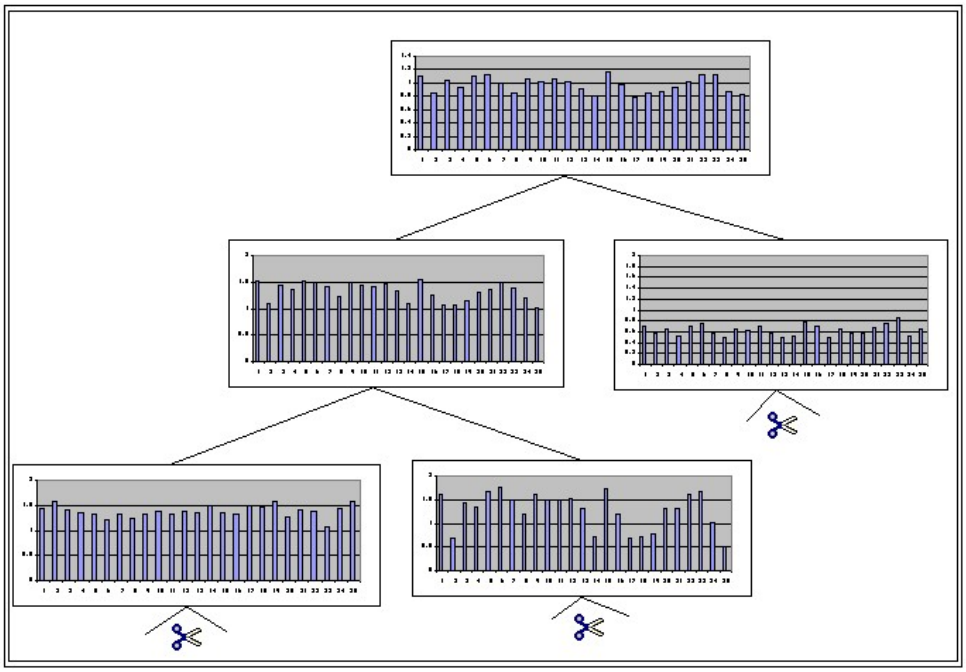
k-divisive Hierarchical Clustering

Master Thesis



Student	Supervisor
Addin Osman Mohamed A.	Jose M. Peña

Faculty of Engineering and Science
Aalborg University

Department of Computer Science

TITLE: k-divisive Hierarchical Clustering
Master Thesis

SEMESTER PERIOD:
DAT6,
February 1st - June 26th, 2003

PROJECT GROUP:
E1-121

GROUP MEMBERS:
Addin O. Mohamed A., aosman@cs.auc.dk

SUPERVISOR:
Jose M. Peña, jmp@cs.auc.dk

NUMBER OF COPIES: 4

NUMBER OF PAGES: 37

SYNOPSIS:

In this master thesis, a novel divisive hierarchical clustering algorithm has been proposed. The algorithm is called the k-divisive hierarchical clustering algorithm. The aim of this algorithm is to overcome some of the disadvantages of the well-known divisive hierarchical clustering algorithms. The proposed algorithm builds the hierarchy by splitting a cluster into tow sub-clusters. The splitting process is performed by implementing the k-means algorithm, and the process of splitting a cluster is stopped by using two proposed stop criteria. The splitting process is halted so as to end up with a high quality dendogram. The algorithm together with the stop criteria are implemented and tested using artificial databases as well as real world databases.

Contents

Chapter 1

Introduction

This project represents the second part of my master thesis. In the first part (dat5 project) we have implemented some data mining techniques in a real database which we have got from a company [31]. Our goal in dat5 project was to use data mining techniques to get better understanding of dealer behaviors, investigate factors influence the amounts of clothes a dealer sells, and to discover what causes so many dealers to stop their business. By finishing the first part of the master thesis we have gained good experience in data mining, and we have realized that data mining is a very interesting field of research. In this project I have moved to another problem in data mining regarding hierarchical clustering.

In the rest of this section a brief description of data mining, clustering, and hierarchical clustering are presented. The objective of the project is presented at the end of this section.

1.1 Data Mining

Normally, organizations worldwide are collecting data and storing their valuable information in databases so that, they can access and use them in the future. Data access solutions provided the ability to retrieve information from databases through specified queries. But still there are some valuable information which can not be retrieved by using the well known query engines like SQL. The retrieval of these valuable information is becoming more sophisticated as the mass of the data become overwhelming.

It is now very clear that to retrieve meaningful or valuable hidden information from such kind of databases, automated tools must be developed. Moreover, these tools must be sophisticated enough to search for correlations among the data not determined by the user, as the potential of unforeseen relationships to exist among the data is very high. A successful tool set to achieve these goals will find useful golds of information in the disorderly mass data space, and present them to the user in easy understandable format.

> "As an example, consider a marketing director trying to associate demographics in a database of one million record. Since multiple dependencies surely exist in the data, the marketer will quickly get lost in one or both of two ways: (1) overwhelmed by the sheer amount of data, the number of profiles and associations that emerge are incomprehensible; (2) the resultant narrowing of objectives to reduce the effects of being overwhelmed, which results in the loss of critical relationships. This effect brings to mind the old adage of "being up to your eyebrows in alligators, with no ability drain the swamp". Fortunately, a group of techniques now exist that find real, usable information in the data glut, with the capability of presenting it to the decision-maker in a useful format. This set of tools utilize a technique called "data mining" and supports the goal of knowledge discovery (finding sometimes unexpected information in the data)" [3].

As the result, data mining can be defined as a process of analyzing such huge data from different perspectives and summarizing it into useful information that can be used for increase revenue, cuts costs, or both.

Data mining is applicable in many research and science areas as well as in the industry and business applications. It suits perfectly within application areas where there is a huge amount of factors each affecting the application area. The biology and medical research societies which deals with a huge amount of data such as DNA-profiles, diseases, symptoms, blood-types and drugs, have been using data mining with success. Data mining has also pulling attention from the common business areas such as supermarkets and mobile phone providers.

Data mining techniques can be divided into different groups according to different aspects such as, knowledge to be discovered, the kind of databases to be mined, and the kinds of techniques to be implemented.

For example, for the kinds of knowledge to be discovered, one may classify data mining techniques into clustering, classification, association, etc., or based on the level of concepts to be discovered, into primitive level, high level, multiple-level, etc.

Clustering is an unsupervised (instances are unclassified) learning process which divides data instances into groups such that instances in a group are similar to each other and dissimilar to instances in other groups. Data modeling puts clustering in a historical perspective rooted in mathematics, statistics, and numerical analysis. From a machine learning perspective, clusters correspond to hidden patterns, where the resulting system represents a data concept. From a practical perspective clustering plays an outstanding role in data mining applications. For instance, medical researchers use cluster analysis to determine which diseases have similar patterns of incidence, market researchers use cluster analysis to determine which brands or trademarks of products attract the public perceives in a similar way, etc.

There are several approaches to distinguish the data clustering techniques, one of these approaches distinguish them as partitional and hierarchical. These two types can be defined as:

- **Partitional:** Develops a partition of the data such that instances in a cluster (or group) are more similar to each other than they are to instances in other clusters.
- **Hierarchical:** Is a sequence of partitions in which each partition is nested into the next partition in the sequence. Hierarchical clustering algorithm constructs a tree of nested clusters based on proximity information. The primary purpose for building a cluster hierarchy is to structure and present data at different levels of abstraction.

In the next part of this project the clustering is discussed generally in more detail, specifically the *divisive* hierarchical clustering.

1.2 The Objective

The *divisive* hierarchical clustering opposing to *agglomerative* hierarchical clustering, has great lack in research and has taken a very few space in literature. One of the main problems with the valid *divisive* algorithms is the running time, because most of the algorithms build a complete hierarchy while some others try all of the combinations between the instances to split a cluster, which magnifies the running time of the algorithm.

The aim of this project is to develop a new divisive hierarchical algorithm which builds the hierarchy (dendgoram) using a splitter who is responsible for splitting a cluster, and the second goal is to modify some methods to stop splitting a cluster, if the splitting is not going to improve the quality of the cluster. And the last goal is to compare these methods in artificial data as well as in real data. The stopping criterion for splitting a cluster is the main issue in the algorithm. Because it automatically determines the number of clusters and plays an important role in building high quality dendogram.

Chapter 2

Data Clustering

As discussed in Section 1.1, clustering is the unsupervised (instances are unclassified) learning process of discovering hidden patterns or meaningful sub-groups in a database, which can not be discovered by using normal databases query engines, e.g. SQL. It is widely used in geographic information systems, medical image analysis, satellite imagery, etc. So as to discover hidden patterns; it assigns a set of given data instances into a number of groups or clusters, such that:

- The inter-cluster similarity is maximized.
- The intra-cluster similarity is minimized.

The goal of clustering is to reduce the amount of data by grouping similar data items together. Such grouping helps humans to process information and discover hidden patterns in the data, which can be used in decision making.

In order to assign elements to clusters a similarity measurement is needed and the selection of the similarity measurement is highly dependent on the type of the attributes, e.g. Euclidean distance measurement can be used if attributes are continuous.

In the rest of this section the requirements of good clustering techniques, types of clustering methods, k-means algorithm as partitional clustering method, and hierarchical methods are discussed in more detail. The project focuses in hierarchical clustering algorithm, because it represents the base of the project objectives.

2.1 Requirements of Good Clustering Techniques

The good clustering techniques must satisfy the following requirements:

- **Scalability**: The cluster algorithm should be applicable to huge databases and performance should decrease linearly with data size increase.
- **Versatility**: Clustering objects could be of different types - numerical data, boolean data or categorical data. Ideally a clustering method should be suitable for all different types of data objects.
- **Ability to discover clusters with different shapes**: This is an important requirement for spatial data clustering. Many clustering algorithms can only discover clusters with spherical shapes.
- **Minimal input parameter:** The method should require a minimum amount of domain knowledge for correct clustering.
- **Robust with regard to noise:** This is important because noise exits everywhere in practical problems. A good clustering algorithm should be able to perform successfully even in the presence of a great deal of noise.

- **Insensitive to the data input order:** The clustering algorithm should give consistent results irrespective of the order the data is presented.
- **Scalable to high dimensionality:** The ability to handle high dimensionality is very challenging but real data sets are often multidimensional.

Historically, there is no single algorithm that can fully satisfy all the above requirements. It is important to understand the characteristics of each algorithm so the proper algorithm can be selected for the clustering problem at hand. Recently there are several new clustering techniques offering useful advances, possibly even complete solutions and there are different types of clustering methods which are roughly discussed in the next section.

2.2 Types of Clustering Methods

Clustering methods or algorithms can be divided into different main types, one of these divides them into *hierarchical* and *partitional* clustering. Each one of these types can be divided into different sub types.

- *Partitional*: Is a non-hierarchical clustering method and decomposes a partition of the dataset, which results into a set of k clusters, such that instances in a cluster are similar to each other and dissimilar to instances in other clusters. A commonly used partitonal clustering method is the k-means algorithm [2, 13, 14, 19, 25, 33].
- *Hierarchical* Is a sequence of partitions in which each partition is nested into the next partition in the sequence. It constructs a tree of nested clusters based on proximity information. It proceeds building the tree recursively by merging smaller clusters into larger ones, or by splitting larger clusters, until all data items resides in one cluster or every item resides in a different cluster. The different clustering types of this method, differ in the way by which they select the two clusters to be merged or the cluster to be splitted. The end result of the algorithm is a tree of clusters called dendogram. This type of clustering is divided into:
  - *Agglomerative (bottom-up)*: Starts with each instance as separate cluster, and successively merge clusters according to a distance measure. The clustering may stop when all instances are in a single cluster or some user defined stop criterion is met.
  - *Divisive (top-down)*: Starts from one cluster containing all instances and successively split clusters into smaller ones, until each instance belongs to one cluster, or some stop criterion is met. This is similar to the approach followed by divide-and-conquer algorithm

In the rest of this section the k-means algorithm and hierarchical clustering algorithms are discussed in more detail.

2.3 k-means Clustering Algorithm

k-means algorithm is the most famous, classical, and iterative partitional clustering algorithms [2, 13, 14, 19, 25, 33]. It partitions the dataset into k clusters such that clusters are mutually exclusive (non overlapping), exhaustive (every item belongs to a cluster), and non-empty. The clustering criterion used in the k-means algorithm is based on the distance measurement (e. g. Euclidean distance), which allows the assignment of an instance to a cluster with similar instances. The distance is measured between instance and cluster centroid which, is the representative of the cluster (it can be the mean value of the instances in the cluster).

