Hierarchical Agglomerative Clustering

Hierarchical clustering is based on the dendrogram. In agglomerative approach, we start with each observation (data) in singleton cluster, and then sequentially merging cluster by some "linkage" methods in bottom-up fashions.

We show the two linkage methods, which are single link and complete link. For single linkage, two clusters in each step are combined together whose two closet members have the smallest "distance". For complete linkage, two clusters in each step with the smallest maximum pairwise "distance" are merged.

To measure the "distance", datasets are converted to relational data before clustering. We implemented four distance measurements, which are Euclidean distance, Manhattan distance, cosine, and correlation. The first two and the last two are methods for dissimilarity and similarity measurements respectively. As for cosine and correlation measures, we do a mapping relation from similarity to dissimilarity (i.e., one minus similarity result).

We take Iris dataset for clustering, and the feature space is shown in Figure 1. One can find that there is only one class linearly separable from other two classes, whereas the others are not linearly separable from each other.

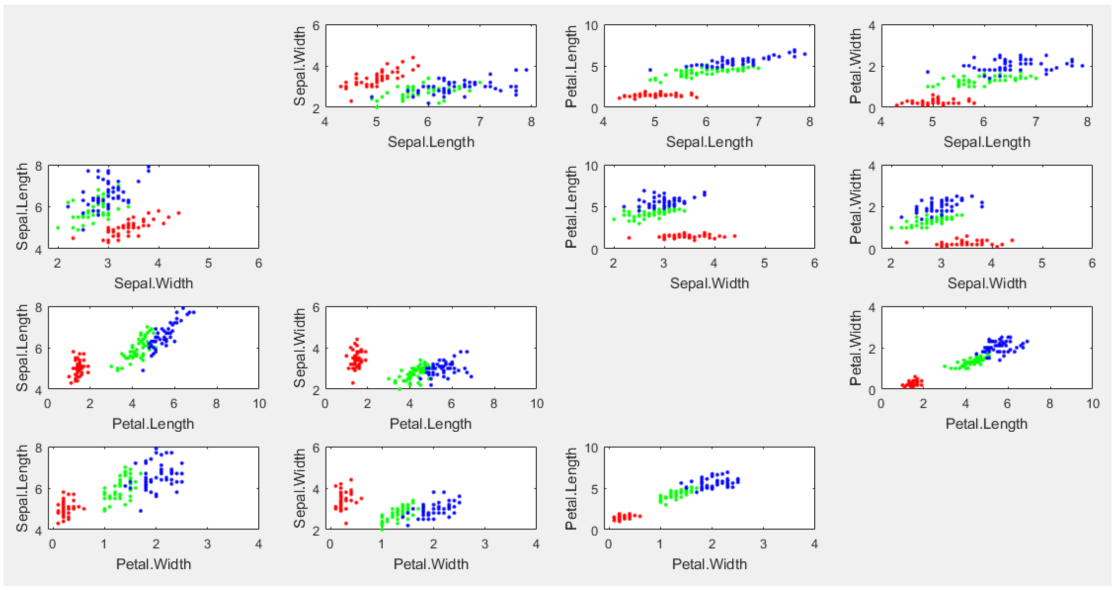


Figure 1. The feature space of Iris dataset.

In terms of linkage, if there is a cut between the second-from-last and third-from-last linkages in one dendrogram, then we can divide the dataset into three clusters.

Applying single-linkage clustering to Iris dataset, it tends to form two clusters as depicted in Figure 2 (i.e., one of the three clusters will include few data points). From Figure 3, we can see that data points lying in the lower right diagonal area are almost clustered into the same group. In each step for single-linkage clustering, merging two clusters with highest similarity will encourage "chaining effect". However, similarity is usually not transitive. That is to say, if A is similar to B and B is similar to C, then it does not imply that A is similar to C. For complete-linkage clustering, it tends to break large clusters. Therefore, it works far worse than single-linkage when the data points we mentioned above belong exactly to the identical class.

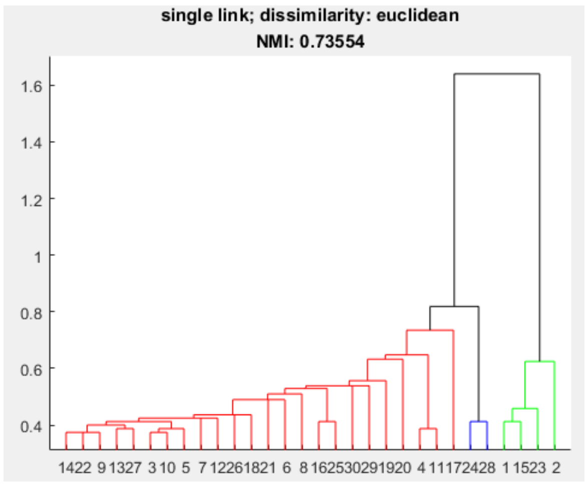


Figure 2. The dendrogram for single-linkage clustering with Euclidean measurement.

Regarding the distance measures, it makes an influence on the clustering results. A feature vector is consisted of two components, "magnitude" and "orientation". Euclidean distance is the shortest path between two data points while Manhattan distance is the path length along each feature. Thus, they are judgments for "magnitude", independent of orientation. For data with zero mean, cosine similarity and correlation test are exactly the same measurement. They are perceptible to "orientation".

Table 1. Correlation coefficients of Iris dataset.

	Attribute 1	Attribute 2	Attribute 3	Attribute 4
Attribute 1	1	-0.1094	0.8718	0.8180
Attribute 2	-0.1094	1	-0.4205	-0.3565
Attribute 3	0.8718	-0.4205	1	0.9628
Attribute 4	0.8180	-0.3565	0.9628	1

As shown the correlation coefficients of each two attributes listed in Table 1, some components are in high correlation. That is to say, if two data points belong to the same cluster, then their feature vectors might close to each other (i.e., with lower dissimilarity). However, for Euclidean distance, different vector magnitudes cause higher dissimilarity compared to that generated from correlation measurement. From Figure 3 and Figure 4, clustering results with cosine and correlation measurements are a little bit better than those adopting Euclidean distance. Notably, for Iris dataset, Manhattan distance performed the best clustering results among the four proximity measures we adopted. It can be said that this approach takes more information about the dissimilarity between two data points along each feature. Generally speaking, for high dimension data, Manhattan distance works better than Euclidean distance. Note that data points before clustering are normalized, while those shown in clustering results are actual data points.

