NEW APPROACHES TO HIERARCHICAL MODELING — FRAMEWORKS, ALGORITHMS, AND APPLICATIONS

A Dissertation Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirement for the Degree

Doctor of Philosophy

By

Paul K. Amalaman

December 2015

NEW APPROACHES TO HIERARCHICAL MODELING — FRAMEWORKS, ALGORITHMS, AND APPLICATIONS

_____

Paul K. Amalaman

APPROVED:

_____

Dr. Christoph Eick, Dept. of Computer Science, UH

_____

Dr. Ricardo Vilalta, Dept. of Computer Science, UH

_____

Dr. Weidong Shi, Dept. of Computer Science, UH

_____

Dr. Shishir Shah, Dept. of Computer Science, UH

_____

Dr. Tim Cooper, Dept. of Biology, UH

_____

Dean, College of Natural Sciences and Mathematics

ACKNOWLEDGMENTS

My present status in life, this Ph.D. included, is the result of extraordinary efforts, and sacrifices, from my parents; themselves illiterate. To them, I direct many thanks!

I would like to thank my wife for providing valuable support to me and dedicated care to our children while I was pursuing my educational goals. My brothers and sisters deserve special thanks for the encouragements and supports they have always given me.

I would like to express special gratitude to my advisor, Dr. Christoph F. Eick, for all his guidance, encouragement, and patience throughout my Ph.D. I owe him the recognition of a great teacher. I would like to thank my committee members: Dr. Ricardo Vilalta, Dr. Shishir Shah, Dr. Tim Cooper, and Dr. Weidong (Larry) Shi, for taking the time to read this thesis and for their helpful comments. Special thanks goes to my lab team members at UH-DMML lab, with whom I shared great times and valuable relationships. Particular thanks to Dr. Nouhad Rizk, for her valuable advice. Finally, I would like extend special gratitude to Yvette Elder for her valuable administrative support and to the faculty and staff members of the Computer Science Department of the University of Houston for the guidance and assistance they provided me throughout my Ph.D.

NEW APPROACHES TO HIERARCHICAL MODELING — FRAMEWORKS, ALGORITHMS, AND APPLICATIONS

An Abstract of a Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirement for the Degree

Doctor of Philosophy

By

Paul K. Amalaman

December 2015

ABSTRACT

Obtaining hierarchical organizations of knowledge is important in many domains. To create such hierarchies, improved techniques for subdividing entities hierarchically according to similarities and differences are needed. New techniques for organizing documents in hierarchies, for automatic document retrieval and for hierarchical query clustering are being made available at a fast pace. In this work, we investigate new methods to induce hierarchical models with the goal of obtaining better predictive models, to facilitate the creation of background knowledge with respect to an underlining class distribution, to obtain hierarchical groupings of a set of objects based on background knowledge they share, to detect sub-classes within existing class distribution, and to provide methods to evaluate hierarchical groupings. The results of this effort has led to the development of (1) TPRTI, a new regression tree induction approach which uses turning points, candidates split points computed before the recursive process takes place, to recursively split the node datasets; (2) PATHFINDER, a new classification tree induction capable of inducing very short trees with high accuracies for the price of not classifying examples deemed difficult to classify; (3) AVALANCHE, a new hierarchical divisive clustering approach which takes as input a distance matrix and forms clusters maximizing inter-cluster distances; (4) STAXAC, a new agglomerative clustering approach which creates *supervised taxonomies* that unlike traditional agglomerative clustering, which only uses proximity as the single criterion for merging, uses both proximity and class labels information to obtain hierarchical groupings of a set of objects. We applied the techniques we developed, (1) to molecular phylogenetic-based taxonomy generation and found that this new approach and the obtained supervised taxonomies can help biologists better charac-

terize organisms according to some characteristics of interest such as diseases, or growth rate; (2) to data editing; we were able to enhance the accuracy of the k-nearest neighbor classifier by removing minority class examples from clusters that were extracted from a supervised taxonomy; (3) to meta learning; we developed new algorithms that operate on supervised taxonomies and compute both the distribution of the classes within a dataset, and the difficulty of classifying examples belonging to a particular dataset.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The accessibility and increasing abundance of data today makes knowledge discovery in databases a matter of increasing necessity and considerable importance. Knowledge discovery in databases is the process of discovering useful knowledge from large collections of data. A wide variety of methods has been proposed, and many more are being made available. Algorithms that organize datasets hierarchically and methods for analyzing, evaluating, and comparing the obtained hierarchies are becoming quite popular. This dissertation is concerned with both problems: (1) given a collection of data objects, how to efficiently organize the objects in hierarchical sets with respect to some predefined objective function, and (2) given a hierarchical dataset, how to efficiently extract interesting knowledge from it. With respect to organizing the objects hierarchically, the grouping may be done in two ways: one may group the objects in relation to their relatedness to a given target class or one may use the objects pairwise relatedness as a criterion for grouping. For example, documents may be organized hierarchically based on their relatedness to a particular topic — target class — to facilitate their retrieval. The aim is to form hierarchical groups based on the degree to which each document relates to the topic. In the second scenario, a target class is not given but instead a relationship among the objects is provided; for example similarity between the objects. In this case one must use that knowledge to organize the objects in hierarchical sets with the most closely related sets merged first, and the least related sets merged last. For example, a biologist may form

hierarchical groups of species based on pairwise similarities among them. More advanced rules can be used to generate hierarchical clusters such as a combination of proximity of the objects and their class labels. For example, we may form hierarchical groups based not only on proximity of gene sequences but also on the species phenotypes, or their particular properties, such as being radiation resistant, high temperature resistant, etc. The assumption in using an advanced criterion for grouping is that if the criterion is chosen carefully, some hard-to-reach background knowledge about the dataset can be revealed. To the best of our knowledge, approaches that use complex criteria to obtain hierarchical groupings have not yet been explored in the literature.

The goal of this research is to investigate new methods to organize objects hierarchically with the aim of:

1. Developing better predictive models capable of assigning a label to previously unseen examples (classification tree) or a real value to a target attribute (regression tree) with improved accuracy rate.
2. Obtaining hierarchical groupings of a set of objects based on background knowledge they share.
3. Facilitating the creation of background knowledge with respect to an underlining class distribution by detecting sub-classes within existing class distribution, by assessing the difficulty of classifying examples belonging to a particular dataset and by providing methods to evaluate hierarchical organizations.

In particular, the specific contributions of this research include:

1. The development of *TPRTI* [1], a new regression tree approach that relies on using turning points to split an input dataset. It uses sliding window technique to compute turning points (split-point candidates) before the recursive process takes place.

2. The development of *PATHFINDER* [5], a new classification tree-induction approach that uses a grid-based clustering approach to first group the examples in high purity clusters then defines a top down method to identify such clusters. Classification trees induced by this approach are capable of rejecting examples that are difficult to classify—these are examples that do not belong to any of the identified clusters.

3. The introduction of *Supervised Taxonomy* [4], a new approach for taxonomy generation that uses both proximity information and background information in the form of class labels to create a hierarchy for a given set of objects.

4. The development of *STAXAC* [4], a supervised taxonomy generating algorithm that uses a bottom up approach to develop the hierarchical tree (maximizing purity at each step of the merging process).

5. The development of *HC-edit* [3], a data-editing approach for the k-NN classifier which improves the accuracy of the k-NN classifier by first generating supervised taxonomies from the input dataset then uses information from the tree to edit the dataset.

6. The development of *AVALANCHE* [2], a new divisive hierarchical clustering that uses a distance matrix dataset as its input. To the best of our knowledge, AVALANCHE is the first hierarchical divisive algorithm of this kind.

7. The development of *EUREKA*[4], a series of algorithms that operate on supervised taxonomies and create background knowledge about a dataset, such as characterizing the

distribution of the classes within a dataset, addressing the question if particular classes in the dataset are uni- or multi-modal, and assessing the difficulty of classifying the examples belonging to a particular dataset.

The dissertation is organized as follows. Chapter 2 discusses the *TPRTI* approach to Linear Regression Tree construction. Chapter 3 introduces the *PATHFINDER* approach to Classification Tree induction and chapter 4 discusses *AVALANCHE*, a new divisive approach to hierarchical clustering. Chapter 5 introduces *Supervised Taxonomy*, a new approach to taxonomy generation, its associated *STAXAC* algorithm, and methods for generating background knowledge from a dataset. Chapter 6 describes the *HC-edit* algorithm while chapter 7 provides summary of our work and proposes directions for future work.

References

1. Paul K. Amalaman, and Christoph F. Eick; *"Using Turning Point Detection to Obtain Better Regression Trees"*; In Proceeding of International Conference on Machine Learning and Data Mining (MLDM), New York City, New York, July 2013.

2. Paul K. Amalaman, and Christoph F. Eick; *"Avalanche: A Hierarchical, Divisive Clustering Algorithm";* In Proceeding of International Conference on Machine Learning and Data Mining (MLDM), Hamburg, Germany July, 2015.

3. Paul K. Amalaman, and Christoph F. Eick; "*HC-edit: A Hierarchical Clustering Approach To Data Editing";* In Proceeding of International Symposium on Methodologies for Intelligent Systems Lyon, France, October, 2015.

4. Paul K. Amalaman, and Christoph F. Eick; *"Supervised Taxonomies — Benefits, Algorithms and Applications";* submitted to Society for Industrial and Applied Mathematics Conference on Data Mining (SDM) 2016.

5. Paul K. Amalaman, and Christoph F. Eick*; "PATHFINDER: A New Bivariate Decision Tree Induction Approach*"; submitted to the Journal of Knowledge and Information System (KAIS).

Chapter 2

TPRTI: Using Turning Point Detection to Obtain Better Regression Trees[1]

2.1. Introduction

Regression trees are widely used machine learning techniques for numerical prediction. Among the regression trees, we may distinguish those that associate a constant to each leaf node, for instance CART [2], and those that fit a less-trivial model to each leaf node. Among the latter, we further distinguish a class of regression trees that fit a linear model to each leaf node, such as linear regression trees. Linear regression tree induction has been intensely researched. One of the first approaches, M5[12], induces the tree using a CART-like splitting decision which is a binary split based on mean values and uses constant-regression functions in the nodes; the attribute that best reduces the variance in the nodes is chosen for the split, and its mean value is selected as the split value. After the tree is fully developed M5 fits a linear model to each leaf-node during pruning phase. This splitting method is also used by many regression approaches which associate non-constant models with the leaf-nodes. However, in conjunction with non-constant regression models, using mean values of attributes as split points and using variance reduction as an objective function does not necessarily obtain the best model [7].

[1]Published in Proceeding of International Conference on Machine Learning and Data Mining (MLDM), New York City, New York, July 2013.

To address this issue, Karalic proposed RETIS [7] which fits a linear model in each node and uses minimization of the residual sum of squared errors (RSS) instead of the variance as splitting function. However, although the approach yields significantly better accuracy and smaller regression trees than M5 [12], it has been labeled "intractable" because to find the best pair {split attribute/split value} all potential split-values for each input attribute need be evaluated, making this approach too expensive even for medium-sized datasets. Therefore, many more scalable algorithms for inducing linear regression trees have been proposed [1, 5, 6, 7, 11, 12, 15, 17] which rely on heuristics, sampling, approximations, and different node evaluation functions to reduce the computational cost of the RETIS algorithm.

Detecting turning points which indicate locations where the general trend changes direction can be useful in many applications. In this chapter, we propose a new approach for Linear Regression Tree construction called Turning Point Regression Tree Induction (TPRTI) that infuses turning points into a regression tree induction algorithm to achieve improved scalability while maintaining accuracy and low-model complexity. TPRTI induces decision trees using the following procedure. First, a general trend is derived from the dataset by dividing the dataset into a sequence of subsets of equal size using a sliding window, and by associating a centroid with each subset. Second, using those centroids, a set of turning points is identified, indicating points in the input space in which the piecewise linear function associated with neighboring subsets changes direction. Finally, the identified turning points are used in two novel top-down linear regression tree induction

algorithms as potential split points. The two algorithms which are called TPRTI-A and TPRTI-B are discussed in section 2.

Fig. 2.1 illustrates our proposed approach to detecting turning points. The input dataset is shown in panel (A). In panel (B), the dataset is sorted by the input attribute and divided into subsets of equal size using a sliding window approach. In panel (C), the general trend in the dataset is derived by connecting the centroids of neighboring subsets, obtaining a piecewise linear function. Finally, in panel (D), points m1 and m3 are selected as turning points as they exhibit sharp turns in the piecewise linear function. The selected turning points are later fed to a linear regression tree algorithm as potential split points. Algorithm 1 gives the pseudo code of turning point detection algorithm.

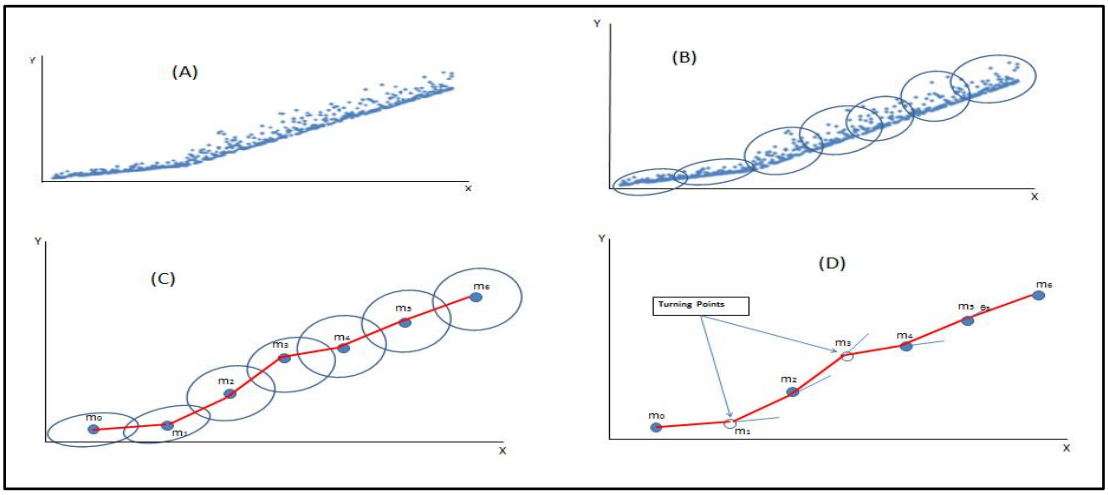


Fig. 2.1. Illustrating how the turning points are detected. Panel (A) represents hypothetical dataset in a plane (x,y). (B) The dataset is subdivided into overlapping subsets of equal size. (C) Centroids are joined by straight lines to form a general trend in the dataset. In panel (D) m_1 and m_3 are detected as turning points.

Panel (A) represents hypothetical dataset in a plane (x,y). (B) The dataset is subdivided into overlapping subsets of equal size. (C) Centroids are joined by straight lines to form a

general trend in the dataset. In panel (D) m_1 and m_3 are detected as turning points.

Algorithm 1: Determining turning points

1 Inputs:

2 plane (X_k,Y) where X_k's are input attributes (k=1,2,..p)

3 X_k; real valued discrete or continuous variable

4 Y: target variable

5 Outputs:

6 Turning points set

7

8 Project dataset onto each plane (X_k,Y)

9 For each plane (X_k,Y)

10 Sort the data per attribute X_k

11 IF X_k discrete attribute THEN

12 Compute centroids for each distinct value of X_k

13 Label centroids as turning points

14 ELSE

15 Use a sliding window of fixed size subsets for the input attribute to split the data into neighboring subsets from small values of X_k to the largest value of X_k

16 Determine general trends by computing the centroids of each subset and connect them to obtain piecewise linear functions

17 Identify turning points by analyzing the angle θ between neighboring subsets of the piecewise linear function

18 Output the set of turning points

The main contributions of this research include:

1. A novel approach for turning-point detection which relies on window sub-setting is introduced.
2. Two novel linear regression tree induction algorithms called TPRTI-A and TPRTI-B which incorporate turning points into the node evaluation are introduced.
3. State-of-the-art linear regression tree algorithms are compared with each other and with TPRTI-A and TPRTI-B for a challenging benchmark involving 12 datasets.
4. The experimental results indicate that TPRTI is a scalable algorithm that is capable of obtaining a high-predictive accuracy using smaller decision trees than other approaches.

The rest of the chapter is organized as follows. Section 2 contains the description of our proposed methods for linear regression trees. In section 3, we show results of experimental study and we conclude in Section 4.

Table 2.1. Notations used in the remainder of the chapter

K	User defined overlapping parameter characterizing the number of examples pertaining to two consecutive subsets.
S	Size of each subset
θ	Angle at a centroid
β	User-defined threshold angle such that if $\cos\theta < \cos\beta$ then the centroid with angle θ is a turning point
Stp_{XY}	Set of turning points in the XY-plane
Stp	Union of all Stp_{XY} (for all planes)
Stp_left	Turning Points set for left sub-node such that Stp=Stp_left $\cup$ Stp_right
Stp_right	Turning Point set for right sub-node such that Stp=Stp_left $\cup$ Stp_right
RSS	Residual Sum of Squared errors

2.2. Related Work

Linear regression algorithms can be divided into three groups: The first group fits a constant regression model to each intermediate node. M5 [12] is an example of this approach. The second group fits a more complex regression model to each node; usually at least a model is needed per input attribute. RETIS [7] is one such example since it fits multiple regression models per input attribute; one regression model for each distinct value of each input attribute. The third group uses linear regression models in the leaves, but at each node transforms the regression problem into a classification problem in order to use more efficient node evaluation function. SECRET [5], GUIDE [9], and SUPPORT [3], are examples of such approaches. Each group can be distinguished by how well it performs with respect to model accuracy, model complexity, and runtime complexity, which is how scalable the approach is when data sizes increase. To evaluate a node, the first group is computationally more efficient since the evaluation function in each node is the simplest; however it often yields complex models and low accuracy. The second group has a much better accuracy, and model complexity but it comes at the expense of a higher-cost node evaluation. Between both ends of the spectrum lies the third group. The major limitation of the first group of algorithms is illustrated next.

2.2.1. Illustrating the Limitations of the Variance-Based Approaches

Many widely used first-group algorithms use variance as node evaluation function and we will refer to them as "variance-based approaches". M5 is one such an example. As pointed out in [7], variance based algorithms have a fundamental imperfection to induce linear regression trees that are optimal because the variance is not a good node-evaluation

criterion. To illustrate this point, let us consider a dataset called dataset#2 whose instances were generated with respect to the function defined below without using any noise, assuming a uniform distribution of input values x1 in [0,250]: x1 $\in [0,250]$ and x2=0, and y $\in R$

$$\begin{cases} y = x1; & if\ 0 \le x1 < 50 \\ y = 100 - x1; & if\ 50 \le x1 < 250 \end{cases}$$

It is clear that the best split value is located at x1=50. The variance based approaches, like M5, will split at x1=125 (the mean value of x1) leading to the necessity to introduce further possible split to the data in the left node. RETIS however, which tests each distinct value of the input variable x1, selects the optimal split value 50.

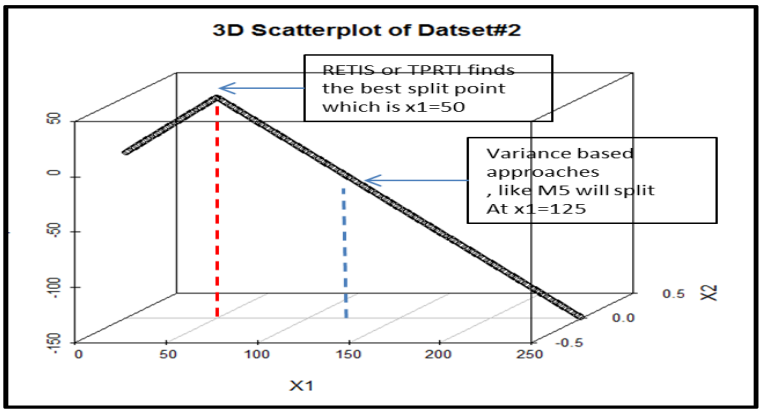


Fig. 2.2. Illustrating the imperfection of the M5 node-split method. M5 splits at x1=150 which is not optimum. Optimum split point is x1=50.