The algorithm: k-means. For partitioning based on the centroids of the clusters.
Input: The number of clusters k and n instances of a database $(d_1, \ldots, d_n)$.
Output: A set of k clusters $(c_1, \ldots c_k)$.
Method:

```
Randomly assign class labels to all instances.
Calculate the centroids of all clusters.
Repeat
  For every instance in the database and preserving the instances order Do
      Reassign the instance to the nearest cluster to it;
      if the instance changes its cluster, recalculate the
         centroids of the previous and current cluster;
until no instance can change its cluster or some stop criterion is met.
```

The above pseudo-code shows the k-means algorithm used in this project and it is described as:

1. The algorithm has two inputs, the instances to be clustered and the number of clusters.
2. In the first stage the algorithm randomly assigns clusters to all instances.
3. After the random assignment, the algorithm calculates the centroids of the clusters (the mean values of the instances in every cluster).
4. Then the algorithm repeatedly and preserving the order of the instances, re-assigns class labels to all instances (using some distance measurement, like Euclidean distance), and whenever any instance changes its cluster the centroid of the previous and current clusters of the instance are immediately recalculated. This process continues until no instance can change cluster or some stop criterion is met.
5. The algorithm returns the centroids of the clusters and the class labels of the instances as output.

The computational complexity of the algorithm is O(npt) where n is the number of samples, p is the number of features, and t is the number of iterations. In practice, the number of iterations is generally much less than the number of samples. So k-means algorithm is a simple and systematic algorithm for data clustering [12].

2.3.1 Disadvantages of the k-means

Even though it has been used in many applications and the most classical algorithm, the k-means algorithm has some disadvantages. The common ones are:

- The algorithms is very sensitive to the initial starting centroids.
- As many clustering algorithms, needs the number of clusters to be specified at the beginning.
- The performance of the algorithm is highly dependent in the order of the instances in the input.

2.4 Hierarchical Clustering

A general question facing researchers in many areas of data query like biology and medicine, is how to organize observed data into meaningful structures (groups or clusters). For example biologists have to organize the different species of animal before a meaningful description of the differences between animals is possible. According to the modern system employed in biology, man belongs to the primates (high rank of animals, which have hands, feet, large brain, etc.), mammals, and the animals. If we want to cluster a group of animals which have a man among them, it is obvious that man has more in common with all other "primates" (e.g. apes) than it does with "mammals" (e.g. dogs), man has more distance to mammals than primates. Therefore, hierarchical clustering which is widely used to find patterns in multi-dimensional datasets is appropriate to be used in this case. For example, by implementing the hierarchical clustering in this data, a good representation for the data can be found, in which the man and the other primates can be divided into one cluster which, is different than the one contains the mammals. Subsequently, the man and primates cluster can be divided again into sub clusters.

Hierarchical clustering techniques produce a nested sequence (tree like structure) of partitions, which graphically can be displayed as a tree, known as dendogram. The node on the top (root) of the dendogram

represents a single cluster, which includes all of the instances (all-inclusive), and each node in the bottom (leaf) of the dendogram represents a single cluster which includes an instance from the dataset (known as singletons). The number of singletons is equal to the total number of instances. A cluster in an internal node composes of combination of two nodes in the next lower level or from the splitting of a cluster in a node from the next higher level. The algorithm that builds the first type of the dendogram is known as *agglomerative* and the second one as *divisive* hierarchical clustering algorithm. The *agglomerative* hierarchical algorithm builds the dendogram in away known as bottom-up, where *divisive* hierarchical algorithm known as top-bottom clustering. All the clusters formed at any stage in the dendogram are mutually exclusive, exhaustive, and non-empty. Figure 2.4 shows graphically the two types of the hierarchical algorithms.

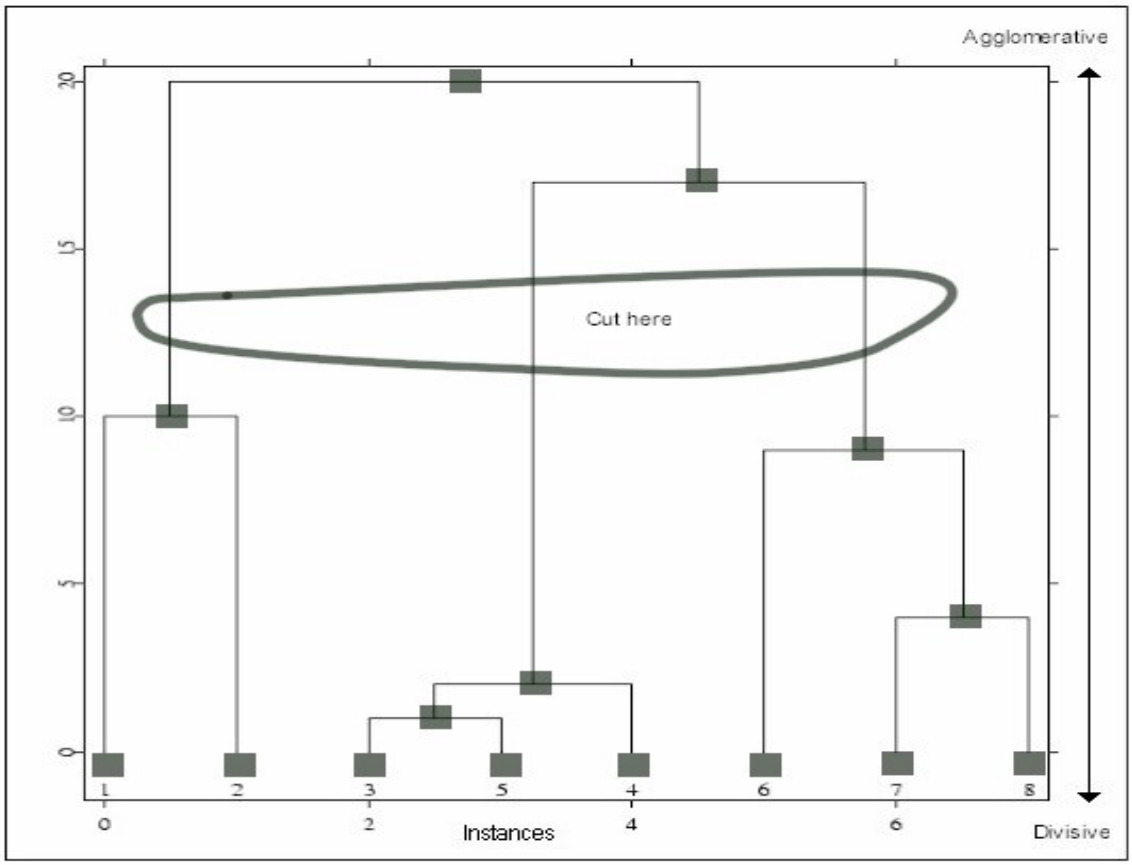


Figure 2.1: The figure shows a dendogram built by *agglomerative* and *divisive* hierarchical algorithms. The bottom nodes show 8 instances as individual clusters and the top node represents the final cluster, which includes the 8 instances. The height of an internal node represents the Euclidean distance or similarity between each two child nodes in the dendogram. "Cut here" shows the place to cut the dendogram for a specific number of clusters determined by an expert.

The general properties of the two hierarchical clustering methods can be depicted by the mathematical formulas shown below.

Let D be a dataset with n instances, such that:

$$D = (d_1, d_2, \ldots, d_n) \tag{2.1}$$

where d_i is the ith instance of the dataset, which represents an m-vector contains the values of m attributes of the instance:

$$d_i = (x_{i1}, x_{i2}, x_{i3}, \ldots x_{im}) \tag{2.2}$$

If h represents a level in the dendogram, k the number of clusters in that level, and E_h is the sub-set of the data in that level ($E_h \subseteq D$), which is splitted into k clusters ($C_{h1}, C_{h2}, \ldots, C_{hk}$).

The following must be satisfied:

$$C_{hi} \cap C_{hj} = \phi, \ for\ all\ \ \leq i, j, \leq h, i \neq j \tag{2.3}$$

$$C_{hi} \neq \ for\ all\ 1 \leq i \leq h. \tag{2.4}$$

$$\bigcup_{i=1}^{h} E_i = D \tag{2.5}$$

2.4.1 Agglomerative Hierarchical Clustering

Any agglomerative hierarchical clustering can be completely described by means of a *cophenetic matrix*. A cohpenetic similarity (or distance) of two instances is defined as the similarity level at which these instances appear for the first time to be members of the same cluster. Any dendogram can be uniquely represented by a matrix in which the similarity for a pair of instances is their cophenetic similarity.

There are numerous methods used in agglomerative algorithms to determine the pairs of clusters to be merged. In most agglomerative algorithms, this is accomplished by selecting the most similar pair of clusters, and numerous approaches have been developed for computing the similarity between two clusters, e.g. single-link, complete-link, and UPGMA schemes [2, 5, 28]

1. Single Linkage: Also known as nearest neighbor method. Similarity between two clusters is computed based on the minimum distance between the instances belonging to the corresponding clusters.

2. Complete Linkage: In this scheme the dissimilarity between two clusters is equal to the greatest dissimilarity between an instance of one cluster and an instance of another cluster. In this scheme, the maximum distance between the instances in the analyzed clusters is considered.

3. UPGMA (Unweighted Pair-Group Method using Averages): In this scheme, two clusters are merged based on their average distances between the instances in the clusters.

The algorithm: Agglomerative Hierarchical Clustering Algorithm.
Input: A set of n instances $(d_1, \ldots, d_n)$.
Output: A dendogram.
Method:

```
Initially assign each instance to a separate cluster.
While all instances are not in one cluster
    Merge a pair of nereast clusters to create a new cluster.
```

The above pseudo-code shows the *agglomerative* hierarchical clustering algorithm.

2.4.2 Divisive Hierarchical Clustering

Divisive hierarchical clustering algorithm as mentioned before, builds the dendogram in top-bottom way. It is less popular than the agglomerative hierarchical algorithm, therefore a brief description about it in this report is presented.

Divisive hierarchical algorithms can be divided into two types, *monothetic* in which the divisions are based on just a single attribute, and *polythetic* in which the divisions are based on all attributes. *Monothetic* algorithms are more faster in operation than *polythetic* algorithms, but tend to give power results.

Figure 2.2 shows a dendogram built by using a divisive algorithm. In this figure, at the first level of the dendogram there is one cluster. It is obvious that this cluster can be divided into three clusters, one big cluster contains 2000 instances and another two clusters contain 1000 instances each. In the second level of the dendogram the cluster is divided into two clusters, one contains the big cluster and the other one contains the other two small clusters. In the third level the big cluster is very compact for that reason it has not been divided. The other cluster is divided into another two compact clusters. The implemented stop-criterion plays a vital role in catching the structure of these clusters.

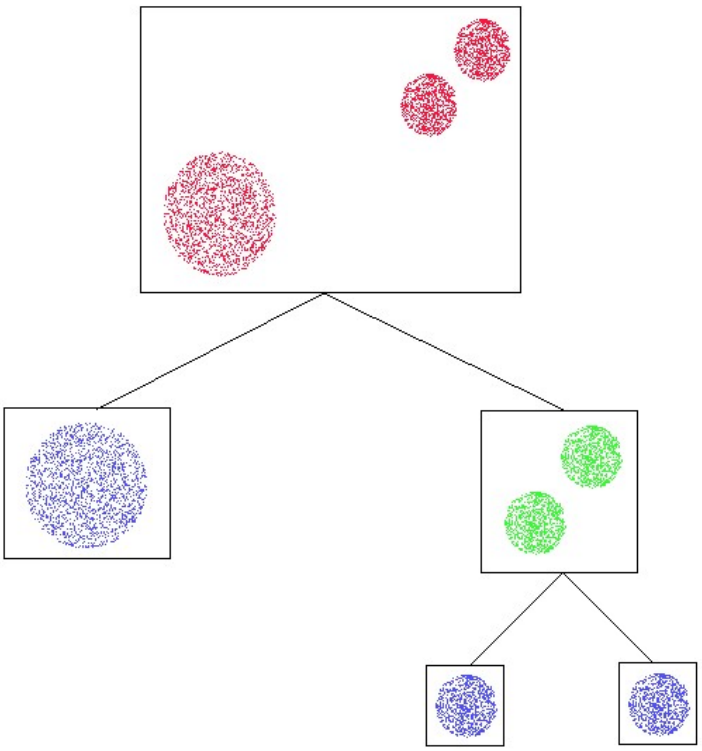


Figure 2.2: The figure shows an artificial dendogram built by a divisive algorithm. The leaf nodes (in blue color) show the final compact clusters which are not splitted. The big cluster contains 2000 instances, and the other two contain 1000 instances each.

The algorithm: Divisive Hierarchical Clustering Algorithm.
Input: A set of n instances $(d_1, \ldots, d_n)$.
Output: A dendogram.

Method:

```
Initialize one cluster with all instances.
While not each instance resides in a separate cluster
    Split in the best way a cluster with more than one instance into two clusters.
```

The above pseudo-code shows the *divisive* hierarchical clustering algorithm.