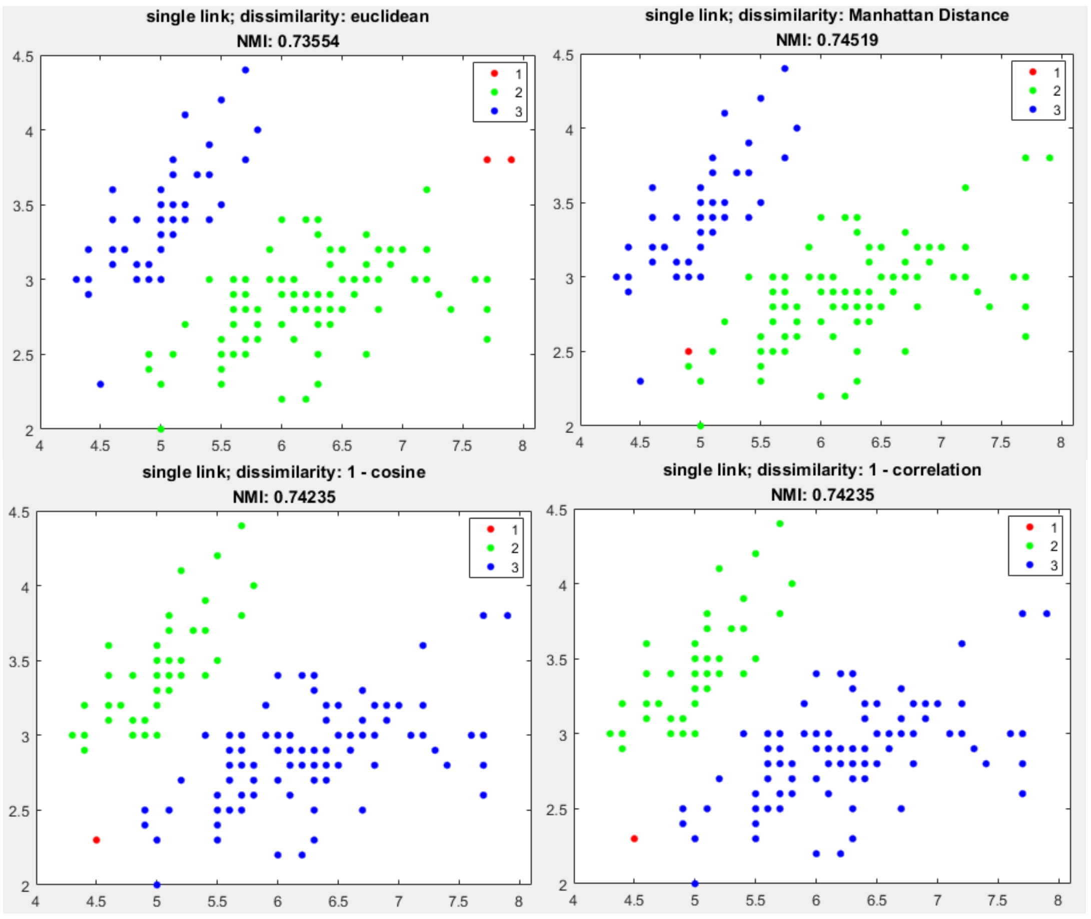


Figure 3. The clustering reuslts for single link.

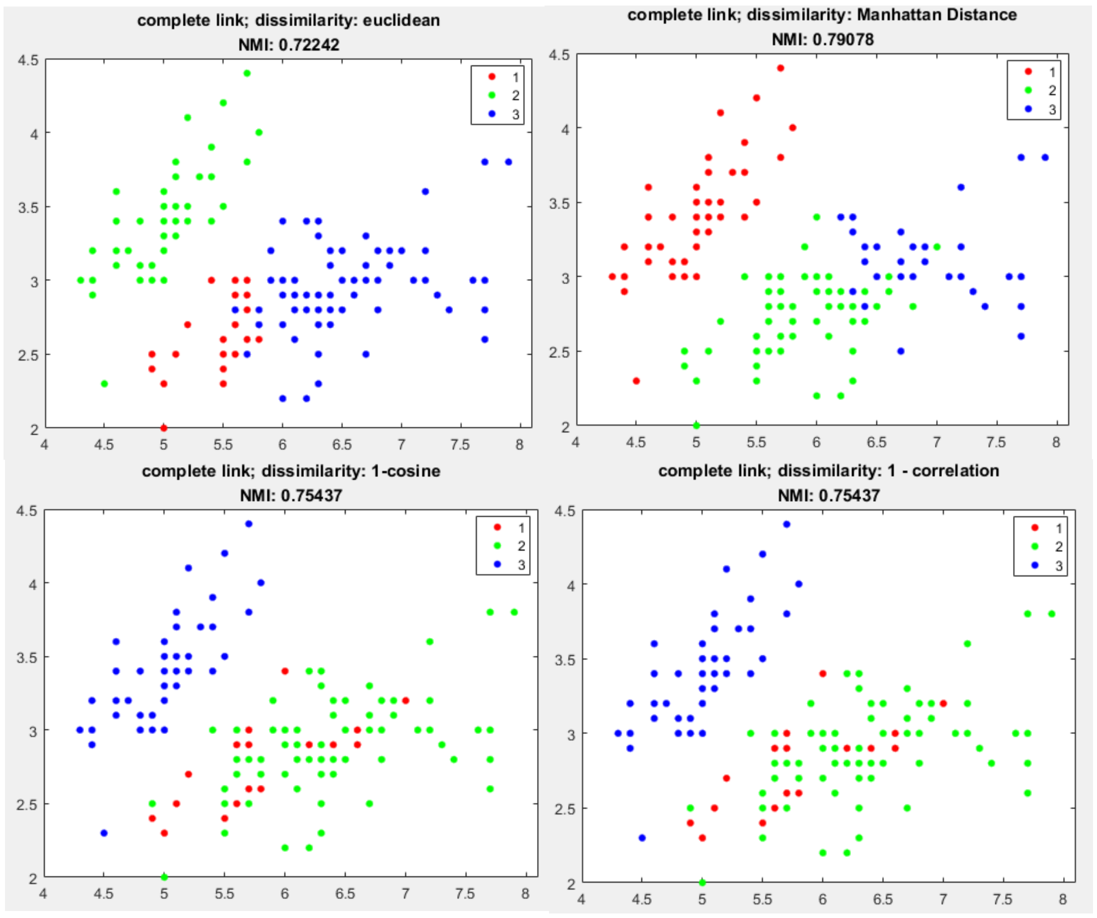


Figure 4. The clustering reuslts for complete link.