As pointed out earlier, many second-group approaches like RETIS yield more accurate, and shorter linear regression trees but do not scale well to large dataset. Likewise, the third group of approaches resolves the scalability issue of RETIS but at a cost to accuracy and/or model complexity.

2.3. The TPRTI Approach

In this chapter, we use the term "look-ahead RSS" to mean the following: First, the dataset associated with current node is split into two sub-sets; one pertaining to each sub-node. Second, each sub-node is fitted with a linear regression model and the Residual Sum-of-Squared errors (RSS) are computed for both nodes. Next, a weighted sum of both RSS is computed. This is done for all potential split pairs {attribute variable, attribute value} available. One such split is retained and the rest discarded. Likewise, we use the term "split point" to designate the pair {split attribute, split-attribute value}. We also use it to refer to the actual point in the plane, or point in the input space. When used as a point in a plane, it refers to the pair (x_q,y) where x_q is one of the input variables, $q \in \{1,..p\}$. When used to mean a point in the input space, it refers to a tuple; $(\mathbf{x},y)$ where $\mathbf{x}$ is an input vector $(x_1, ..x_p)$ and y the associate response. Key notations used in the remainder of the chapter are provided in Table 2.1. Algorithm1 presents the turning-point-detection approach. We provide next, in section 2.2 detailed description of how turning points are computed.

2.3.1. Centroids and Turning Points Computation

First, a general trend is derived from the dataset by sorting the examples based on the input attributes, by dividing the dataset into subsets of equal size using a sliding window, and by associating a centroid with each subset. Second, using those centroids, a set of turning points is identified, indicating points in the input space in which the piecewise

linear function, which was obtained by connecting centroids of neighboring subsets, significantly changes direction.

The input attributes to the algorithm are real-valued attributes that are either discrete or continuous. Lines 8, 9, and 10 in algorithm 1 project the dataset onto the p x_ky-planes (k=1,..,p). For the remaining of the lines, line 11 to 17, we consider one plane at a time. Line 10 ensures that the dataset associated with the plane is sorted with respect to the input attribute. Lines 11, 12, and 13 treat the case of discrete attributes. First, the algorithm queries the distinct values of the discrete attribute. It then computes the centroids for each attribute. Next, each centroid is labeled turning point.

Lines 14 to 17 treat the case where the input attribute is continuous. There are three user-defined parameters K, S, β that need to be set. Let assume that a centroid has angle θ. β is a user-defined angle such that if $\cos\theta < \cos\beta$ then the centroid is a turning point. K is the over-lapping parameter characterizing the number of examples pertaining to two consecutive subsets. S is the size of each subset. In line 15 subsets of equal size S are created using the sorted dataset as follows: S_0 is composed of the first S examples. At any given step i, (i>0), the examples in subset S_i are determined by dropping the first K examples in S_{i-1} and by adding the next K new examples. When K=S, the subsets are disjoint. When 0<K< S the subsets overlap. In line 17 turning points are computed for each plane by analyzing the angle at each centroid.

2.3.2. Node Evaluation

We introduce TPRTI-A, which is a mixture of the first group and second group approach in that its node evaluation avoids exhaustive search by evaluating only a supplied list of turning points. We also introduce TPRTI-B which is a mixture of all 3 groups in that it uses a two-step node evaluation. It avoids exhaustive search by evaluating only a supplied set of turning points. It first fits a model to the current node, and uses a simple evaluation function which is the distance of each turning point to the fitted model. The turning-point with the largest distance is selected to split the node. TPRTI-A and TPRTI-B differ by their node evaluation function. They both use as input, a set of predetermined turning points.

2.3.2.1 Node Evaluation for TPRTI-A

The first approach, TPRTI-A, evaluates all turning points by a look-ahead strategy and selects the one that yields the minimum RSS.

Algorithm 2: Node evaluation for TPRTI-A

1. Inputs: Stp_{XY}: Set of all turning points in the XY-plane
2. Stp: Union of all Stp_{XY} (for all planes)
3. $TP_{XY}(x,y)$: Turning point in the XY-plane with coordinate (x,y)
4. Outputs: Left_Stp, Right_Stp; left and right node datasets
5. IF stopping criteria is reached THEN
6. RETURN
7. FOR each Stp_{XY} in Stp
8. FOR each $TP_{XY}(x,y)$ in Stp_{XY}
9. Split data in S_{Left} and S_{right}

10. Compute look-ahead weighted RSS

11. Select the turning point (x_α,y_α) that minimizes weighted RSS for the split

12. Split Stp into Left_Stp, and Right_Stp based on x_α

2.3.2.2. Node Evaluation for TPRTI-B

The second approach, TPRTI-B, is a multi-step evaluation approach. With this approach, first the current node is fitted with a model and the distances of the turning points (actual tuples) to the fitted model are computed. The turning point for which the distance to the model is the largest is selected as split point. Next, each coordinate (value of input attributes) of the split point needs be evaluated by a look-ahead RSS minimization method to determine the best pair {split variable, split value}. That is, in contrast to TPRTI-A, only a single split value per input attribute and not a set of split values is considered; reducing runtime complexity. Fig. 2.3 illustrates the general idea. In Fig. 2.3, the dotted line represents a linear model F fitted to current node. In this hypothetical node, the turning point set has three turning points TP1, TP2, and TP3. We assume d1 < d2 and d3 < d2. Hence, TP2(x_{21},x_{22},y_2) is chosen as split point. Its coordinates x_1= x_{21}, and x_2= x_{22} need next, to be evaluated to select the pair {split variable, split value} that best minimizes RSS. Algorithm 3 summarizes the concept.

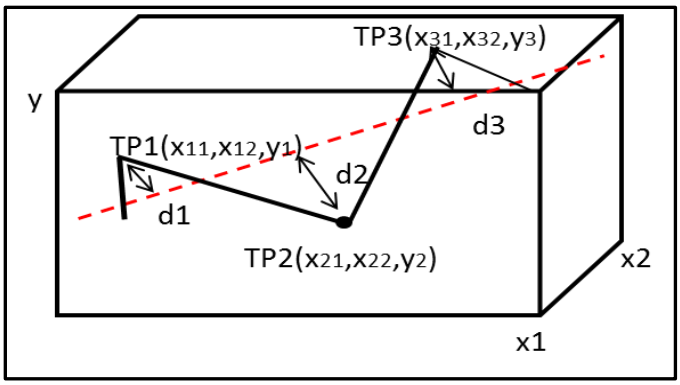


Fig. 2.3. Illustrating the node-evaluation of TPRTI-B. We assume d1 < d2 and d3 < d2. Hence, TP2(x_{21},x_{22},y_2) is chosen as split point because it is the turning point further away from the fitted model F

In Fig. 2.3, the dotted line is a model fitted to current node. TP2 is selected as a split point because it is the turning point further away from the fitted-model F.

Algorithm 3: Node evaluation for TPRTI-B

1. IF stopping criteria is reached then
2. RETURN
3. Fit current node with linear model F
4. FOR each turning point tp in Stp
5. Compute distance to F
6. Select tp_max the point with the largest distance to F
7. FOR each input coordinate of tp_max
8. Split data in S_{Left} and S_{right}
9. Compute look-ahead weighted RSS
10. Select the turning point that minimizes weighted RSS as split point
11. Split Stp into Left_Stp, and Right_Stp

In line 3, a model F is fitted to the node. Lines 4, 5, and 6 select the point with the largest distance to F. Line 7 to 11 are similar to what was presented for TPRTI-A in 2.3.2.1 except that here the points are represented in the actual space; not in a plane. Hence, the turning points set Stp, although computed as presented previously in algorithm 1, is formed of points with their coordinates in the data space. This is so because line 5 needs to compute the distance to F in the entire input space.

2.3.3. Runtime Complexity

We compute the runtime cost to evaluate a node. Let n be the total number of training examples in the node, m, the number of subsets, p the number of input attributes, and t total number of turning points. Assuming n>>p, evaluating each split candidate costs $O(p^2n)$; If TPRTI-A method is used, all the turning points are evaluated and the cost to evaluate a node is $O(p^2nt)$. If TPRTI-B is used, the distance of each turning point in the data space to the fitted curve cost O(p) which leads to O(pt) for the t turning points. $O(p^2n)$ is needed to evaluate each of the p input attribute, and $O(p^2n)$ is as well needed to fit a model to the current node; obtaining: $O(pt+p^2n+np^3) = O(p^2n(p+1))$. With M5, the split point is the mean point of each p variable. Hence, t=p obtaining $O(p^3n)$; In the worst case, RETIS will test each value of each variable leading to t=pn ; thus $O(p^3n^2)$. TPRTI-A worse case happens when each centroid is a turning point; which leads to t=pm, hence, $O(p^3nm)$. Table 2.2 summarizes the runtime complexity of each approach.

Table 2.2. Node runtime complexity of TPRTI in comparison to M5 and RETIS

	RETIS	TPRTI-A	TPRTI-B	M5
Runtime complexity	$O(p^3n^2)$	$O(p^3nm)$	$O(p^2n(p+1))$	$O(p^3n)$

2.3.4. Stopping Criteria

RETIS, M5 and TPRTI share the following stopping criteria: The algorithm stops if the number of examples in the node is less than a minimum number set by user. The algorithm stops if the subsequent split sub-nodes do not improve the error function more than a minimum value set by user. However, TPRTI has an additional stopping condition: The algorithm may terminate if there is no more turning points in the node dataset to evaluate.

2.4. Experimental Evaluation

In this section, results of extensive experimental study of TPRTI-A and TPRTI-B are provided. We compare both algorithms with each other and against the well-known M5 [12], SECRET [5], GUIDE [9], and RETIS [7] algorithms in term of accuracy, scalability, and model complexity. The experimental result published in [5], for GUIDE and SECRET, two state-of-the-art scalable linear regression tree induction algorithms are used in this study for comparison. The GUIDE and SECRET algorithms were built to be both fast and accurate. Model complexity results for previously used datasets were not available for comparison. We performed all the experiments reported in this chapter on a 64-bit PC i7-2630 CPU at 2 Ghz running Windows 7.

2.4.1. Datasets

Five artificial and 7 real-world datasets were used in the experiments; Table 2.3 and 2.4 give a summary for the 12 datasets. The last column in both tables contains in parenthesis the number of attributes.

Table 2.3. Artificial datasets

Dataset	Description	Number of examples
dataset #1	x1, x2 are the input variables and y the output variable; x1 ∈ R ,x2 ∈ R, y∈ R $\begin{cases} Y = -2 * x1 & \text{if } 0 \le x1 < 100 \text{ and } x2 = 0 \\ Y = -700 + 5 * x1 & \text{if } 100 \le x1 < 200 \text{ and } x2 = 0 \\ Y = 300 - 3 * x2 & \text{if } 0 \le x2 < 100 \text{ and } x1 = 200 \end{cases}$	300 (3)
dataset #2	x1 ∈ [0,250] and x2=0, and y ∈ R $\begin{cases} y = x1; & if\ 0 \le x1 < 50 \\ y = 100 - x1; & if\ 50 \le x1 < 250 \end{cases}$	2500 (3)
CART dataset	This dataset was found in [2] with 10 predictor attributes:X1 ϵ {−1, 1} with equal probability that X1 =1 or X1 = -1; Xi ϵ {−1, 0, 1}, with i ϵ {2, . . ,10} and the predicted attribute y determined by $\begin{cases} Y = 3 + 3X2 + 2X3 + X4 & if\ x1 = 1 \\ Y = -3 + 3X5 + 2X6 + X7 & if\ x = -1 \end{cases}$ A random noise ε between [-2 and 2] was added to Y	750 (11)
3DSin dataset	This dataset has two continuous predictor attributes x1, x2 uniformly distributed in interval [−3, 3] determined by Y = 3 sin(X1) sin(X2).	500(3)
Fried dataset	This dataset has 10 continuous predictor attributes with independent values uniformly distributed in the interval [0, 1] Y = 10 sin(πX1X2) + 20(X3−0.5)^2 + 10X4+5X5;A random noise ε between [-1;1] was added	700(11)

Table 2.4. Real world datasets

Dataset	Description	Number of examples(number of attributes)
Abalone	This dataset was obtained from UCI [16] machine learning repository.	4177 (8)
Auto-mpg	This dataset obtained from UCI [16] repository. Tuples with missing data were removed.	392 (8)
Boston Housing	This dataset obtained from UCI[16] repository	506 (14)
Kin8nm	This dataset was obtained from the DELVE [4] repository.	8192 (9)
Normalized Auto-mpg	This is the auto-mpg dataset from UCI [16] repository that has been normalized with z-score values	392 (8)
STOCK	This dataset is from SatLib [14] . The dataset contains 950 examples. However, the first tuple was removed because it did not appear correctly formatted	949 (10)
Tecator	This dataset originated from the StatLib [14] repository.	240 (11)

2.4.2. Experimental Methodology

All the experiments were done with a 10-fold cross validation and repeated 10 times with different seeds for each run. The average values are reported in Table 2.7 along with corresponding standard deviation. Five artificial datasets were used three of which were previously used and results for GUIDE and SECRET are available in [5]. We also used 7 real world datasets of which 6 were previously used and results for GUIDE and SECRET available in [5]. The datasets which have not been previously used are dataset#1, dataset#2, and the normalized auto mpg dataset. We implemented TPRTI making use of R software [13] libraries. We run M5 using Weka [18]. R and Weka are publicly available software. Our implementation of RETIS relies on running TPRTI-A with all input attributes set as discrete attributes. Input parameters and stopping rules used for the experiments are in Table 2.5.

Table 2.5. Input and stopping parameters for TPRTI

	TPRTI-A				**TPRTI-B**			
	Input parameters		**Stopping rules*		*Input parameters*		*Stopping rules*	
	Subset Size	cos β	Min. Node Size (in %)	Min. RSS imp.(in%)	Subset size	cos β	Min. Node Size (in %)	Min. RSS imp. (in %)
Dataset #1	3	0.8	10	10	3	0.8	10	10
Dataset #2	9	0.8	10	10	9	0.8	10	10
CART	5	0.8	10	10	5	0.8	10	10
3DSin	4	0.8	10	10	5	0.8	10	10
Fried	3	0.85	10	10	3	0.85	10	10
Abalone	55	0.8	10	10	21	0.8	10	10
Auto mpg	4	0.85	10	10	4	0.85	10	10
Boston Housing	14	0.8	12	12	14	85	12	12
Normalized auto mpg (z-score)	4	0.85	10	10	4	0.85	10	10
Stock Data	4	0.8	10	10	13	0.85	10	10
Tecator	8	0.8	10	10	21	0.7	10	10

Kin8nm	250	0.95	3	1	250	0.95	3	1

**Stopping rules:* The stopping parameters set for TPRTI-A are same parameters used for RETIS.

For each dataset, different input parameters and stopping rules were used for TPRTI-A and TPRTI-B. Table 2.5 summarizes the parameters used for each dataset where cosβ is the cosine threshold used to determine the turning points, "Min. Node Size" is the minimum- required size of a node expressed as 100*(current node size)/(initial dataset), and "Min. RSS imp." is the minimum-improvement of the sum of the weighted RSS of both sub-nodes required to split a node. It is expressed as 100*(Parent RSS - weighted sum of children nodes RSS)/Parent RSS. The input parameter "Subset Size" is used to subdivide the input data into subsets of equal size in order to compute the centroids. RETIS was run without post pruning.

2.4.3. Results

Accuracy was measured by the MSE (Mean Squared Error). Model Complexity was measured by the number of leaf-nodes. However, a bad model may have small number of leaf-nodes. Thus complexity was slightly redefined as number of times an approach obtained the combination (best accuracy, fewest leaf-nodes). Both the number of leaf-nodes and MSE are provided as $\mu \pm c$ where μ is the average MSE (or number of leaf-node) and c the standard deviation over several runs. Let $\mu_1 \pm c_1$ and $\mu_2 \pm c_2$ be two results such that $\mu_1 < \mu_2$. We consider a tie between the two results if $\mu_1 + c_1 > \mu_2$. Both accuracy and number of leaf-nodes are reported in Table 2.7 with the number of leaf-nodes in parenthesis. The main findings of our study are provided in 2.4.3.1, 2.4.3.2, and 2.4.3.3.

2.4.3.1. Model Accuracy

Table 2.6. Comparison between TPRTI and state-of-the-art approaches with respect to accuracy

	M5	TPRTI-A	TPRTI-B	RETIS	GUIDE	SECRET
TPRTI-A	(6/5/1)	-	(4/6/2)	(4/6/0)	(5/1/2)	(4/2/2)
TPRTI-B	(4/6/2)	(2/6/4)	-	(3/7/0)	(5/1/2)	(1/4/3)

TPRTI-A, and TPRTI-B are compared with the approaches in the columns. The number in each cell denotes (number of wins/number of ties/number of losses). For example, (6/5/1) in the first column of the first row means that TPRTI-A is more accurate than M5 on 6 datasets, ties M5 on 5 datasets and loses on 1 dataset. Overall, TPRTI yields comparable result as or slightly better result than RETIS. It has better accuracy than GUIDE and SECRET.

Table 2.7. Accuracy results

	M5	RETIS	GUIDE	SECRET	TPRTI-A	TPRTI-B
Dataset #1	446.899 ±36.45 (11±0.00)	**0.000 ±0.0000 (3±0.00)**	N.A	N.A	0.089 ±0.0000 **(3±0.00)**	0.089 ±0.0000 **(3±0.00)**
Dataset #2	4.75 ±0.239 (11±0.00)	**0.000 ±0.0000 (2±0.00)**	N.A	N.A	**0.000 ±0.0000 (2±0.00)**	**0.000 ±0.0000 (2±0.00)**
CART	0.0932 ±0.0009 **(2±0.00)**	0.085 ±0.0125 (4.1±0.32)	N.A	N.A	**0.07614 ±0.0100** (4.1±0.32)	0.098±0.33 (6.1±0.32)
3DSin	**0.001 ±0.0002** (20±0.00)	0.01 ±0.0074 (4±0.00)	0.0448 ±0.0018	0.0384 ±0.0026	0.0074 ±0.01 (4±0.00)	0.0063 ±0.01 **(3±0.00)**
Fried	4.888 ±0.1536 **(3±0.00)**	4.773 ±0.3893 **(3±0.00)**	**1.21 ±0.0000**	1.26 ±0.010	3.1114 ±0.80 (4±0.00)	1.4968 ±0.60 (6.7±0.48)
Abalone	4.691 ±0.586 **(2±0.00)**	*N.A	4.63 ±0.04	4.67 ±0.04	4.3806 ±2.71 (4±0.00)	**4.1527±2.59** (5.1±0.45)
Auto mpg	8.507 ±0.3105 (5±0.00)	8.8470 ±7.2183 **(3.1±0.32)**	34.92 ±21.92	15.88 ±0.68	**7.6021 ±6.33** (5±0.00)	8.4493 ±6.39 (4.6±0.52)
Boston	28.839	24.569±20.090	40.63	24.01	**16.0922±10.29**	19.6237 ±9.24

Housing	±30.889 (10±0.00)	**(4.2 ±0.92)**	±6.63	±0.69	(5.5±0.53)	(4.8±0.92)
Normalized Auto mpg (z-score)	0.139 ±0.0051 (5±0.00)	0.1186±0.0895 (4.0 ±0.00)	N.A	N.A	**0.1169 ±0.07 (3.8±0.63)**	0.1342 ±0.09 (4.7±0.82)
Stock Data	1.038 ±0.100 (19±0.00)	11.977±7.884 (3.9 ±0.32)	1.49 ±0.09	1.35 ±0.05	**0.2067 ±0.10 (3±0.47)**	4.8867 ±3.09 (4.9±0.88)
Tecator	9.451 ±2.9519 (6±0.00)	6.6310±6.3036 (5.4 ±0.51)	13.46 ±0.72	12.08 ±0.53	**2.8315 ±1.412 (3.1 ±0.31)**	7.1266 ±8.20 (6.4±0.70)
Kin8nm	0.030 ±0.0009 (24±0.00)	*N.A.	0.0235 ±0.0002	**0.0222 ±0.0002**	0.0303 ±0.001 (5.33±0.57)	0.0227±0.0020 (25.5±0.17)

*N.A is used to express the fact that the program runs more than 3 hours without outputting a result or runs out of memory whereas N.A is used to express the fact that the result is not available.

