteration
while(coherence > tau)
    for i = 1:size(X,2)
        for j = 1:k
            A(j,i) = norm(X(:,i) - centroid(:,j),2);
        end
        [Distance, cluster] = min(A(1:k,i));
        A(k+1,i) = cluster;
        A(k+2,i) = Distance;
    end

    for i = 1:k
        num = A(k+1,:) == i;
        centroid(:,i) = mean(X(:,num),2);
    end
     [q_new, cluster] = min(centroid,[],2);
      Qnew = sum(q_new.^2);
      coherence = abs(Qnew - tightness);
      tightness = Qnew;
end
partition_index = A(k+1,:)
end
```

Appendix II

```
% k-medoids algorithm
function [medoids,cluster] = k_medoids(k,D,tau)
% initialization
  cluster = zeros(1,size(D,2));
  coherence = 10^16;
  MAX_ITER = 100000;
  % picking randomly k data vectors to be the initial medoids
  medoids_index = (floor((size(D,2)).*rand(k,1) + 1)');
  medoids = D(:,medoids_index);
  for r = 1:size(D,2)
      % assign every vector into different clusters
      [val,ind] = min(medoids(r,:));
     cluster(:,r) = ind;
  end
  % computing the corresponding tightness.
  tightness = 0;
  for l = 1:k
      cluster_index = find(cluster==l);
      for j = 1 : size(cluster_index,2)
          tightness = tightness+D(cluster_index(:,j),medoids_index(l));
      end
  end
 % Repeat the initialization 20 times
 % pick the initial clustering that corresponds to the lowest tightness.
  cluster_t = zeros(1,size(D,2));
  for t = 1:20
    medoids_index_t = (floor((size(D,2)).*rand(k,1) + 1))';
    medoids_t = D(:,medoids_index_t);
    for r = 1:size(D,2)
      [val,ind_t] = min(medoids_t(r,:));
      cluster_t(:,r) = ind_t;
    end
    tightness_t = 0;
    for l = 1:k
      cluster_index_t = find(cluster==l);
      for j = 1 : size(cluster_index_t,2)
         tightness_t = tightness_t+D(cluster_index_t(:,j),medoids_index_t(l));
      end
    end
    if tightness_t<tightness
       tightness = tightness_t;
       medoids = medoids_t;
       cluster = cluster_t;
    end
  end
  D_m = D(:,medoids_index);
  [q, cluster] = min(D_m,[],2);
   tightness = sum(q.^2);
 % iteration
 t=0;
 while (coherence > tau && t<MAX_ITER)
```

```
    for l = 1: k
      I_{l} = find(cluster == l);
      D_{l} = D(I_{l},I_{l});
      [~,j] = min(sum(D_{l}));
      D_m(:,l) = D(:,j);
      medoids_index(l) = j;
    end
    [q_new, cluster] = min(D_m,[],2);
     Qnew = sum(q_new.^2);
     coherence = abs(Qnew - tightness);
     tightness = Qnew;
     t= t+1;
 end
medoids = D(:,medoids_index);
cluster= cluster';
end
```