2.4.3 Problems of Hierarchical Clustering

One of the drawbacks of the two algorithms is that clusters are usually not optimal. This happens because a tree does not allow branches to split and then merge back or merge then split back. The divisive algorithm always splits but does not merge and the agglomerative algorithm always merges but does not split.

Agglomerative algorithms have gained more attention from the researchers in the literature. Unfortunately the divisive algorithm has been ignored in the literature and very few researches have been done about it. The main reason for that is the running time of the divisive algorithm. As discussed before, the general divisive algorithm starts with one cluster which contains all the instances and before splitting the instances into two clusters, it checks all of the possible combinations between the instances. For example, if the algorithm clusters n instances from a dataset, then in the first stage it checks $2^{(n-1)} - 1$ combinations to find the combination that best splits the dataset. This number grows exponentially fast and computationally burdensome.

Nevertheless, it is obvious that the running time of the divisive algorithm can be improved by ignoring some divisions, most of which is totally in appropriate. Therefore, in this project, the k-divisive algorithm (discussed in detail in the next section) has been proposed as a modified divisive clustering algorithm which uses the k-means algorithm as splitting algorithm.

Chapter 3

k-divisive Hierarchical Clustering Algorithm

As mentioned in Section 2.4, most of the common *divisive* hierarchical clustering algorithms begin at the first stage with a unique cluster (all-inclusive) which contains all instances of the dataset, then recursively split each cluster into two sub clusters until every instance resides in a separate cluster. In common *divisive* hierarchical clustering algorithms a cluster splitting process is done by checking all of the possible splits of the instances in the cluster, then selects the best one which is believed to end up with a meaningful partitioning. Hence the computational cost of the algorithm will increase and the algorithm ends up with a very big dendogram which is very difficult to interpret.

The splitting process of any divisive hierarchical algorithm can be categorized by one of the approaches discussed below:

- Complete splitting of the clusters until every instance resides in a single cluster.
- Split the cluster with the highest number of instances which is greater than some threshold.
- Split the cluster that doesn't satisfy a selected stop criterion.

The first approach is very simple. When implemented, a complete dendogram will be formed, but it has the disadvantage of forming a big dendogram or tree, which is difficult to visualize, also it is difficult for an expert to decide a place to cut the dendogram. The second approach is also very simple, it does not give a full dendogram, and builds a "balanced" dendogram that gives the leaves a chance to end up with the same size, but ignores the quality of the dendogram. The third one, is the commonly used approach. The problem of avoiding "meaningless" partitioning of clusters can be solved by implementing a stop criterion. The quality of the dendogram is highly dependable on the selected stop criterion.

The aim of this project was to illustrate a new divisive hierarchical algorithm for clustering to overcome the above mentioned problems. The splitting process applied to solve these problems is based on the third approach discussed above. Our proposed algorithm is the k-divisive algorithm where, the classical k-means algorithm is used as splitter, which is responsible for splitting a cluster into two sub clusters, and two independent stop criteria are proposed to halt the cluster splitting process. These two stop criteria are based on hypothesis-test and k-fold cross-validation, and they play a vital role in the proposed k-divisive hierarchical clustering algorithm. The first stop criterion is known as *hypo-test* and the second one as *artilab k-folds*.

In the rest of this section, the k-divisive hierarchical algorithm is presented first, the *hypo-test* stop criterion second, and the *artilab k-folds* stop criterion last.

3.1 The Algorithm

The k-means algorithm is the most classical, widely used, and most celebrated clustering technique, hence it is one of the best representatives of the partitional clustering algorithms. For this reason it has been selected as a splitter in the k-divisive hierarchical clustering algorithm. The pseudo-code below shows the algorithm.

The algorithm: k-divisive hierarchical clustering to build a dendogram.
Input: n instances of a dataset $(d_1, d_2, \ldots, d_n)$.
Output: A dendogram that represents the top-down hierarchy of the instances.
Method: Start(instances)

```
Step 1: Initialize a dendogram with a root node which contains all instances.
Step 2: Call Split(instances, root).
```

Method: Split(instances, node)

```
Step 1: Implement k-means to split instances into two clusters CL and CR.
Step 2: Implement the stop criterion on the dataset.
      Step 3: If the stop criterion is no satisfied:
              Add new right node R to ''node'' containing the instances of CR.
              Add new left node L to ''node'' containing the instances of CL.
              Call Split(instances of CR, R).
              Call Split(instances of CL, L).
      Step 4: else halt splitting (return 0).
```

The above pseudo-code shows the k-divisive hierarchical algorithm. The input of the algorithm is n instances. At the beginning the algorithm calls the "Start" method. This method initializes the dendogram with a root node. This node contains the n instances of the data set as one cluster. Then the method calls the "Split" method, which takes two parameters, the instances to be splitted and the node in the dendogram which contains these instances.

The "Split" method implements the k-means algorithm (shown in Section 2.3) on the instances to split them into two clusters CR and CL. The method then runs the stop criterion on the instances. If the split criterion is satisfied then it adds two nodes to the node received as a parameter in the method. One of the nodes represents the right cluster and the second one the left cluster. Then the algorithm recursively calls the "Split" method two times with the right and left clusters. Otherwise the splitting process is halted.

The output of the algorithm is the dendogram which represents the divisive hierarchy of the data set and the centroids of the clusters formed by the algorithm at any level of the dendogram.

3.2 Hypo-Test Stop Criterion

The *hypo-test* (hypothesis-test) stop criterion is one of the two stop criteria we suggest for the k-divisive algorithm and it is mainly based on cluster-validity.

Generally cluster-validity is defined as a quantitative evaluation of clustering results and measuring the quality of the obtained data partitions. The k-divisive algorithm builds the dendogram by splitting a cluster into two sub clusters. The main issue in this case is to determine whether the new structure resulted from the splitting process improves the quality of the dendogram or not. Therefore, the splitting process of a cluster can be treated as the validity of the structure recovered by the splitting process.

In the beginning of this sub-section some preliminaries about cluster-validity are presented. These preliminaries will help understanding the *hypo-test* stop criterion, which is presented later in this sub-section.

3.2.1 Indices of Cluster Validity

A clustering structure is "valid" if it is "unusual" in analytical way, and its quality can be assessed by using various cluster validity indices. It is very easy to define indices of cluster validity. However, it is very difficult to establish accurately thresholds on such indices that determine the "unusualness" of the index.

The quality of a structure recovered through cluster analysis can be determined by using an index of cluster analysis in order to be interpreted objectively, e.g. square error, and Hubert's statistic (shown in Section 3.2.5). The validity of clustering structure can be showed in terms of three criteria as follows:

1. **External Criterion**: Measures the performance by comparing a clustering structure to a priori information. This criterion validates a clustering result by comparing it to another partition or hierarchy of the instances obtained by an independent process based on information other than the given data set. For example, this criterion measures the degree of correspondence between cluster numbers, obtained from a clustering algorithm, and class labels, assigned a priori.

2. **Internal Criterion**: Uses only the data themselves to determine the fit between the structure and the data. For example, this criterion would be used to measure the degree to which a partition, obtained from a clustering algorithm, is justified by the given proximity matrix.

3. **Relative Criterion**: This criterion decides which of two structures is better in some sense, such as being more stable or more appropriate for the data. For example, a relative criterion would measure quantitatively whether a single-link or a complete-link hierarchy fits the data better.

A criterion shows the approach by which a clustering structure is to be validated, while an index is a statistic that determines which validity is to be tested.

In this project we focus on the internal criterion.

3.2.2 Hypothesis-Test

Testing up a hypothesis is an essential part in statistics. It can be defined as the evaluation (or testing) of a claim (or hypothesis) about a true value of an index.

A statistic function S is a random variable and its distribution describes the relative frequency (the proportion of times that the event occurs) with which values of S occur under some hypothesis. Statistic S can be a compactness measure of a cluster or square error. A hypothesis is a statement about the relative frequency of events in the sample space that expresses one's concept of phrases such as "the data are random" or "the data are clustered". Based on the distribution of S, a hypothesis is tested by observing a value of S and decide about its "unusualness".

The hypothesis we are testing is called the null hypothesis (H_0) and it is the complement of a second hypothesis about the index value, called the alternative hypothesis (H_1). The two hypothesis must be mutually exclusive.

3.2.3 Null Hypothesis

The null hypothesis (H_0) in cluster validity is a statement of "no structure" that expresses the frequency with which a member of a baseline population occurs. A major problem in cluster validity is establishing the distributions of statistics under the H_0. The three most common null hypothesis in cluster validity are:

- **Random Graph Hypothesis**
  H_0 : All $n \times n$ rank order proximity matrices are equally likely (n is the number of instances in the dataset).

  The baseline population for this hypothesis is the set of all ordinal proximities matrices of the n instances. An n x n proximity matrix has $n(n-1)/2$ inputs, which can be integers from 1 to $n(n-1)/2$. Each explicit ordinal proximity matrix has a probability $1/[n(n-1)/2]!$.

- **Random Label Hypothesis**
  H_0 : All permutations of the labels on n instances are equally likely.

  The baseline population for this hypothesis is the set of $n!$ permutations of the labels on the n instances under study. And each permutation is assigned probability $1/n!$.

- **Random Position Hypothesis**
  H_0 : All sets of n locations in some region of n-dimensional space are equally likely.

  This hypothesis requires all sets of n points in some region of d-dimensional space be equally likely and is appropriate for ratio data, e.g. the region could be hypercube having side one or hyper-sphere

of fixed radius. Another approach for this hypothesis is to require that each of the n points be entered randomly into the region. The main idea of this hypothesis is sampling a set of points with no structure such that the meaning is clear and the distribution of the interesting statistics can be determined or estimated.

Hypothesis tests apply indices in order to decide the appropriateness of the recovered structure for the data. This is done in the external and internal criterion by testing whether the value of the index is either unusually large or small, consequently this requires the baseline to be established.

Given the distribution of S under H_0, and assuming that a large value of S indicates that H_0 should be rejected, we can place a critical value s_α, on S. The critical value for a hypothesis is a threshold to which the value of the test statistic in a sample is compared to determine whether or not the H_0 is rejected.

The critical region is a set of values of the test statistic for which the H_0 is rejected. The sample space for the test statistic is partitioned into two regions separated by the critical value, one critical region leads to the rejection of the H_0, and the other does not. The testing situation is shown in Figure 3.1.

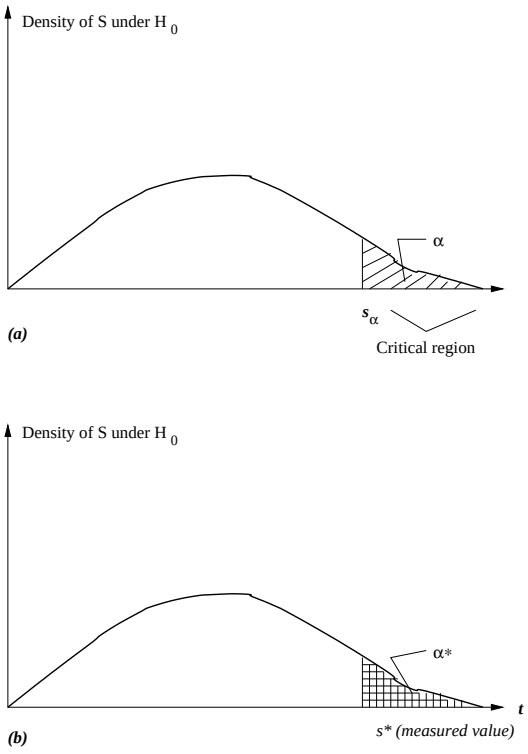


Figure 3.1: Hypothesis testing: (a) critical region for testing H_0; (b) critical level of significance α^*

Assume that the measured value of S is s^*. In this case we want to check whether s^* is large enough for the unusualness of H_0, we apply the following rule:

$$If\ s^*\ \geq s_\alpha,\ reject\ H_0 \tag{3.1}$$

In cluster validity, statistics are selected such that either large values or small values show structure, but generally any kind of critical region can be defined.

3.2.4 Alternative Hypothesis

The alternative hypothesis (H_1) is a statement of what a statistical test is set up to establish. But once the test has been carried out, it is always given in terms of the null hypothesis. We either "reject H_0 in favour of H_1" or "do not reject H_0"; we never conclude "reject H_1", or even "accept H_1" [34].

The steps of cluster validity defined in this section can be summarized as:

- A null hypothesis which is depicting the idea of no structure should be defined according to the data type.
- An index sensitive to the existence of structure in the data should be chosen and the distribution of the statistic under the null hypothesis should be established.
- A threshold can be formed, which creates a formal test of hypothesis.
- The alternative hypothesis helps in determining the presence of the structure.

