TREC: tree reduced ensemble clustering

by

Wenqing Liu

Research Paper Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Math

in the
Computational Mathematics
Faculty of Math

University of Waterloo
Winter 2019

Approval

Name:	**Wenqing Liu**
Degree:	**Master of Math (Computational Mathematics)**
Title:	**TREC: tree reduced ensemble clustering**
Examining Committee:	**Wayne Oldford** Supervisor Professor Department of Statistics and Acturial Science
Date Defended:	**Apr 15, 2019**

Abstract

In this paper we introduce a R package TREC: tree reduced ensemble clustering, a R package which combines multiple clustering outcomes and generates a cluster tree. We discussed our motivation and reasoning for TREC, and adopt a new algorithm to accelerate implementation of TREC. We created a R package named 'TREC', which simplifies procedure of using TREC. This package can also calculate distance between clusterings and support plotting functionality.

Keywords: tree reduced ensemble clustering; R package; cluster tree plot; clustering distance

Dedication

I hereby declare that I am the sole author of this report. This is a true copy of the report, including any required final revisions, as accepted by my examiners. I understand that my report may be made electronically available to the public.

Acknowledgements

I would like to thank my supervisor, Professor R. Wayne Oldford. Thanks for his patience and I learned a lot from him. He gave me a lot of useful suggestions and technical help in R. He taught me how to write sustainable and efficient R code. This paper is impossible without his help.

Table of Contents

List of Figures

Chapter 1

Introduction

Nowadays, clustering methods have become a popular technique in data science. Clustering is a statistical technique which divides data into useful, hopefully meaningful, groups. As a machine learning methodology, it is in the class of unsupervised learning algorithms and has broad applications including estimating densities, detecting anomalies, segmenting images, and many others.

Clustering methods can be roughly divided into two different groups. There are hierarchical methods, like single linkage. Single linkage grows a hierarchy of clusters in a bottom-up fashion by combining 'closest' clusters [11]. There are also partitional methods which, instead of producing a hierarchy of clusters, produce only a single partition of the data into clusters. Examples of partitional methods include DBSCAN and Kmeans. DBSCAN selects data points that are closely gathered in a neighborhood, leaving out outliers that don't lie near any clusters [6]. Kmeans algorithm partitions all observations into k clusters in which each observation belongs to the cluster with nearest mean [10].

Because of different mechanisms behind algorithms, different clustering results arise. The notion of a 'cluster', is not universally defined and every clustering algorithm has its own idea of what defines one. Different metrics, such as Euclidean distance, Minkowski distance and max norm can be used within these algorithms, leading to more possible clusterings. Choosing between algorithms and between metrics can be tricky.

Combining algorithms that integrate different clusterings may lead to a better understanding of the data.

Clustering algorithms are generally hard to compare with each other because it's subjective to determine which clustering is better. In this paper, we refer to the ensemble method `TREC` [20] for combining multiple clustering outcomes. `TREC` provides

a general framework which makes combination of any clustering algorithm outcomes possible, regardless of the number of clusters or whether it's a partitional or hierarchical method. W.Zhou [20] describes the way of integrating multiple clusterings and generate a series of ordered adjacency matrices as intermediate steps. These adjacency matrices correspond to graphs which naturally create nested structure contained in `TREC`'s result. The framework offers a unified way of understanding and comparing different clustering results. All clustering algorithms can be converted into a cluster tree and distances between those clusterings can be calculated.

We have implemented a new R package `TREC` to realize ideas [20] referred to in this paper. `TREC` stands for Tree Reduced Ensemble Clustering, which simplifies and combines different clusterings to provide a nested structure called a cluster tree. To select meaningful clusters and get rid of outliers, `TREC` implements a pruning strategy to simplify the nested structure. To make `TREC` package easier and faster to use, we improve the original algorithm of W.Zhou [20]. Given the output of most clustering algorithms in R as arguments, `TREC` will automatically combine these clustering results to produce a single, simplified, nested structure cluster tree. `TREC`'s output of nested clusters can be visualized as a cluster tree by `TREC`. R's plot function has been specialized to provide a list of arguments such as labels, colour of labels to visualize the result of `TREC`. `TREC` also provides some functionalities to understand clustering results. W.Zhou [20] describes a distance function between clusterings which we have implemented in `TREC`. This allows us to compare clusterings with one another. The distances can also be used to get coordinates for clusters. Now we are able to calculate distances between multiple clusterings with `TREC`. By using `TREC` package, we are able to integrate multiple clusterings easily and understand clustering results by clustering distance and plotting functionality with simple function calls.

This paper separates into four major parts. The first part is Section 2 where we introduce our ideas and reasoning behind `TREC`. This part explains why `TREC` is a general framework suitable for understanding clustering. Some literature review is used to motivate the approach. Once explained, we go through an example to illustrate how to apply `TREC` on a dataset. The second major part is Chapter 3 where the formal graph algebraic framework is abstracted mathematically and summarized. Algorithm for constructing cluster tree is also abstracted in this section. The distance measure is also introduced and applied to cluster trees constructed in `TREC`. These distances are used in multidimensional scaling to position the various cluster trees in a two dimensional space. This provides a quick visual means to assess various

characteristics of the resulting cluster trees [15]. In chapter 4, we delve into the `TREC` package and describe the general design of the `TREC` package. In chapter 5, we apply `TREC` on real datasets to evaluate its performance. Chapter 6 provides a summary of the work.

Chapter 2

Rationale behind the algorithm

2.1 Motivation - multiple clusterings

Multiple clusterings arise for many reasons. For example, all hierarchical clustering methods inherently produce multiple clusterings as a set of nested partitions [9]. Other methods intentionally generate multiple non-nested clusterings to understand different perspectives of similarity between data (one old example being **ADCLUS** [12]).

Various clustering algorithms have been developed to group data together, and each of them provide different insights into the structure of data. These include, for example, [15] seperating apart different concepts of humanly perceived similarity between stimuli [19], fitting mixture distributions to sample measurements [7], optimizing an objective function designed to capture 'natural' spatial structure in dimensional data[13], determining high density modes from a sample (e.g [9], [5], [17]), and determining approximate graph components in a similarity graph(e.g. [4], [14]), to mention a few common approaches [15]. Multiple clusterings can also arise from multiple local optima (e.g. [13]) or simply from multiple values of some 'tuning parameter' (e.g. [1]). Sometimes multiple clusterings are even induced by resampling (e.g. [18]).

A natural idea is to apply an ensemble method to different clustering outcomes. Ensemble refers to taking a majority vote or integrated information on clustering results. Our method intentionally takes advantage of results from multiple clustering outcomes and constructs a nested structure among clusters. Multiple clusterings can be summarized as a single partition [16][3], by a single tree [8], or even by several trees as in [2]. In multiple clusterings, clustering outcomes itself could be used to generate a new similarity matrix which is then used as input to some clustering algorithm, rerun

specific algorithm on new similarity matrix to generate a new clustering outcome. The new clustering combines the knowledge of previous clusterings from different algorithms and can be interpreted as a comprehensive understanding of clustering of data from different perspective. W.Zhou [20] describes the method of combining multiple clustering outcomes into one similarity matrix and clustering this similarity matrix. Next we will introduce motivations for W.Zhou's [20] idea.

Consider a set of m graphs G_k which represent clustering outcomes, and their adjacency matrices $\mathbf{A_k}, k = 1, \cdots, m$. Our idea is to summarize clustering information from this set of graphs as $\mathbf{A}_\omega = \sum_{i=1}^{k} \mathbf{A_k}$. To explain rationality of $\mathbf{A}_\omega$ as summarized information, we first consider how **ADCLUS** [12] constructs multiple overlapping clusters.**ADCLUS** [12] interprets nested clustering structure by additive clustering. More specifically, an additive clustering can be represented by an $n * m$ matrix of binary entries $P = \{p_{ij}\}$, where:

$$\begin{cases} 1 & \text{data i belong to cluster j} \\ 0 & \text{otherwise} \end{cases}$$

Under this representation, similarity matrix $S = \{s_{ij}\}$ can be derived from the cluster assigned in matrix P. Moreover, assume salience of kth cluster is ω_k, then similarity matrix can be represented using the following structure [12]:

$$s_{ij} = \sum_{k=1}^{K} \omega_k p_{ik} p_{jk}$$

For kth cluster, we define a binary vector $p_k = (p_{1k}, p_{2k}, \cdots, p_{nk})$, then the additive clustering can be rewritten as $S = \sum_{k=1}^{K} \omega_k p_k p_k^T$. One interpretation of this algorithm is a voting mechanism that each cluster has some weighted vote in consisting of a similarity matrix. Therefore, we imagine each clustering result as a 'vote' for the ensemble clustering result. Assume ω_k for each cluster is one, then all clustering information can be interpreted as a sum of each clustering outcomes. Clustering is not exactly a graph, but we may interpret cluster as a graph by adding edges on two data points if and only if those two data points belong to same cluster. Therefore, each clustering outcome is a graph , hence equivalent to an adjacency matrix. We summarize all information related to clustering as a sum of those adjacency matrices.

Another intuition is to create nested structure through multiple clustering outcomes. Data points i, j with closer distance in similarity matrix tend to stay at higher

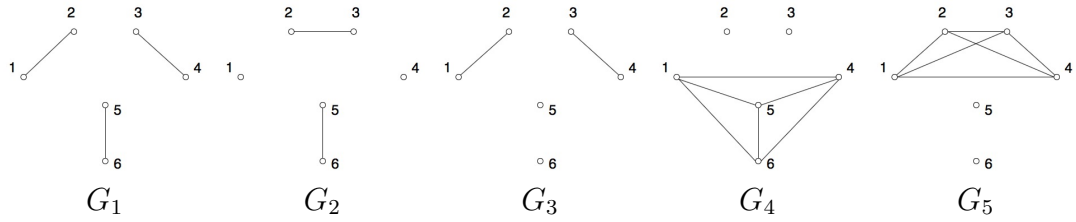


Figure 2.1: a graph family

hierarchy while i, j with smaller distance stay at lower hierarchy. It motivates us to split similarity matrix in a way that has hierarchical properties. Hierarchy can be considered as an inequality relation. Mathematically speaking, we consider two matrices $\mathbf{A} = [a_{ij}], \mathbf{B} = [b_{ij}], \mathbf{A} \geq \mathbf{B}$ if and only if $a_{ij} >= b_{ij}$ for $\forall i, j$, Therefore, our idea is to split similarity matrix into a series of matrices which sum up to similarity matrix, and those matrices have an ordered inequality relation between them.