Interestingly, for single-linkage clustering, we try to partition Iris dataset into two clusters through a horizontal cut between the last and second-from-last linkages in the dendrogram. NMI for the clustering results are higher than those shown in Figure 3. We presented these NMIs in Table 2.

Table 2. NMIs for 2-clusters with respect to different distance measurements.

	Euclidean	Manhattan	Cosine	Correlation
Single-link	0.76117	0.76117	0.76117	0.76117

In this section, our implementation includes four proximity measures (Euclidean distance, Manhattan distance, Cosine similarity, and Correlation test), single-link clustering, complete-link clustering, and linkage for dendrogram. We use existing Matlab functions for NMI evaluation and dendrogram visualization. For either single-linkage clustering or complete linkage clustering, they all perform inferior results for the dataset we used in the previous assignments (NMIs are around 0.01 to 0.1).

DBSCAN

For DBSCAN, we first need to determine two parameters, *Radius* and *Pthres*. Radius is defined as the maximum distance between two samples for one to be considered as in the neighborhood of the other. Pthres is defined as the number of samples (including the point iteself) in a neighborhood for a point to be considered as a core point. The point reachable from one core point is clustered into the same group and the point assigned to no cluster is labeled as the noise point.

Distance meansurement makes great influence on the choice for *Radius*. Assume that *Pthres* is held fixed, it is still hard to find the suitable value for *Radius*. For convenience, we take only the mean of the distance matrix (denoted as *M*) as basis, and test the all selected multipliers. In our experiments, we adopted values from 0.01*M* to *M* with the step size 0.01*M*. In terms of *Pthres*, the larger value is, the more the number of cluster will be generated. It is because the larger *Pthres* makes it hard to find more points in the same clusters. In our experiments, we used a naïve way to go through all possible values for *Pthres* (i.e., from 1 to the number of total data points).

As shown in Figure 5, we take Iris dataset for clustering and DBSCAN perfoms acceptable clustering results through the four proximity meaures. Except for the complete-linkage clustering result with Manhattan distance measurement, DBSCAN outperforms single-linkage and complete linkage clusterings. Note that data points before clustering are normalized, while those shown in clustering results are actual data points (noise points are colored in red and labeled as 0).

Since there are too much possible combinations for two clustering parameters and we only simulated for some values, we cannot give the concrete performance comparisons between these four distance measurements. Furthermore, DBSCAN is sensitive to clustering parameters. We take one of the dataset we used in the previous assignments, Wireless Indoor Localizatio dataset (4-class 7-dimension dataset), for clustering. As shown in Figure 6, a little change of *Radius* (0.01*M*) causes larger decrease for NMI (0.10806) and so does *Pthres*. Therefore, to choose the appropriate parameters for DBSCAN, sufficient understanding of the dataset must be required.

All in all, compared to those agglomerative clustering approach we mentioned above, DBSCAN shows the flexibility of clustering shapes and performs relatively acceptable clustering results.

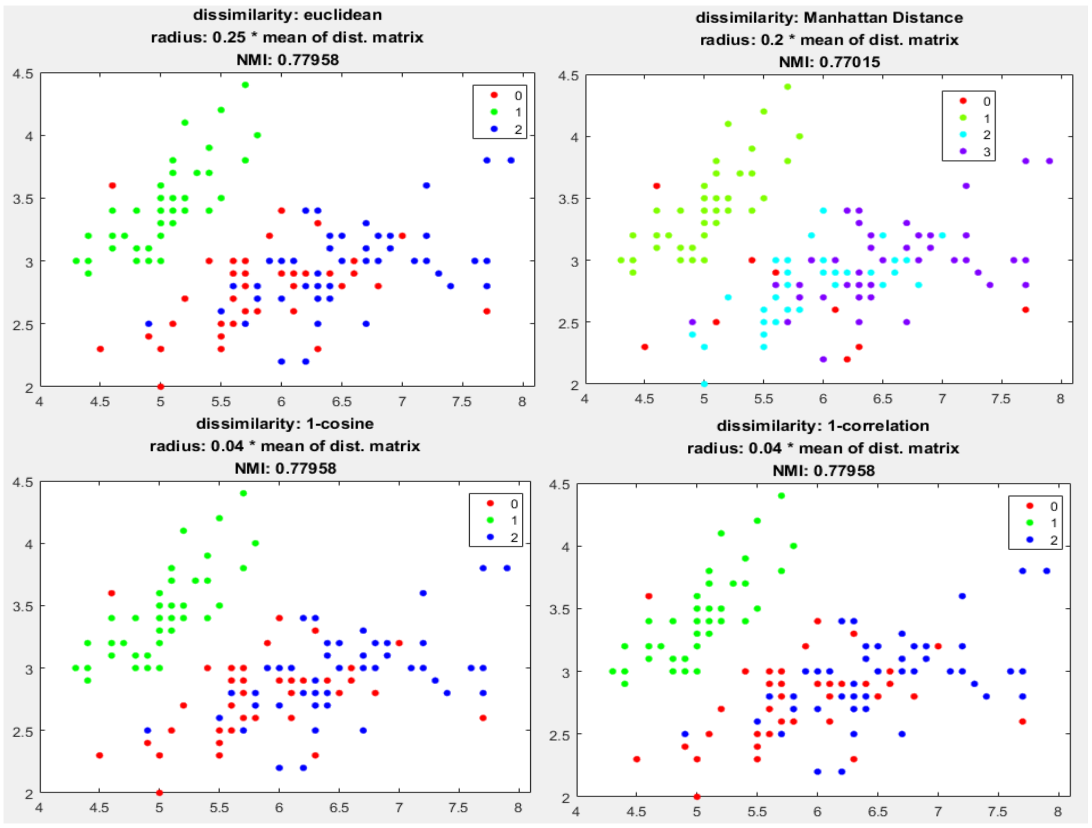


Figure 5. The clustering reuslts for DBSCAN.