2.4.3.2. Model Complexity

In this study, we consider a linear regression model to have a good model-complexity when it is both accurate and has a small number of leaf nodes. Table 2.8, which is compiled from Table 2.7, presents the number of cases where an approach obtained both best accuracy and fewest nodes at the same time.

Table 2.8. Number of time an approach obtained the combination (best accuracy, fewest leaf nodes) for a dataset

M5	TPRTI-A	TPRTI-B	RETIS	GUIDE	SECRET
0	5	3	5	N.A	N.A

RETIS and TPRTI-A won the combination (best accuracy, fewest leaf nodes) five times while M5 never won, and TPRTI-B won 3 times. This suggests that TPRTI hold comparable model complexity as RETIS.

2.4.3.3. Model Scalability

We use a direct comparison of runtime in seconds for TPRTI-A, TPRTI-B, and RETIS because they were implemented and run in the same machine. We use an indirect comparison to compare the different approaches. The indirect comparison consists of setting a baseline dataset size, and measuring the percent increase in runtime in relation to percent increase in baseline-dataset size. Fig. 2.4 summarizes our findings. TPRTI-B outperforms M5 consistently on all dataset sizes and number of input attribute. This suggests that TPRTI-B is a more scalable approach than M5. This is because models generated by TPRTI tend to have fewer nodes. On small to medium size datasets, there are no significant differences between TPRTI and SECRET. Overall SECRET outperforms TPRTI consistently on all dataset sizes. Fig. 2.5 summarizes our result for the direct comparison. In Fig. 2.5, Panel (A) shows that RETIS has the worst performance even on dataset with small number of input attribute. Panel (B) provides evidence that as the number of input attribute increases, performance decreases. Panel(C) and (D) demonstrate that TPRTI-B consistently outperform TPRTI-A.

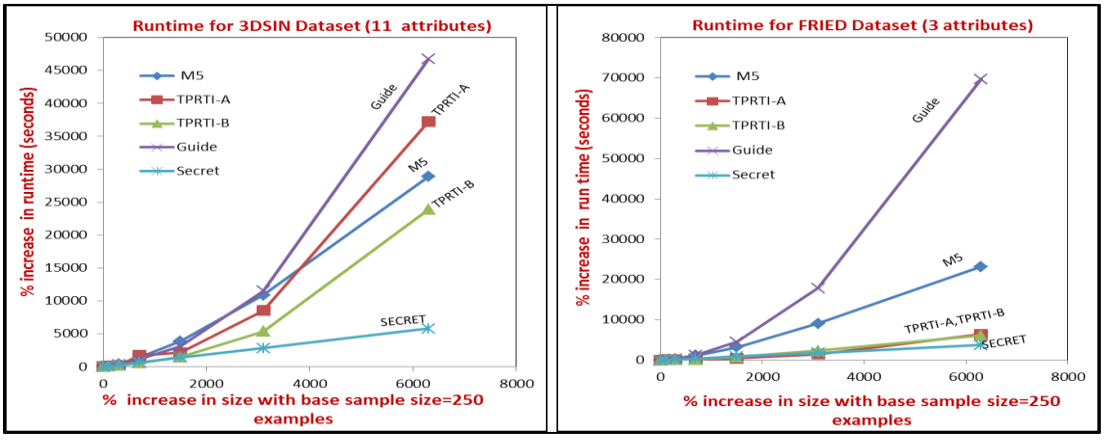


Fig. 2.4. Indirect runtime comparison: Percent increase in baseline dataset size and the resulting percent increase in runtime with baseline size set to 250 examples

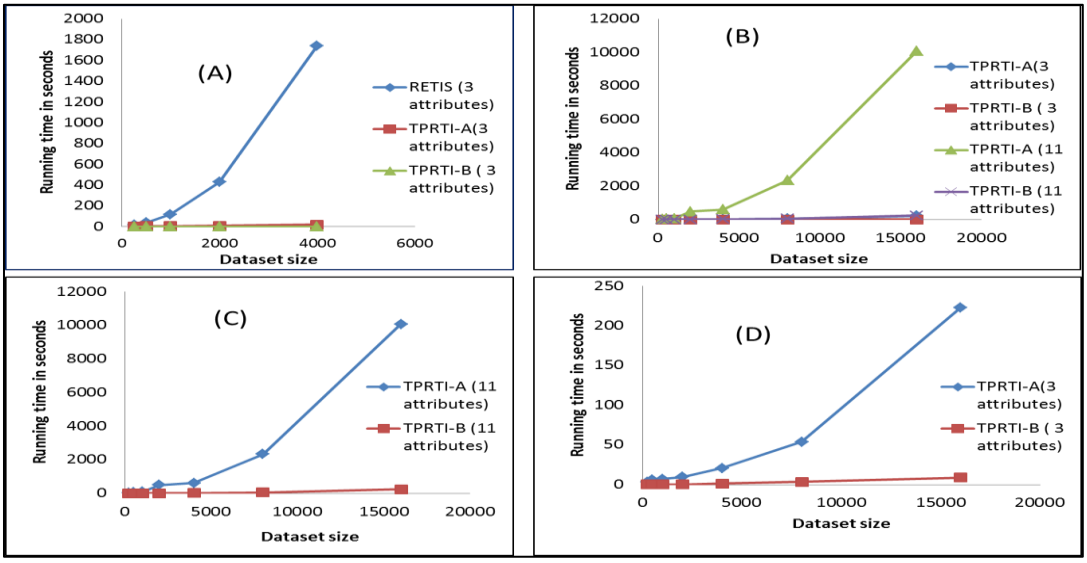


Fig. 2.5. Direct runtime comparison among TPRTI-A, TPRTI-B and RETIS

2.5. Conclusion

This research proposes a new approach for Linear Regression Tree construction called Turning Point Regression Tree Induction (TPRTI) that infuses turning points into a regression tree induction algorithm to achieve improved scalability while maintaining high accuracy and low model complexity. Two novel linear regression tree induction algorithms called TPRTI-A and TPRTI-B which incorporate turning points into the node evaluation were introduced and experimental results indicate that TPRTI is a scalable algorithm that is capable of obtaining a high predictive accuracy using smaller decision trees than other approaches.

References

1. W.P. Alexander and S.D. Grimshaw; *"Treed regression";* Journal of Computational and Graphical Statistics, 5:156-175, 1996.

2. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone; *"Classification and Regression Trees"*; Wadsworth, Belmont, 1984.

3. P. Chaudhuri, M.-C. Huang, W.-Y. Loh, and R. Yao; *"Piecewise-polynomial regression trees"*; Statistica Sinica, 4:143–167, 1994.

4. DELVE repository of data http://www.cs.toronto.edu/~delve/ as of 12/04/2012.

5. Alin Dobra, Johannes Gehrke; *"SECRET:a scalable linear regression tree algorithm"*; In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining -2002.

6. J. Friedman; *"Multivariate adaptive regression splines (with discussion)"*; Annals of Statistics, 19:1-142, 1991.

7. A. Karalic; *"Employing linear regression in regression tree leaves"*; In European Conference on Artificial Intelligence, pages 440-441, 1992.

8. Li, K.C., Lue, H.H., Chen, C.H.; "*Interactive Tree-Structured Regression via Principal Hessian Directions"*; Journal of the American Statistical Association, vol. 95, pp. 547-560, 2000.

9. W.-Y. Loh; "*Regression trees with unbiased variable selection and interaction detection*"; Statistica Sinica, 12:361-386, 2002.

10. Loh, W.-Y., and Shih, Y.-S. (1997); "*Split Selection Methods for Classification Trees*"; Statistica Sinica, Vol.7, 1997, pp. 815-840.

11. D. Malerba, F. Esposito, M. Ceci, and A. Appice; "*Top-down induction of model trees with regression and splitting nodes*"; IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(5):612-625, 2004.

12. J. R. Quinlan; "*Learning with Continuous Classes";* In 5th Australian Joint Conference on Artificial Intelligence, pages 343–348, 1992.

13. http://www.r-project.org/ The R Project for Statistical Computing official website as of 8/8/2012.

14. StatLib repository (Dataset Archive) at http://lib.stat.cmu.edu/datasets/ as of 12/04/2013.

15. L. Torgo; "*Functional models for regression tree leaves*"; In Proc. 14th International Conference on Machine Learning, pages 385–393. Morgan Kaufmann, 1997.
16. UCI repository at http://archive.ics.uci.edu/ml/datasets.html as of 12/04/2012.

17. David S. Vogel, Ognian Asparouhov, Tobias Scheffer; "*Scalable Look-Ahead Linear Regression Trees*"; KDD International Conference on Knowledge Discovery and Data Mining, August 12–15, 2007, San Jose, California, USA.

18. http://www.cs.waikato.ac.nz/ml/weka/; Weka software official website as of 8/8/2012.

Chapter 3

PATHFINDER: A New Bivariate Decision Tree Induction Approach

3.1. Introduction

Decision trees are very important tools for many applications involving prediction such as weather, and financial forecasting. The earlier approaches use a top down hierarchical split strategies based on a single attribute tests [3, 8]. This strategy yields axis-parallel hyperplanes. These simple univariate tests are convenient because a domain expert can easily interpret the decision trees, but they may result in complex trees. Oblique decision trees, on the other hand, [13, 4, 5], use multivariate tests that are not necessarily parallel to the axes, and, in some cases yield much smaller and more accurate trees. However, oblique trees are not as popular as the axis-parallel trees because the tests are more difficult to interpret, and require greater computational resources than the axis-parallel algorithms. At each node, oblique tree inducing approaches split the node dataset in two subnodes by cutting through the entire input space with an oblique hyperplanes. Oblique methods, just like their axis-parallel counterparts, often generate small regions next to a larger region in a single node which may lead to sub-optimal accuracy.

This research proposes PATHFINDER, a new decision tree induction approach that uses piecewise oblique linear segments (pieces of hyperplanes) to split the node dataset at-

tempting to avoid splitting high density regions with respect to a single class. By using pieces of oblique segments (rather than oblique lines) to locally construct decision boundaries separating neighboring regions dominated by different classes, the number of node splits can be reduced; which in turn can improve accuracy and reduce tree complexity. When PATHFINDER is used, it first clusters the dataset into clusters of high purities, then using piecewise linear segments in planes formed by pairs of attributes, it locally splits "opposed clusters", attempting to keep a maximum number of clusters un-split. Opposed clusters are neighboring clusters whose majority of examples pertains to different classes. To illustrate, let us consider the hypothetical 3-classes dataset in 2D (Fig. 3.1) formed by three regions of high purities C_1, C_2, and C_3 with three different dominant classes (c1, c2, and c3). It is further assumed that in Fig. 3.1 the sizes of the clusters are proportional to the number of examples in the regions. The original dataset is depicted by Fig. 3.1.a. The axis parallel method requires a lot of splits (Fig. 3.1.b). When oblique split strategy is used (Fig. 3.1.c) a reduced number of splits is obtained as most of the clusters are kept un-split. However, the oblique approach reduces the number of splits but does not necessarily yield the optimum number of split. As illustrated in Fig. 3.1.c, hyperplane (1) splits C_1 into two nodes with one node dominated by examples of class c3 contaminated by few examples of class c1. The advantage of the proposed piecewise oblique approach is illustrated in Fig. 3.1.d. where the clusters are isolated without splitting them by "making turns" within the input space (at points A, B, and C).

By using pieces of line segments (instead of lines) to divide the input space, a maximum number of clusters can be preserved (un-split) which may improve accuracy rate. The

main goal of PATHFINDER is to avoid as much as possible splitting of clusters during the tree induction recursive process. However, the proposed splitting often generates regions in the input space that do not belong to any cluster. We term these regions *reject regions*, and the examples residing in those regions are referred to as *reject examples*. In Fig. 3.1.d the region delimited by triangle ABC is a reject region because an example residing in that region would not pertain to any cluster. We argue that in some applications where the cost of misclassification is high, such as medical diagnostic, having a high accuracy rate classifier at the expense of not classifying few examples can be beneficial.

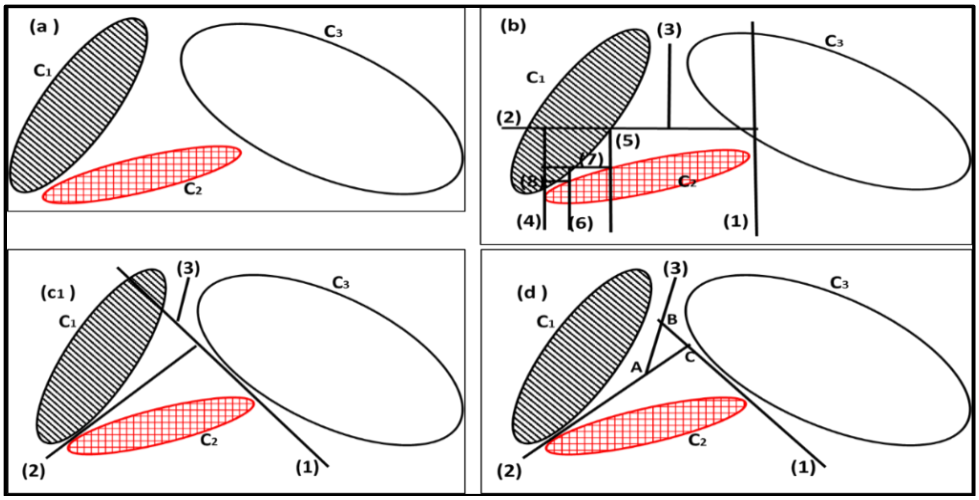


Fig. 3.1. Illustrating axis parallel, oblique, and piecewise oblique decision trees

To achieve its goal, PATHFINDER must find an efficient solution for the following problems:

(1) How to compute the number of clusters in the dataset (if such clusters exist)

(2) How to compute local boundaries separating opposed clusters

(3) How to define test conditions in the node to split the dataset.

PATHFINDER uses traditional methods as well as new techniques to overcome the above challenges. With respect to problem (1), a grid based approach is used whereby 2D planes are formed using pairs of attributes. Then in each plane, pairs of classes are used to cluster one class against the other. That is, only examples whose class labels are one of the selected pair of classes are considered — the rest of the examples are ignored. With respect to issue (2), after the clusters are computed, a line tracing approach is used to determine opposed clusters in 2D, and their mid-points are used as boundary points. Regarding issue (3) a set of candidate tuples *(plane, class label, set of high purity clusters and their boundary points)* — obtained from step (2) — are used in test conditions to split the node dataset. The Gini index is computed for each tuple, and the tuple that has the highest Gini gain is selected to split the node.

The contributions of this research are the following:

1. We have developed a new split strategy for oblique decision tree that is capable of using pieces of segments as opposed to linear oblique lines to split the input space. By using piecewise oblique segments, high purity regions can be preserved which in turn may lead to improvement in accuracy.
2. We introduce a new decision tree induction approach that employs clusters in a grid and their boundary points to recursively split the node dataset.
3. We propose a novel node split test condition for binary tree induction capable of separating instances of a class from instances of other classes in a single step.

4. The approach provides a reject option — examples that do not belong to any cluster will not be classified.

The rest of the chapter is organized as follows. In section 2 we provide related work. Section 3 details our approach. Section 4 contains results of extensive experiments and we conclude in section 5.

3.2. Related Work

Breiman et al. [3] were the first to suggest inducing decision trees with oblique hyperplanes (a linear combination of the attributes) [13]. We assume examples have the form $x=(a_1,..,a_r, c_k)$ where c_k is a class label, and the a_i's real-valued attributes; the test condition in the node is of the form

$$\sum_{i=1}^{r} \lambda_i x_i + \lambda_{r+1} \geq 0 \qquad (1)$$

where the λ_i's are real-valued coefficients and r the number of attributes. We note that when all the coefficients are equal to 0 except one, equation (1) corresponds to the axis-parallel test condition. An optimal hyperplane is found when equation (1) separates the dataset into two sets such that some measure of impurity is minimal. With regards to the complexity of selecting an optimal hyperplane to split the dataset at a node, for n points in r dimensions there are r.n possible tests for the axis-parallel method; one attribute is tested at a time for all possible n split values. For an oblique tree a subset of attributes must be selected and all possible combination of attribute values from the n examples must be tested. There are 2^r possible choices for the attributes and $\binom{n}{r}$ possible ways to

select the attribute values. The task of selecting the optimal hyperplane has an upper bound runtime complexity of $2^r * \binom{n}{r}$. Therefore searching for the optimal oblique hyperplane is a very complex task. In fact it is NP-hard [13].

Current oblique decision tree algorithms use a heuristic to find the values for the coefficients of equation (1). Breiman et al. [3] introduce an oblique decision tree induction algorithm, CART-LC, which induces oblique decision tree as follows. At each node, CART-LC first computes the optimal axis-parallel hyperplane (all λi's but one are 0s). It then iteratively computes local optimal values for each of the λ_i's. The main idea is to perturb the hyperplane around the computed optimal axis-parallel hyperplane until the marginal benefits become smaller than some predefined constant value, ε (ε >0). In some cases, doing so fails to find the optimal split. For example, in Fig. 3.3, the optimum axis-parallel split is assumed to be hyperplane (1). Hyperplanes (2) and (3) are two perturbations of (1) that results in no split improvement. Murthy et al. [13] introduced an improved version of the CART-LC, called OC1, which uses randomization to break from the local optimal trap (when perturbation of the hyperplane at a given location yields no improvement).

The overview of the OC1 algorithm is provided in Fig. 3.2. As in CART-LC, OC1 starts with the best axis-parallel hyperplane (Line 1) then perturbs the hyperplane by finding local optimal values, one coefficient at a time (Step 1) until no improvement is observed. Then unlike CART-LC, it breaks from the local optimal trap by randomly selecting a hyperplane at a different location in the dataset and re-starts the process. For that reason

OC1 may produce a different tree (for each run) for the same dataset. As illustrated in Fig. 3.3, OC1 is capable of finding the optimal hyperplane (hyperplane (4)).

To find a split of a set of examples T:
1. Find the best axis-parallel split of T. Let *I* be the impurity of this split.
2. Repeat *R* times:
3. Choose a random hyperplane *H*.
4. (For the first iteration, initialize *H* to be the best axis-parallel split.)
5. Step 1: Until the impurity measure does not improve, do:
6. Perturb each of the coefficients of *H* in sequence.
7. Step 2: Repeat at most *J* times:
8. Choose a random direction and attempt to perturb *H* in that direction.
9. If this reduces the impurity of *H*, go to Step 1.
10. Let I_1 = the impurity of H. If $I_1 < I$, then set $I = I_1$.
11. Output the split corresponding to *I*.