2.2 Method on families of graphs

Here our method refers to [20]. Suppose we have a collection of graphs $G = \{G_1, G_2, \cdots, G_m\}$ and their corresponding adjacency matrices $A_1, A_2, \cdots, A_m$. Each $G_i = (V_i, E_i)$ is a labelled graph with vertex set V_i representing n data points and edges E_i representing some association between the corresponding pair of data points. Assume each connected component in the graph form a cluster, then each graph represents a clustering outcome. A collection of graphs can be considered as multiple clustering outcomes derived from running multiple clustering algorithms. We call G a graph family, our objective is to transform a collection of clustering outcomes into a single, suitably pruned, nested, cluster tree.

2.2.1 Graph family to component tree

We will go through the algorithm W.Zhou [20] with an example . Consider the following Figure 2.1. Each graph represents a clustering outcome of six data points with each connected component being a cluster. Differenet graphs represent different clustering outcomes. All these graphs together form a starting graph family. The corresponding adjacency matrices are shown in Figure 2.2.

Based on interpretation of additive clustering, each adjacency matrix is considered a vote to similarity matrix. Therefore, multiple clustering outcomes' similarity matrix

$$\begin{bmatrix}0&1&0&0&0&0\\1&0&0&0&0&0\\0&0&0&1&0&0\\0&0&1&0&0&0\\0&0&0&0&0&1\\0&0&0&0&1&0\end{bmatrix} \quad \begin{bmatrix}0&1&0&0&0&0\\1&0&0&0&0&0\\0&0&0&1&0&0\\0&0&1&0&0&0\\0&0&0&0&0&1\\0&0&0&0&1&0\end{bmatrix} \quad \begin{bmatrix}0&1&0&0&0&0\\1&0&0&0&0&0\\0&0&0&1&0&0\\0&0&1&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\end{bmatrix} \quad \begin{bmatrix}0&0&0&1&1&1\\0&0&0&0&0&0\\0&0&0&0&0&0\\1&0&0&0&1&1\\1&0&0&1&0&1\\1&0&0&1&1&0\end{bmatrix} \quad \begin{bmatrix}0&1&1&1&0&0\\1&0&1&1&0&0\\1&1&0&1&0&0\\1&1&1&0&0&0\\0&0&0&0&0&1\\0&0&0&0&1&0\end{bmatrix}$$

$\mathbf{A_1}$ $\mathbf{A_2}$ $\mathbf{A_3}$ $\mathbf{A_4}$ $\mathbf{A_5}$

Figure 2.2: adjacency matrices

$$\begin{bmatrix}0&1&1&1&1&1\\1&0&1&1&0&0\\1&1&0&1&0&0\\1&1&1&0&1&1\\1&0&0&1&0&1\\1&0&0&1&1&0\end{bmatrix} \quad \begin{bmatrix}0&1&0&0&0&0\\1&0&0&0&0&0\\0&0&0&1&0&0\\0&0&1&0&0&0\\0&0&0&0&0&1\\0&0&0&0&1&0\end{bmatrix} \quad \begin{bmatrix}0&1&0&0&0&0\\1&0&0&0&0&0\\0&0&0&1&0&0\\0&0&1&0&0&0\\0&0&0&0&0&1\\0&0&0&0&1&0\end{bmatrix} \quad \begin{bmatrix}0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&0\\0&0&0&0&0&1\\0&0&0&0&1&0\end{bmatrix}$$

$\mathbf{A_{(1)}}$ $\mathbf{A_{(2)}}$ $\mathbf{A_{(3)}}$ $\mathbf{A_{(4)}}$

Figure 2.3: ordered adjacency matrices

is simply the sum of all adjacency matrices. After calculation, similarity matrix is

$$\mathbf{A}_\omega = \sum_{k=1}^{5} \mathbf{A_k} = \begin{bmatrix}0&3&1&2&1&1\\3&0&2&1&0&0\\1&2&0&3&0&0\\2&1&3&0&1&1\\1&0&0&1&0&4\\1&0&0&1&4&0\end{bmatrix}$$

$\mathbf{A}_\omega$ encodes information of similarity. Large entries of $\mathbf{A}_\omega$ imply closer relation between two data points and they tend to stay at higher hierarchy in nested clustering. As a result, we hope to split $\mathbf{A}_\omega$ into a a series of ordered adjacency matrices. Here for matrix $\mathbf{B} = \{b_{ij}\}$ and $\mathbf{C} = \{c_{ij}\}$, we say $\mathbf{B} \leq \mathbf{C}$ if $b_{ij} \leq c_{ij}$ for $\forall i, j$. We formulate our requirement as follows: Find adjacency matrices $\mathbf{A_{(1)}}, \mathbf{A_{(2)}}, \cdots, \mathbf{A_{(k)}}$ such that $\sum_{i=1}^{k} \mathbf{A_{(i)}} = \mathbf{A}_\omega$ and $\mathbf{A_{(k)}} \leq \cdots \leq \mathbf{A_{(2)}} \leq \mathbf{A_{(1)}}$. The resulting adjacency matrices is shown in Figure 2.3. These 'ordered' new adjacency matrices naturally create a nested structure in the consequent graphs $G_{(i)}$. The nested graph family is shown in Figure 2.4. The construction of 'ordered' graphs $G_{(i)}$ whose adjacency matrices sum up to $\mathbf{A}_\omega$ is also unique. We can easily find values of $\mathbf{A_{(i)}}$ in Figure 2.3. Now we hope to create a hierarchy of clusters based on this decomposition. Consider each marix $\mathbf{A_{(i)}}$ as an adjacency matrix. Then we are able to create a series of graph:

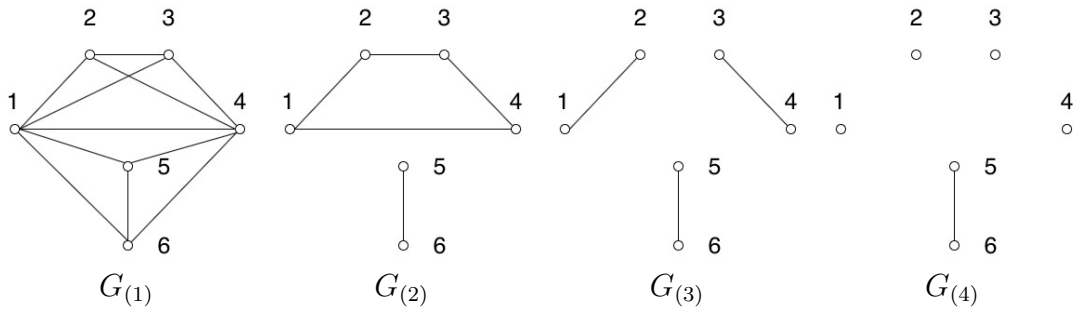


Figure 2.4: The nested sequence of graphs $G_{(1)} \geq G_{(2)} \geq G_{(3)} \geq G_{(4)}$, each graph component is a cluster.

The above graphs provide a hierarchy of clusters, but adjacency graphs are a little bit different from clustering. We simply put all points in the same connected component as a cluster. Then a natural component tree can be generated as Figure 2.5.

Note we only keep the part which retains all changes in cluster component, therefore we obtain cluster tree density plot in Figure 2.6.

2.2.2 Tree reduction

After we form a component tree, we hope to reduce the complexity of component tree and prune redundant component. Here, we refered to the pruning method in [15]. In this way, trivial parts of component tree are removed and we can capture essence of cluster structure. We execute two simple rules to reduce complexity.

1. Prune the trivial components.

2. Contract pure telescopes of components.

These two rules are applied in order. The resulting component tree will be called the cluster tree. These two rules are illustrated in Figure 2.7 using component tree shown in picture 1 of Figure 2.7.

In picture 2 of Figure 2.7, we first apply trivial pruning, which means that we delete all splits that only set aside one data point. Here we call components that only contain one data point as trivial component. In picture 3 Figure 2.7, we delete all telescopic contraction, which means that, if that contraction step does not provide any branches, we will remove that step. In other words, we get rid of those steps that branching is not involved, and only keep track of steps that two or more branches are generated. Components or data points that are pruned will not be shown in our cluster tree. Therefore, our final cluster tree is in picture 4 Figure 2.7.