3.2.5 Hubert's Γ Statistic

The Hubert's Γ statistic is the point serial correlation between two matrices X(i,j) and Y(i,j). The high value of this index indicates high correlation between the two matrices. It can be expressed in raw form as follows when the two matrices are symmetric:

$$\Gamma = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X(i,j)Y(i,j) \tag{3.2}$$

If m_x and m_y denote the sample means and s_x and x_y denote the sample standard deviations of the entries of matrices X and Y the normalized Γ statistic is

$$\Gamma = (1/M) \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} [X(i,j) - m_x][Y(i,j) - m_y]/s_x s_y \tag{3.3}$$

Where M = n(n-1) is the number of entries.

Let g(1), g(2), ..., g(n) denote a permutation of the integers 1, 2, ..., n. The random variable whose distribution is required under H_0 can be written as:

$$\Gamma(g) = (1/M) \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} [X(i,j) - m_x][Y(g(i),g(j)) - m_y]/s_x s_y \tag{3.4}$$

Example 1. *This example demonstrates the estimation of the distribution of* Γ *under the random label hypothesis for two pattern matrices. The 80X data set [18] contains 45 instances, each 15 instances in an 8-dimensional space from different 3 class labels. Suppose that we want to test whether the class labels are matched unusually well to the locations of instances in the 8-dimensional feature space. Matrix* X *is taken to be the matrix of Euclidean distances between the instances and matrix* Y *contains a 1 in position (i,j) if the* i*th and* j*th instances are in different clusters and a 0 if they are in the same cluster. If the first 15 instances are from cluster A, the next 15 are from cluster B, and the last 15 from cluster C, then* Y *can be blocked into nine 15 x 15 sub matrices as indicated below, where* 1 *denotes a 15 x 15 matrix of 1's and* 0 *denotes a 15 x 15 matrix of 0's:*

$$Y = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \tag{3.5}$$

Only the 990 upper diagonal entries are used in the computation. A simple counting argument shows that

$$m_y = \frac{3 \times 15^2}{990} = \frac{675}{990} = 0.682 \quad and \quad s_y = \sqrt{\frac{675}{990} - (\frac{675}{990})^2} = 0.466 \tag{3.6}$$

The mean and standard deviation of the Euclidean distances between patterns were found to be

$$m_x = 9.07 \quad and \quad s_x = 1.88 \tag{3.7}$$

The gamma statistic can be written as follows for this example.

$$\Gamma = \{(\frac{1}{990}) \sum_{i=1}^{44} \sum_{j=i+1}^{45} X[i,j]Y[g(i),g(j)] - (0.682)(9.07)\}/(0.466)(1.88) \tag{3.8}$$

Figure 3.2 shows a histogram of 100 permutations $\Gamma(g)$ of the 45 class labels. The 100 values of $\Gamma(g)$ were divided into 40 bins, and the count of these bins are shown in the Figure. The observed value of $\Gamma(g)$ is 0.33087 and the blip on the right of the histogram. It is clear that the observed value is significantly higher than all the values in the histogram, so we reject the random label hypothesis. That is the evidence suggests a nonrandom assignment of class labels.

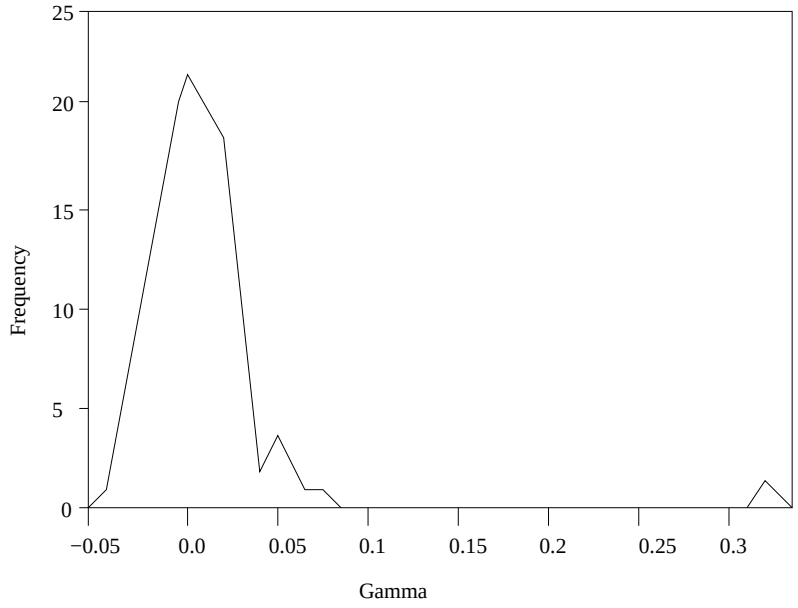


Figure 3.2: Histogram of Γ under the random hypothesis for the 80X data with class labels

3.2.6 Our Approach: Hypo-Test Stop Criterion

In previous sub-sections, we have discussed the basics of hypothesis testing and provided general information on how the test is set up. The test starts by defining a null-hypothesis. Therefore, we define the null-hypothesis of the *hypo-test* stop criterion as:

H_0 : There is no further cluster structure.

The *hypo-test* stop criterion is a random position hypothesis based on the internal criterion of cluster validity. Its baseline population is m random databases. These databases are sampled from a population of h-dimensional hypercube space (h is the number of the attributes in the database). We resort to sampling, because we don't have an analytical expression of the density function of the statistic under H_0. A random database $D_{r(i)}$ ($i \in \{1, 2, \ldots, m\}$) is created by using Definition 1.

Definition 1 (Random Database). *A random database is a database that created from the dataset D by randomly swapping its attribute values. The attribute values of an instance are randomly swapped with attribute values of different instances of the dataset. The swapping process starts from the first instance and continues to the last instance. This process is repeated for f times (user defined constant).*

After determining the null-hypothesis and the baseline population, an index is needed. The index we suggest here is the squared-error measurement S:

$$S = \sum_{k} \sum_{x_j \epsilon c_k} dist(x_j, c_k) \tag{3.9}$$

Where k represents a numerical class label of one of the two clusters (c_k, $k \in \{1, 2\}$) resulted from splitting a dataset using the k-means algorithm, and $dist$ is a distance measurement function (e.g. Euclidean distance), which measures the distance between an instance and the centroid of the cluster.

So as to answer the question: "Should H_0 be rejected?' the index is used to check whether the k-means algorithm can recover many clustering structures from the random databases that are believed to be better than the structure recovered from the original database (D).

If we let $S(D_{r(i)})$ denotes the squared-error of the random database $D_{r(i)}$, $S(D)$ denotes the squared-error of the original database D, and $comp(S(D_{r(i)}), S(D))$ is a function that returns 1 if $S(D_{r(i)}) > S(D)$, otherwise 0. Then the rule which answers the above question can be shown as follows:

$$If s < \alpha, \; reject \; H_0 \tag{3.10}$$

Where: s is the percentage of the number of random databases that show better square-error than the dataset.

The *hypo-test* stop criterion can be depicted by the pseudo-code shown below.

The algorithm: *hypo-test* stop criterion.
Input: dataset D, α, $S(D)$, m, w.
Output: Return true of H_0 is kept, otherwise false.
Method: hypo-test

```
int ns = 0.
For(i = 1 to m)
     RD = Randomly_swap_attributes(D, w).
     Implement_kmeans(RD), returns 2 clusters C1, and C2.
     S = Implement_index_on_2clustes(C1, C2).
     If (S < S(D)) Increase ns by 1.
```

if ($\frac{ns}{m} \geq \alpha$)

```
   Return(true).
else
   Return(false).
```

The input to the *hypo-test* stop criterion algorithm is composed of the data set D which has n instances, the accuracy level α, the number of random databases m, and the square-error of the original dataset $S(D)$, the weight w. The algorithm returns true if H_0 is kept, otherwise false. When the algorithm returns true the splitting process of a cluster is halted.

The algorithm, after receiving the input initializes the variable ns, which keeps the number of structures recovered from the random databases and believed to be better than the structure recovered from D. The algorithm loops m times and in each loop:

1. Runs the Randomly_swap_attributes method. This method takes two inputs D and w. The algorithm creates a random database from D using Definition 1. The method returns the random database RD.
2. Runs the Implement_kmeans method. This method takes RD as input, runs the k-means algorithm in the data, and returns two clusters C_1 and C_2.
3. Runs the Implement_index_on_2clusters method, which takes two inputs C_1 and C_2. This method implements the proposed index on the two clusters and returns the square-error S.
4. Increases ns by 1 if the square-error of S is less than the square-error of D.

At the end, the algorithm returns true if $\frac{ns}{m} \geq \alpha$, otherwise false. This process compares the percentage of the number of structures recovered from the random databases that are believed to be unusually good than the structure of D to a user defined threshold. If the result is true the H_0 is kept and the splitting process of the dataset is halted, because the random databases believed to give better partitions than the dataset, otherwise the dataset is splitted.

3.3 Artilab k-folds Stop Criterion

This section presents the second stop criterion, the *artilab k-folds* (artificial labels k-folds) we propose for the k-divisive hierarchical algorithm to halt the division of a cluster while building a dendogram. As mentioned before, this stop criterion is based on the well known k-fold cross-validation technique.

At the beginning of this section the k-fold cross-validation technique is presented, and second the *artilab k-fold* stop criterion.

3.3.1 k-fold Cross-Validation

The k-fold cross-validation is a technique in cross-validation. And cross-validation can be defined as a supervised accuracy estimation technique, which requires a set of pre-classified training set. The training set is used to build the model, which produces accurate predictions to the value of one attribute, given the value of the remaining attributes.

In k-fold cross-validation, a classifier can be defined as a function that maps an unlabeled instance to a labeled instance using internal data structures, and it can be built from a given dataset.

Let:

- V = the space of unlabeled multidimensional vector instances.
- Y = the set of possible labels.
- $X = V \times Y$ be the space of labeled instances.
- $D = \{x_1, x_2, \ldots, x_n\}$ a dataset of n labeled multidimensional vector instances, where $x_i = (v_i \, \epsilon \, V, y_i \epsilon Y)$.

A classifier C maps an unlabeled instance $v \, \epsilon \, V$ to a label $y \, \epsilon \, Y$. The notation $C(D, v)$ denotes the label assigned to an unlabeled instance v by the classifier that built from the dataset D. We assume that there exits a distribution on the set of labeled instances and that our dataset consists of independently and identically distributed instances.

In k-fold cross-validation the data set D is divided into k mutually exclusive subsets (D_1, D_2, ..., D_k), approximately of equal size. The classifier that built from the given dataset is trained and tested k times, each time using (k - 1) data subset $D \setminus D_t$ (where $t \epsilon$ 1, 2, ..., k) for training the model and the omitted subset (D_t) as a testing set to estimate the accuracy of the model.

There are different accuracy estimation methods relevant for the k-fold cross-validation. For the sake of this project the two-way contingency table is presented below.

3.3.2 Accuracy Estimation by Two-way Contingency Table

A two-way contingency table is an accuracy estimation table used to show whether there exists an association between the row variables and the column variables.

If we assume that the dataset D has only two labels ($Y = \{0, 1\}$), $x_i = (v, y)$ where $v \epsilon D_t$ and $y \epsilon Y$, and $C(D \backslash D_t,\ v)$ is a classifier built from the training set $D \setminus D_t$ which predicts the label of v, the two-way contingency table can be show as:

	Predicted Classes	
Priori Classes	*Class* 0	*Class* 1
Class 0	a	b
Class 1	c	d

Table 3.1: A two way contingency table. Predicted Classes are the class labels assigned to the instances by the classifier, and Priori Classes are the original class labels of the instances.

where:

- $a = \sum_{v \epsilon D_t} \delta(C(D \backslash D_t,\ v),\ 0,\ y,\ 0)$
- $b = \sum_{v \epsilon D_t} \delta(C(D \backslash D_t,\ v),\ 0,\ y,\ 1)$
- $c = \sum_{v \epsilon D_t} \delta(C(D \backslash D_t,\ v),\ 1,\ y,\ 0)$
- $d = \sum_{v \epsilon D_t} \delta(C(D \backslash D_t,\ v),\ 1,\ y,\ 1)$

where:

$\delta(i,\ j,\ m,\ n) = 1$ if $i = j$ and $m = n$ otherwise 0, e.g. a represents the number of the instances predicted by C (shown as i) to be 0 (shown as j) while the prior class label of the instance (shown as m) is 0 (shown as n).

The estimated accuracy of the table can be shown as:

$$Acc = \frac{1}{n} \sum_{t=1}^{k} \max\{a + d, b + c\} \tag{3.11}$$

where n is the number of instances in the data set D.

The above formula calculates the number of the instances which their class labels are believed to be correctly predicted by the classifier and divides it by the number of instances in the dataset. As we know, there are two clusters in the testing set and the classifier creates two clusters too, but it is very difficult to determine "which is which?". For example, cluster number 0 from the testing set, can be number 0 or number 1 for the classifier. For that reason the *Acc* function takes the maximum of (a + d) and (b + c). The *Acc* function can be used to halt the division of a cluster as we see in the next sub-section..