Fig.3.2. Overview of the OC1 algorithm for a single node of a decision tree [13]

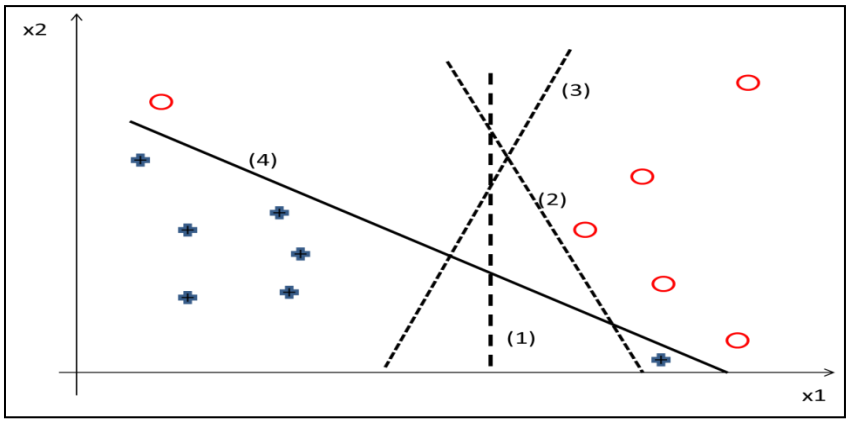


Fig. 3.3. Illustrating the difference between CART-LC and the OC1 algorithm

Recently, a family of six boundaries computation methods for linear classifiers have been proposed by [1]. Evolutionary algorithms have been used as well to induce decision trees [11, 12]. A series of discriminant analysis based multivariate decision tree systems have also been developed by [7, 6]. Linear-programming based oblique tree algorithms have been proposed by [2]. The results of these studies show that oblique decision trees gener-

ally have higher accuracies and a smaller tree sizes than univariate trees, but at the expense of longer computing times.

An area closely related to our work is piecewise linear classifiers which uses combinations of linear classifiers to separate clusters in the dataset [14, 15]. However, in [14] the authors use prototype-based clustering approach to cluster the input dataset and construct hyperplanes to separate the opposed clusters; which requires that a number of k opposed clusters be defined before training. In [15] the authors first identify opposed examples (instead of clusters), and join them with line segments, also called *links*. Opposed examples are nearest neighbor examples with different class labels. In a second step, hyperplanes are constructed such that a hyperplane cut as many links as possible. This approach is too sensitive to outliers. Furthermore both approaches, [14] and [15], compute the hyperplanes in the input space which can be an expensive process (as previously discussed). The general goal of our proposed approach is to partition the input space into high purities regions while avoiding the difficulties associated with finding optimal hyperplanes in the input space. Section 3 provides details of the PATHFINDER approach.

3.3. The PATHFINDER Approach

PATHFINDER is an oblique decision tree induction approach. However, it uses pieces of consecutive oblique line segments (pieces of hyperplanes) to split the input space. For that reason we prefer to refer to it as piecewise oblique decision tree induction approach. Fig. 3.4 illustrates the concept. Fig. 3.4.a represents a hypothetical dataset and Fig. 3.4.d its possible splitting result by PATHFINDER. It can be observed that high purity regions

have been obtained except region ABDC which has poor purity. Examples in region ABDC are reject examples because they do not belong to any cluster.

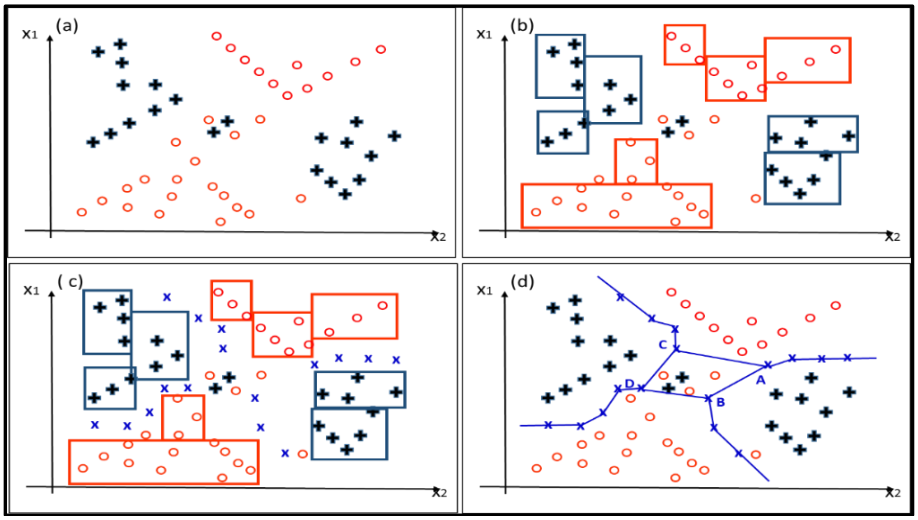


Fig. 3.4. Illustrating the PATHFINDER boundary points computation

The general approach to boundary points computation follows the below steps:

1. Project the dataset onto 2D planes formed by pairs of input attributes.
2. Use a grid-based approach to cluster the dataset into rectangular shape clusters (Fig. 3.4.b). Adjacent clusters of same class are merged into a larger cluster.
3. A line tracing approach is introduced to determine opposed clusters and boundary points are computed as midpoints separating opposed clusters (Fig. 3.4.c).

In Fig. 3.4.c the "x" represent the midpoints between two opposed clusters in a row or column of the grid. We refer to those points as boundary points. The key idea is that a cluster is stored along with its boundary points. Fig. 3.4.d illustrates a possible decision boundary by joining the boundary points into segments. Actual decision boundaries are

created locally at runtime using a subset of the boundary points that were computed in step 3. At each node, using the boundary points and the clustering results PATHFINDER splits the node dataset in two sub-nodes. The recursive process continues until some stopping criteria are met. The PATHFINDER tree generation method is provided in algorithm 1.

Algorithm 1: PATHFINDER Tree Generation

Begin

1. Project the dataset onto 2D planes formed by pairs of input attributes
2. For each plane
3. For every pair of classes
4. Generate clusters using agglomerative grid-based clustering approach
5. Compute opposed clusters and the boundary points that separate them
6. Tentatively split the node dataset using the computed boundary points and
7. Compute Gini gain
8. Select the overall best boundary to split the entire node dataset
9. Repeat step (1) through (8) for the newly generated nodes until some stopping criterion is met

End

Any grid-based method can be used to generate the clusters (line 4). However in this implementation we generate rectangular clusters. It is economical to store rectangular clusters as one only needs to store the two coordinates of the corner points on one of the diagonals. (This technique consumes less memory space than other approaches that store a

cluster by storing information about every cell in the cluster). PATHFINDER is different from previous approaches in many folds: (1) It does not work in the original input space; it reduces the complex task of working in the high dimensional input space to a more manageable set of 2D input space tasks by projecting the dataset onto planes formed by pairs of attributes. (2) It does not split the node datasets with hyperplanes but instead with pieces of hyperplanes (hyperplane segments). (3) It clusters the examples in the node dataset considering examples of two classes at a time (ignoring examples from the other classes).

The detail of the clustering computation (line 4) is provided in section 1. Section 2 describes how the opposed clusters and their boundary points are computed (line 5). Section 3 details the node evaluation method (line 6).

The notation used in the remainder of the chapter is summarized in Table 3.1.

Table 3.1 Notations used in the remainder of the chapter

O	Dataset
y_p	The p^{th} class label; Y =$\{y_1,..,y_q\}$
q	Number of classes in O
r	Number of attributes in O
Π	Set of all attribute pairs (also set of all planes) $\Pi = \{x_i, x_j\}_{i,j=1,..,d}$
$\Pi_{i,j}$	Plane formed by the i^{th} and j^{th} attributes
C_i	The i^{th} cluster
C_{i-j}	The j^{th} sub-cluster of C_i; $C_i = \bigcup_{j=1,..,m} (C_{i-j})$ for some m>0
θ	Cluster purity threshold
α	Minimum row or column purity threshold

$K_{i,j}(y_p,y_q)$	Clustering result for clustering examples of class y_p, against y_q in the plane $\Pi_{i,j}$; $K_{i,j}(y_p)$ set of clusters of dominant class y_p in plane $\Pi_{i,j}$
$K_{i,j}$	Set of all clustering in the plane $\Pi_{i,j}$
K	Set of all clustering for all planes. $K = \bigcup_{i,j} K_{i,j}$

3.3.1. Clustering

The input attributes are real-valued attributes. We let $\{a_i\}_{i=1,..,r}$ be the set of all r input real-valued attributes, and $\Pi = \{a_i, a_j\}_{i,j=1,..,r}$ be the set of all planes formed by two distinct input attributes (with $i \neq j$). The target variable is categorical with q classes and takes values in $Y = \{y_1,..,y_q\}$. Let $\psi = \{(y_i,y_j)\}_{i,j=1,..,q}$ be the set of all pairs of class labels (with $i \neq j$). The set of all clusters obtained from a given plane $\Pi_{i,j}$ is noted $K_{i,j}$ and the set of all clusters for all the planes is noted $K = \bigcup_{i,j} K_{i,j}$ (with $i \neq j$). The PATHFINDER clustering approach is provided in algorithm 2 and 2.1.

Algorithm 2: PATHFINDER Clustering Method

1. Inputs:
2. α; user-defined row/column minimum purity thresholds
3. θ; user-defined cluster minimum purity thresholds
4. O; dataset
5. Outputs:
6. K; array storing the set of all clusterings
7. BEGIN
8. index← 0
9. FOR each plane $\Pi_{i,j}$ in Π
10. Project dataset onto $\Pi_{i,j}$
11. Create grid of equal size cells

12. FOR each pair of class $(y_{p,}y_q)$ in ψ

13. K[index] ← *GenCluster*(y_p ,y_q ,O,α,θ, $\Pi_{i,j}$)

14. K[index+1] ← *GenCluster*(y_q , y_p ,O,α,θ, $\Pi_{i,j}$)

15. index←i+2

16. Return K

17. End

Function *GenCluster*(.) generates rectangular clusters in a grid. *GenCluster* is called twice for each pair of classes (y_p,y_q); once to cluster the examples of class y_p against the examples of class y_q; then a second time to cluster the examples of class y_q against the examples of class y_p. Examples of other classes are ignored during that process. The output K, contains all the generated clusters (for all the planes and all the classes).

Each entry to K is a tuple *($\Pi_{i,j}$, pivot_class, hlist*) where

$\Pi_{i,j}$ is a plane formed by the i^{th} and j^{th} attributes

pivot_class is the dominant class label of the clusters in the clustering result

hlist is the clustering result (a list of clusters C_i for the selected plane, and *pivot_class)* and their associated boundary points.

The detail of *GenCluster*(.) is provided by algorithm 2.1. In algorithm 2.1 a "Pivot cell" is a cell from which the algorithm starts growing a cluster. "A pivot class" is the class whose examples are being clustered. The first argument of GenCluster(.) is the "pivot class" label. The second argument, which is the class the pivot class is being clustered against, is referred to as the "non-pivot" class. The term "class purity", or simply "purity"

represents the number of examples associated with the pivot class divided by total number of examples in a cell (or a cluster). The algorithm adds to the cluster being grown, adjacent blocks of contiguous cells in a row or in a column referred to as "row" or "column" (respectively). The number of cells in a row/column being added increases as the algorithm makes anticlockwise/clockwise trips around the pivot cell. The purity of a cluster must be above a user-defined purity threshold θ. The purity of a pivot cell is greater than θ. The purity of a row/column must be above a user-defined purity threshold, α (with $\alpha <= \theta$). Doing so makes it possible to add some impurity to the clusters as long as overall purity (of the cluster) remains above θ.

Algorithm 2.1 : Function *GenCluster*(y_+ ,y_- ,O,α,θ, $\Pi_{p,q}$)

1. Inputs:
2. y_+ ,y_- ; The two classes being clustered against each other
3. $\Pi_{p,q}$; $x_p x_q$-plane, O; dataset
4. θ; cluster purity threshold, α; row or column purity
5. Outputs:
6. $K_{p,q}$; clustering result
7.
8. Begin
9. Initialization(.); set up rectangular cells
10. DO UNTIL no pivot cell is available
11. Select a pivot cell
12. currentCluster ← pivot cell
13. previousClusterSize ←0

14. WHILE (currentCluster.size ≠ previousClusterSize)
15. previousClusterSize ← currentCluster.size
16. IF SuccessfullMoveRight(.)=true THEN
17. IF purity(right_column) > α and ClusterPurity(.) > θ THEN
18. currentCluster ← currentCluster ∪ right_column
19. IF SuccessfulMoveUp(.)=true THEN
20. IF purity(row_up) > α and ClusterPurity(.) > θ THEN
21. currentCluster ← currentCluster ∪ row_up
22. IF SuccessfulMoveLeft(.) =true then
23. IF purity(left_column) > α and ClusterPurity(.)>θ THEN
24. currentCluster ← currentCluster ∪ left_column
25. IF SuccessfulMoveDown(.) =true THEN
26. IF purity(row_down) > α and ClusterPurity(.)>θ THEN
27. currentCluster ← currentCluster ∪ row_down
28. $K_{p,q} = K_{p,q} \cup \{currentCluster\}$
29. Return $K_{p,q}$
30. End

The algorithm computes high purity clusters with dominant class label y+ (pivot class). Class label y- is a non-pivot class against which class y+ is being clustered. Procedure Initialization(.) (line 9) set up rectangular grid, projects the dataset onto the grid, and computes the purity of each cell with respect to the pivot class. The function ClusterPurity(.) re-computes the purity of the cluster before deciding if the row/column should be added or not. currentCluster.size returns the size of the cluster being grown. Its initial

value is the size of the pivot cell. The algorithm selects a pivot cell then grows a cluster around the pivot cell by adding rows or/and columns until no more row/column can be added (line 14-27). The cells thus clustered are marked so that there are not considered in subsequent clustering. The algorithm then selects another pivot cell (line 11) not yet part of a cluster, to grow a new cluster. It returns a set of high purity clusters (for the pair (plane $\Pi_{p,q}$, pivot class)) . Procedure SuccessfulMoveRight(.) attempts to add cells from the right column to the cluster. If this is the last column on the grid (if it is the rightmost column), the attempt fails. Similarly SuccessfulMoveUp(.), SuccessfulMoveLeft(.), and SuccessfulMoveDown(.), respectively tests if there is a row up, a columns to the left, or a row down to add to current cluster. If a row or column is available, it is added only if its purity > α. A new cluster purity is computed (tentatively) after the row/column has been added. If cluster's purity is still greater than θ then it is accepted; otherwise the row/column is not added. The algorithm makes anti-clockwise (or clockwise) loops adding blocks of cells in rows or/and columns. If after a round trip, no block of cells has been added, the loop condition in lines 14 ensures that the algorithm exits from the WHILE loop and add the new cluster to the cluster set. It then selects a new pivot cell (line 11) not yet part of a cluster to grow a new cluster. Line 10 ensures that when all potential pivot cells have been used, the algorithm returns the cluster set. Adjacent clusters of same dominant class label are merged into larger cluster ($C_i=\cup_{j=1,..,m}(C_{i-j})$). When several adjacent clusters of same dominant class are merged we refer to their union as a *merged cluster* or simply a cluster. We refer to the individual component clusters (of a merged cluster) as *sub-clusters* C_{i-j} (with C_i being the cluster).

Pivot cells can randomly be selected or can be selected in some predefined order without significant impact on the final clustering result. Fig. 3.5 gives a detailed illustration of algorithm 2.1.

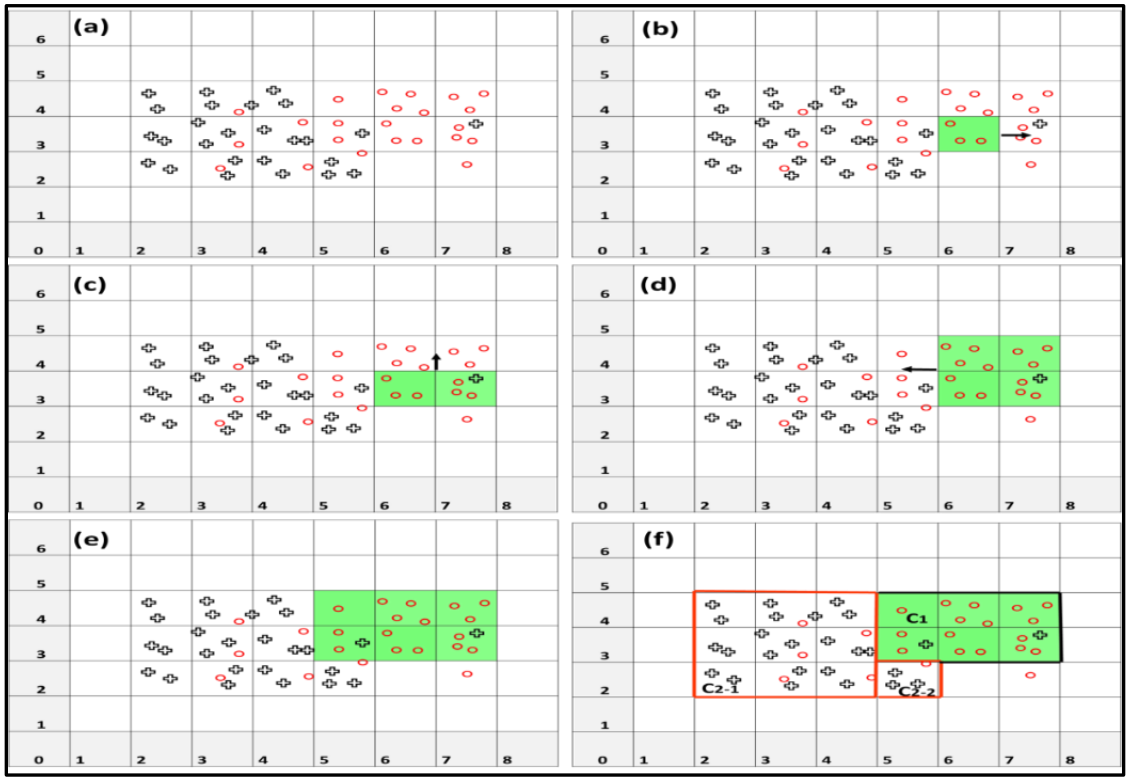


Fig. 3.5. PATHFINDER clustering approach for a hypothetical pivot class (circle) and non-pivot class (hallow cross)

In Fig. 3.5, the pivot class is the class which examples are represented by the circles. The cluster is being grown under the following condition: $\alpha = 70\%$; $\theta = 80\%$. The initially selected pivot cell is cell(6,3) with purity 100 %> 70% (Fig. 3.5.b) . From the selected pivot cell the algorithm attempts to grow the initial cluster by adding the one-cell column to the right of the pivot (cell(7,3)). Cell(7,3)'s purity equals 100*(3/4)%. Adding cell(7,3)

to cell(6,3) does not decrease the cluster purity below the threshold of 80%. Hence, the move is successful and the resulting new cluster is shown in Fig. 3.5.c. Next, a move up is contemplated. This time a two-cell block {cell(6,4), cell(7,4)} in row 4 is tentatively added. This attempt is successful because both conditions are met: the row purity $\alpha = 100$ %> 70% and overall cluster purity after the addition is $\theta = 100*(13/14) > 80\%$ (Fig. 3.5.d). A move to the left is successful because the column being added is the two-cells block {cell(5,3), cell(5,4)} which purity is $100*(4/5) > 70\%$ (Fig 5.e). Next, a move downward is attempted. At this point we must add a three-cell row block to the cluster. Since the purity of the row block {cell(5,2), cell(6,2),(7,2)} is $100*(2/5) < 70\%$ this attempt fails. The first anti-clockwise loop is thus completed. However, since during the round trip, at least one row or column block was added, a second round trip is necessary to break from the WHILE loop (line 14 of algorithm 2.1) and the process ends. The clustering result is shown in Fig. 3.5.e. Fig. 3.5.f illustrates the clustering result for both classes with cluster C_2 having two sub-clusters $C_{2\text{-}1}$ and $C_{2\text{-}2.}$

3.3.2. Boundary Points Computation

Two clusters are horizontally neighboring in a grid, if any part of their contours are facing each other in a row; this means if there is no other cluster between them in that row. Similarly two clusters are vertically neighboring if any part of their contour is facing each other in a column. Horizontal or vertical neighboring clusters are said to be direct neighbors. When a cluster has no direct neighbor, an oblique neighbor is searched for. Central to the PATHFINDER approach is the fact that boundary points are assigned to the clusters they separate.