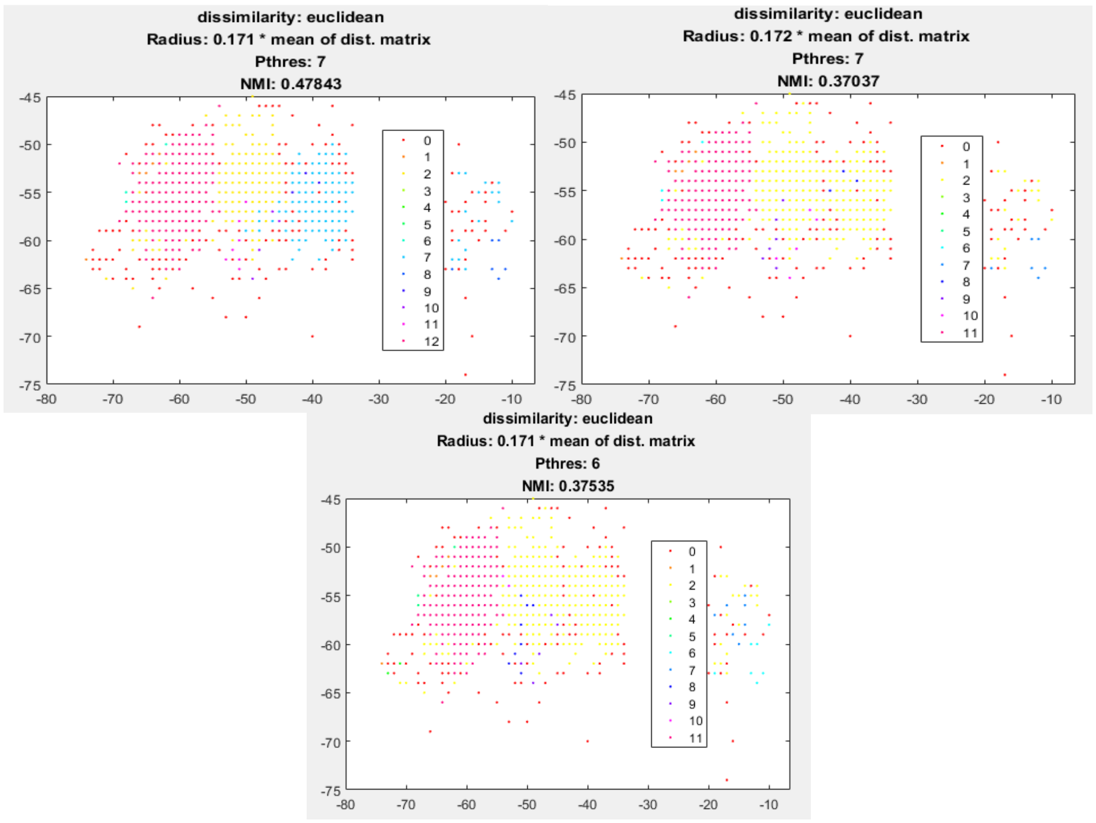
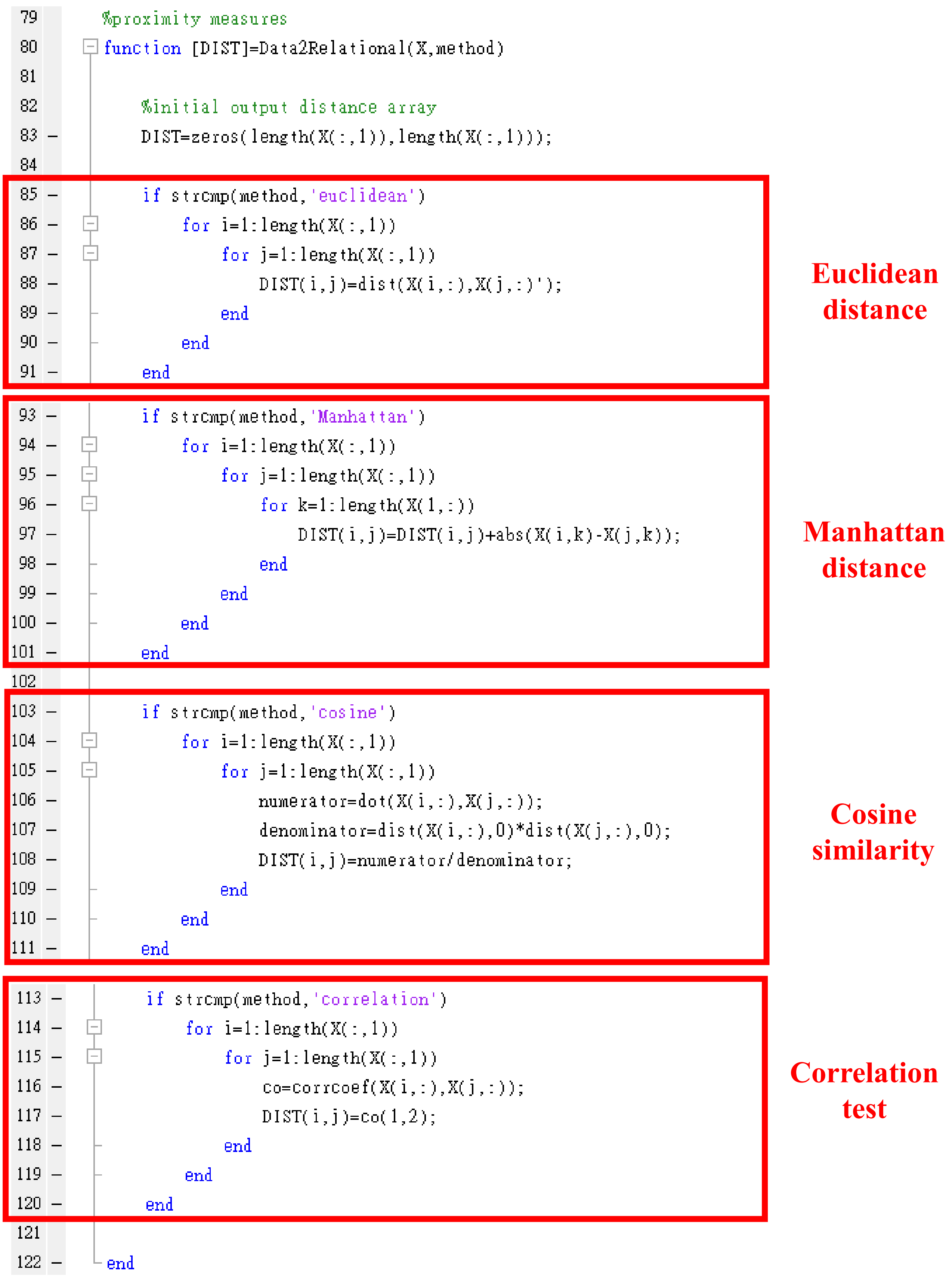
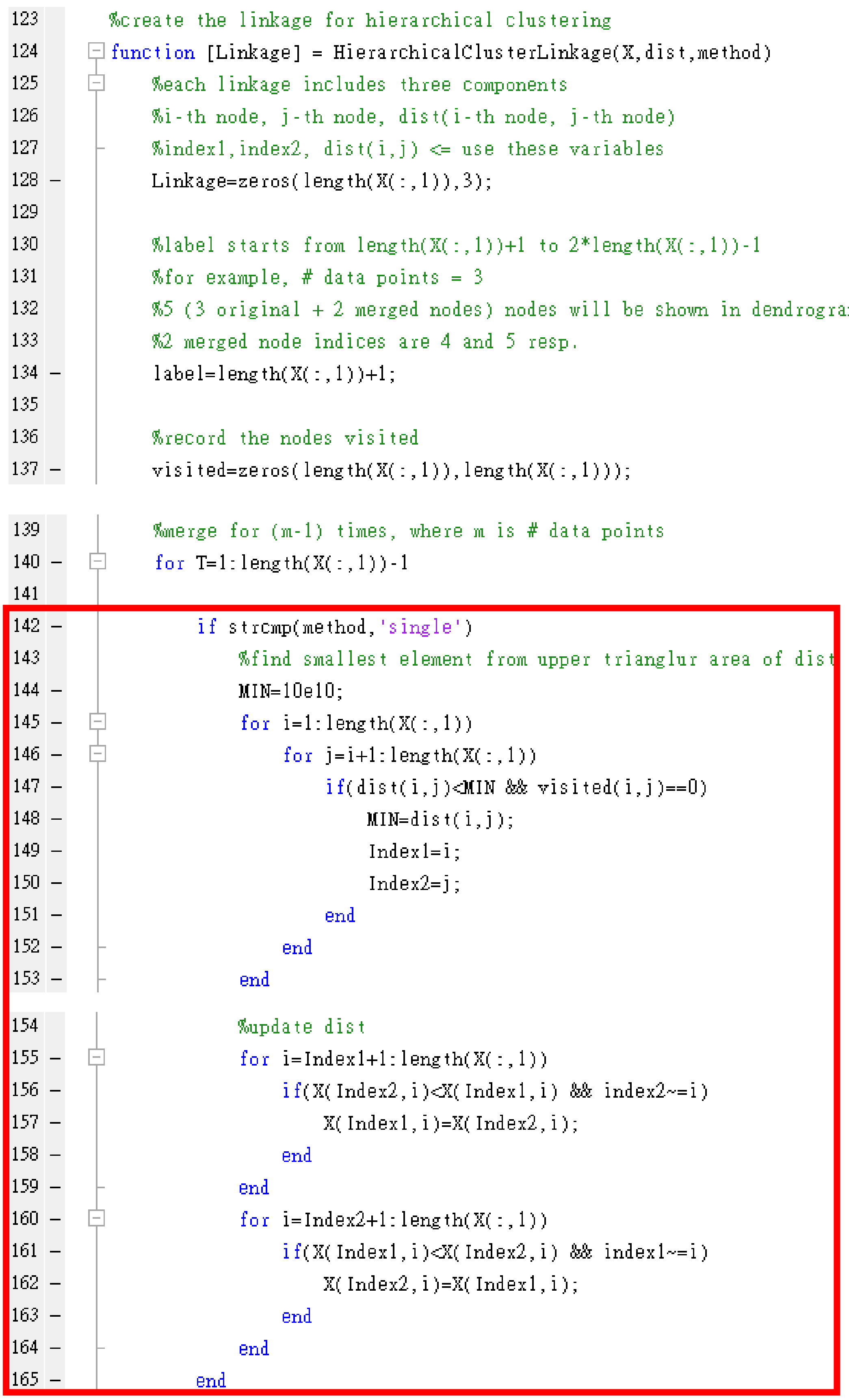