3.3.3 Our Approach: Artilab k-folds Stop Criterion

The *k-folds* stop criterion generally works the same way as the k-folds cross validation. It begins by dividing the dataset into training and testing sets ($D \setminus D_t$, D_t). However the k-fold cross-validation technique is a supervised learning technique and the instances of the data set used by the k-divisive algorithm are not labeled. At this stage the stop criterion needs some class labels to be assigned to the instances. So as to solve this problem an assumption has been made about the dataset. And based on this assumption artificial class labels can be assigned to the instances.

The assumption made is that when the training and testing sets are sampled, if there is a structure in the data set, both the training and testing sets can catch this structure when they have been partitioned separately by the the k-means algorithm. This assumption can be depicted by the graphs shown below.

Figure 3.3, shows a dataset with a structure (two clusters). Each of the training and testing sets are divided by the k-means algorithm into two clusters. From the graph it is very obvious that the structure recovered from the training set some how is similar to that recovered by the testing set.

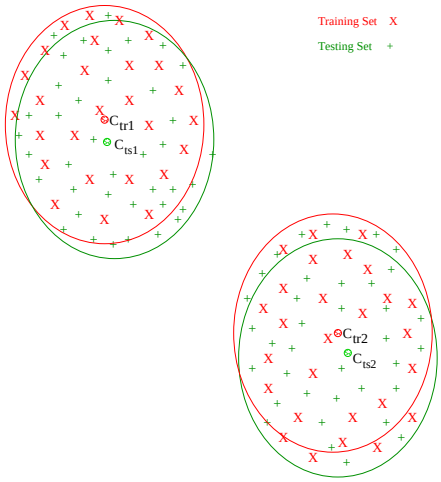


Figure 3.3: Sampling a dataset with a structure into training and testing set, and dividing each set into two clusters.

Figure 3.4, shows a dataset with no structure (only one cluster). As we see from the graph, there is high overlapping between the clusters formed by the training and testing sets. That means, the instances of a cluster from the testing set which is supposed to correspond to a cluster from the training set are assigned approximately equally to the two clusters of the testing set.

Based on the assumption made about the data set, the *artilab k-folds* stop criterion implements the k-means algorithm separately on the training and testing sets to form two partitions, each with two clusters. If there is a structure in the dataset, these class labels are assumed to be the same as the class labels of this structure.

The *artilab k-folds* stop criterion builds a classifier for learning, which predicts the class labels of the instances in the test dataset. So as to measure the accuracy of the classifier, the algorithm builds a contingency table which shows the correlation between the class labels assigned to the testing set by applying k-means and the ones that predicted by the classifier.

The classifier predicts the class labels of the testing set by measuring the distance between the instances

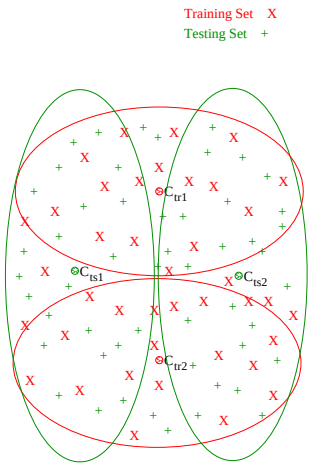


Figure 3.4: Sampling a dataset with no structure into training and testing sets, and dividing each set into two clusters.

and the centroids of the two clusters formed from the training set, and according to this distance it assigns a label to the instance.

To assess the accuracy of the classifier, the stop criterion builds a contingency table as shown in Section 3.3.2. In this table, the rows represent the labels of the testing set, and the columns represent the labels predicted by the classifier. The stop criterion uses this table to calculate the value of Acc (Equation 3.11). If the value of Acc is less than a user defined threshold, the division process of the dataset is halted.

The algorithm: *artilab k-folds* stop criterion.
Input: dataset D , number of folds k, number of instances in the data set n, and user defined threshold $user\_def$.
Output: True or false.
Method:

```
int num_predicted_instances = 0;
Divide the dataset D into k folds approximately with equal sizes.
For(i = 1 to k)
   testint_set  = D_i.
   training_set = D \ D_i.
   Run the k-means algorithm on training_set and testing_set.
   Build a classifier C from training_set to predict the labels of
     testint_set by using the centroids of the  training_set.
   Form the contingency table (a, b, c, d).
   num_predicted_instances = num_of_predicted_instances + max(a + d, b + c).
return( (num_predictions / n) < user_def).
```

The k-divisive hierarchical algorithm has been implemented and tested using the two proposed stop criteria in different databases. The implementation, evaluation, and tests are discussed in the next section.

Chapter 4

Implementation and Evaluation

This section is devoted to present the implementation, and experimental evaluations we have done in order to assess effectiveness of our k-divisive hierarchical clustering algorithm in artificial databases as well as in real world databases.

Experiments were run on relatively small samples and low dimensional databases. From a data mining viewpoint this may not be interesting. In reality, the opposite is true. Small sample sizes are the most challenging problems for methods which seek to extract structures from data. These databases may assist evaluating the methods on problems where the structure can easily be visualized. Finally due to time limitations the results are summarized and only preliminary evaluation has been done. The preliminary evaluation of the clusters is done by using their centroids (the averages of the attribute values) and the number of instances in the clusters.

In the rest of this section, first we describe the implementation, and second the evaluation of the algorithm with the two stop criteria using artificial and real databases.

4.1 Implementation

In order to be able to test and evaluate the proposed k-divisive hierarchical clustering algorithm, we have designed two graphical user interface programs using Java JDK 1.3 programming language to implement the k-divisive algorithm. One program is intended to implement the algorithm with *hypo-test* stop criterion and the other one with the *artilab k-folds* stop criterion. While running the programs some input must be pre-specified by the user, e.g. input file name. In order to get good results suitable values must be fixed for some of these inputs which may need some kind of expertise.

Figure 4.1 shows the graphical user interface program for the *hypo-test* stop criterion. The first input to the program is the "Input Filename". The input data file must be a tab-delimited text file. The first row in the input file contains the number of instances to be clustered and the number of attributes. The rest of the file contains the instances to be clustered. The "Stop Accuracy" input is a user defined threshold or accuracy level (a value between 0 and 1). The last input is the number of folds. The "Run" button is responsible for creating the dendogram which is built by extending the "JTree" in Java "Swing" component. Another input is the number of random databases. The program also shows as an output the starting and ending time of the clustering. One important aspect of using the "JTree" to represent the dendogram is that every node in the tree can be expanded or collapsed to minimize the tree. Another aspect is, when clicking in any node or cluster, the centroid of the cluster appears with the estimated accuracy of the stop criterion and the accuracy level α. The program can be closed by pressing on the exit button.

Figure 4.2 shows the graphical user interface program for the *artilab k-folds* stop criterion. The general idea and the design of this program are similar to that one of the *hypo-test* stop criterion mentioned above, just this one implements a different algorithm. In this program instead of the number of random databases, the number of the folds is needed as input. The input file is the same as the previous one.

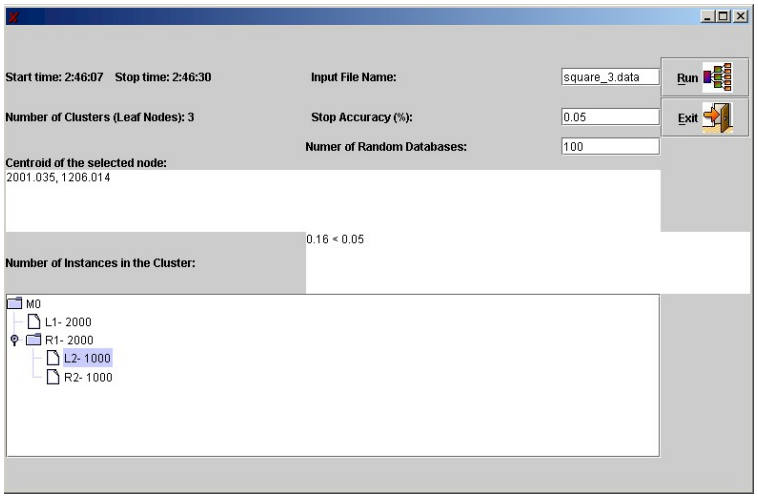


Figure 4.1: The user interface program for the *hypo-test* stop criterion

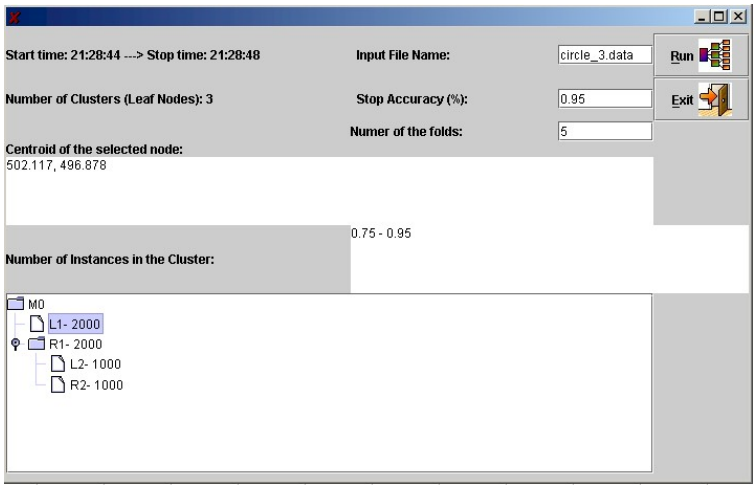


Figure 4.2: The user interface program for the *artilab k-folds* stop criterion

4.2 Evaluation in Artificial Databases

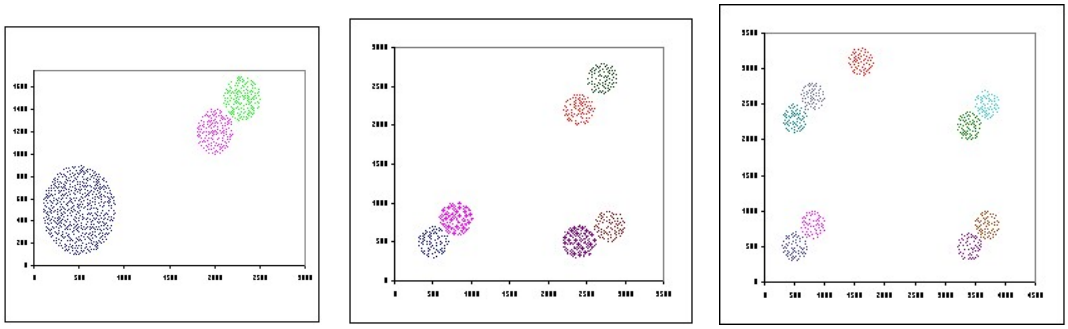
In real world problems, data often can not be arranged in tidy clusters. Most of the time algorithms decline to join clusters which we believe should have been grouped together, or the algorithms will join clusters which we believe should have been split. This because of the different requirements of the clustering algorithms (e.g. stop criteria, cluster indices). Therefore, so as to assess the quality of our algorithm we decided to start testing with 2-dimensional artificial data sets, which we can randomly disperse points independently, to determine how well the algorithm performed. It is beneficial to start the experiments with artificial data sets to estimate the confidence in a layer which we know its structure. First we discuss the 2-dimensional data which we created, then we move to the waveform artificial data [6].

4.2.1 2-Dimensional Databases

The 2-dimensional datasets we created contain clusters of two types, one type created as circles and the other one as squares. These two types are discussed below.

- **Artificial Circle Clusters**

  For every cluster, given a number of random data points n, circle center (x, y), and circle diameter d, we generated three groups of data sets for testing. The first group has 3 clusters in which, one cluster has 2000 points and the other two has 1000 points each. The second group has 6 cluster and the third one has 9. Each cluster has 1000 points. This group of artificial data can be depicted by Figure 4.3.

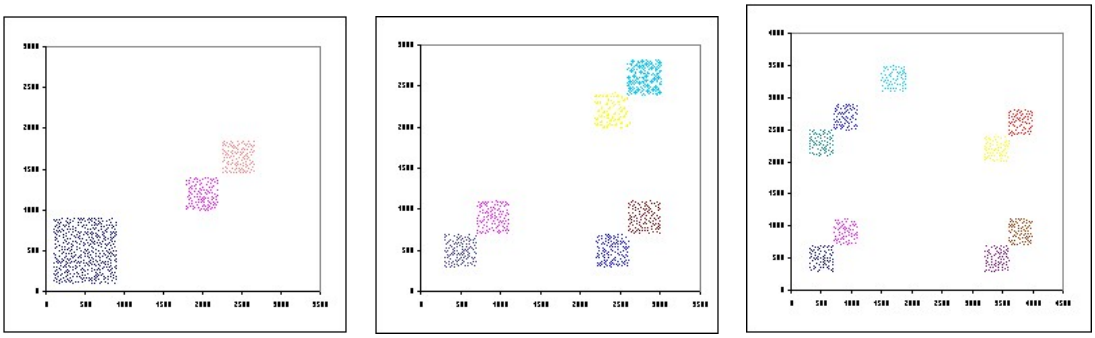


(a) 3 Circle Clusters. (b) 6 Circle Clusters. (c) 9 Circle Clusters.