The list of the boundary points for a cluster C_i ((C_i=$\cup_{j=1,..,m}(C_{i-j})$) is noted Lpoint(C_i) and is the union of all the points of individual sub-clusters C_{i-j} (LPoint(C_i) = $\cup_{j=1,..,m}(Lpoint(C_{i-j}))$). A point separating two direct neighbors is assigned to both clusters. Points separating oblique clusters are assigned as follows. A point is assigned to its closest cluster, and points sitting midway between the two clusters are assigned to both. The following sections provide further detail on the boundary point computation and assignment method.

Boundary Points Determination for Direct Neighbors

Given two clusters C_i and C_j in a grid. A boundary point in row r, separating two horizontal neighbors C_i and C_j is the midpoint between both clusters in row r. Likewise, a boundary point in a column say, c, is the midpoint between both clusters in column c.

Boundary Points Determination for Oblique Neighbors

Let C_j be a cluster and C_i its oblique neighbor, say to the northwest. Let C_i and C_j be denoted by their respective coordinates ($x_{1\_i\_low}$, $x_{2\_i\_low}$), ($x_{1\_i\_high}$, $x_{2\_i\_high}$) and ($x_{1\_j\_low}$, $x_{2\_j\_low}$), ($x_{1\_j\_high}$, $x_{2\_j\_high}$), three oblique boundary points $P_1(b_{1x1},b_{1x2})$, $P_2(b_{2x1},b_{2x2})$, and $P_3(b_{3x1},b_{3x2})$ are used to separate the two clusters. They are computed as follows

$b_{1x1} = (x_{1\_i\_high} + x_{1\_j\_low})/2$; $b_{1x2} = (x_{2\_i\_low} + x_{2\_j\_high})/2$

$b_{2x1} = x_{1\_i\_high}$; $b_{2x2} = (x_{2\_i\_low} + x_{2\_j\_high})/2$

$b_{3x1} = (x_{1\_j\_low})$; $b_{3x2} = (x_{2\_i\_high} + x_{2\_j\_high})/2$

The assignment of the boundary points to both clusters is done as follows. The middle point, P_1 is shared by both clusters. P_2 is a boundary point associated with C_i; P_3 is associated to C_j. (LPoint(C_i)← LPoint(C_i) ∪ {P_1, P_2}; and LPoint(C_j)← LPoint(C_j) ∪ {P_1, P_3}).

To illustrate let us consider Fig. 3.6. The boundary points P_{11}, P_{10} and P_9 separate the oblique clusters C_1 and C_4. P_{11} is associated to C_1, P_9 is associated to C_4 whereas P_{10} is shared by both clusters.

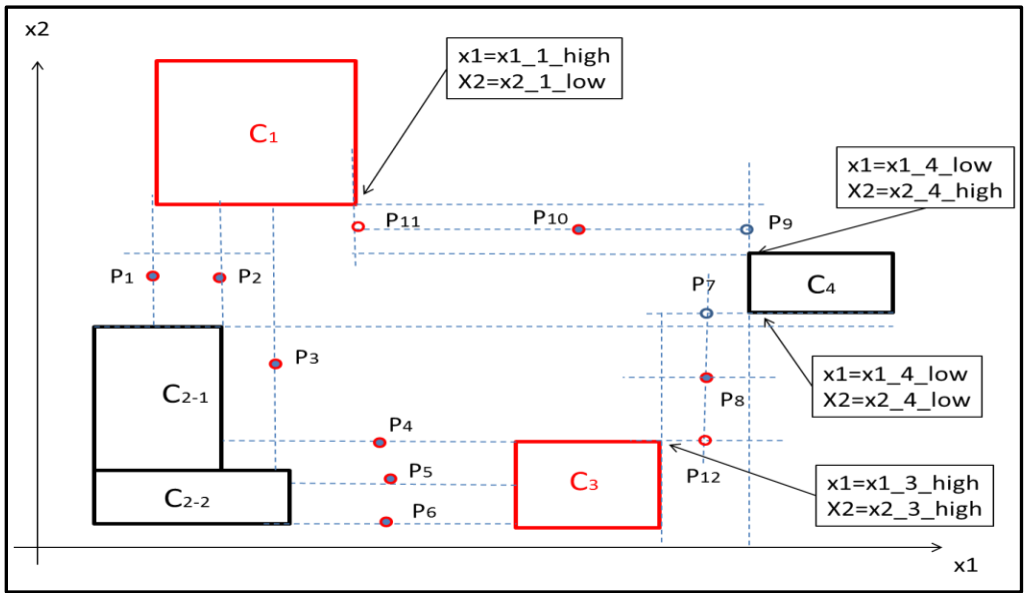


Fig. 3.6. Illustrating the computation of boundary points

In Fig. 3.6, cluster C_2 and C_3 are horizontally neighboring; C_2 and C_1 are vertically neighboring. The points P_4, P_5 and P_6 are shared boundary points for both cluster C_2 and C_3. The points P_1, P_2 and P_3 are shared boundary points for clusters C_2 and C_1. The points P_7, P_8 and P_{12} are boundary points separating the oblique clusters C_3 and C_4 with P_8 being

their shared point. Thus Lpoint($C_{2\text{-}2}$) = {P_5, P_6}, Lpoint($C_{2\text{-}1}$)={P_1,P_2,P_3,P_4}, Lpoint(C_1) = {P_1, P_2, P_{11}, P_{10}}, Lpoint(C_4) = {P_9, P_{10}, P_7, P_8} and Lpoint(C_3) = {P_4, P_5, P_6, P_8 , P_{12}}.

3.3.3. Node Split Method

The key idea to PATHFINDER node evaluation is to use the clustering result, K, obtained from algorithm 2 to split the node dataset. Each entry K[i], of K, ($i \geq 0$) contains a tuple *(plane, pivot_class, hlist*) where *hlist* is an array containing the clustering result for the pair (*plane, pivot_class)*. For each tuple in K, each example in the node dataset is tested against the clusters in *hlist* to determine if the example resides within the boundary of a cluster or not. The node dataset is thus tentatively split into two sub-nodes with one sub-node containing the examples that reside inside the boundary of a cluster and the other hosting the examples that do not pertain to any cluster in *hlist* (for the selected *plane, pivot_class*). Gini gain is computed to select the best split of all tuples in K. Given an example s, a clustering result in *hlist*, many strategies can be used to test if s lies within the boundary of a cluster in *hlist*. We propose a method, detailed in algorithm 3 that uses the two nearest neighbor boundary points of s, to compute a local boundary separating s and its closest cluster.

Algorithm 3: PATHFINDER Node Splitting Method

1. Inputs: O; node dataset
2. K; list of tuples *(plane, pivot_class, hlist*)
3. Outputs: split node dataset O, into two sub-nodes
4. Begin
5. FOR each tuple *t* in K

6. FOR each s, in O

7. Compute the closest boundary point $P_{closest}$ to s

8. Retrieve the sub-cluster $C_{closest}$ associated with $P_{closest}$

9. Compute $P_{closest2}$, the second nearest neighbor of s in LPoint($C_{closest}$)

10. Compute the equation of boundary line ($P_{closest}$, $P_{closest2}$)

11. IF s and centroid($C_{closest}$) lie on the same side of line ($P_{closest}$, $P_{closest2}$) THEN

12. Assign s to left node

13. ELSE

14. Assign s to right node

15. Compute Gini gain

16. Select the tuple *(plane, pivot_class, hlist)*$_{best}$ that maximizes Gini gain for the split

17. Split node in two nodes with all examples within a cluster in *hlist* in left node and the rest in right node

18. End

The algorithm first finds the closest boundary points to s, $P_{closest}$ (line 7), and its associated sub-cluster $C_{closest}$ (line 8). It then computes the second nearest neighbor of s in Lpoint($C_{closest}$) (line 9). Using line ($P_{closest,}$ $P_{closest2}$) as a local boundary it determines if s and the sub-cluster's centroid, reside on the side of the local boundary or not (line 11). If s and the cluster's centroid reside on the same side of the boundary then s is assigned to the left node; otherwise it is assigned to the right node. The tuple *(plane, pivot_class_label, hlist*) that maximizes Gini gain is selected to split the node dataset. The tuple is stored in the node.

To classify an unlabeled example s, PATHFINDER starts from the root node and tests s against each stored tuple to find out if s lies within a boundary of a cluster in *hlist*. If s lies within the boundary of a cluster in a leaf-node, then s is assigned the class label of the dominant class of the leaf-node cluster.

Fig. 3.7 illustrates a node test for the dataset in Fig. 3.6. In Fig. 3.7 three examples s1, s2, and s3 are being evaluated by PATHFINDER. It is assumed that *hlist* contains {C_2, C_4} (C_2 and C_3 are assumed to be of the same dominant class label). The closest boundary point to s_1 is P_3. Hence, the selected cluster $C_{closest}=C_{2\text{-}1}$. The closest boundary point to P_3 in Lpoint($C_{2\text{-}1}$) is P_4. Since s_1 and cluster $C_{2\text{-}1}$ reside on the same side of line (P_3,P_4), s_1 is assigned to the left sub-node. In the same manner, s_2 nearest neighbor is assumed be P_8 which belongs to C_4. The second nearest neighbor of s_2 in Lpoint(C_4) is P_7 and s_2 is assigned to the right node because it lies on the opposite side of the boundary as the C_4.

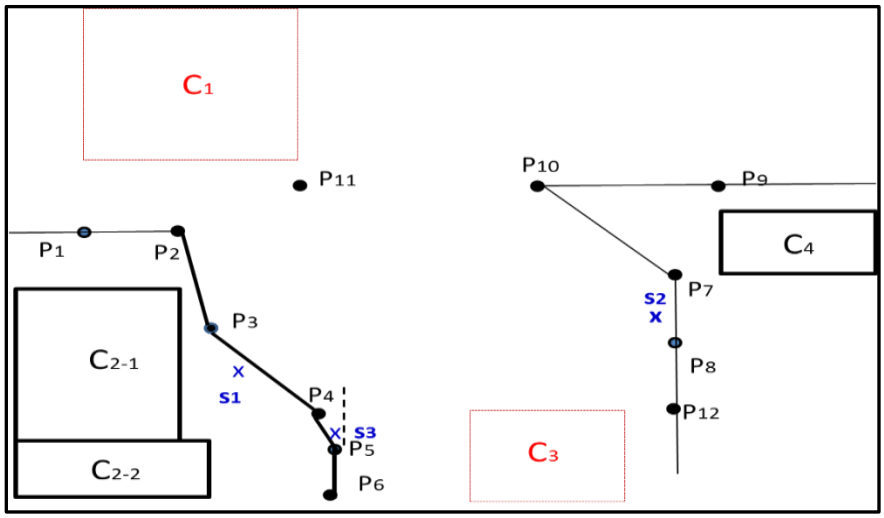


Fig. 3.7. A merged cluster and its boundary points

PATHFINDER splitting method is not perfect. For example s_3 closest boundary point is P_5. P_5 belongs to $C_{2\text{-}2}$. The second closest point to s_3, in Lpoint($C_{2\text{-}2}$) is P_6. Thus s_3 is assigned to left node though it is clearly outside cluster C_2 local boundary.

3.3.4. Runtime Complexity

We let n be the node dataset size, q the number of classes in the dataset, r the number of input attributes. We have r(r-1)/2 planes. In each plane the clusters are formed two classes at a time leading to 2q(q-1)/2 steps. To compute the clusters the proportion of examples in each grid cell must be computed. Hence, each example must be visited at least once. It cost n[(r(r-1)/2]*[q(q-1))] to compute all the clusters for all the planes in a node. It cost O(1) to test if an example lies on same or opposite side of the boundary as a cluster. To compute the nearest neighbor boundary point of an example, all the boundary points for all the clusters must be tested; which cost O(knr(r-1)q(q-1)/2) where k is the average number of boundary points per cluster (Finding the nearest neighbor of an example is the largest cost during a node split operation). Thus the cost for a node split operation is $O(knr^2q^2)$ where k, the average number of boundary points is dependent on the grid size.

3.4. Experimental Evaluation

In this section, an experimental evaluation of PATHFINDER is provided for 11 datasets. The result is compared against those found in [1] on 7 real world datasets. Performance is evaluated in terms of tree leaf-nodes and model accuracy.

3.4.1. Datasets

Four artificial and 7 real-world datasets with 6 previously used as benchmarks were used in the experiments. Artificial datasets are generated by functions provided in Table 3.2. The datasets size, number of classes, and number of input attributes are provided in Table 3.3. All datasets were normalized.

Table 3.2. Artificial datasets

2DSin	$x_1 \in]0,1[$; $x_2 \in [-3,3]$ where x_1, x_2 are randomly generated real numbers. $d = x_2 - \sin(x_1 * 2\Pi)$ if $d > 0.3$ then example $s(x_1,x_2)$ is labeled "C+" if $d < -0.3$ then $s(x_1,x_2)$ is labeled "C-"

4Patches	$x_1 \in]0,1[$; $x_1, x_2 \in R$
	if($x_1 > 0.1$ and $x_2 > 0.1$) then "C-"
	if ($x_1 < -0.1$ and $x_2 > 0.1$) then "C+"
	if ($x_1 < -0.1$ and $x_2 < -0.1$) then "C-"
	if($x1 > 0.1$ and $x_2 < -0.1$) then "C+"

6Patches	d = 0.1; $x_1 \in]-3,3[$ $x_1, x_2 \in R$
	if (x2 > x1 + 0.2)
	if ((x1 > d and x2 > d)) then "C-";
	else if (x1 < -d and x2 > d) then "C+";
	else if (x1 < -d and x2 < -d) then "C-";
	if (x2 < x1 - 0.2)
	if (x1 > d and x2 > d) then "C+";
	else if (x1 > d and x2 < - d) then "C-";
	else if (x1 < - d and x2 < - d) then "C+";

2DCircles	$x_1, x_2 \in]-5,5[$; $d \in R$ if $((x_1^2 + x_2^2)^{1/2} > 3.5)$ then "C+" if $((x_1^2 + x_2^2)^{1/2} < 3)$ and $(x_1^2 + x_2^2)^{1/2} > 1.5)$ then C- if $((x_1^2 + x_2^2)^{1/2} < 1.2)$ then C+

Table 3.3. Real world datasets [10]

	Dataset code	No. of classes	# of records	No. input of attributes
Real World Datasets				
Iris	Irs	3	150	4
Bupa Liver Disorder	Bld	2	345	6
Boston Housing	Bos	3	506	12
Wisconsin Breast Cancer	Bcw	2	683	9
PIMA Indian Diabetes	Pid	2	532	7
*Silhouette	Veh	4	846	18

Image Segmentation	Seg	7	2310	19
Artificial Datasets				
4Patches	4ps	2	1082	2
6Patches	6ps	2	1118	2
2DSin	2ds	2	834	2
2DCircles	2dc	2	1670	2

All real world datasets were obtained from UCI repository [10]. In [1] the Silhouette dataset size was 3772 records whereas the UCI Silhouette dataset used in this experiment has 846 records.

3.4.2. Experimental Setup and Methodology

All the experiments were done with a 10-fold cross validation and repeated 10 times with different seeds for each run. Accuracy and reject result are reported in Table 3.6 and Table 3.7 as $\mu \pm c$ where μ is the mean value of the rate, and c the standard deviation. In this experiment, the input purity threshold parameters were set to α =100, and θ =100 for all datasets so as to maximize accuracy rate. The stopping parameters were set to *minimum data size* = 5% and *minimum Gini Improvement* =1% for all datasets. The grid size was adjusted for each dataset as shown in Table 3.4.

Table 3.4. Grid sizes

Datasets	Number Of Cell Per Row /Column
Iris (Irs)	16
Bupa Liver Disorder (Bld)	28
Boston Housing (Bos)	20
Wisconsin Breast Cancer (Bcw)	10
PIMA Indian Diabetes (Pid)	24
Silhouette (Veh)	16
Image Segmentation (Seg)	16

4Patches (4ps)	20
6Patches (6ps)	20
2DSin (2ds)	12
2DCircles (2dc)	18

Grid size selection

To select an optimum grid size we use a semi-automatic method consisting of running the algorithm on training set for increasing values of grid size (starting from 2 with 1 increment). The grid size that yields minimum error rate is selected.

For each of the six benchmark datasets, we compare the PATHFINDER accuracy rate against the best result of twenty two widely used approaches obtained from [1]. The twenty two approaches are listed in Table 3.5.

Table 3.5. Twenty two approaches [1]

Abbrev	Algorithm	Abbrev	Algorithm
QU0	QUEST Versions (Loh,Shih 1997)	IC0	CART versions (Breiman,Friedman,Olsen,Stone
QU1		IC1	
QL0		0CU	OC1 versions (Murthy, Kassif,Salzberg 1994)
QL1		0CL	
FTU	FACT Versions (Loh,Vanichsetakul 1998)	0CM	
FTL		ST0	S-PLUS versions (Clark,Pregibon 1993)
C4T	C4.5 (Quinlan1993)	ST1	
C4R		IB	IND versions (Buntine 1992)
LMT	LMDT (Brodley,Utgoff 1995)	IBO	
CAL	CAL5 (Müller,Wysotzki 1997)	IM	
T1	T1 single split (Holte 1993)	IM0	

3.4.3. Experimental Results

Error rate and reject rate were computed as follows. If we let n be the test sample size

Error rate= 100* number of misclassified examples/n

Reject rate= 100* number of rejected examples/n

Accuracy Rate=100 - error rate

Reported error and reject rates were obtained using 10-fold-cross validation.

We note that because our approach provides a reject option, it is not directly comparable to existing approaches. However, to highlight the benefit of PATHFINDER we compare its result to those obtained with CART on artificial dataset and the best results obtain using twenty two other classifiers.

The result shows that PATHFINDER yields consistently high accuracy rate on all datasets. This result is to be expected since PATHFINDER does not classify examples that are difficult to classify.

3.4.3.1. Results on Artificial Datasets

The result is summarized in Table 3.6. We use the result of CART as a baseline comparison. Trees obtained by PATHFINDER are illustrated in Fig. 3.8.

Table 3.6. Accuracy and complexity results on artificial datasets

	PATHFINDER			SIMPLE CART	
	Accuracy Rate%	Reject Rate (%)	Complexity	Accuracy Rate	Complexity
4Ps	100 ±0.00	0.00 ±00	2.00±00	99.72	8
6Ps	98.56 ± 1.14	2.32 ± 2.17	2.20 ±0.42	99.01	11
2Ds	100.00 ± 0.00	0.00± 0.00	2.00 ±0.00	98.8	7
2Dc	99.46 ± 0.66	6.14 ± 3.07	2.40 ± 0.70	98.56	15

On the selected artificial datasets PATHFINDER consistently yields high accuracy rate and low complexity result at the same time. On two datasets, it yields 100% accuracy rate with no reject.