Figure 6. The clustering reuslts for DBSCAN with different clustering parameters.

Appendix

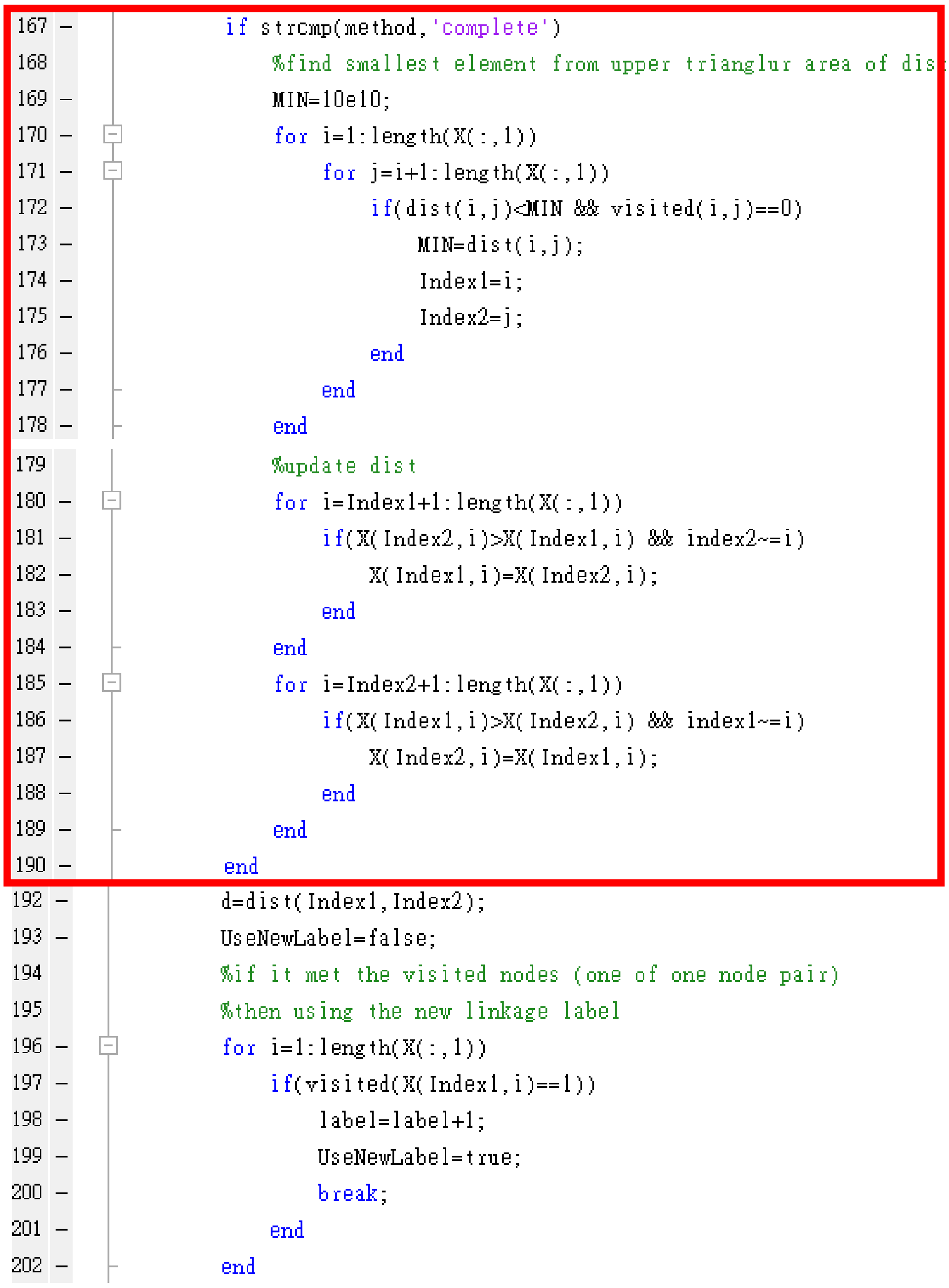


```matlab
%proximity measures
function [DIST]=Data2Relational(X,method)

    %initial output distance array
    DIST=zeros(length(X(:,1)),length(X(:,1)));

    if strcmp(method,'euclidean')
        for i=1:length(X(:,1))
            for j=1:length(X(:,1))
                DIST(i,j)=dist(X(i,:),X(j,:)');
            end
        end
    end
    if strcmp(method,'Manhattan')
        for i=1:length(X(:,1))
            for j=1:length(X(:,1))
                for k=1:length(X(1,:))
                    DIST(i,j)=DIST(i,j)+abs(X(i,k)-X(j,k));
                end
            end
        end
    end

    if strcmp(method,'cosine')
        for i=1:length(X(:,1))
            for j=1:length(X(:,1))
                numerator=dot(X(i,:),X(j,:));
                denominator=dist(X(i,:),0)*dist(X(j,:),0);
                DIST(i,j)=numerator/denominator;
            end
        end
    end
     if strcmp(method,'correlation')
         for i=1:length(X(:,1))
             for j=1:length(X(:,1))
                 co=corrcoef(X(i,:),X(j,:));
                 DIST(i,j)=co(1,2);
             end
         end
     end

end
```