Figure 4.3: Artificial Circle Clusters.

- **Artificial Square Clusters**

  For every cluster, given a number of random data points n, square top-left corner (x_1, y_1), and square bottom-right corner (x_2, y_2), we have generated data sets with the same number of groups, number of clusters and random data points as in the circle clusters, just with squares surrounding the clusters. This group of artificial data is shown in Figure 4.4.



(a) 3 Square Clusters (b) 6 Square Clusters (c) 9 Square Clusters

Figure 4.4: Artificial Square Clusters

Hypo-Test Stop Criterion

The k-divisive hierarchical clustering algorithm with *hypo-test* stop criterion has run many times on all sets of the artificial data. The algorithm has successfully captured the structures of the artificial data most of

the time. Great success in capturing the structures was registered when the accuracy level was 0.05 and the number of random databases was 1000.

Only 2 dendograms are shown here as examples, one formed by using the data set of the three circle clusters, and the other one formed by using the data set of the three square clusters. The discussion and the interpretation of the other dendograms can be based on these examples.

Figure 4.5 shows the dendogram we attained by running the k-divisive hierarchical clustering algorithm with *hypo-test* stop criterion on the artificial data sets. The measured accuracy used to reject the null-hypothesis together with the user defined threshold are shown under each cluster. The first split partitions the entire data of 4000 points into two clusters, the first one is the cluster that contains 2000 instances, and the second one contains the two clusters with 1000 instances each. This split has an accuracy of 0.0 which means perfect split of the cluster (the null-hypothesis is totally rejected). In contrast, the split of the inner cluster of 2000 instances (left most grandchild of the root) yielded a measured accuracy of 0.62 (the null-hypothesis is kept), which resulted in halting the split. In the other hand, the other inner cluster (right most grandchild of the root) has an accuracy of 0.0, which resulted into two sub clusters where both of them kept the null-hypothesis with accuracy of 0.55 and 0.52 respectively.

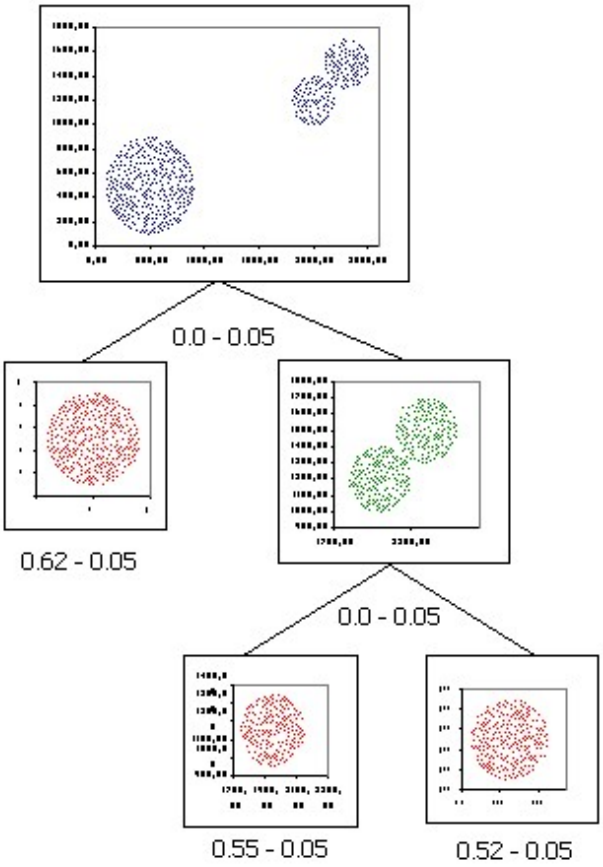


Figure 4.5: A dendogram built by the *hypo-test* stop criteria for three circle clusters. The red nodes show the leaf nodes

Figure 4.6 shows the dendogram we attained by running the algorithm with the *hypo-test* stop criterion on the artificial data set that has three square clusters. The interpretation of this dendogram is similar to the interpretation of the dendogram of the three circle cluster mentioned above.

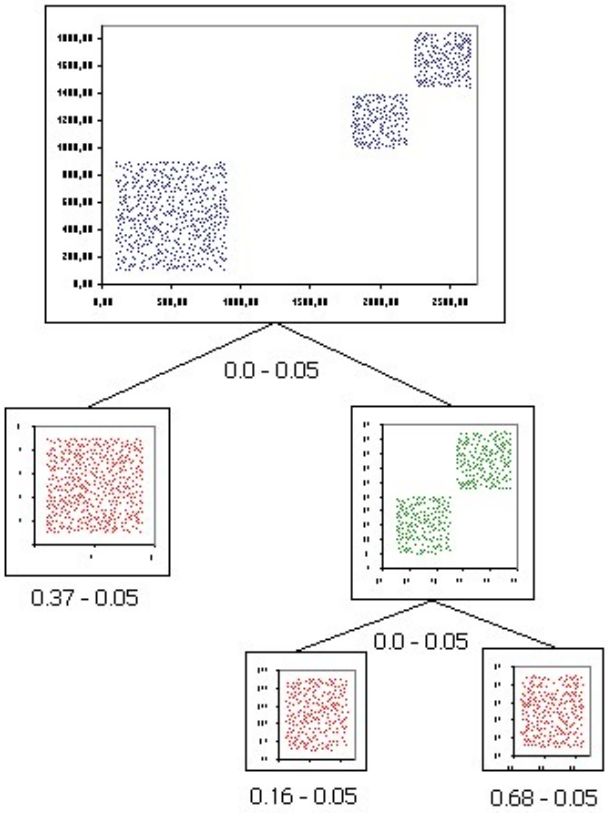


Figure 4.6: A dendogram built by the *hypo-test* stop criteria for the three square clusters. The red nodes show the leaf nodes

Artilab k-folds Stop Criterion

The k-divisive algorithm with *artilab k-folds* stop criterion has run the same way as the *hypo-test* stop criterion many times on the artificial data sets. Most of the time the structures were perfectly detected when the user defined threshold was 0.95 and the number of folds was 5.

As with the *hypo-test* stop criteria only 2 dendograms are shown.

Figure 4.7 shows the dendogram created by using the data set of the three circle clusters. The best results were attained when the accuracy level was 0.95 and the number of folds was 5. The division of the cluster on the root node resulted into two sub-clusters, one contains the cluster with 2000 instances and the other one contains the two clusters with 1000 instances each. This split has an accuracy of 1.0 which resulted in a perfect division of the cluster. This shows a complete agreement between the classifier's prediction to the labels of the instances in the testing set and the ones that assigned by the k-means algorithm. The cluster on the left most grandchild of the root yielded an accuracy of 0.75 which is below the threshold. This resulted in stopping the division of the cluster. The cluster on the right most grandchild of the root is perfectly divided into two sub-clusters with an accuracy of 1.

Figure 4.8 shows the dendogram we attained by running the k-divisive hierarchical algorithm with *artilab k-folds* stop criterion on the artificial data set that has three square clusters. The interpretation of this dendogram is similar to the interpretation of the dendogram of the three circle clusters mentioned above.

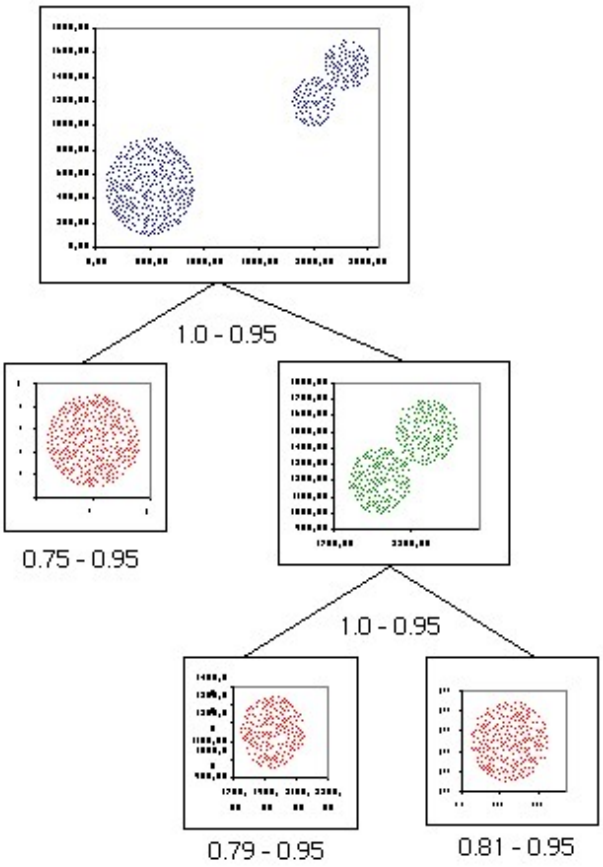


Figure 4.7: A dendogram built by *artilab k-folds* stop criterion for 3 circle clusters. The red nodes show the leaf nodes. The inner node cluster are shown with different colors.

4.2.2 Waveform Database

The waveform database is an artificial database created by C language [6] and it contains 5000 instances. Each instance has 21 attributes with continuous values between 0 and 6. Each instance is generated with added noise in each attribute. Each attribute is associated with one of a three class labels, these class labels are generated from a combination of 2 out of 3 "base" waves, e.g. $c_1 = w_1 + w_3$. Each class contains 33% of the instances. Figure 4.9 shows three clusters formed from these waves represented by their centroids.

Experimental evaluation has been done in this database by running both stop criterion. As it can be seen from the dendograms in Figure 4.10 and 4.11, the two stop criteria have captured the same structure, even though the *artilab k-folds* formed 4 clusters while the *hypo-test* formed 6 clusters. The *hypo-test* at the beginning formed the same dendogram as that formed by the *artilab k-folds* (the clusters have the same number of instances in the two dendograms) and later on divided more two clusters which resulted in 6 clusters.

By investigating the two graphs we can see that the centroids of the original waveforms and most of the centroids of the clusters we got are symmetric with peaks. As we mentioned before the class labels are generated by a combination of 2 waves out of 3. It is very obvious that the class labels are generated by a combination of 50% from the first wave and 50% from the second wave, that is why the centroids are symmetric. If it happens that the class labels are generated, for example by a combination of 30% from the first wave and 70% from the second wave, it could be able for our proposed stop criteria to catch the structure of the dataset.

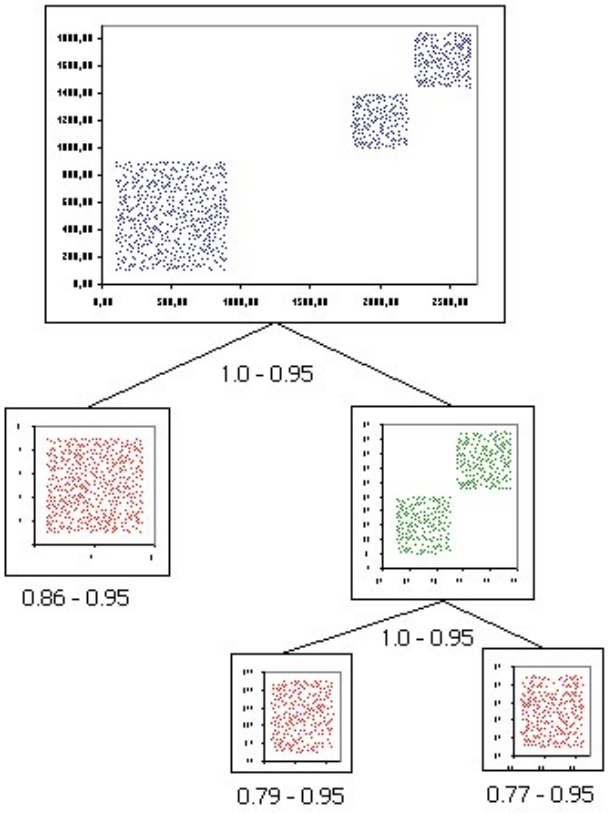


Figure 4.8: A dendogram built by *artilab k-folds* stop criterion for 3 square clusters. The red nodes show the leaf nodes

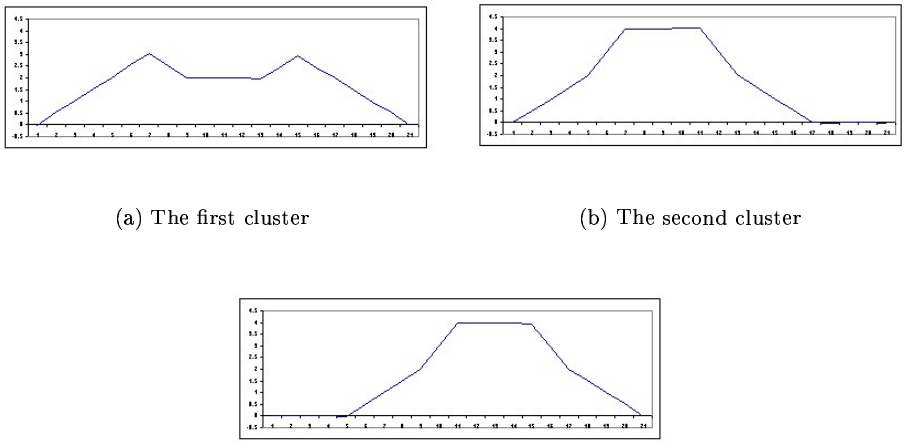
4.3 Evaluation in Real Data: Leukemia Database

Since the development of microarray technology in 1995, there has been an enormous increase in gene expression data. Thinking genes as a network of nodes that influence each other's expression levels, scientists want to reconstruct the precise interaction network from the expression data obtained with large scale arraying. Many techniques have been developed for quantitative monitoring of gene expression patterns with a complementary DNA, e.g. clustering [9].