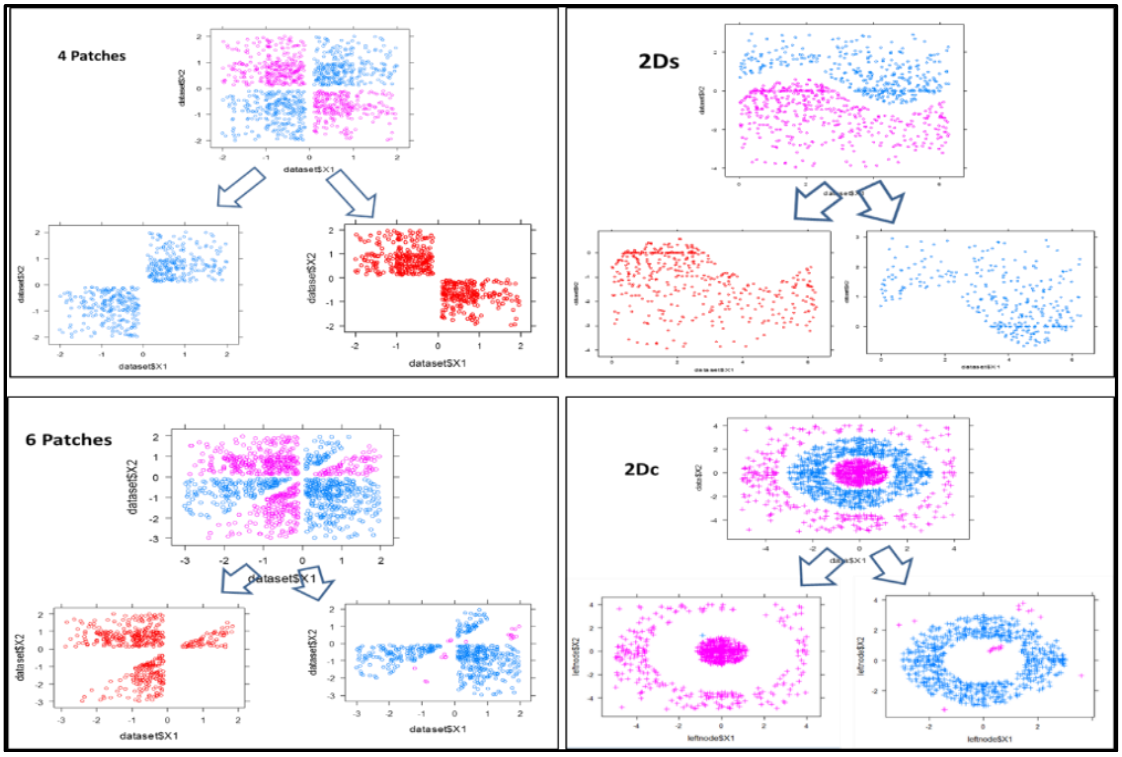


Fig. 3.8. Illustrating PATHFINDER ability to isolate high-purity clusters in a node in a single step while yielding high-accuracy rate

3.4.3.2. Results on Real World Datasets

Accuracy Results

Table 3.7. Accuracy results

	PATHFINDER		Twenty two Approaches	SIMPLE CART
	Accuracy Rate%	Reject Rate (%)	Best Accuracy Rate	Accuracy Rate
Bld	81.63 ± 3.73	21.63 ± 24.68	72	64.05
Bos	82.65 ± 3.63	27.96 ± 7.50	78	73.12

Bcw	96.12 ± 3.00	8.36 ± 4.78	97	94.87
Pid	77.50 ± 4.32	21.45 ± 4.89	78	75
Veh	82.32± 8.07	35.37 ± 14.82	85	69.03
Seg	95.97 ± 3.29	14.76 ± 7.23	98	96
Irs	100.00 ± 0.00	17.33± 6.44	N.A	95.33

PATHFINDER yields higher accuracy rate than the best result obtained from the twenty two algorithms on two datasets (Bld and Bos), ties on two dataset (Bcw,Pid), and under-performs on 2 datasets (Veh, Seg). Depending on the dataset, a small increase in reject rate may yield a high improvement in accuracy rate. For example, on the Bld dataset we have 14% increase in accuracy rate for the price of 21.63% reject rate while on the Bos dataset we obtained 6% increase for the price of 28% reject rate.

Reject rate is influenced by the grid size or purity threshold. In general when purity parameters are low, reject rate and accuracy rate decrease but not linearly. Therefore, for a given dataset, a user may adjust those parameters to find a trade- off that best suits his purposes. Another advantage of PATHFINDER is that by identifying the reject examples (examples difficult to classify) a domain expert may be prompted for further analysis (or lab tests).

Complexity Results

Model complexity was obtained with 10-fold cross validation and was measured by the number of leaf-nodes. Fig. 3.9 shows the tree complexity of PATHFINDER in compari-

son with previously known best results obtained from [9]. The result shows that PATHFINDER consistently has low complexity.

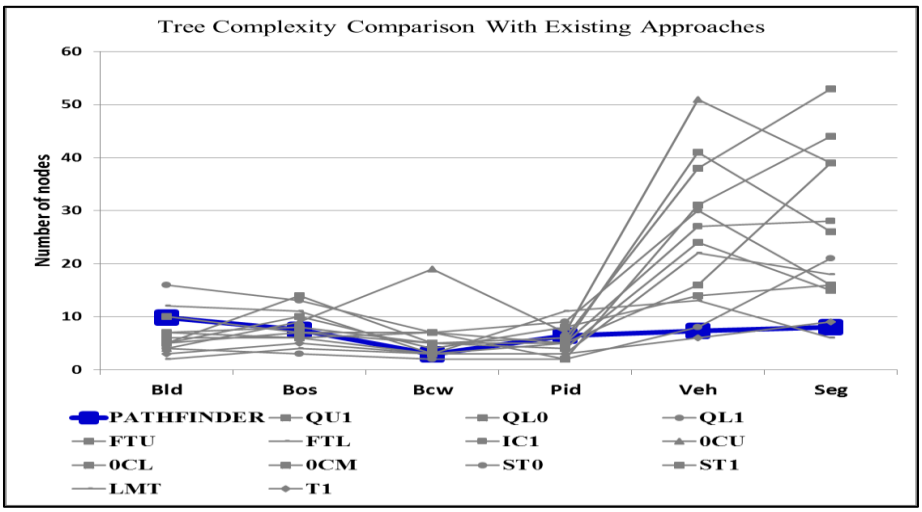


Fig. 3.9. Complexity results

3.5. Conclusion

PATHFINDER addresses several critical issues related to multivariate classification tree induction algorithms. First, a novel grid-based clustering algorithm is introduced which computes rectangular clusters, whereas past research relies on prototype-based clustering algorithms to identify opposed clusters. We claim that the grid-based approach has advantages over the prototype-based clustering approach, as the number of clusters does not need to be specified in advance and that our approach is not sensitive to outliers. Second, we introduce a new methodology to create piece-wise linear decision boundaries to separate regions which are dominated by instances of different classes. In contrast to previous work, our approach stores a set of boundary points in a node, and not the boundaries

themselves, and uses a novel node evaluation technique which computes local boundaries at runtime by connecting pairs of the boundary points between opposed clusters. Moreover, computed decision boundaries might differ based on the location of the example to be classified which makes our approach a mixture between a nearest neighbor approach and a decision tree approach. Third, it is capable of fitting several clusters into a single leaf-node and able to keep most clusters in the dataset un-split. Consequently, PATHFINDER induces very short trees compared to other widely known approaches. Fourth, it provides a reject option to identify examples that are difficult to classify and therefore induces trees with very high accuracy at a cost of some reject rate. We claim that such a classifier can be particularly useful in areas, such as computer-aided medical diagnostics where the cost of misclassification is often prohibitively high.

References

1. M. Fatih Amasyali and Okan Ersoy; "*Cline: A New Decision-Tree Family"*; IEEE Transaction on Neural Networks, Vol. 19, No. 2, February 2008.

2. K. Bennett; *"Decision tree construction via linear programming"* ; Proceeding of 4th Midwest Artificial Intelligence Cognitive Science Soc. Conf., Utica, IL, 1992, pp. 97–101.

3. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone; *"Classification and Regression Trees"*; Belmont, CA: Wadsworth, 1984.

4. B. A. Draper, C. E. Brodley, and P. E. Utgoff; *"Goal-directed classification using linear machine decision trees"*; IEEE Trans. Pattern Anal. Machine Intelli. vol. 16, pp. 888–893, Sept. 1994.

5. H. Kim and W.-Y. Loh; *"Classification trees with unbiased multiway splits"*; J. Amer. Statistical Assoc., vol. 96, pp. 589–604, 2001.

6. W.-Y. Loh and Y.-S. Shih; *"Split selection methods for classification trees"*; Statist. Sinica, vol. 7, pp. 815–840, 1997.

7. W.-Y. Loh and N. Vanichsetakul; *"Tree-structured classification via generalized discriminant analysis"*; J. Amer. Statist. Assoc., vol. 83, no. 403, pp. 715–728, 1988.

8. J.R. Quinlan; *"Simplifying decision trees"*; Int. J. Man-Mach. Stud., vol. 27, pp. 221–234, 1987.

9. Tjen-Sien Lim, Wei-Yin Loh; *"A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms"*; Machine Learning, 40, 203-229, 2000 Kluwer Academic Publishers, Boston.

10. UCI Repository of Machine Learning Databases; Department of Information and Computer Science, University of California, Irvine, CA. http://archive.ics.uci.edu/ml/ (2014).

11. Cantú-Paz, E. and Kamath C. (2000a); *"Using Evolutionary Algorithms to Induce Oblique Decision Trees"* ; Genetic and Evolutionary Computation Conf. (GECCO) 2000, Las Vegas, NV, July 8-12, 2000.

12. Cantú-Paz, E and Kamath, C. (2003); *"Inducing oblique decision trees with evolutionary algorithms"*; IEEE Transactions on Evolutionary Computation. 7(1), 54-68.

13. Murthy, S. K., Kasif, S., & Salzberg, S. (1994); *"A system for induction of oblique decision trees"*; Journal of Artificial Intelligence Research, 2(1), 1-32.

14. Jack Sklansky and Leo Michelotti; *"Locally trained piecewise linear classifiers"*; IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI-2(2):101--111, March 1980.

15. Park Youngtae and Jack Sklansky; *"Automated design of multiple-class piecewise linear classifiers"*; Journal of Classification December 1989, Volume 6, Issue 1, pp 195-222.

Chapter 4

AVALANCHE: A Hierarchical, Divisive Clustering Algorithm[2]

4.1. Introduction

This research proposes a divisive approach to hierarchical clustering for applications that use as input a dissimilarity matrix. To the best of our knowledge such a method has not yet been proposed in literature. Clustering is the unsupervised grouping of examples into clusters. The clustering problem has received a lot of attention by researchers in many disciplines. Hierarchical clustering algorithms aim at organizing datasets hierarchically as dendrograms based on the distances of examples and clusters. Hierarchical clustering approaches are either agglomerative or divisive. In the general case, agglomerative (bottom-up) approaches start by merging the most similar examples/clusters and continue until the last two clusters have been merged. Divisive approaches — which are less popular —, on the other hand, start with a cluster containing all the examples of the dataset, and recursively split the dataset until a termination condition has been met. Both agglomerative and divisive clustering approaches are greedy algorithms, making decisions based on local patterns or based on global objective functions. The ability of the top-down approaches to use global information about the dataset to select splits is often viewed as a potential advantage [16]. On the other hand, the same publication claims that bottom-up approaches are perceived to produce better clustering result than the top-down methods

[2] Published in Proceeding of International Conference on Machine Learning and Data Mining (MLDM), Hamburg, Germany July, 2015.

but usually run significantly slower. The ability of the top-down methods to stop growing the tree when a predefined termination condition is met has made them very popular in applications such as document searching/indexing, web query. However, thus far, most of the divisive approaches involve the computation of a centroid (K-mean like or Principal Direction Divisive Partitioning (PDDP) based approaches) which restricts their use to flat file datasets with numerical attributes. In applications that use dissimilarity matrices as inputs, centroids cannot be computed due to the lack of numerical attributes; consequently, novel divisive methods are needed for such datasets and introducing such methods is the main focus of this chapter.

This research proposes AVALANCHE, a novel top-down hierarchical clustering algorithm which uses as input a dissimilarity matrix. The problem is to come up with a test that splits a set into two subsets, maximizing an underlying objective function. During the top-down process a node is split into two sub-nodes such that the distance between the two sub-node clusters is maximized, and the sum of the "intra-cluster distances" of both clusters is minimized. To split a node, initially the example that is furthest away from the other examples — the *anti-medoid* — is assigned to the right sub-node and then — using the one Nearest Neighbor Chain approach (1-NNC) — additional examples are progressively assigned to this node which are nearest neighbors of the previously added example as long as the objective function improves. Given a set of objects, the *medoid* is the centermost object (the one with the shortest total distance to the other objects in the set); the *anti-medoid* on the other hand, is the outermost object (the object with the largest total distance to the other objects in the set).The heuristic used by AVALANCHE to find

a solution for the set split is a two-step method based on the One Nearest Neighbor Chain approach (1-NNC). First, it selects the anti-medoid to start the node split and assigns it to the right node. Next, it follows the 1-NN chain that originates from the anti-medoid to add further examples to the right node until doing so degrades the objective function.

The general idea of the AVALANCHE algorithm is illustrated in Fig. 4.1. Fig. 4.1.a displays an initial dataset where s1 is the anti-medoid since s1 contribution to total intra-cluster distance is 24 (largest value). Next, the nearest neighbor of s1, s2, is selected as candidate for assignment. The objective function which will be explained in detail in section 3 is computed to determine if adding s2 to the right node improves the inter-cluster distance between the two sets. The answer is "yes" and s2 is added. At this point the right node dataset is {s1, s2} and the left node dataset is {s3, s4, s5}. Fig. 4.1.b shows the first node split. Since 1-NN(s2)=s3 the algorithm attempts to add s3 to the right node. However, adding s3 to set {s1, s2} will decrease total inter-cluster distance and increase total intra-cluster distance. Therefore the node split stops and the obtained clusters for this step are: {s1, s2} and {s3, s4, s5}. The tree construction continues in Fig. 4.1.c where the sub-node {s3, s4, s5} is further split.

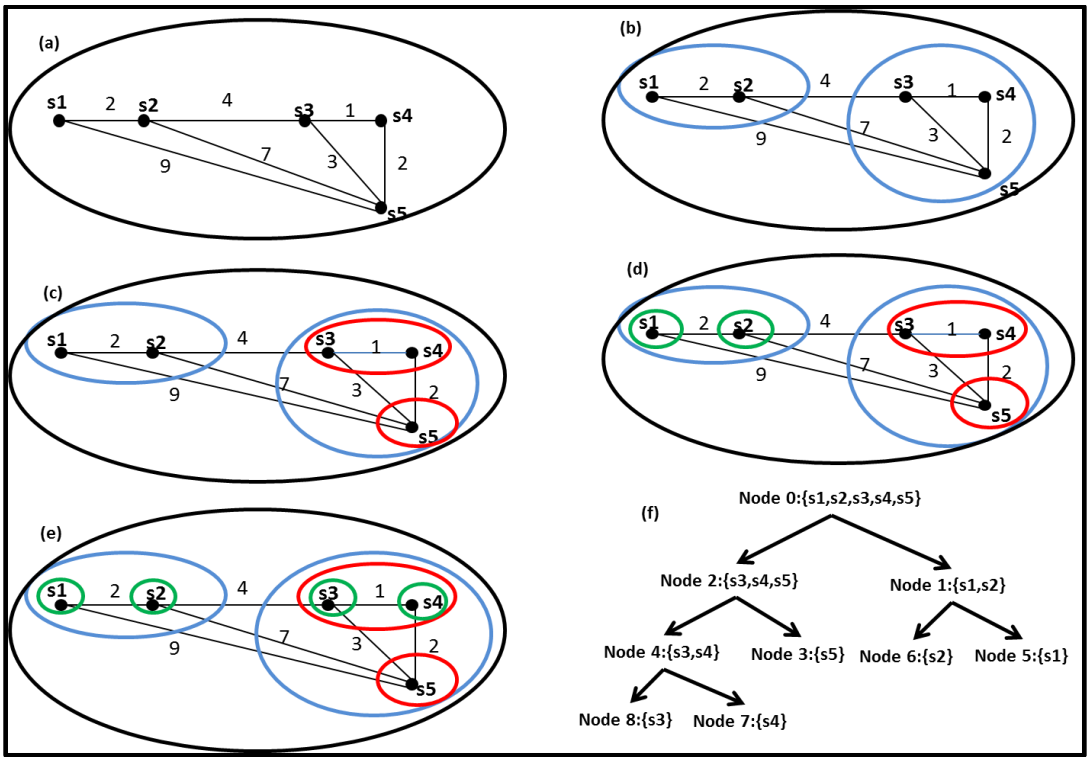


Fig. 4.1. AVALANCHE splitting process starts from the outermost example toward the center of the cluster

In Fig. 4.1.c s5 is the anti-medoid. 1-NN(s5)=s4 however, adding s4 to right node would increase total intra-cluster distance and decrease inter-cluster distance, which would decrease the objective function. Therefore the split stops and we obtain {s3, s4} and {s5}. Fig. 4.1.d and 4.1.c show that the two-object clusters are further split. The final tree is shown in Fig. 4.1.f. More details of this approach are provided in section 3.

The main contributions of this chapter include:

1. The introduction of a new unsupervised approach to divisive clustering which uses both global and local information in its splitting decision. Most divisive approaches make splitting decision based on global statistics information about the dataset (vari-

ance, sum of square errors). Likewise, most agglomerative approaches make merging decision based on proximity of pair of examples which is perceived to be local information. AVALANCHE splitting decision is based on both concepts since it computes the anti-medoid from the dataset (global) then uses 1-NNC approach to merge neighboring examples in the proximity of the anti-medoid (local).

2. A new unsupervised approach to top-down hierarchical clustering which takes as input a dissimilarity matrix. Divisive algorithms that have been proposed in the literature so far cannot be applied to datasets where examples do not have any attributes.
3. The introduction of a novel objective function which considers the inter-cluster distance in addition to the intra-cluster distance when evaluating different clustering solutions.
4. Incremental methods are proposed which save time in the objective functions computations.
5. The algorithm runs faster than its main competitor while maintaining comparable or better clustering results!

The rest of the chapter is organized as follows. In section 4.2 we present related work. Section 4.3 details the proposed algorithm. Finally, in section 4.4 we discuss results of experimental evaluation.

4.2. Related Work

Hierarchical clustering approaches are either agglomerative or divisive. In order to decide which clusters should be combined/divided they require some evaluation measure. Divisive approaches rely heavily on Ward's minimum variance method [19] which states that the cluster with the largest Sum of Square Error should be chosen for the split. The agglomerative approaches, on the other hand, rely on dissimilarity among the clusters/examples to make their merging decision. An important choice required in agglomerative hierarchical clustering is how to measure the distance between the clusters. Commonly used distances — often known as "linkages" — include complete link, single link, average link, centroid link [9, 8, 2, 17, 5]. Given two clusters C_i, and C_j, and x_{ip}, x_{jq} examples in cluster C_i, and C_j respectively; the different distance measures are defined as follows:

- Single link: $d(C_i, C_j) = \min\{d(x_{ip}, x_{jq})\}$ which is the smallest distance between an element in one cluster and an element in the other
- Complete link: $d(C_i, C_j) = \max\{d(x_{ip}, x_{jq})\}$ which is the largest distance between an element in one cluster and an element in another cluster
- Average link: $d(C_i, C_j) = \text{avg}\{d(x_{ip}, x_{jq})\}$ which is the average distance between elements in one cluster and elements in the other
- Centroids/medoids link: $\text{cen}\{d(x_{ip}, x_{jq})\}$ or $\text{med}\{d(x_{ip}, x_{jq})\}$ which is the distance between two centroids/medoids of the two clusters. A centroid is the central point of a cluster (mean point), while a medoid is an actual example which has the smallest largest distance to any point in the cluster.