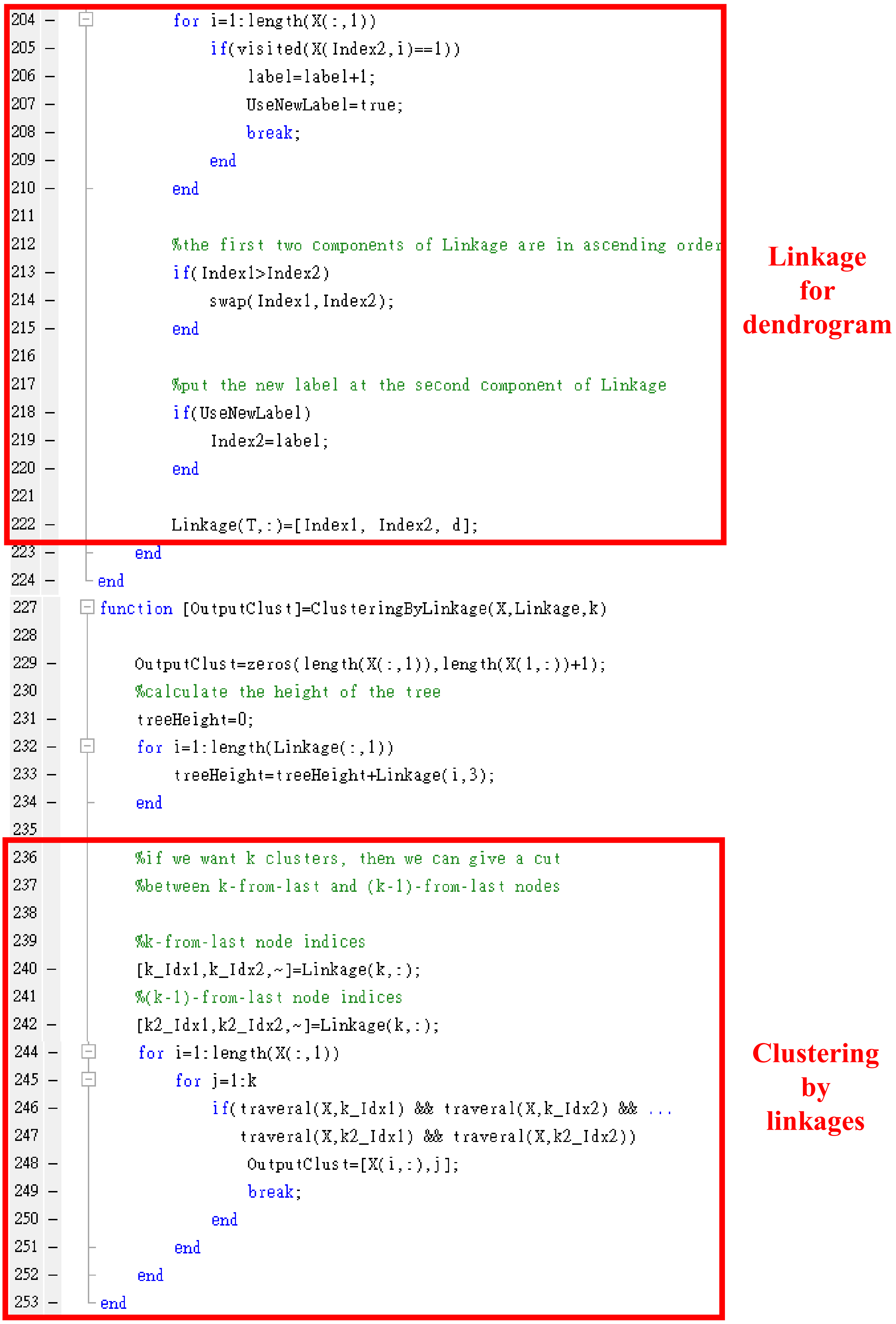
```
%create the linkage for hierarchical clustering
function [Linkage] = HierarchicalClusterLinkage(X,dist,method)
    %each linkage includes three components
    %i-th node, j-th node, dist(i-th node, j-th node)
    %index1,index2, dist(i,j) <= use these variables
    Linkage=zeros(length(X(:,1)),3);

    %label starts from length(X(:,1))+1 to 2*length(X(:,1))-1
    %for example, # data points = 3
    %5 (3 original + 2 merged nodes) nodes will be shown in dendrogram
    %2 merged node indices are 4 and 5 resp.
    label=length(X(:,1))+1;

    %record the nodes visited
    visited=zeros(length(X(:,1)),length(X(:,1)));

    %merge for (m-1) times, where m is # data points
    for T=1:length(X(:,1))-1

        if strcmp(method,'single')
            %find smallest element from upper trianglur area of dist
            MIN=10e10;
            for i=1:length(X(:,1))
                for j=i+1:length(X(:,1))
                    if(dist(i,j)<MIN && visited(i,j)==0)
                        MIN=dist(i,j);
                        Index1=i;
                        Index2=j;
                    end
                end
            end

            %update dist
            for i=Index1+1:length(X(:,1))
                if(X(Index2,i)<X(Index1,i) && index2~=i)
                    X(Index1,i)=X(Index2,i);
                end
            end
            for i=Index2+1:length(X(:,1))
                if(X(Index1,i)<X(Index2,i) && index1~=i)
                    X(Index2,i)=X(Index1,i);
                end
            end
        end
```