The output of a DNA microarray data is presented generally in the form of a matrix with many rows as instances (e.g. tissues), and many attributes as genes in the experiment or vice versa. Hierarchical clustering techniques are the most widely used methods for the analysis of patterns of gene expression. It produces a representation of the data with the shape of a binary tree, in which the most similar patterns are clusters in a hierarchy of nested subsets, and it can help to identify clusters of samples sharing the same gene expression profile and/or clusters of genes sharing the same expression profile across samples.

The leukemia database [32] has been selected for the evaluation, with the aim of identifying clusters of genes with similar expressions.

This database has become as a standard test bed for gene expression data analysis techniques [1] [21]. It contains 72 instances from leukemic patients, with each instance being characterized by the expression level of 7129 genes. In addition, each instance is labeled with either acute lymphoblastic leukemia (ALL) or acute myeloid leukemia (AML) meaning the specific type of acute leukemic the patient suffers from. The database is divided into two discretized sets, the first one contains 47 instances out of the 72 labeled as ALL, and the



(a) The first cluster

(b) The second cluster

(c) The third cluster.

Figure 4.9: The three clusters of the waveforms database represented by centroids.

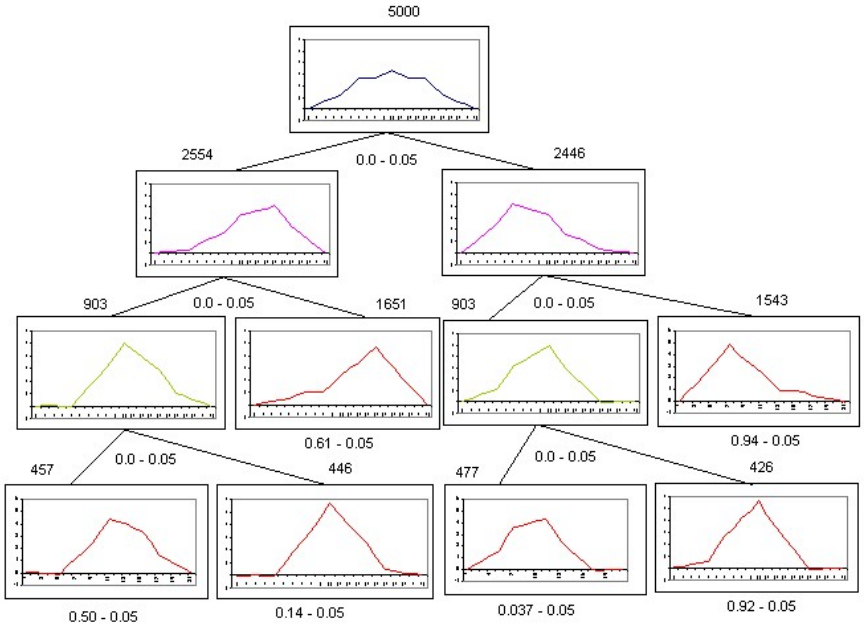


Figure 4.10: A dendogram built by hypo-test stop criterion on waveform database.

second one contains the rest 25 instances as AML. The discretization of the database was done following the most common approach in the literature, the gene expression levels were discretized into three states. This discretization distinguishes the gene being either under-expressed, baseline, or over-expressed [21]. The interpretation of the results in this section is based on this discretization.

The ALL and AML databases were transposed, such that 7219 genes were the instances and the measurements for the corresponding 47 and 25 patients were the predictive attributes. These two data sets are

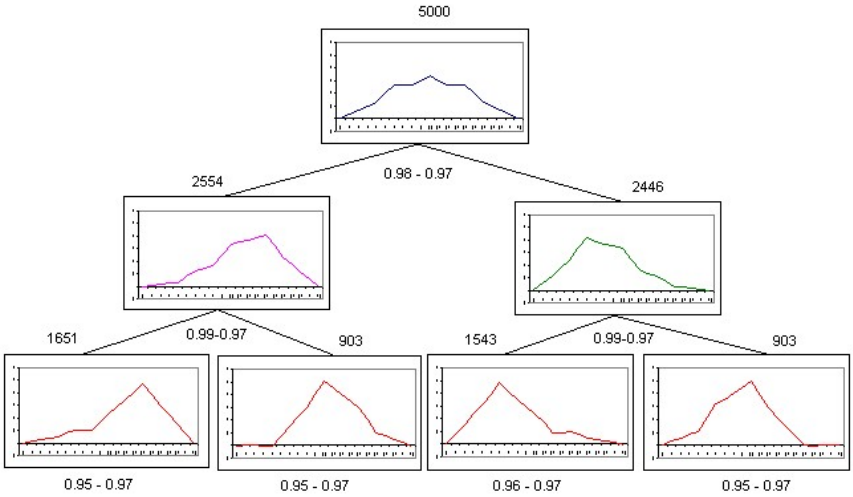


Figure 4.11: A dendogram built by artilab k-folds stop criterion on waveform database.

indicated as D_{ALL} and D_{AML} in the forthcoming, where the patients are indicated by the subscription. Finally, the instances in D_{ALL} and D_{AML} are treated as independent and identically distributed, although some genes may be co-regulated and therefore some cases may be correlated. This implies that the analysis may not change the essence of the results. In fact, this approach is taken in many other gene expression data analysis applications.

Experimental evaluations have been done for the D_{AML} and D_{ALL} databases using both proposed stop criteria of the k-divisive hierarchical algorithm. First the experiments of the D_{AML} database is presented and next the experiments of the D_{ALL} database.

Preliminary experiments for the *artilab k-folds* with different number of folds ($fold$) and different values of accuracy level (acc) show that $fold = 5$ is well suited for the D_{AML} and D_{ALL}, $acc = 0.95$ is well suited for the D_{AML}, and $acc = 0.92$ is well suited for the D_{ALL}.

It is worth to note that the clusters on the dendograms in this section are shown with different colors. The red color is used to show clusters on the leaf nodes, and the other colors to show the level of the dendogram except the red (every level has different color). The red clusters are numbered and the number of each cluster is shown under the left-bottom corner of the cluster. The clusters are represented by their centroids which are shown in the dendograms.

Figure 4.12 and 4.13 show dendograms built on D_{AML} by the *hypot-test* and *artilab k-folds* stop criterion respectively. The two dendgorams show great agreement on the structure found in the D_{AML}, even the number of the instances in all clusters are approximately the same. The two stop criteria start by splitting the cluster in the root node into two sub-clusters, a red cluster named as cluster 1 and a green cluster. Genes in cluster 1 have the tendency towards the under-expressed and the genes in the green cluster have the tendency towards baseline and/or over-expressed. The green cluster has been divided again into two sub-clusters, one with red color (cluster 2) and the other one with the pink color. The genes in cluster 2 are very diverse, and the interpretation of this cluster may be difficult. But it may happen that there are some outliers or some noise in the data, which resulted in forming this cluster. The pink cluster has the tendency again to be baseline and/or over-expressed. Lastly the pink cluster has been divided into two red clusters (cluster 3 and 4). The genes in cluster 3 has the tendency towards to be over-expressed while the genes in cluster 4 has the tendency towards to be baseline.

Figure 4.14 and 4.15 show dendograms built on D_{ALL} by the *hypot-test* and *artilab k-folds* stop criterion respectively. The two dendgorams show great agreement on the structure found in the D_{ALL}. It is very obvious from the dendograms that the genes in D_{ALL} has a tendency towards be either, under-expressed, baseline, or over-expressed. The interpretation of these dendgorams can be done the same way as done

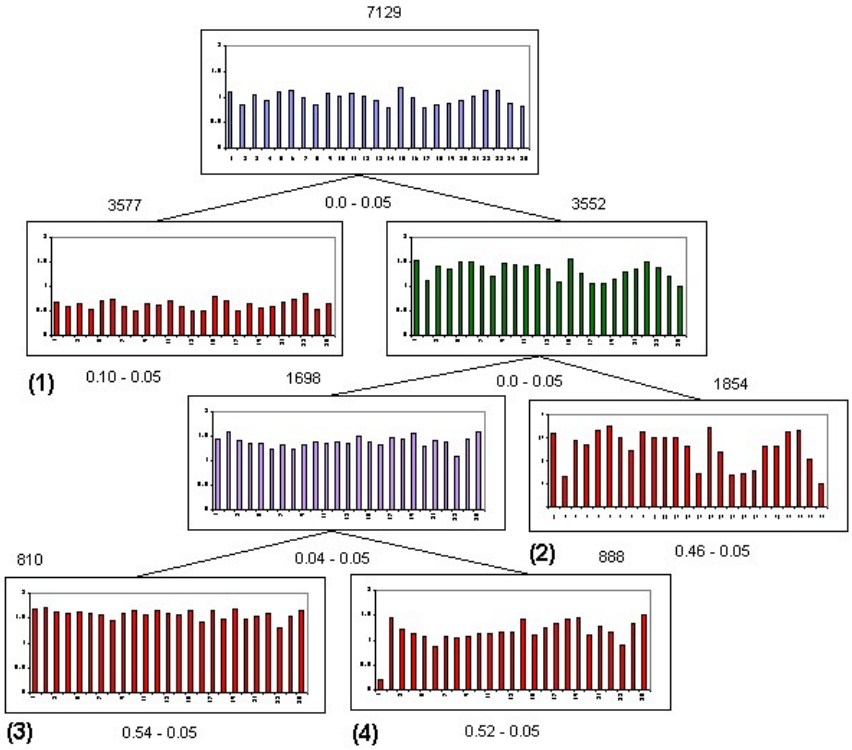


Figure 4.12: A dendogram built by hypo-test stop criterion for on D_{AML}.

with the D_{AML}. The only think we notice here is that, both cluster 3 and 4 have the tendency towards be over-expressed.

As a conclusion, the results of these tests make sense to believe that the generative mechanisms underline the D_{AML} and D_{ALL} consists of three physical processes. Each physical process tends towards genes being either under-expressed, baseline, or over-expressed for all patients. This can be convincing if we take into account that all patients in D_{AML} and D_{ALL} are suffering from the same type of acute leukemia.

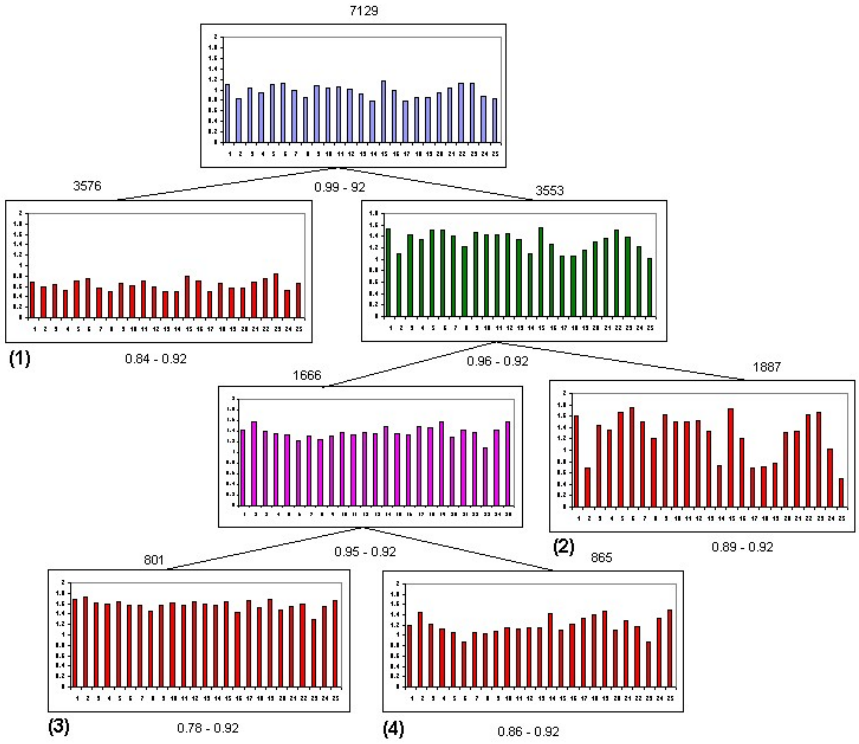


Figure 4.13: A dendogram built by artilab k folds stop criterion on the D_{AML}.