The most popular agglomerative approach is the average link approach also known as the **U**n-weighted **P**airs **G**roup **M**ethod with **A**rithmetic Mean (UPGMA) algorithm. It takes as input a distance matrix. When UPGMA is used the distance between two clusters C_1, and C_2 is defined as

$$d(C_1, C_2) = \frac{1}{|C_1||C_2|}\sum_{x\in C_1}\sum_{y\in C_2} d(x, y)$$

where $|C_1|$ and $|C_2|$ denote the number of examples in cluster C_1, and C_2 respectively. First, it locates the two closest examples/clusters in the matrix, and merges them into a cluster, say C_k; that is $C_k=C_1 \cup C_2$. The rows and columns occupied by the examples/clusters thus merged are removed from the matrix and a new row and column for C_k are entered. For any cluster C_i the distance between C_i and C_k is computed as

$$d(C_i, C_k) = \frac{|C_1|}{|C_1|+|C_2|} d(C_1, C_i) + \frac{|C_2|}{|C_1|+|C_2|} d(C_2, C_i)$$

Divisive hierarchical clustering is a top-down approach which starts with the root node having all the data associated with it, and the approach recursively splits it into two node/set pairs until sets containing one example are obtained or it terminates earlier if a predefined termination condition is met. The most popular approach is the "Bisecting K-mean" approach [6, 7,10,11,14, 16]. To split a node, it first computes the centroid of the dataset. Next, it iteratively computes the centroids K_1, and K_2 of two regions of the dataset using K-mean (K=2). Then it bisects the data with a hyper-plane passing through the centroid and perpendicular to segment K_1K_2. The process continues until a termination condition is reached. Another divisive approach is the Principal Direction Divisive Partitioning (PDDP) algorithm [3]. It computes the singular value decompositions (SVD) and

derives the principal direction then divides the dataset with a hyper-plane passing through the origin and perpendicular to the principal direction vector. The principal direction of the data is its direction of maximum variance [15]. Recently a hybrid method has been proposed [12]. It uses both agglomerative and divisive hierarchical clustering algorithms to get the best of both approaches. In a first step, the algorithm uses the Bisect K-means approach to cluster the dataset in K' clusters and then uses UPGMA on the centroids of the computed K' clusters. If two centroids end up in same cluster then all of their examples are merged in one cluster. This approach is different from our proposed method in that it applies sequentially the top-down method then the bottom-up method, whereas our approach incorporates bottom-up merging criteria into its top-down splitting decision. One of the main drawbacks of both the Bisect K-mean and PDDP is that they are sensitive to outliers.

Divisive algorithms proposed thus far are unsuitable for clustering datasets where the only input is a distance matrix because they rely on centroids computation which is not possible with distance matrix. For those applications we propose AVALANCHE, a top-down hierarchical algorithm that does not use centroid in its splitting method. The detail of the AVALANCHE algorithm is provided in section 3.

4.3. The AVALANCHE Approach

The general idea of the AVALANCHE approach is that at each intermediate node the algorithm tries to split the node into two child-nodes (binary split) such that the distance between the clusters associated with the child nodes is maximum and the sum of the

"within cluster" distances is minimum. Table 4.1 summarizes the notations that are used throughout this chapter.

Table 4.1. Notations used in the remainder of the chapter

Notation	Description
D	Input matrix
$D(x_i, x_j)$	Distance between object x_i, and x_j
T_p, T_L, T_R	Respectively parent node and its associated left, and right nodes
$I(T_L,T_R)$ or I	Total inter-cluster distance between node T_L, and T_R
$U(T_P)$ $U(T_L)$,$U(T_R)$	Sum of all distances between objects within a given set; respectively in parent node T_P, left node T_L, and right node T_R
$U(x_i)$ or $U(x_i ,T)$	Sum of distances from object x_i to every object in a given set T.
$H(T_L,T_R)$ or H	The objective function
1-NN(x_i),	The nearest neighbor object to x_i

4.3.1. Problem Definition

Let T_P be a node, and T_L, and T_R its associated sub-nodes (to be computed). We let $I(T_L,T_R)$ be the inter-cluster distance between the sub-nodes and $U(T_L)$ and $U(T_R)$ be respectively the intra-cluster distance of node T_L and T_R. The problem is to split T_P into T_L, and T_R in such a way that inter-cluster distance between T_L and T_R, $I(T_L,T_R)$, is maximized and total intra-cluster distance, $U(T_L)+U(T_R)$, is minimized.

This splitting problem can formally be defined as follows:

Given a set T_P of objects $\{x_1,...,x_n\}$ find two sub-sets T_L and T_R, with $T_P=T_R \cup T_L$ and $T_R \cap T_L=\varnothing$ such that:

$$H(T_L ,T_R) = (1- \alpha)\ I(T_L,T_R) – \alpha [\ U(T_R) + U(T_L)] \qquad (1)$$

is maximized.

α is a parameter between 0 and 1 which balances the importance of each objective.

When α=1 one favors to minimize the intra-cluster distance over maximizing the inter-cluster distance. Similarly when α=0 one favors to maximize the inter-cluster distance over the intra cluster distance. We note that intra-cluster distance of the parent node is equal to the sum of intra-cluster distances of both sub-nodes plus inter-cluster distance between both sub-nodes.

$$U(T_P)= I(T_L,T_R) + [U(T_R) + U(T_L)] \quad (2)$$

We also remark that $U(T_P)$ is a constant value because assigning an example to left or right node, does not change total intra-cluster distance of the parent node. Substituting $I(T_L,T_R)=U(T_P) - [U(T_R) + U(T_L)]$ into equation (1) we obtain $H(T_L ,T_R) = - [U(T_R) + U(T_L)] + U(T_P)(1- \alpha)$. Since $U(T_P)(1- \alpha)$ is a constant value, maximizing H is equivalent to minimizing $[U(T_R) + U(T_L)]$. Likewise replacing $[U(T_R) + U(T_L)]= U(T_P) - I(T_L,T_R)$ into equation (1) we obtain $H(T_L ,T_R) = I(T_L,T_R) - \alpha U(T_P)$. Since $\alpha U(T_P)$ is a constant value, maximizing H is equivalent to maximizing $I(T_L,T_R)$.Therefore the objective function expressed in equation (1) is equivalent to the objective function given by equation (3) or equivalently by equation (4).

$$\text{Maximize } H_1(T_L, T_R) = I(T_L,T_R) \quad (3)$$

$$\text{Minimize } H_2(T_L,T_R) = [U(T_L)+U(T_R)] \quad (4)$$

We note that maximizing/minimizing the objective function is independent of α. The total inter-cluster distance, $I(T_L,T_R)$, is given by equation (5) and total intra-cluster distances for node T_L is by given by equation (6).

$$I(T_L,T_R) = \sum_{\substack{x \in TL \\ y \in TR}} D(x,y) \quad (5)$$

$$U(T_L) = \sum_{\substack{x,y \in T_l \\ x \neq y}} D\,(x,y) \qquad (6)$$

Finding the optimum solution for H_1 or H_2 is computationally complex. A somewhat similar problem is the well-known "The balanced number partitioning problem" [20] which is known to be NP-hard. Computing the optimum solution for our objective function may require that all possible assignments to either set be tried out. AVALANCHE proposes a heuristics to split the dataset which is based on identifying first, the anti-medoid and then assigning it to a cluster (previously empty set). Next, neighboring examples to the anti-medoid are added one at a time to grow the cluster so long as the objective function improves. Therefore unlike traditional bottom up hierarchical approaches that consider only proximity of clusters as sole criterion for merging (local information), or top-down hierarchical clustering which use variance reduction as splitting strategy (global information), the proposed algorithm uses both local as well as global information about the data to recursively split it with the aim of obtaining improved clustering result. Our approach incorporates global information about the dataset because it computes the anti-medoid of the dataset, and local information is considered by identifying examples in the neighborhood of the anti-medoid. The detail of this method is provided in section 4.3.2. Since both equation (3) and (4) are equivalent, for the rest of the chapter we consider equation (3). AVALANCHE computes an approximate value for the objective function which we describe in section 4.3.2.

4.3.2. Node Evaluation

An approximated solution to H_1 is provided in this section. We use $T_{L,}$ $T_{R,}$ and T_P to not only designate the parent node and both child-nodes but also to refer to the datasets in the respective nodes. Given T_P a parent node, the algorithm starts with one of the sub-node being an empty set, (say $T_R = \emptyset$) and the other sub-node filled with all the data of the parent node, (say $T_L = T_P$). Next, it attempts to move examples from T_L to T_R one example at a time, maximizing H_1. One important challenge is how to determine the examples that need be removed from T_L to T_R. A brute force approach would be to try all possible splits then select the one that maximizes the objective function. Such an approach would be impractical. We use the Nearest Neighbor Chain approach (1-NNC) for this purpose. Firstly, it is cost effective to compute 1-NNC from an input distance matrix. Secondly, by adding the examples one at a time incremental optimization of the objective function can be achieved. With this approach, first the anti-medoid is computed and assigned to T_R. Then its nearest neighbor, S_{last}, is computed and tentatively assigned to T_R. If H_1 improves when S_{last} is assigned to T_R then the assignment holds and the nearest neighbor of S_{last} residing in T_L is computed and tentatively assigned to T_R; the chain of assignment continues until H_1 does not improve. To compute the anti-medoid (the outermost object) AVALANCHE first computes the contribution of each example to total intra-cluster distance of T_L then selects the object with the largest value. This process is detailed in algorithm 1 which gives the pseudo code of the AVALANCHE algorithm.

Algorithm 1: Node Evaluation for the Top-Down Tree

1. Input: T_P ; Parent node
2. Outputs: T_L, T_R ; Sub-nodes
3. BEGIN
4. $T_L \leftarrow T_P$; $T_R \leftarrow \emptyset$
5. IF size of T_L = 1 THEN
6. RETURN;
7.
8. $S_{last} \leftarrow$ Anti-Medoid(T_L)
9. current_H$\leftarrow$ Compute H(T_L,T_R)
10. WHILE (TRUE)
11. previous_ H$\leftarrow$ current_H
12. $T_R \leftarrow T_R \cup \{S_{last}\}$
13. $T_L \leftarrow T_L - \{ S_{last}\}$
14. current_H$\leftarrow$ Compute H(T_L,T_R)
15. IF ((current_H - previous_H) > 0) THEN
16. $S_{last} \leftarrow$ 1-NN(S_{last})
17. ELSE
18. EXIT
19. END WHILE
20. END

If T_L has only one example, there is no need for a split (line 5-6); otherwise the anti-medoid is computed. The algorithm then enters the loop and iteratively uses 1-NN ap-

proach to move objects from T_L to T_R as long as H_1 improves (line 10-18). In line 12 to 14 both sets are updated (tentatively) and a new H_1 computed. If there is no improvement in H_1 the algorithm exits the loop (line 18); otherwise the assignment is confirmed and the algorithm fetches from T_L the nearest object to the object last assigned to T_R (1-NN(S_{last})), and assigns it to S_{last} (line 16).

4.3.3. Implementation

The anti-medoid (line 8) is computed using equation (7). Equation (7) computes each object x_i's ($x_i \in T_L$) contribution to total intra-cluster distance of T_L.

$$U(x_i,T_L) = \sum_{k=1}^{N} D\,(x_i,x_k) \qquad (7)$$

$U(x_i,T_L)$ can be stored as a vector say, vU. Matrix D is augmented with a row representing vU (last row of D).When object x_k located at row k of matrix D is moved to T_R a new row is entered in matrix D as new_vU ← old_vU – row(k,D) where row(k,D) means "row k of matrix D". Doing so updates both the intra-cluster distances for T_L and the inter-cluster distances at the same time in vector vU. Thus to compute $I(T_L,T_R)$ we only need to sum the values in the cells of vU corresponding to the objects already moved to T_R; and to compute $U(T_L)$ we sum the values in the cells corresponding to objects still remaining in T_L.

4.3.4. Runtime Complexity

Given a node dataset of size t, it cost $O(t^2)$ to compute the anti-medoid (to construct vector vU and to select the largest value from it). It cost O(t) to compute H_1 from vector vU;

therefore the cost to split a node is $O(t^2)=O(t^2)+ m*O(t)$ where m (m<t) is the number of objects assigned to T_R. When only one example is assigned per node split, H is computed only 1 time (per node split); therefore it cost $O(n^2)$ for splitting the root node, then $O((n-1)^2)$ to split the two nodes at depth 1, etc. and hence, $O(n^2+ (n-1)^2+.. + 2^1) = O(n^3)$ to build the tree. In case half of the examples are assigned per node split, it cost $O(n^2) =O(n^2) + n/2*O(n)$ to split the root node, $O([(n/2)^2+ (n/2)^2] =O(2*(n /2)^2)$ to split both nodes at depth 1, etc. Therefore the cost to build the tree in this scenario is $O(n^2+ 2^1*(n/2^1)^2 + 2^2 * (n/2^2)^2 +..+ 2^p*(n/2^p)^2]$ where p=log(n). This value can further be simplified as $O(n^2(1+ (1/2^1)^2+..+ (1/2^{p-1})^2) = O(n^2 +n^2 s_n)$ with $s_n=(1/2^1)^2+..+ (1/2^{p-1})^2 < 1$. Therefore total cost is $O(n^2)$. That it cost $O(n^2)$ for the best case scenario and $O(n^3)$ for the worst case scenario to build the tree.

4.3.5. Illustrating AVALANCHE Node Splitting Method

Fig. 4.2 demonstrates one implementation of algorithm 1. Initially, we assume that $T_L \leftarrow T_P$ and $T_R=Ø$. The highlighted columns contain data moved to T_R and the white columns represent data still in T_L. We use H_1 as objective function for this illustration. Column H hosts the values of the objective function and the last row of the table contains the current vector vU.

Fig. 4.2.a hosts the input matrix of dataset shown in Fig. 4.1, augmented with the initial vector vU in the last row. We note row(U_0) to mean "vector vU located in row U_0 of the table ". At this early stage no example has yet been assigned to T_R. Next, the algorithm

computes Umax, the largest value in row(U_0) and found that U(s1)=Umax=24. Hence, the anti-medoid is s1 and s1 is assigned it to T_R.

In Fig. 4.2.b row(s1) and column(s1) are highlighted to signify that T_L={s2,s3,s4,s5}, T_R= {s1}. Next, a new vU is computed. To do so, a new row is entered at the bottom of the table and updated as row(U_1) ←row(U_0) – row(s1). This is so because after s1 has been assigned to T_R, we must subtract its contribution to T_L's intra-cluster distances (U(T_L) ←U(T_L)- U(s1)) leading to U(s2)=16, U(s3)=8, U(s4)=8, and U(s5)=12. Hence, the new U(T_L) = U(s2) + U(s3) + U(s4)+ U(s5) = 44. U(S_1) contains the contribution of s1 to total inter-cluster distance. Since T_R only contains s1, it comes that H= I(T_R,T_L) = I({s1}, {s2,s3,s4,s5}) = U(s1) =24.

In Fig. 4.2.c the algorithm computes the nearest neighbor of s1; 1-NN(s1)=s2 (circled value); s2 is tentatively assigned to T_R.' This is shown in Fig. 4.2.d where column(s1) and column(s2) are highlighted. A new row is entered and its values updated as row(U_2) ← row(U_1) - row(s2). H= I({s1,s2},{s3,s4,s5}) =U(s1)+U(s2)=22+16=38. (We remark that the intra-cluster distance of T_L is now U(T_L) =U(s3) +U(s2) +U(5)= 4+3+5=12) .

Next, in Fig. 4.2.e 1-NN(s2) = s3. A new row, (row(U_3), is entered and updated in the same manner as previously done for row(U_2) and H=16+12+4= 32 < 38. Therefore s3 candidacy is rejected and the node split stops. The result of the node split is shown in Fig. 4.2.f.

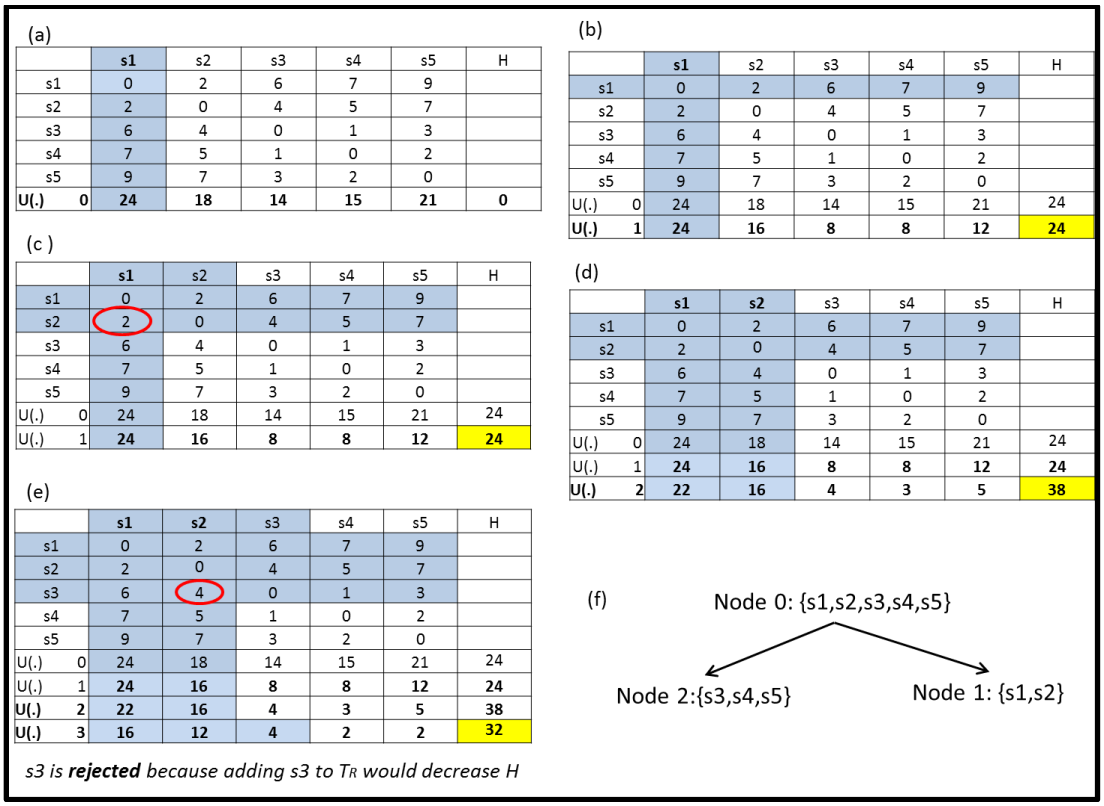


Fig. 4.2. Illustrating AVALANCHE node-split method

4.4. Experimental Evaluation

We compare the results of our proposed method to those obtained with the widely used UPGMA algorithm. This is because a direct comparison to divisive approaches that have been proposed in the literature is not possible as our approach uses dissimilarity matrices whereas current divisive approaches use the pair (attribute/attribute values). The current approach was evaluated using two types of performance criteria: (1) runtime complexity, (2) intra-cluster distances of the generated clusters. One artificial dataset and five real world datasets were used to evaluate the algorithms. All experiments reported in this chapter were performed on a 64-bit PC i7-2630 CPU at 2 Ghz running Windows 7. With

respect to the runtime evaluation, we generated various size datasets (artificial dataset). We implemented AVALANCHE and UPGMA in the same machine. UPGMA was implemented in its canonical form without optimization. We ran both algorithms and compared the average speeds (section 4.4.2). We used total average intra-cluster distances to measure the quality of the clustering result (section 4.4.3).

4.4.1. Datasets

We used the datasets summarized in Table 4.2 throughout the experiments

Table 4.2. Datasets

Dataset Name	***Description***	***Size***	***Number of Class Labels***
E.coli	Niche breadth	82	3
AV	Archaea growth rate in natural environment	70	4
BE	Bacteria ecosystem class: engineered environment	120	4
BV	Bacteria ecosystem class: environmental	311	4
BH	Bacteria ecosystem class: host-associated environment	571	3
Art	Randomly generated sequences	*many sizes	3

*We generated various sizes of this dataset

- **Real-world Datasets**

Distance measure used for distance matrices were the patristic distance for all real-world datasets.

E. coli: This dataset was obtained by measuring the growth of 82 strains of *Escherichia coli* in 10 distinct environments. Strains were then classified as specialists (S), intermediate (I), or generalists (G) depending on arbitrary divisions of the standard deviation of their growth in the environments.

Ecosystem datasets: Datasets characterize principle ecosystem type of bacteria (engineered environment, BE; environmental, BV; host-associate, BH). Ecosystem type and sequence information were downloaded from the Joint Genome Institute website [21].

In addition, dataset for growth rate of various types of archaea in natural environment was used in this experiment (AV).

- **Artificial Datasets**

Art dataset: Artificial dataset was generated using a random sequence generator and have a sequence length of 200. The publicly available software MEG6 (Molecular Evolutionary Genetics Analysis version 6) [13] was used to compute the distance matrix using p-value distance.

4.4.2. Runtime

We used the artificial dataset with various sizes to evaluate the runtime speed of the approaches. The trees were fully grown until each leaf-node contains one example. Fig. 4.3 summarizes the obtained results.