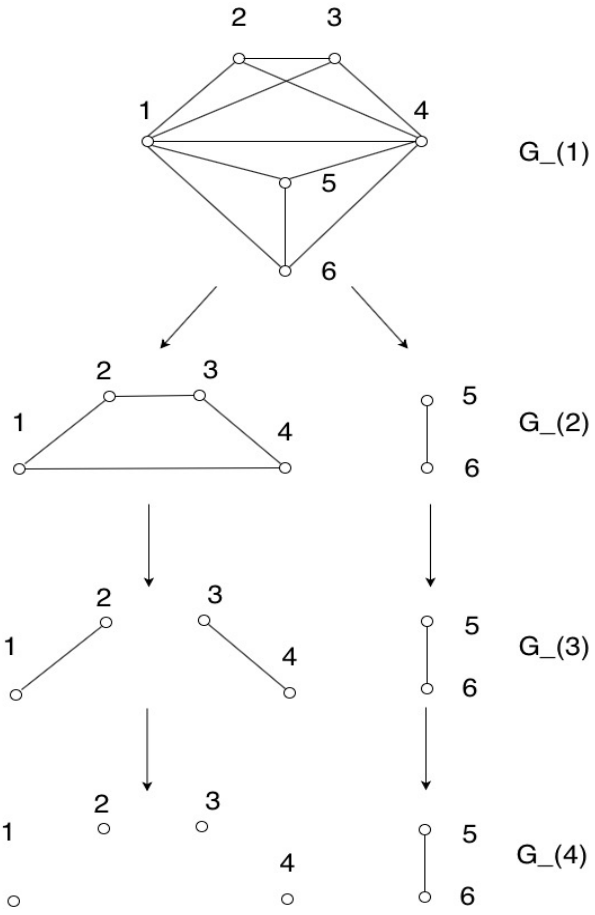


Figure 2.5: component tree

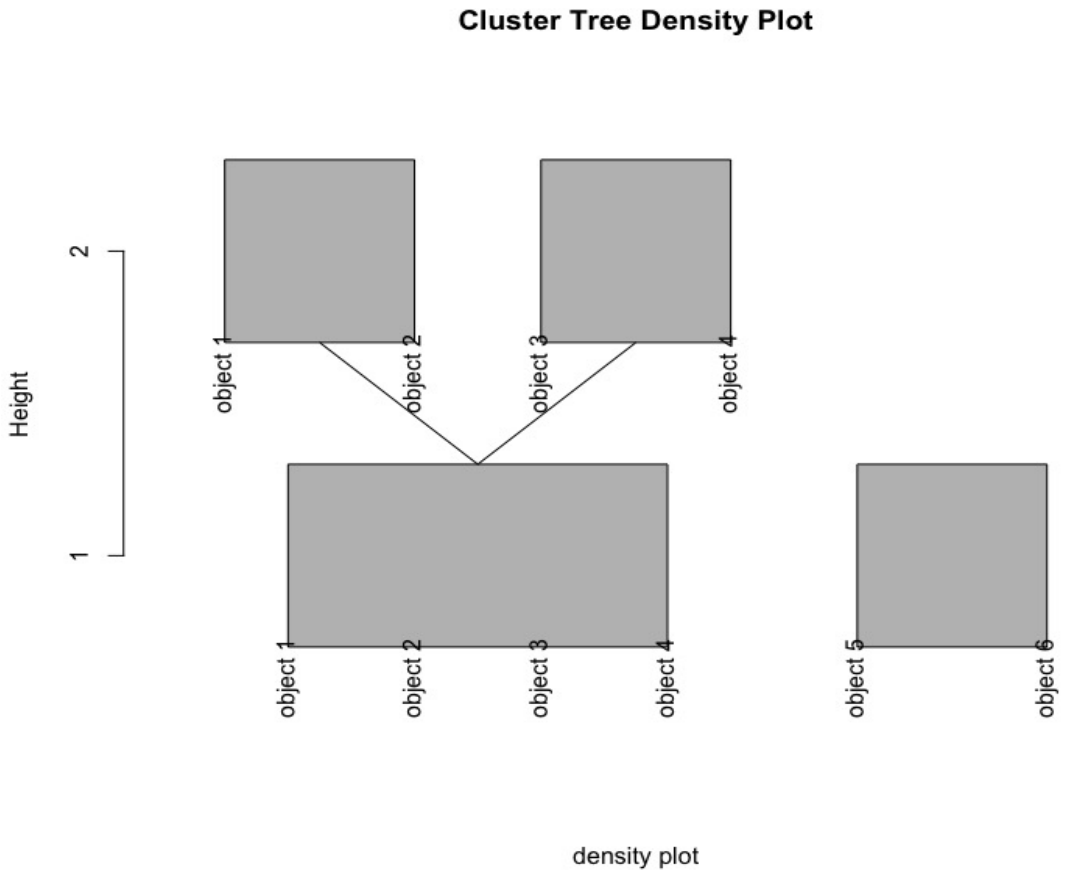
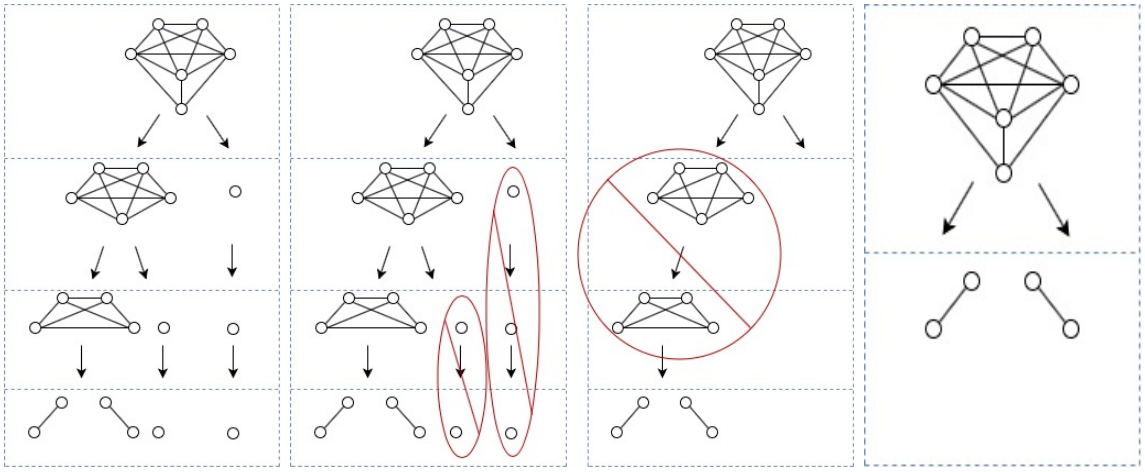


Figure 2.6: cluster tree density plot



1. Component tree 2. Trivial pruning 3. Telescopic contraction 4. Cluster tree

Figure 2.7: Steps in reducing the component tree to a cluster tree

2.2.3 Difference between TREC and single linkage

Note in figure 2.7, the final cluster tree is in picture 4. This result is completely different from applying single linkage on distance matrix. TREC differs from single linkage in two aspects.

1. Single linkage always splits into two branches at a time, while this is not necessarily true for TREC. Due to algorithm of single linkage, it always has two branches at each split. TREC can split into any number of branches based on the concrete situation.

2. Single linkage does not have pruning process, while TREC removes trivial components and does telescopic contraction. With pruning, number of points in the next layer of cluster tree may differ from the current layer since unimportant data points are removed. TREC will only retain those important data points that reveal the structure of data.

Chapter 3

Algorithms in 'TREC'

3.1 Combining Clusterings

Given the example gone through in Chapter 2, now we will summarized full steps of `TREC` from W.Zhou [20]. We will describe our algorithm by developing the methodolgy on a general family of graph so that `TREC` applies to any multiple clustering problems. Assume we are given a series of clustering outcomes $C_1, \cdots, C_m$. Each C_i corresponds to a graph G_i. we refer to $G = \{G_1, G_2, \cdots, G_m\}$ as a family of graphs. Each $G_i = (V_i, E_i)$ is a labelled graph, with each vertex representing a data point and two vertexes are linked if and only if two vertexes belong to the same cluster. Each graph has its corresponding adjacency matrix, let $A = \{\mathbf{A_1}, \mathbf{A_2}, \cdots, \mathbf{A_m}\}$ be those adjacency matrices. Then we start to walk through our algorithm.

Step 1: Transform clustering outcomes into adjacency matrices.

The first step is to transform clustering outcomes into a part of additive clustering. We construct a graph G_i based on clustering outcome: connecting two data points if and only if those two points belong to the same cluster. Then each clustering outcome becomes a graph. Determine the corresponding adjacency matrix A_i based on graph. Here edges symbolize certain association between two data points which belong to the same cluster. This representation gives entry 1 in adjacency matrix if two points are in the same cluster and entry 0 in adjacency matrix if two points are not in the same cluster, which is exactly what we construct for the idea of additive clustering.

Step 2: Sum all adjacency matrices $\mathbf{A}_\omega = \sum_{i=1}^{m} \mathbf{A_k}$

Based on the idea of additive clustering [12], since all information related to multiple clustering outcomes is adjacency matrices, our way to interpret multiple clusterings is to sum all adjacency matrices together. Taking sums have absorbed

all information as it's exactly the definition of 'ensemble'. Assume $\mathbf{A}_\omega = \sum_{i=1}^{n} A_i$, Now our problem of multiple clustering has transformed to how to apply clustering methodology on sum of adjacency matrices, which is $\mathbf{A}_\omega$.

Step 3: Split $\mathbf{A}_\omega$ into a series of adjacency matrices $\mathbf{A_{(k)}} = \{a_{ij}^{(k)}\}$, and $a_{ij}^{(k)} = I(a_{ij} >= k)$

As we mentioned above, we hope to generate a hierarchy based on $\mathbf{A}_\omega$. Since each entry in $\mathbf{A}_\omega$ represents similarity between two data points, a natural idea is to put data points with larger similarity on higher hierarchy. We construct a series of adjacency matrices which sum up to $\mathbf{A}_\omega$ in a ordered relation. We let $\mathbf{A_{(k)}} = \{a_{ij}^{(k)}\}, a_{ij}^{(k)} = I(a_{ij} >= k)$. Then we can observe matrices $\mathbf{A_{(k)}}$ has relation $\mathbf{A_{(1)}} \geq \mathbf{A_{(2)}} \geq \cdots \geq \mathbf{A_{(m)}}$. We may eliminate duplicate matrices among them to simplify to a non-repetitive sequence of matrices. Anyway, this sequence of adjacency matrices create hierarchical structure using information from multiple clustering outcomes.