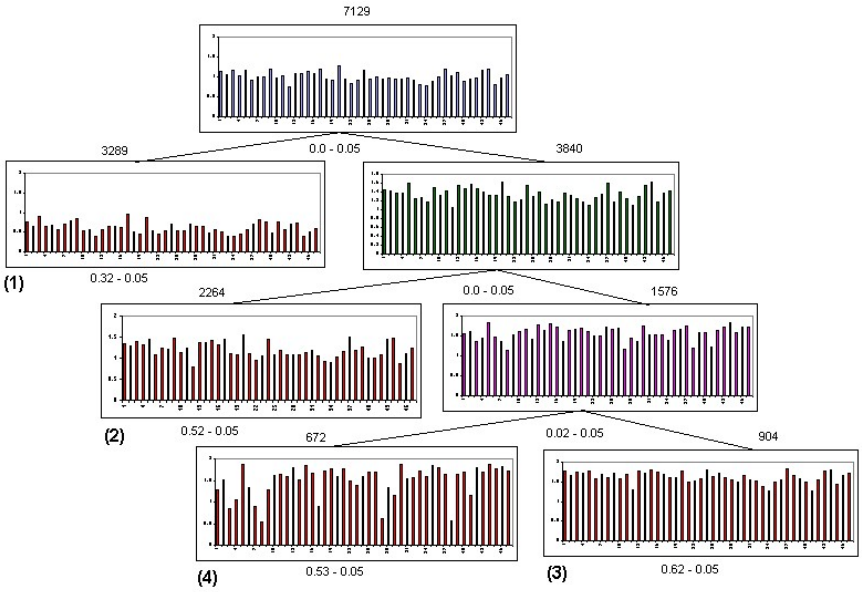


Figure 4.14: A dendogram built by hypo-test stop criterion on the D_{ALL}.

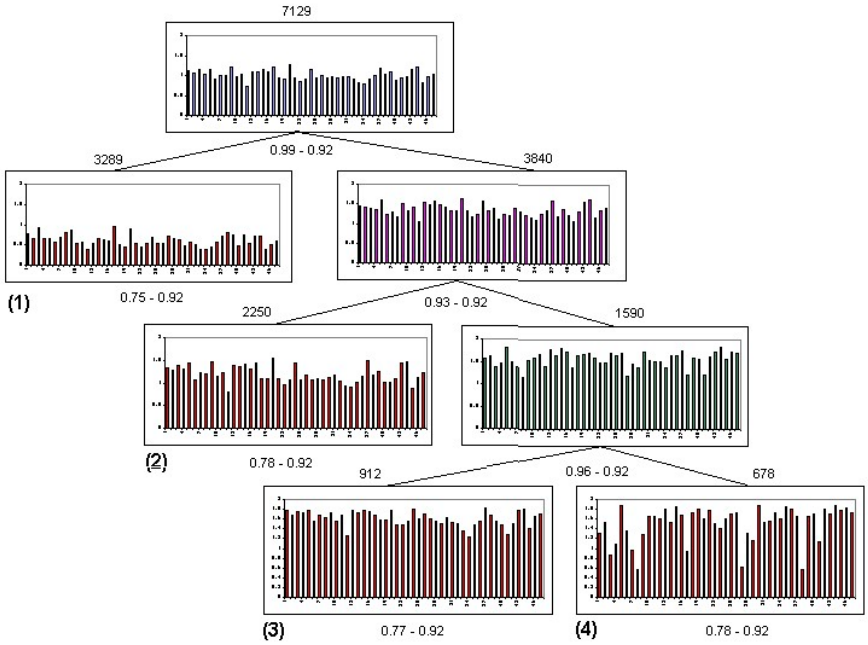


Figure 4.15: A dendogram built by artilab k-folds stop criterion on the D_{ALL}.

Chapter 5

Conclusion and Future Work

Data mining is a new discipline lying at the interface of statistics, database technology, pattern recognition, and machine learning. It can be defined as the process of discovering hidden patterns in a given database by applying some techniques. Clustering is one of the unsupervised learning techniques used in data mining to discover hidden patterns. Generally, clustering techniques can be divided into many types, one of them is the hierarchical clustering. In hierarchical clustering, the divisive hierarchical is the most challenging problem.

The objective of this project was to develop a new divisive hierarchical clustering algorithm which stops building meaningless dendogram. In order to satisfy these objectives, the k-divisive hierarchical algorithm was proposed. This algorithm uses the well-known k-means algorithm as a splitter to divide a cluster into two. Two stopping criteria were suggested to stop building meaningless dendograms. The first one is the *hypo-test* stop criterion, which is based on the hypothesis test and the second one is the *artilab k-folds* stop criterion, which is based on the k-fold cross validation.

The k-divisive algorithm has been implemented together with the two stop criterion on Java $JDK1.3$ platform. The algorithm has been tested into two types of databases, artificial databases and real world databases. Experimental evaluation has shown great agreement between the *hypo-test* and *artilab k-folds* stop criteria in discovering similar structures on the artificial databases as well as in the real world databases.

The DNA microarray database known as leukemia database was used as a real world database to test the algorithm. After testing the algorithm on this data with the two stop criteria, both of the stop criteria supported the claim that the genes can be classified as under-expressed, baseline, or over-expressed.

Regarding the running time, the *artilab k-folds* over-performed the *hypo-test*. While the first one takes only few seconds to complete running, the second one takes more than 24 hours in a database with thousands of instances.

One of the big problems in the two stop criteria is how to fix thresholds in the indices used, so as to determine how big is "big" or how small is "small". Fixing up these values may need some kind of expertise, but still it can be difficult to determine. Consequently, the program needs to be run many times and compare the results so as to select the most suitable one. In the future we plan to develop a new technique to run the program with different indices values so as to get different results, and validate the most suitable one automatically.

Chapter 6

Acknowledgement

"If I have seen further it is by standing on the shoulders of giants."
Sir Isaac Newton, cited in The Oxford Dictionary of Quotations.

I would like to thank my supervisor Jose M. Peña for his nice suggestions and unlimited helps and regards.

Bibliography

[1] Ben-dor, Friedman and Z. Yakhini *Class Discovery in Gene Expression Data.* (In Proceedings of the Fifth Annual International Conference of Computationl Molecular Biology, 2001).

[2] Jain and R. C. Dubes., *Algorithms for Clustering Data.* (Prentice Hall, 1988).

[3] Ben A., Hitt *Unsupervised Clustering for Data Prospecting and Data Mining* (http://www.tdan.com/i008hy01.htm)

[4] Bjornar Larsen and Chinatsu Aone, *Fast and Effective Text Mining Using Linear-time Document Clustering.* (Int'l Conference on Knowledge Discovery and Data Mining, 1999).

[5] B. King., *Step-wise clustering procedures.* (Journal of the American Statistical Association, 1967).

[6] Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J.. *Classification and Regression Tress.* (Wadsworth International Group: Belmont, California, 1988).

[7] Brian S. Everitt. *Third Edition Cluster Analysis.* (Oxford University Press Inc., 1993).

[8] Aggarwal, Stephen C. Gates, and Phili S. Yu., *On the Merits of Building Categorization Systems by Supervised Clustering* (In Proc. of the Fifth ACM SIGKDD Int'l Conference on Knowledge Discovery, 1999).

[9] Cheng Y. and Church. *BiClustering of Exression Data.* (http://ismb00.sdsc.edu/tutorials/kohane.html).

[10] Cutting, J.O. Pederson, D. R. Karge, and J. W. Tukey, *A Cluser-Based Approach to Browsing Large Document Collections.,* (In Proceedings of the ACM SIGIR, 1999).

[11] Efron, B. and Tibshirani. *Improvements on cross-validation.* (J. of the American Statictical Association, 1997).

[12] Feng Zheng. *kmeansBased Fuzzy Classifier.* (http://www.isip.msstate.edu/publications/courses/ece_8443/papers /2001/kmean/p02_paper_v0.pdf).

[13] Forgy, E., *Cluster Analysis of Multivariate Data: Efficiency versus Interpretability of Classification.* (Biometrics, 1965).

[14] Gose, E., R. Johnsonbaugh, S. Jost, *Pattern Recodnition & Image Analysis.* (Prentice-Hall, 1996).

[15] Hjorth, J.S.U. *Computer Intensive Statistical Methods Validation, Model Selection, and Bootsrap.* (London, Chapman & Hall, 1994)

[16] In Le Cam, L. M. and Neyman, J. *Some Methods for Classification and Analysis for Multivariate Observations.* (Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. Volume I, 1967).

[17] Berry and Gordon Linoff. *Data Mining Techniques for Marketing, Sales and Customer Support.* (John Willey & Sons, Inc, 1996).

[18] Jain, A. and Dubes. *Algorithms for Clustering Data* (Prentice Hall, 1988).

[19] Jain, A.K., M.N. Murty, P.J. Flynn, *Data Clustering: a Review.* (ACM Computing Surveys, 1999).

[20] Dunn J. C. *Well separated clusters and optimal fuzzy partitions.* (J. Cybern, 1974).

[21] Jose. M. Peña, J. A. Lozano and P. Larrañnaga. *Unsupervised Learning of Bayesian Networks Via Estimation of Distribution Algorithms: An Application to Gene Expression Data Clustering.* (International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 2003).

[22] Kearns, M. *A bound on the error of cross-validation using the approximation and estimation rates, with consequences for the training-test split.* (Neural Compuation, 1997).

[23] MacNaughton-Smith, P. *Some Statistical and Other Numerical Techniques for Classfying Indiviudals.* (Home Office Research Unit Report No. 6. H.M.S.O., London, 1965).

[24] Michal Linial1, Nathan Linial2, Naftali Tishby and Golan Yona2.*http://citeseer.nj.nec.com/cache/papers/cs/601/http:zSzzSzwww.cs.huji.ac.ilzSzlabszSzlearningzSzPaperszSzproteins.pdf/linial97global.pdf.*.

[25] Steinbach, G. Karyipos, and V. Kumar, *A comparison of doument clustering techniques,* (Copenhagen: IN KDD Workshop on Text Mining, 2000).

[26] Padhraic Smyth. *Clustering Using Monte Carlo Validation.* (KDD, 1996).

[27] Pahdraic Smyth. (http://citeseer.nj.nec.com/cache/papers/cs/59/http:zSzzSzftp.ics.uci. eduzSzpubzSzsmythzSzpaperszSzkdd96.pdf/smyth96clustering.pdf).

[28] Sneath and R. R. Sokal, *Numerical Taxonomy* (London: Freeman, 1973).

[29] Pereira, F. Tishy, N. & Lee L. *Distributional Clustering of English Words.* (30th Annual Meeting of the Association fro Computational Linguistics, 1993)

[30] Plutowski, M., Sakata, S., and White, H. *Cross-validation Estimates IMSE.* (Morgan Kaufman, 1994)

[31] Rasa Juergelenaite, Addin O. Mohamed A., Nicolaj Sndberg-Madsan, Casper Thomsen. *Induterial Data Mining: A Data Mining Project in Cooperation with Green House.* (Aalborg University, 2003).

[32] Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Goller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield and E. S. Lander. *Molecular Classification of Cancer: Class Discovery and Class Prediction by Gene Expression Monitoring.* (Science **268** (199) 531-537)

[33] Selim, S.Z., M.A. Ismail, *K-means-type Algorithms: a Generalized Convergence Theorem and Characterization of Local Optimality.* (IEEE Trans. on Pattern Analysis and Machine Intelligence, 1984).

[34] Valerie J. Easton & John H. McColl. *Hypothesis Testing* (http://www.cas.lancs.ac.uk/glossary_v1.1/hyptest.html#h0)

[35] Vapnik, V.N. *Estimation of Dependences Based on Empirical Data* (Springer-Verlag, New-York, 1982)

[36] Weiss, S.M. and Kulikowski. *Computer Systems That Learn.* (Morgan Kaufman, 1991)

[37] Yannis Batistakis, and Michalis Vazirgiannis. *On Clustering Validation Techniques.* (Journal of Intelligent Information Systems, 2001).