Single-linkage clustering

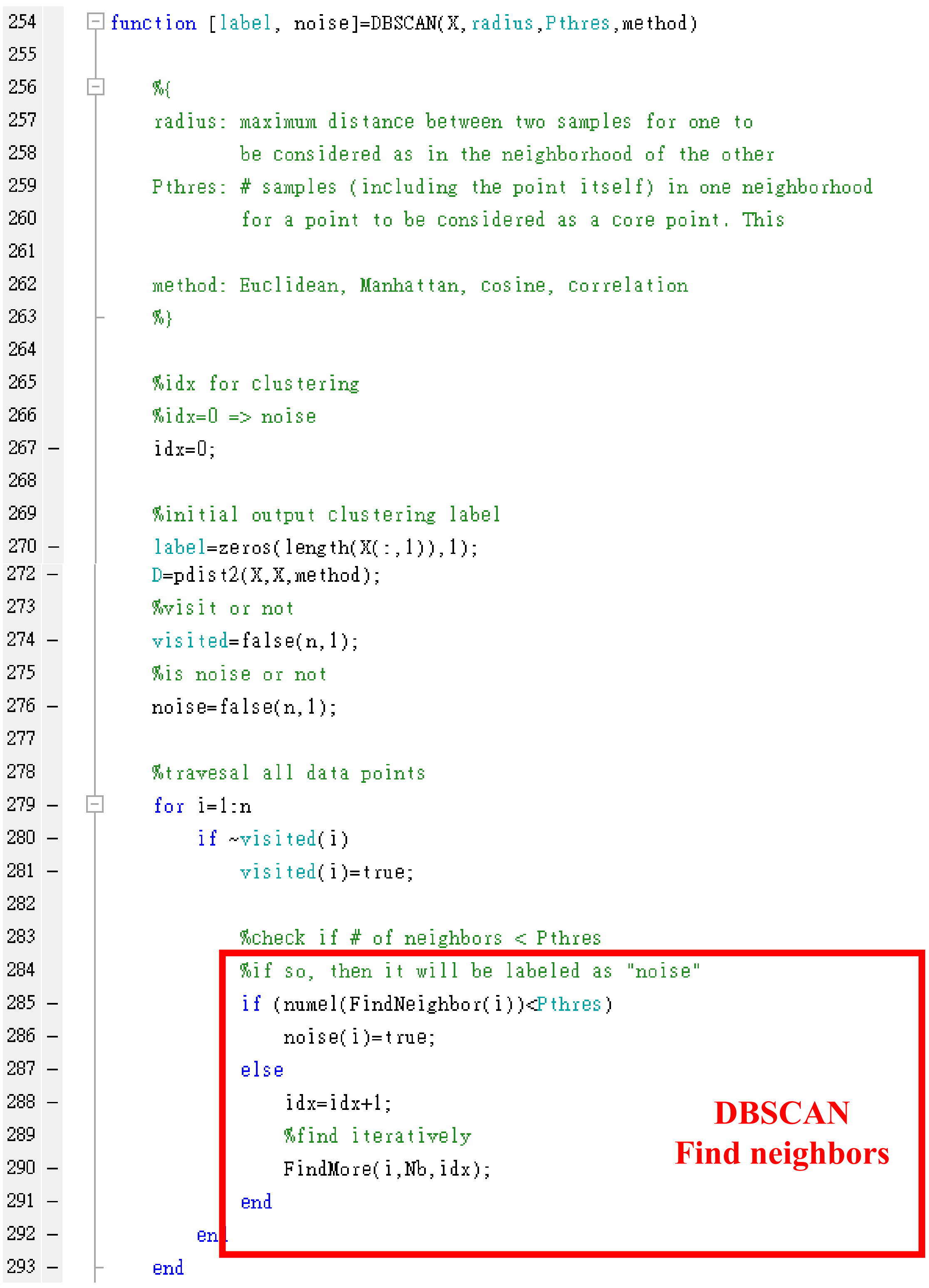


```
if strcmp(method,'complete')
    %find smallest element from upper trianglur area of dis
    MIN=10e10;
    for i=1:length(X(:,1))
        for j=i+1:length(X(:,1))
            if(dist(i,j)<MIN && visited(i,j)==0)
                MIN=dist(i,j);
                Index1=i;
                Index2=j;
            end
        end
    end
    %update dist
    for i=Index1+1:length(X(:,1))
        if(X(Index2,i)>X(Index1,i) && index2~=i)
            X(Index1,i)=X(Index2,i);
        end
    end
    for i=Index2+1:length(X(:,1))
        if(X(Index1,i)>X(Index2,i) && index1~=i)
            X(Index2,i)=X(Index1,i);
        end
    end
end
d=dist(Index1,Index2);
UseNewLabel=false;
%if it met the visited nodes (one of one node pair)
%then using the new linkage label
for i=1:length(X(:,1))
    if(visited(X(Index1,i)==1))
        label=label+1;
        UseNewLabel=true;
        break;
    end
end
```