Step 4: Plot graphs $G_{(k)}$ based on adjacency matrix $\mathbf{A_{(k)}}$, assign each connected component into one cluster

Now we try to reverse what we do in Step 1. Here we transform adjacency matrices to graphs. Then we divide graphs into clusters based on connected components. We assign data points in the same connected component to the same cluster. Note each connected component may not be a complete subgraph, but we forcibly put connected component in a cluster. This split natrually arises a nested structure among clusters. In this way, we also follow the idea of density cluster tree. In theory of density cluster tree, one of the most concrete and intuitive ways to define clusters when data are drawn from a density f is on the basis of level sets of f, *i.e* for any λ the connected components of $\{x : f(x) \geq \lambda\}$. We select each λ as each unique different entries of $\mathbf{A}_\omega$. These values are the only possible 'λ'. Follow this way, we create a series of clustering outcomes which form a hierarchical relation.

Step 5: Construct hierarchical structure based on ordered relation between $G_{(k)}$

In the series of graphs $G_{(k)}$, it is a process of separating entire data points into individual points. Record the process and construct hierarchical relations involved in this process. It will be our first nested structure of multiple clustering.

Step 6: Prune cluster tree if there are trivial components and contract pure telescopes of components.

In the above hierarchical structure formed by the series of graphs $G_{(i)}$, prune tree based on two rules mentioned above: remove trivial components and contract

pure telescopes of components. Pruning trees get rid of unimportant subcluster and redundant nested structure which interferes us from seeing the true structure of data more clearly. After pruning, we retain the remaining branches and get the final cluster tree.

Note this is a general framework for any number of clustering algorithms. Multiple clustering outcomes can be fit into this method while a unique clustering algorithm can be considered as multiple clustering with number of clustering algorithms equal to one. Therefore, all clustering outcomes can generate a cluster tree using the above algorithm. In `TREC` R package implementation, we design R package in a way such that any number of clustering outcomes can be simply tossed into `TREC` and a cluster tree will be returned. Wu Zhou implemented this algorithm, but his implementation is a straightforward execution using the above steps, which actually consumes a lot more time and memory space. Next we introduce another implementation of this algorithm which improve both time and space complexity.

3.2 A New Implementation of Combine Clusterings Algorithms

The above algorithm takes 'similarity' between data points as input to clustering process, and 'similarity' encodes information for hierarchy of clusters. Larger entries of similarity matrix correspond to higher hierarchy and then we move along to smaller entries. This relates naturally to hierarchical clustering, especially single linkage algorithm, which also uses 'distance' as input and produce hierarchical clustering outcomes. In implementation of `TREC` package, we hope to take advantage of the existing single linkage clustering algorithm in R to help implement `TREC`. In step 4,5,6 of the Algorithm, what we do is actually very similar to executing hierarchical algorithm. All we need to do to accomodate to `TREC` is to shrink the result of single linkage. After executing single linkage on $\mathbf{A}_\omega$, we first need to shrink all splitting that happens for the same value. On the other hand, we also need to shrink as long as pruning process is involved. Therefore, a split which generate a single data point will not be recorded in the result of `TREC`. Furthermore, we also need to make sure telescopic contraction will be done by removing unnecessary splitting. After those steps, We can summarize our new algorithms for `TREC`:

1. Obtain clustering results from different algorithm

2. Transform clustering results into adjacency matrices

3. sum all adjacency matrices and result is $\mathbf{A}_\omega = \{a_{ij}^\omega\}$

4. execute single linkage hierarchical clustering algorithm on $-\mathbf{A}_\omega$

5. shrink splitting if splitting happens for the same distance

6. shrink splitting if there are trivial components and contract pure telescopes of components

7. obtain final cluster tree

Note in R, single linkage is impllemented in fortran, which is much faster than R. Therefore `TREC` is implemented much faster in this fashion. We no longer need to save all 'level sets' which splitting happens. Corresponding A_k does not need to be saved in `TREC`'s process as well. From memory's perspective of view, All we need to store information is similarity matrix and result from single linkage. Other data can be derived from the result of single linkage.

3.3 Clustering distance

3.3.1 Clustering distance algorithm

To interpret clustering result, W.Zhou [20] designed a metric to measure distance between clustering outcomes. This metric also helps us evaluate the clustering quality by calculating distance between clustering outcomes and true clustering. Note the concept of graph family is applicable to all clustering outcomes, any clustering outcome can be represented with a cluster tree. As a result, what we need to do is to set up a metric for different cluster trees. From cluster tree we are able to revert to recover each adjacency matrices $\mathbf{A_k}$. Note $\mathbf{A_k}$ is obtained by connecting all data points in the same cluster. Then we sum those adjacency matrices together to recover original $\mathbf{A}_\omega$. Note all information is encoded in the upper half triangle of matrix $\mathbf{A}_\omega$ as mentioned above. Assume there are n data points in total, then there are $\frac{n*(n-1)}{2}$ upper triangle elements. Our idea is to expand those elements into a long vector. Each entries in this vector represent similarity between a pair of data points. As long as those long vectors align with each other, those vectors can be considered as a representation for clustering outcome. Now we can compare clustering outcome by comparing those vectors derived from cluster tree. Suppose for two clustering outcomes T_1, T_2, their

expanded vectors are s_1 and s_2. We define a distance $d(T_1, T_2)$ to be :

$$d(T_1, T_2) = ||\frac{s_1}{||s_1||} - \frac{s_2}{||s_2||}||$$

Note each entry of s_1 or s_2 represent similarity between two data points. By normalizing the similarity vector, we are comparing the importance weight of each entry. Therefore we are measuring distance of importance weight for two types of clustering outcomes. Following is steps for comparing clustering distance:

1. transform two clustering outcomes into cluster trees using algorithms in section 3.2
2. revert cluster tree to a series of adjacency matrices
3. sum those adjacency matrices to obtain $\mathbf{A}_\omega$
4. expand upper half of $\mathbf{A}_\omega$ to a long vector
5. normalize vectors and calculate euclidean distance between two vectors

3.3.2 Gram matrix

Given the algorithm for calculating distance between clustering outcomes, we are able to derive distance matrix for multiple clustering outcomes. Each entry of distance matrix is the distance between two clustering outcomes. Now we hope to map distance matrix to n-dimensional space. Assume distance matrix is $\mathbf{D} = [d_{ij}]$ and n-dimensional coordinates or each clustering outcome is $x_1, x_2, \cdots, x_m$, then we have $d_{ij} = ||\mathbf{x}_i - \mathbf{x}_j||$. The Gram matrix is defined to be $\mathbf{G} = [g_{ij}]$ where $g_{ij} = \mathbf{x}_i^\mathsf{T}\mathbf{x}_j$. Then we have formula: $G = -\frac{1}{2}(\mathbf{I}_n - \frac{1}{n}\mathbf{1}\mathbf{1}^T)\mathbf{D}^\star(\mathbf{I}_n - \frac{1}{n}\mathbf{1}\mathbf{1}^T)$ where $\mathbf{D}^\star = [d_{ij}^2]$. In TREC package, we create a function called `distToGram` which can transform distance function to Gram matrix with one function call.

3.3.3 plot cluster tree

To visualize cluster tree obtained from TREC algorithm, we provide plot function to cluster tree. In `TREC`, we organize cluster trees into `clusterTree` object. To plot cluster tree, we design two functions, one is `clusterTreeToPlotInfo`, the other is `plot.clusterTreePlotInfo`.

As function name suggests, `clusterTreeToClusterTreePlotInfo` transform a cluster tree into plot info. Plot info includes coordinates of rectangles, lines, text

etc for plotting cluster tree. Cluster trees are generally hard to plot since cluster trees are inherently nested, they usually have multiple layers and each layers must be somehow connected. To plot cluster tree, we uniformly create a data structure called `clusterTreePlotInfo`. Function `clusterTreeToClusterTreePlotInfo` transforms a `clusterTree` object to `clusterTreePlotInfo` object.

`clusterTreeToClusterTreePlotInfo` takes `clusterTree` object as input, and `clusterTreePlotInfo` object as output. We design `clusterTreePlotInfo` object has three attributes: `treeMatrix`, `plotLayerInfo` and `subtrees`.

`treeMatrix` is a matrix which represents the nested structure. Each entry in matrix represents a mapping between data point id to cluster id. In each column, each value corresponds to mapping between data id and cluster id in that layer. For example, let $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 2 \\ 1 & 2 \\ 2 & NA \\ 2 & NA \end{bmatrix}$ is a `treeMatrix`, then `treeMatrix` $\mathbf{A}$ represents data 1,2,3,4 belong to cluster 1, data 5,6 belong to cluster 2 in the first layer. In second layer data 1,2 belong to cluster 1 while data 5,6 belong to second layer. Data 5,6 simply disappear in the second layer. Therefore nested structure of clusterTree is shown below(Figure 3.1):

`plotLayerInfo` represents plot related information in this layer. `plotLayerInfo` consists of four parts, rectangles, lines, labels and runtsFlag. We represent each cluster with a rectangle. Lines are used to suggest affiliation between each cluster. labels represent labels for each data point. runtsFlag suggests whether those data is pruned during pruning steps. A plot of `treeMatrix` A is shown in (Figure 3.2).

Since cluster tree has nested structure, we design `subtrees` attribute of `clusterTreePlotInfo` to be 'child' of `clusterTreePlotInfo`. Therefore, subtrees are a list of `clusterTreePlotInfo` which has exactly the same structure as original object.

To construct `clusterTreePlotInfo` object, we employ a recursive algorithm. We first organize `treeMatrix` and `plotLayerInfo` for the first column of treeMatrix, then we apply recursive calls to the rest of columns.

`plot.clusterTreePlotInfo` function plots `clusterTreePlotInfo` object. This function also takes a recursive fashion to plot as well. For each `clusterTreePlotInfo` object, we first plot `plotLayerInfo` for this object, and then we will apply recursive function calls to each object in `subtrees` attribute.