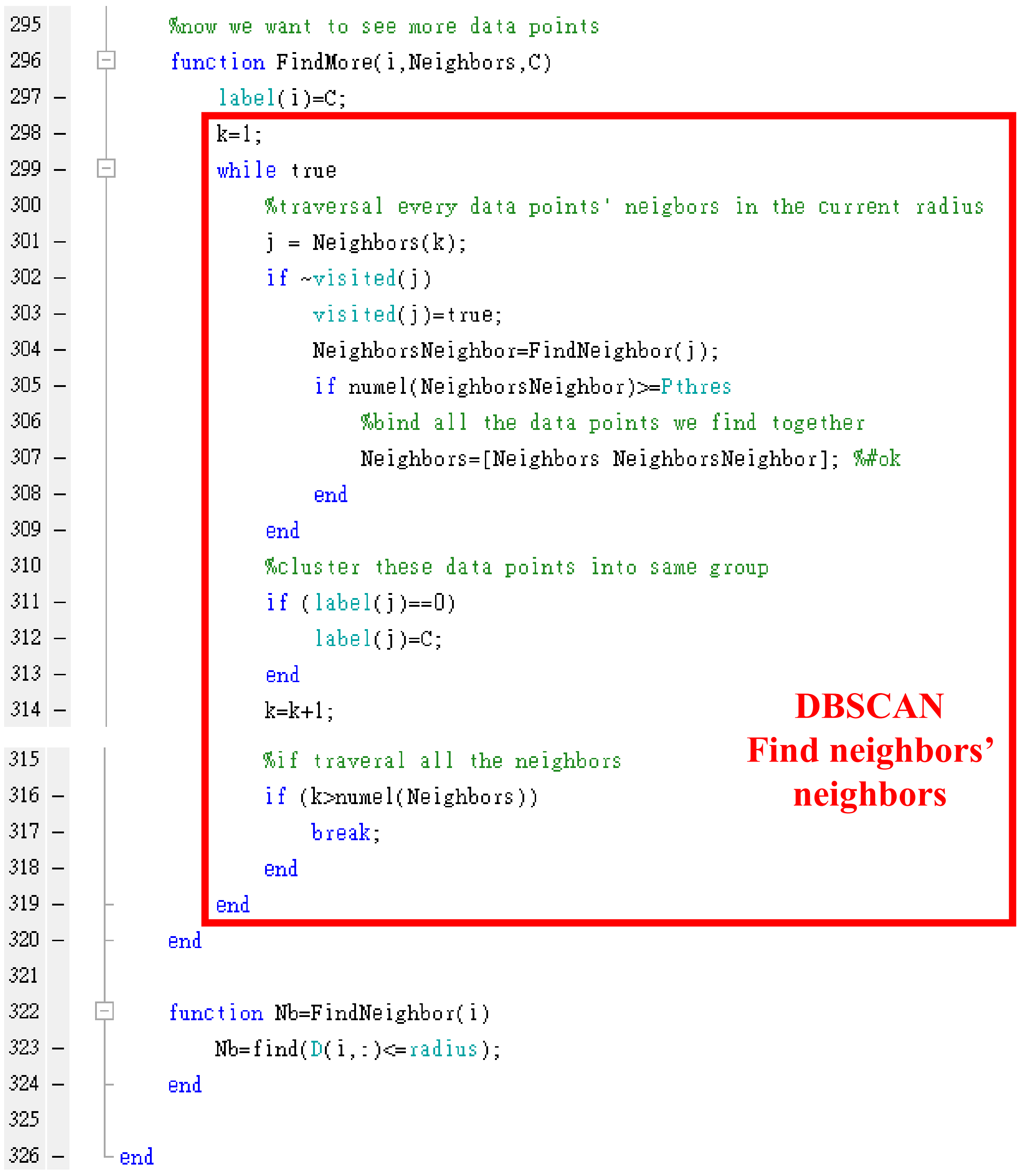
Complete-linkage clustering



```
        for i=1:length(X(:,1))
            if(visited(X(Index2,i)==1))
                label=label+1;
                UseNewLabel=true;
                break;
            end
        end

        %the first two components of Linkage are in ascending order
        if(Index1>Index2)
            swap(Index1,Index2);
        end

        %put the new label at the second component of Linkage
        if(UseNewLabel)
            Index2=label;
        end

        Linkage(T,:)=[Index1, Index2, d];
    end
end
function [OutputClust]=ClusteringByLinkage(X,Linkage,k)

    OutputClust=zeros(length(X(:,1)),length(X(1,:))+1);
    %calculate the height of the tree
    treeHeight=0;
    for i=1:length(Linkage(:,1))
        treeHeight=treeHeight+Linkage(i,3);
    end

    %if we want k clusters, then we can give a cut
    %between k-from-last and (k-1)-from-last nodes

    %k-from-last node indices
    [k_Idx1,k_Idx2,~]=Linkage(k,:);
    %(k-1)-from-last node indices
    [k2_Idx1,k2_Idx2,~]=Linkage(k,:);
    for i=1:length(X(:,1))
        for j=1:k
            if(traveral(X,k_Idx1) && traveral(X,k_Idx2) && ...
               traveral(X,k2_Idx1) && traveral(X,k2_Idx2))
                OutputClust=[X(i,:),j];
                break;
            end
        end
    end
end
```



```
function [label, noise]=DBSCAN(X,radius,Pthres,method)

    %{
    radius: maximum distance between two samples for one to
            be considered as in the neighborhood of the other
    Pthres: # samples (including the point itself) in one neighborhood
            for a point to be considered as a core point. This

    method: Euclidean, Manhattan, cosine, correlation
    %}

    %idx for clustering
    %idx=0 => noise
    idx=0;

    %initial output clustering label
    label=zeros(length(X(:,1)),1);
    D=pdist2(X,X,method);
    %visit or not
    visited=false(n,1);
    %is noise or not
    noise=false(n,1);

    %travesal all data points
    for i=1:n
        if ~visited(i)
            visited(i)=true;

            %check if # of neighbors < Pthres
            %if so, then it will be labeled as "noise"
            if (numel(FindNeighbor(i))<Pthres)
                noise(i)=true;
            else
                idx=idx+1;
                %find iteratively
                FindMore(i,Nb,idx);
            end
        en
    end
```



```
    %now we want to see more data points
    function FindMore(i,Neighbors,C)
        label(i)=C;
        k=1;
        while true
            %traversal every data points' neigbors in the current radius
            j = Neighbors(k);
            if ~visited(j)
                visited(j)=true;
                NeighborsNeighbor=FindNeighbor(j);
                if numel(NeighborsNeighbor)>=Pthres
                    %bind all the data points we find together
                    Neighbors=[Neighbors NeighborsNeighbor]; %#ok
                end
            end
            %cluster these data points into same group
            if (label(j)==0)
                label(j)=C;
            end
            k=k+1;
            %if traveral all the neighbors
            if (k>numel(Neighbors))
                break;
            end
        end
    end

    function Nb=FindNeighbor(i)
        Nb=find(D(i,:)<=radius);
    end

end
```