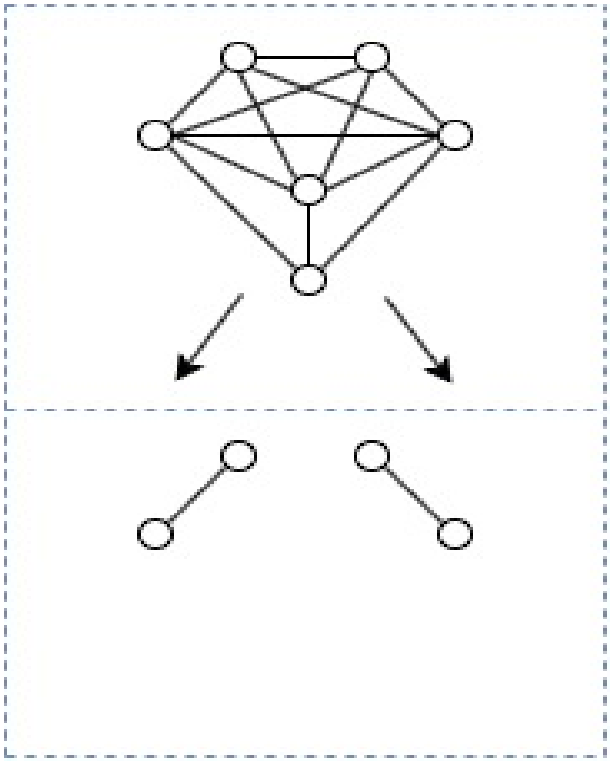


Figure 3.1: cluster tree example

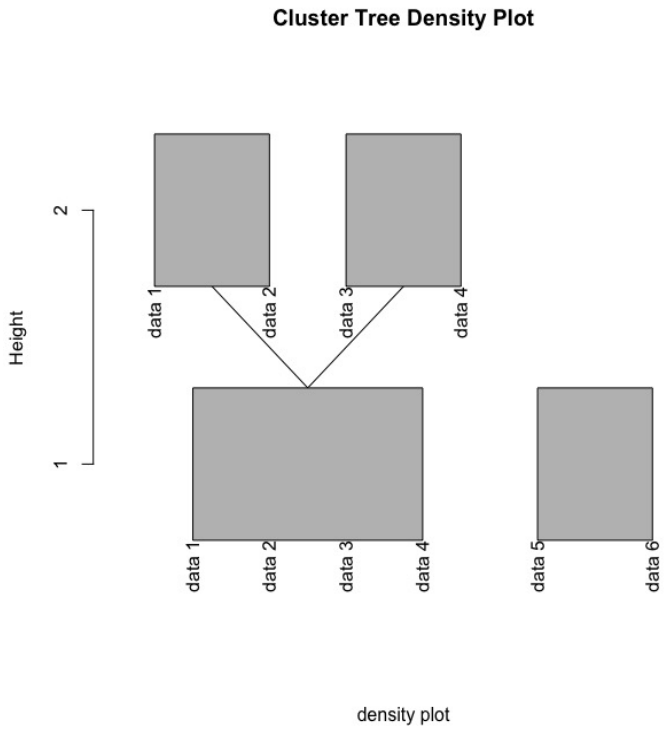


Figure 3.2: cluster tree example

As a result, to plot `clusterTree` object, all we need to do is to call two functions `clusterTreeToClusterTreePlotInfo` and `plot.clusterTreePlotInfo` to generate the plot.

Chapter 4

R package 'TREC'

In this chapter, we mainly talk about our design of R package `TREC`. `TREC` has four files, next we'll introduce design of each files separately.

4.1 getCluster.R

Like we mentioned above, any clustering outcome can be transformed into cluster tree. We hope to uniformly process different clustering outcomes generated by clustering algorithm in R and transform each of them into cluster tree. We design function `getCluster` to transform clustering algorithm's output in R to a uniform data structure called `clusterTree`. Each `clusterTree` object contains a `treeMatrix` attribute which is a matrix mapping data points id to cluster id. Note `treeMatrix` stores the nested structure of cluster tree. Labels can be passed into `clusterTree` object as well. Note `getCluster` function accepts output of different clustering algorithms as input, therefore `getCluster` function must use `useMethod` in R so it can process different class of input differently.

4.2 clusterTree.R

clusterTree.R is the most important files among `TREC` package. It contains three functions: `mergeToMatrix`, `combineClustering` and `reOrderClusterTreeMatrix`.

Because single linkage algorithm only return a data structure called `merge` for hierarchical structure, we design function `mergeToMatrix` to transform merge into `treeMatrix` attribute of cluster tree.

`combineClusterings` is the most important function in `TREC`. Input of this function is output of various clustering algorithms. Output is the cluster tree object based on `TREC` algorithms. This function implements the algorithm in Section 3.2.

As suggested by the name, function `reOrderClusterTreeMatrix` reorders `treeMatrix` attribute of `clusterTree` object. It reorders `treeMatrix` in a way such that data points belong to the same cluster will be next to each other. In other words, index of data points that belong to same cluster will be consecutive numbers. This function helps plot function mentioned later on.

4.3 distance.R

This file provides functions related to caculating distance between different clustering algorithms. It includes three functions: `clusDist`, `getClusDis` and `distToGram`.

The main function is `clusDist`. This function transforms a number of output of clustering algorithms into a distance object. Multiple clustering outcomes are supported.

`getClusDist` function takes only two argument: two clustering outcomes. It implements algorithm in section 3.3.1. With this function , we are able to calculate distance between any two clustering outcomes. This function helps implementation of `clusDist`. Implementation of `clusDist` is actually calling `getClusDist` function repeatedly to construct distance object.

After obtaining distance object between clusterings, `distToGram` function is provided to transform distance matrix into gram matrix. After we have gram matrix, we are able to visualize data on 2-dimensional space.

4.4 plot.R

`plot.R` provides two main functions: `clusterTreeToClusterTreePlotInfo` and `plot.clusterTreePlotInfo`.

`clusterTreeToClusterTreePlotInfo` function transforms `clusterTree` object into `clusterTreePlotInfo` object. This object includes three parts: `treeMatrix`, `plotLayerInfo` and `subtrees`. With those information recorded, we are able to plot in the future.

`plot.clusterTreePlotInfo` provides functionality for plotting `clusterTreePlotInfo` object.

Note both above two functions are written in recursive way. So those two functions both call two other functions, `clusterTreeToClusterTreePlotInfoRecursiveHelper` and `plotClusterTreePlotInfoRecursiveHelper` to recursively generate or plot functions.

Note structure of `clusterTreePlotInfo` is nested. `treeMatrix` and `plotLayerInfo` are information for the first layer, while `subtrees` are child of `clusterTreePlotInfo` which has exact the same structure as `clusterTreePlotInfo`.

Other functions are provided to support two main functions. Because structure of `clusterTreePlotInfo` object is nested, it's pretty complicated to examine it. Helper functions are provided to support easier look up in `clusterTreePlotInfo` data structure. For example, `showLabels` function shows coordinates of labels in `clusterTreePlotInfo` object and `showLines` function shows coordinates of lines in `clusterTreePlotInfo` object. `getBranchInfo` function returns plot information on each layer.

Chapter 5

Examples

5.1 Gaussian mixture data

We start with a data set from a mix of 3 Gaussians. Figure 5.1 shows a sample of 300 points, each 100 drawn from one of three distinct two dimensional Gaussians. Circles are drawn from the first Gaussian(Group 1), the squares from the second(Group 2), and triangles from the third(Group 3). The first and second Gaussians are located closer to each other because Gaussians are generated using the underlying nested hierarchical structure as shown in Figure 5.2. First we attempt to cluster data using traditional clustering algorithms, kmeans with 3 clusters. We can observe that despite we are able to discover three Gaussian groups, no hierarchical structure will be observed. Same thing will happen again if we apply DBSCAN algorithm on the data. Other methods for generating nested structure of clustering, such as hierarchical clustering algorithm is applied on dataset and result is shown in figure(). As we see, hierarchical clustering creates an extremely detailed dendogram that it's make it harder to see no higher level overview. Therefore, we hope to apply `TREC`, a combination of different clustering outcomes to integrate features of different clustering algorithms and reveal true structure of data set.

Here, We apply kmeans, single linkage(cut top three splits) and dbscan algorithms as input to `TREC` algorithm. We choose those three algorithms because we want to retain both non-hierarchical and hierarchical information at the same time. Result is shown in Figure 5.4. `TREC` combines the features of three different clustering algorithms and output a cluster tree which integrate those features. In the figure, red, green and blue labels under rectangles represent group 1,2 and 3. As shown in Figure 5.4, despite some minor errors, `TREC` is able to capture nested structure of data set

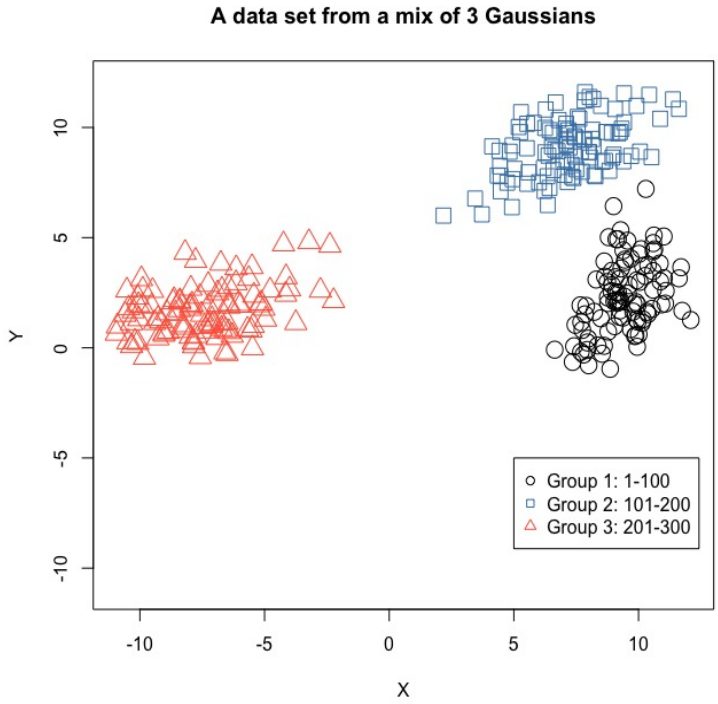


Figure 5.1: a data set of mix of 3 Gaussians

as well as rough division of three groups. Here, `TREC` does provide further information about clustering and wisely prune branches that are not necessary.

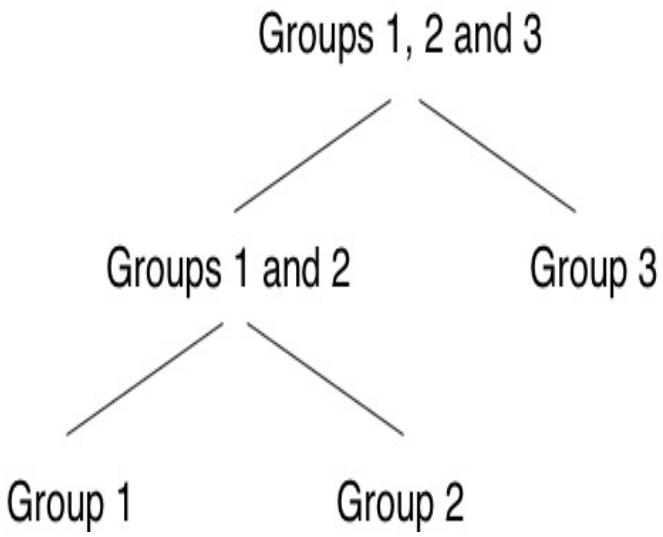


Figure 5.2: true structure of cluster tree

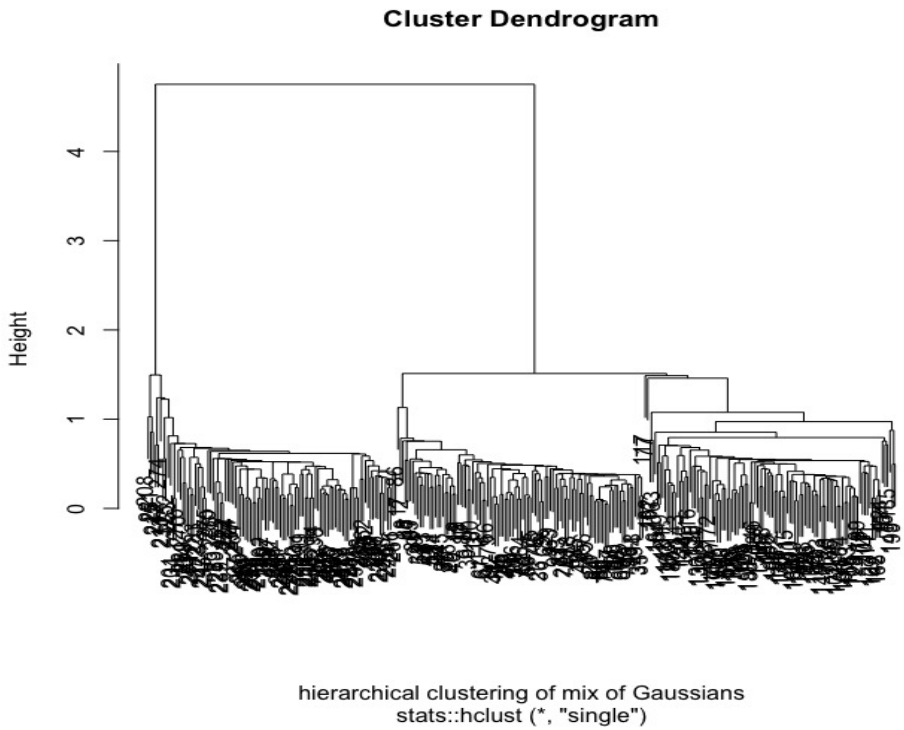


Figure 5.3: hierarchical clustering result

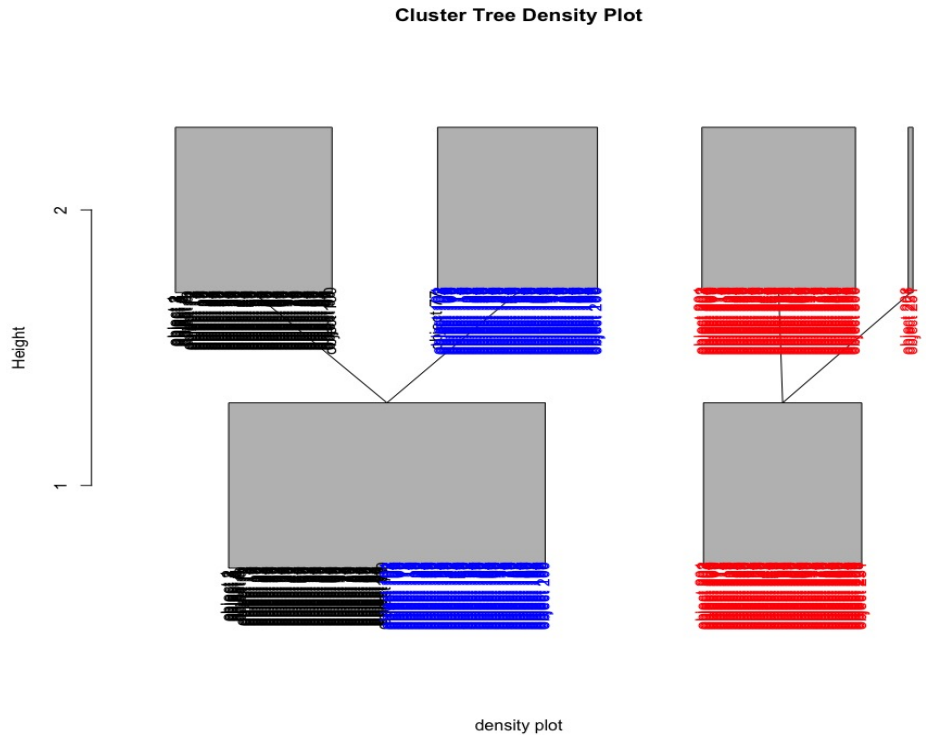


Figure 5.4: TREC on mix of Gaussians

5.2 spiral data set

In this section, we experiment on a spiral data set. Spirals data is generally hard to deal with because data are tangled with each other and euclidean distance along is not enough to reveal structure of clustering. Data points that are closer to each other may not belong to same cluster due to special geometrical structure of data. Here our dataset is consisted of a red curve of circles and a green curve of rectangles, as shown in Figure 5.5. If we simply apply kmeans with two clusters on our data, result is shown in right side of Figure 5.6. Kmeans alone will consider left side data as a cluster and right side data as another cluster. Therefore, we try to apply kmeans with more clusters and see whether it makes a difference. Here we apply kmeans with 3-20 clusters and plot their corresponding cluster trees. The result is shown in Figure 5.7 and Figure 5.8. In Figure 5.7 and Figure 5.8, red represents the red spiral and green represents the green spiral. Note those clustering outcomes are entirely random and don't have any common clue. Then we apply TREC on those kmeans' results. The result is shown in Figure 5.9. Red labels represent the first spiral and green labels represent the second spiral. Cluster tree shown in Figure 5.9 can somehow find two

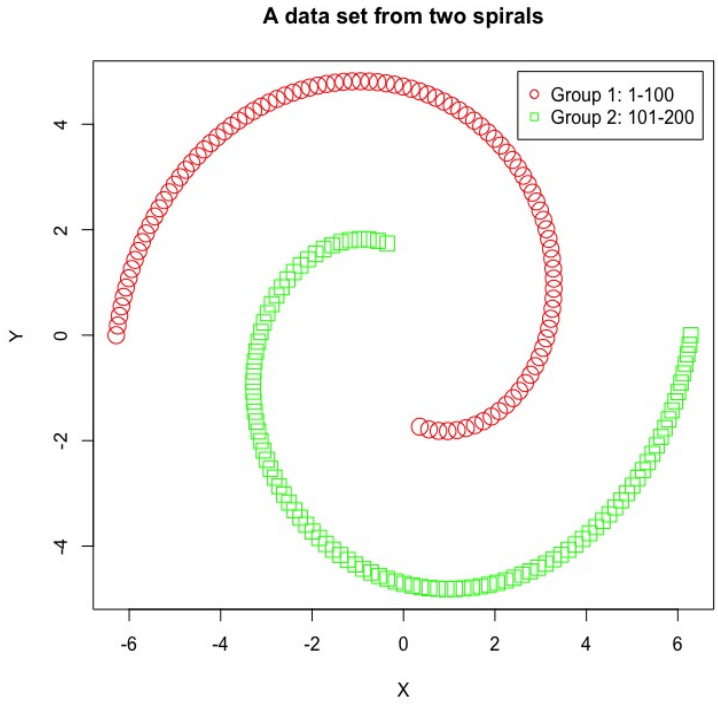


Figure 5.5: spiral dataset

spirals in the first layer. Here we observe that even kmeans' result is not correct, it will be corrected by ensemble rules and produce a reasonable result. Here `TREC` does work like an ensemble algorithm, which consider a variety of prediction results and choose the most frequent clustering outcomes of all prediction results.

5.3 Illustration of use of TREC package

In this section, we'll illustrate how to use R package TREC. Let's start with a dataset.

```
data <- rbind(matrix(rnorm(100, mean = 10, sd = 2), nrow = 50),
              matrix(rnorm(100, mean = 0, sd = 1), nrow = 50),
              matrix(rnorm(100, mean = -10, sd = 3), nrow = 50))
```

Obviously this is a mixture of three Gaussians. Therefore, true clustering is three clusters with Gaussian distribution. We generate the starting clustering outcomes by applying four existing clustering algorithms:

```
clustering1 <- kmeans(data,centers=3)
clustering2 <- dbscan::dbscan(data,eps=.8)
```

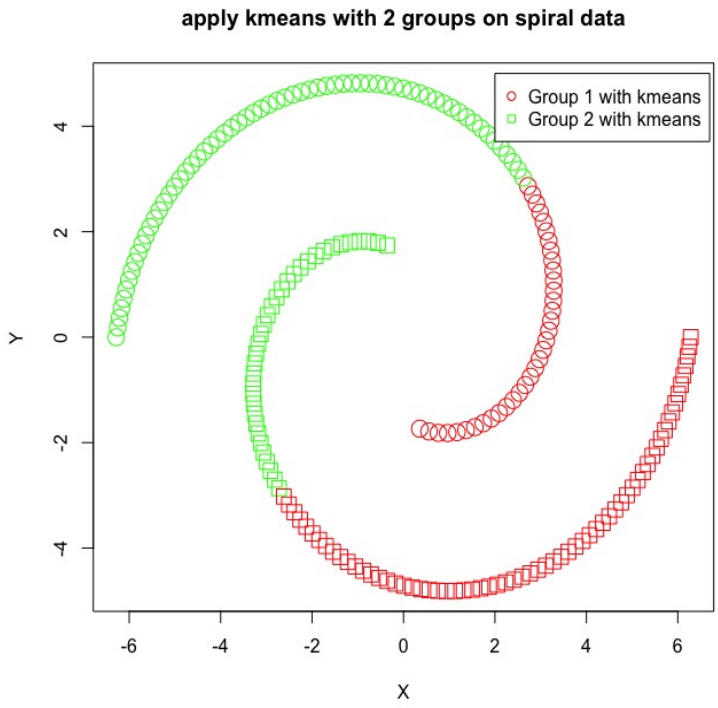


Figure 5.6: kmeans' result

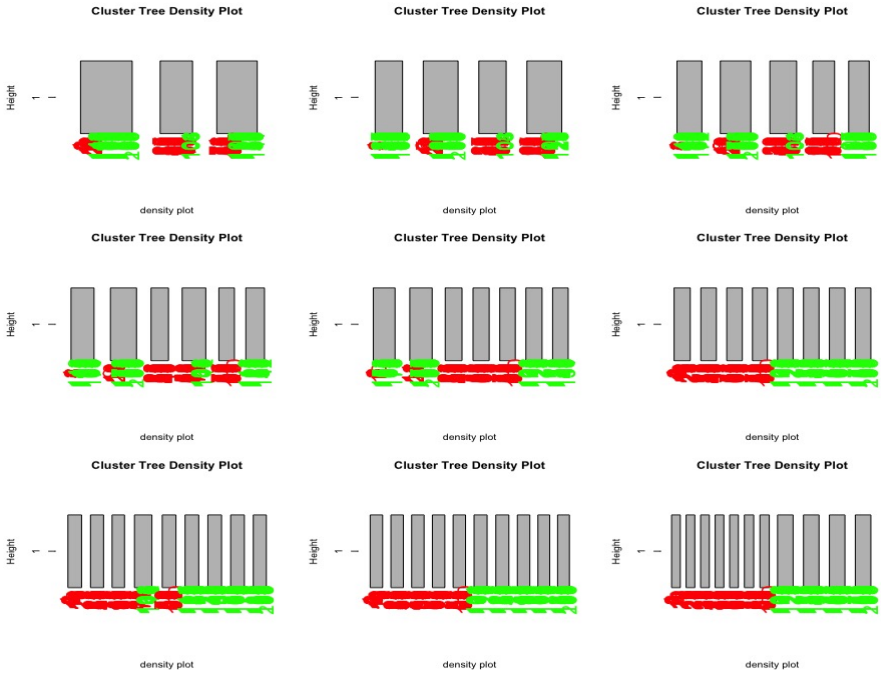


Figure 5.7: kmeans with 3-11 clusters

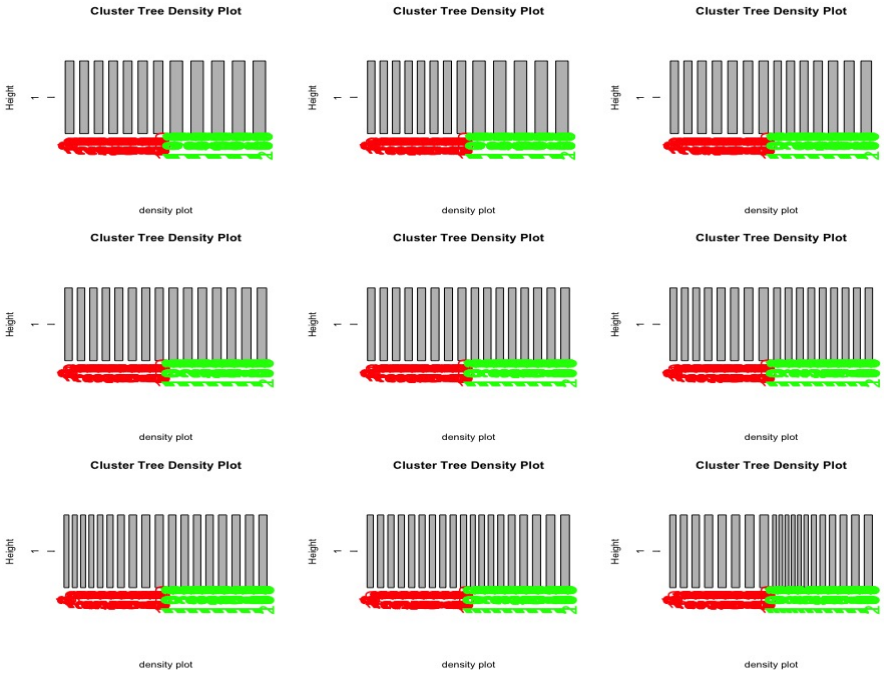


Figure 5.8: kmeans with 12-20 clusters

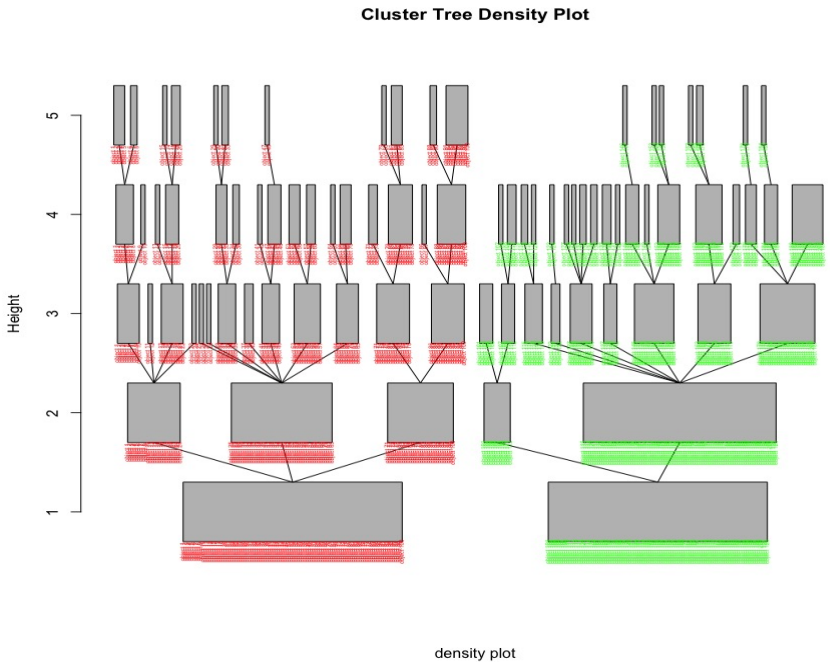


Figure 5.9: TREC result on spiral data

```
library(mclust)
clustering3 <- Mclust(data)
```

Then, with `TREC` package, we are able to generate cluster tree directly with one function call:

```
library(TREC)
clustering4 <- combineClusterings(clustering1,clustering2,
                                  clustering3)
```

Now you may plot cluster tree, as shown in Figure 5.10: Note, in the figure, red labels belong to the same cluster in true cluster. Similarly, green and blue labels also belong to the same cluster.

```
plot(clustering4,labels = TRUE, labels.col = c(rep('red',50),
                                  rep('green',50),rep('blue',50)))
```

We can calculate distance between clustering outcomes based on graph family framework with one function call:

```
distance <- clusDist(clustering1,clustering2,clustering3,clustering4)
```

Also, we may map distance matrix to 2d coordinates by selecting the most important two eigenvalues to show their relation:

```
gram <- distToGram(distance)
decomp <- eigen(gram)
evals <- eigen(gram)$values
coords <- eigen(gram)$vectors
savePar <- par(mfrow = c(1,2))
plot(evals/max(evals), type ="b", ylab = "contribution",
      main = "Contributions to dimensionality",
      sub = "Only two dimensions needed")
plot(coords[, 1:2], pch = 0:3, cex = 3,
      xlab = "Var 1", ylab = "Var 2",
      main = "Comparing clusters in cluster space",
      sub = "kmeans and model based agree")
legend(x=0,y=0,pch = 0:3, legend = c('kmeans','dbscan','mclust','trec'))
par(savePar)
```

The resulting graph is shown in Figure 5.11. As we see above, using `TREC` is simple and straightforward, which also provides quantative measure as distance and 2d visualization.

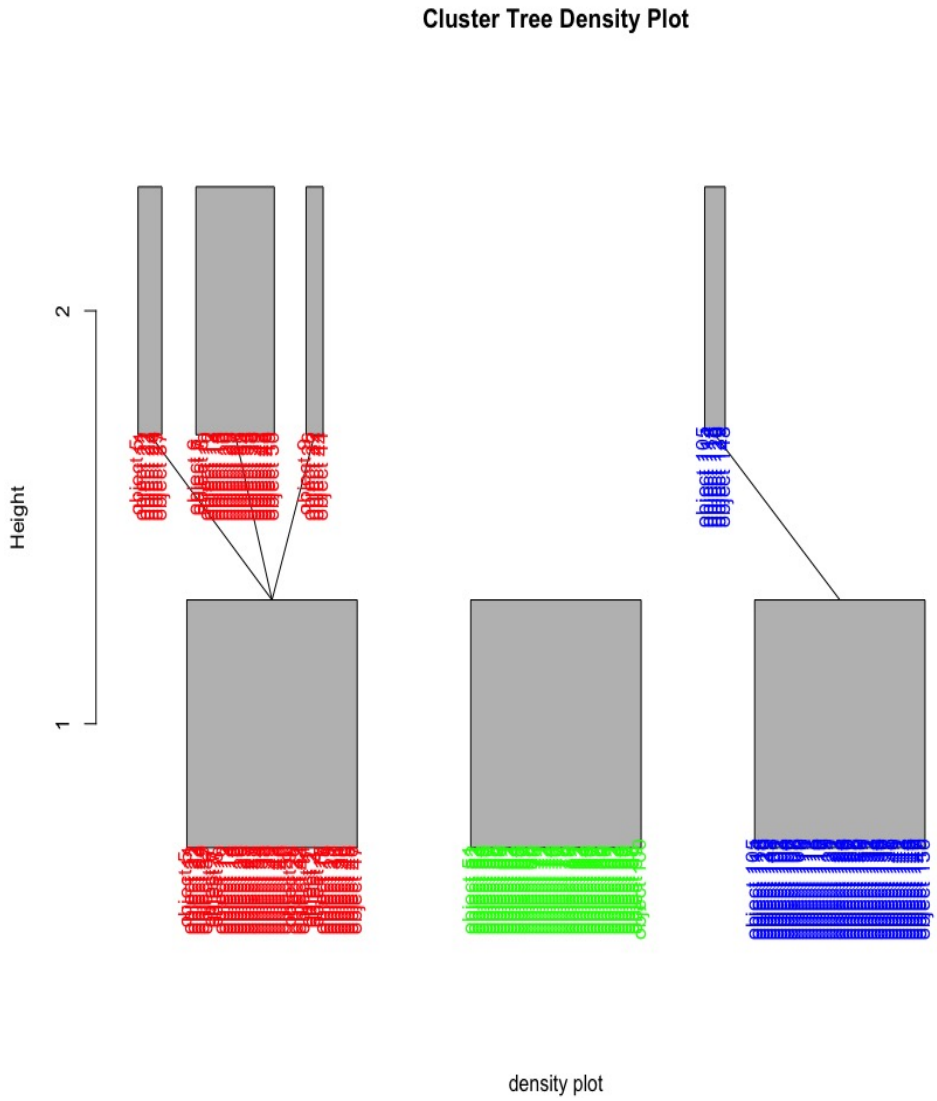


Figure 5.10: illustration plot

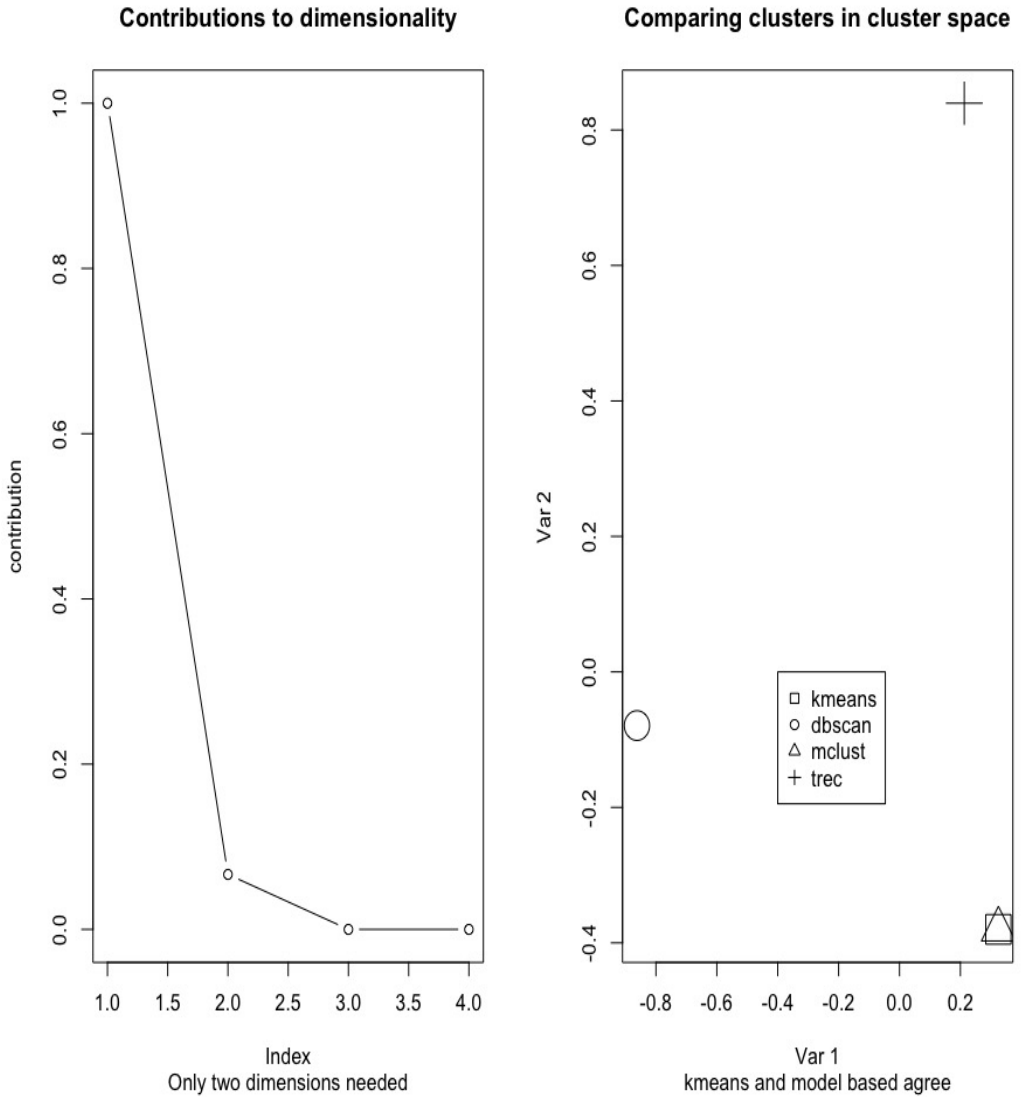


Figure 5.11: illustration of transforming distance to Gram matrix and mapping to 2d coordinates

Chapter 6

Conclusion

In this research paper, we introduce a R package `TREC` which combines multiple clustering outcomes as a unique cluster tree. `TREC` uses a general graph family framework which can be applied to all clustering outcome. To understand clustering qualities, we define the notion of clustering distance between two clustering outcomes. Moreover, we devise plot function to visualize multiple clusterings as a cluster tree. We modified our algorithm to accelerate computation of `TREC`. We apply `TREC` on real data sets, it turns out `TREC` is able to combine features of different algorithms and retain meaningful clusters.

Bibliography

[1] D. Ashlock, E.Y. Kim, and L. Guo. Multi-clustering: Avoiding the natural shape of underlying metrics. *Smart Engineering System Design: Neural Networks, Evolutionary Programming, and Artificial Life*, 15:453–461, 2005.

[2] J. D. Carroll and J. E. Corter. A graph-theoretic method for organizing overlapping clusters into trees, multiple trees, or extended trees. *Journal of Classification*, 12:283–313, 1995.

[3] E. Dimitriadou, A. Weingessel, and K. Hornik. Voting-merging: An ensemble method for clustering. *Lecture Notes in Computer Science*, 2130:217 – 224, 2001.

[4] W. Donath and A. Hoffman. Lower bounds for the partitioning of graphs. *IBM J. Res. Develop.*, 17, 1973.

[5] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of KDD-96*, 1996.

[6] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[7] C. Fraley and A. E. Raftery. Mclust: Software for model-based cluster analysis. *Journal of Classification*, 16:297–306, 1999.

[8] A. Fred and A. K. Jain. Evidence accumulation clustering based on the k-means algorithm. In *Structural, Syntactic and Statistical Pattern Recognition*, pages 442–451, 2002.

[9] John A Hartigan. Statistical theory in clustering. *Journal of classification*, 2(1):63–76, 1985.

[10] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[11] Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[12] Michael D Lee. A simple method for generating additive clustering models with limited complexity. *Machine Learning*, 49(1):39–58, 2002.

[13] J. MacQueen. Some methods for classification and analysis of multivariate observation. *Proc. 5th Berkeley Symp. Math. Stat. Prob.*, 1:281–297, 1967.

[14] M. Meila and J. Shi. A random walks view of spectral segmentation. In *8th International Workshop on Artificial Intelligence and Statistics*, 2001.

[15] R.W. Oldford and W.Zhou. Tree reduced ensemble clustering and distances between cluster trees based on a graph algebraic framework. 2014.

[16] A. Strehl and J. Ghosh. Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583 – 617, 2002.

[17] W. Stuetzle. Estimating the cluster tree of a density by analyzing the minimal spanning tree of a sample. *Journal of Classification*, 20:25–47, 2003.

[18] A. Topchy, B. Minaei-Bidgoli, A. K. Jain, and W. F. Punch. Adaptive clustering ensembles. In *Proceedings of the 17th International Conference on Pattern Recognition*, pages 272 – 275, 2004.

[19] A. Tversky. Features of similarity. *Psychological Review*, 84(4):327 – 352, 1977.

[20] Zhou, Wu. A new framework for clustering. 2010.

Appendix A

Installation of R package TREC

For `TREC`'s code, you can visit github repository `https://github.com/rwoldford/trec`. You can follow the following instructions to install `TREC`.

1. Click `https://github.com/rwoldford/trec`, click green button "Clone or download" and download zip.
2. Open terminal, go to directory where you download , and enter following command in your terminal:

   ```
   tar -zcvf trec.tar.gz your_file_name
   ```

3. Open RStudio, click `Packages`, then click `Install`, install `trec.tar.gz` you just created.
4. Now you can use TREC!