	Time in Seconds	
Dataset sizes	UPGMA	Avalanche
300	0	1
500	3	2
800	16	5
1200	44	14
1500	78	21
1800	136	34
2000	186	44
2200	250	54
2400	319	65

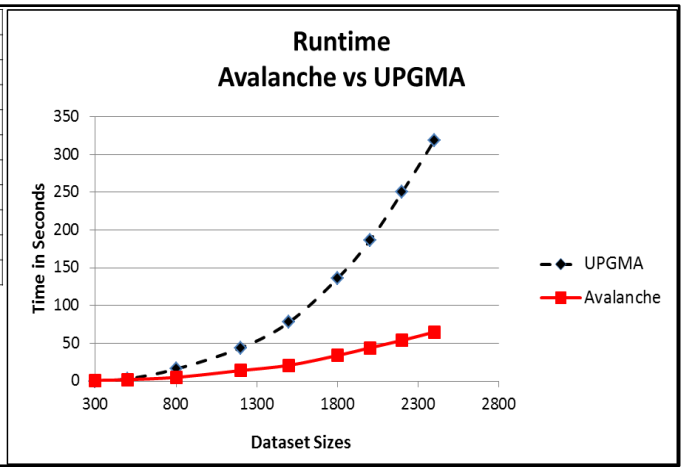


Fig. 4.3. AVALANCHE runtime complexity

As expected, the result in Fig. 4.3 confirms that AVALANCHE is a lot faster than the UPGMA algorithm (not optimized form of the UPMGA).

4.4.3. Intra-Cluster Distance

To measure the compactness of clusters created by both algorithms, we use the average intra-cluster distances criterion. The weighted average method is used to compute overall average "within cluster" distances

$$Av\_Intra(X) = = \sum_{C_i \in X} \left(\frac{|C_i|}{N} * intraclusterDistance(C_i) \right)$$

where N is the sum of the cardinality of all clusters C_i. $N=\sum_{C_i \in X}(|C_i|)$, X is the clustering result (set of clusters generated by the algorithm), $|C_i|$ cardinality of cluster C_i. We made the assumption that the lower the average intra-cluster distance, the better the algorithm with respect to generating compact clusters. Table 4.3 summarizes the obtained results. The result shows that AVALANCHE produces smaller size clusters on four datasets, tie on three datasets with UPGMA and lost on one dataset (slightly under perform

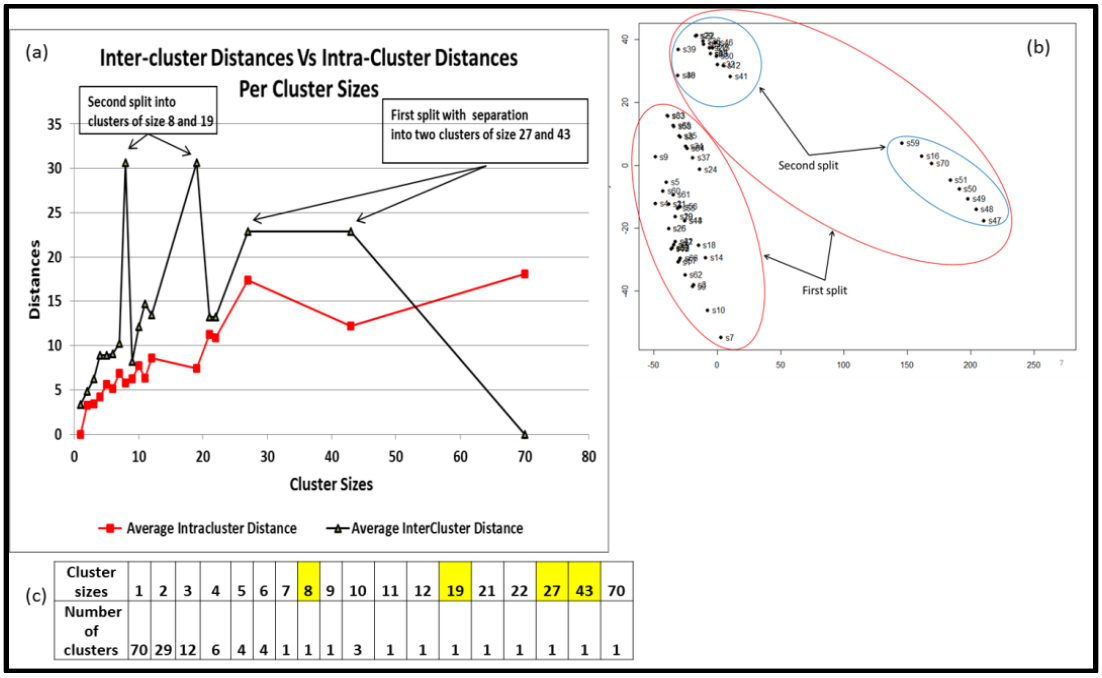
on the E.coli dataset). The obtained result therefore supports the assumption made that incorporating local and global information into the splitting decision leads to improved clustering result.

Table 4.3. Total number of clusters and average intra-cluster distance

			Average Intra Cluster Distances		Average Inter Cluster Distances	
Dataset	*Size*	*Total Number of Clusters*	UPGMA	AVALANCHE	*UPGMA*	*AVALANCHE*
E.coli	82	163	**10.34**	10.5	15.32	14
AV	70	139	8.32	**7.98**	12.1	10.13
BE	120	239	26.25	**24.7**	43.96	52.9
BV	311	621	17.28	**14.67**	31.34	17.09
BH	571	1141	29.05	**24.31**	49.1	29.47
Art 20	20	39	0.6	0.59	0.61	0.6
Art 50	50	99	0.62	0.62	0.63	0.63
Art 100	100	199	0.63	0.63	0.64	0.64

Dataset Art20, Art50, and Art100 are three different sizes of the Art dataset. Using the AV dataset, we illustrate in Fig. 4.4 the number of clusters generated by AVALANCHE, the clusters' sizes, the inter cluster distance that was generated after the cluster was created, and the resulting average intra-cluster distance. The table in Fig. 4.4.c contains in its first row sizes of clusters generated by AVALANCHE (on the AV dataset). The second row contains the number of clusters per size. For example, there are 70 clusters of size 1 (first column) and one cluster of size 70 (last column). It can be observed from Fig. 4.4 that with the AV dataset, the small size clusters do not generate large inter-cluster distances (in general). However, cluster of size 8 and cluster of size 19 generated after split, a large inter cluster distance of 30 (Fig. 4.4.a). This indicates that the two clusters are well separated clusters. Likewise, cluster of size 27 and 43 generated another spike. The

actual clusters (size 8 and 19) and (size 43 and size 27) are shown in Fig. 4.4.b. As shown in Fig. 4.4.b the first split generated clusters of size 27 and 43. Then the second split divided the cluster of size 27 into cluster of size 8 and cluster of size 19. By analyzing the inter-cluster distances and the size of the generated clusters the structure in the dataset can be understood.



(c)

Cluster sizes	1	2	3	4	5	6	7	8	9	10	11	12	19	21	22	27	43	70
Number of clusters	70	29	12	6	4	4	1	1	1	3	1	1	1	1	1	1	1	1

Fig. 4.4. Illustrating AVALANCHE ability to generate clusters that are both compact and far apart

4.5. Conclusion

This research introduces a novel hierarchical clustering algorithm called AVALANCHE. AVALANCHE forms clusters by splitting node datasets using a nearest neighbor chaining approach that originates from the anti-medoid of the dataset to be split and the chaining approach is controlled by a novel objective function that can consider both intra-

cluster and inter-cluster distances. An incremental method to save time in the objective functions computations is proposed. The approach takes as input a dissimilarity matrix and therefore can be a useful tool for applications where the dataset is formed by pairwise distances among the examples; taxonomy generation tools and molecular biology application need such capabilities. Divisive clustering algorithms that have been proposed in the literature cannot be used for such datasets because they rely on centroid computation which is not feasible when the input is a distance matrix. Unlike other approaches, AVALANCHE incorporates in its splitting decision local as well as global information which provides the algorithm with the capability of generating better clustering results. Experimental evaluations confirm that the new approach generates comparable or better clustering results than the well-known UPGMA algorithm.

References

1. Ao, S. I., Yip, K., Ng,M., Cheung, D., Fong, P.-Y., Melhado, I., and Sham, P. C. ; *"Clustag: Hierarchical Clustering and Graph Methods for Selecting Tag Snps"*; Bioinformatics, 21 (8), 1735–1736.

2. Jacob Bien and Robert Tibshirani; *"Hierarchical Clustering With Prototypes via Minimax Linkage"*; Journal of the American Statistical Association 2011.

3. Boley, D.L. (1998); *"Principal Direction Divisive Partitioning"*;Data Mining and Knowledge Discovery, vol.2, n.4, pp. 325-344.

4. R. Chitta and M. Narasimha Murty; *"Two-level k-means clustering algorithm for k–T relationship establishment and linear-time classification"*; *Pattern Recognition,* vol. 43, no. 3,pp. 796–804, March. 2010.

5. Defays; *"An efficient algorithm for a complete link method";* The Computer Journal (British Computer Society) 20 (4): 364–366. (1977).

6. Forgy, E. (1965); *"Cluster Analysis of Multivariate Data: Efficiency versus Interpretability of Classification"*; Biometrics, pp.768-780.

7. Gose, E., R. Johnsonbaugh, S. Jost (1996); *"Pattern Recognition & Image Analysis"*; Prentice-Hall.

8. Hastie, T., Tibshirani, R., and Friedman, J. (2009); *"The Elements of Statistical Learning; Data Mining, Inference and Prediction"*; (2nd ed.), New York: Springer-Verlag.

9. Everitt, B., Landau, S., and Leese, M. (2001); "*Cluster Analysis (4th ed.)"*; London: Arnold.

10. Jain, A.K., R.C. Dubes (1988); *"Algorithms for clustering data"*; Prentice-Hall advance reference series. Prentice-Hall, Upper Saddle River, NJ.

11. Jain, A.K, M.N. Murty, P.J. Flynn (1999); *"Data Clustering: a Review"*; ACM Computing Surveys, Vol.31,n.3, pp.264-323.

12. Keerthiram Murugesan, Jun Zhang; *"Hybrid Bisect K-Means Clustering Algorithm"*; 2011 Second International Conference on Business Computing and Global Informatization pp,216-219.

13. Koichiro Tamura, Glen Stecher, Daniel Peterson, Alan Filipski, and Sudhir Kumar; (2013) *MEGA6: Molecular Evolutionary Genetics Analysis version 6.0. Molecular Biology and Evolution: 30 2725-2729*; http://www.megasoftware.net/ (as of October 2014).

14. Selim, S.Z., M.A. Ismail (1984); *"K-means-type algorithms: a generalized convergence theorem and characterization of local optimality"*; IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol.6, n.1, pp. 81-86.

15. Sergio M. Savaresi, Daniel L. Boley, Sergio Bittanti, Giovanna Gazzaniga; *"Choosing the Cluster to Split in Bisecting Divisive Clustering Algorithms";* SIAM International Conference on Data Mining 2002.

16. Steinbach, M., G. Karypis, V. Kumar (2000); *"A comparison of Document Clustering Techniques"*; Proceedings of World Text Mining Conference, KDD 2000, Boston.

17. Sibson *"SLINK: an optimally efficient algorithm for the single-link cluster method"*; The Computer Journal (British Computer Society) 16 (1): 30–34. (1973).

18. P.-N. Tan, M. Steinbach, and V. Kumar; *Introduction to Data Mining*; (First Edition), May 2005, vol. chapter 8.

19. Ward, J. H., Jr. (1963); *"Hierarchical Grouping to Optimize an Objective Function"*; Journal of the American Statistical Association, 58, 236–244.

20. Stephen Mertens Computational; *"The Easiest Hard problem"*; Complexity and Statistical Physics edited by Allon Percus, Gabriel Istrate, Cristopher Moore.

21. The Joint Genome Institute, https://img.jgi.doe.gov/cgi-bin/w/main.cgi (as of April 2015)

Chapter 5

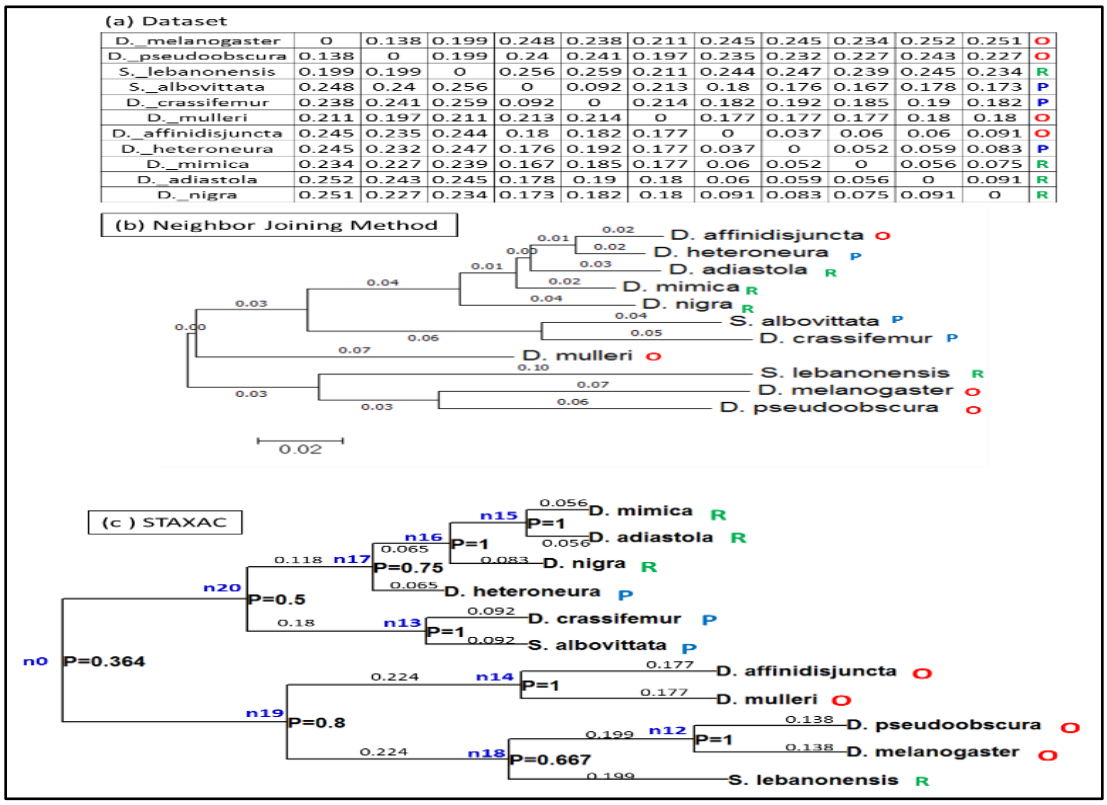
Supervised Taxonomies — Benefits, Algorithms and Applications

5.1. Introduction

The goal of taxonomies is to produce a formal system for naming and classifying species. Molecular taxonomists generate phylogenetic trees — computed based on gene sequences dissimilarities — revealing predicted evolutionary relationships of a group of organisms. Such phylogenetic trees are not only useful tools to categorize species, but also represent powerful predictive tools that are often used to predict class membership with respect to phenotypes.

This work proposes a new type of taxonomy we term *supervised taxonom*y (ST). Supervised taxonomies are generated considering background information concerning class labels in addition to distance metrics, and are capable of capturing class-uniform regions in a dataset. More formally, supervised taxonomy generation deals with the following problem: given a set of organisms $B=\{b_1,b_2,..,b_n\}$ with class labels drawn from a finite set of classes $C=\{c_1,c_2,..,c_m\}$ one would like to construct hierarchical clusters, such that at each step class purity is maximized under the constraint that the merged clusters are neighboring.

The merit of supervised taxonomies is illustrated in Fig. 5.1 which depicts a dissimilarity matrix containing 11 *Drosophila* species randomly assigned to three hypothetical classes: {O,P,R}. The same dataset is processed with the well-known Neighbor Joining (NJ) algorithm [21], a hierarchical agglomerative clustering approach that joins two examples into their possible common ancestor (Fig. 5.1.b), and using STAXAC (Fig. 5.1.c), a supervised taxonomy generation algorithm which will be introduced in section 5.3.



D._melanogaster	0	0.138	0.199	0.248	0.238	0.211	0.245	0.245	0.234	0.252	0.251	O
D._pseudoobscura	0.138	0	0.199	0.24	0.241	0.197	0.235	0.232	0.227	0.243	0.227	O
S._lebanonensis	0.199	0.199	0	0.256	0.259	0.211	0.244	0.247	0.239	0.245	0.234	R
S._albovittata	0.248	0.24	0.256	0	0.092	0.213	0.18	0.176	0.167	0.178	0.173	P
D._crassifemur	0.238	0.241	0.259	0.092	0	0.214	0.182	0.192	0.185	0.19	0.182	P
D._mulleri	0.211	0.197	0.211	0.213	0.214	0	0.177	0.177	0.177	0.18	0.18	O
D._affinidisjuncta	0.245	0.235	0.244	0.18	0.182	0.177	0	0.037	0.06	0.06	0.091	O
D._heteroneura	0.245	0.232	0.247	0.176	0.192	0.177	0.037	0	0.052	0.059	0.083	P
D._mimica	0.234	0.227	0.239	0.167	0.185	0.177	0.06	0.052	0	0.056	0.075	R
D._adiastola	0.252	0.243	0.245	0.178	0.19	0.18	0.06	0.059	0.056	0	0.091	R
D._nigra	0.251	0.227	0.234	0.173	0.182	0.18	0.091	0.083	0.075	0.091	0	R

Fig. 5.1. Traditional phylogenetic tree vs supervised taxonomy

In Fig. 5.1.b the distance from the ancestor to an existing organism is annotated on the edges that connect them. In contrast, trees generated by supervised taxonomy generating algorithms, such as STAXAC, are built balancing both the phylogenetic proximity (dis-

similarity) of the examples within the clusters and the cluster purity. There are several merits to the proposed approach: (1) it provides an alternative way to categorize species by analyzing the class proportion of each sub-tree. For example, in Fig. 5.1.c, the class proportions for node n19 are (0.8, 0.2, 0.0) with respect to the underlying class structure {O, R, P}, indicating that 80% of the examples in the sub-tree belong to class O and 20% belong to class R. (2) Another advantage of supervised taxonomies is that they can be used as predictive tools. For example using Fig. 5.1.c, if an unlabeled species is found to pertain to sub-tree rooted at, say node 17, its chances of exhibiting class R or P can be estimated to be larger than that of exhibiting class O because it pertains to a sub-tree with associated class proportions (0, 0.75, 0.25). With the phylogenetic tree in Fig. 5.1.b, the class composition (of a sub-tree) is likely to be more diverse. Although the discussion presented above relates to organisms, ST can be generated from any dataset provided a dissimilarity matrix can be computed from the input dataset.

Moreover, in this research we are interested in the question if the distribution of the classes in a dataset is random, unimodal or multi-modal, and algorithms that address this problem are introduced which operate on top of supervised taxonomies. To obtain an answer to this question, EUREKA, a set of measures and algorithms that operate on top of supervised taxonomies is introduced later in this chapter. It first extracts from a given STs, sub-trees of high purity such that the union of all the examples in the sub-trees is equal to the original dataset. It then characterizes the class distribution by analyzing the composition of the obtained clusterings. Throughout this chapter, a high purity cluster/region with respect to a single class is referred to as subclass. According to this defini-

tion, a cluster is viewed to be a subclass if most of its instances belong to a single class; that is, if the cluster's purity is high.

A closely related problem is the problem of determining the number of natural clusters hidden in a dataset [17, 26, 22]. However, traditional clustering is about discovering high density regions with respect to proximity whereas the problem investigated in this work is about discovering class uniform regions in a dataset. Fig. 5.2 illustrates these differences. Fig. 5.2.a represents the original dataset for which subclasses need be uncovered.

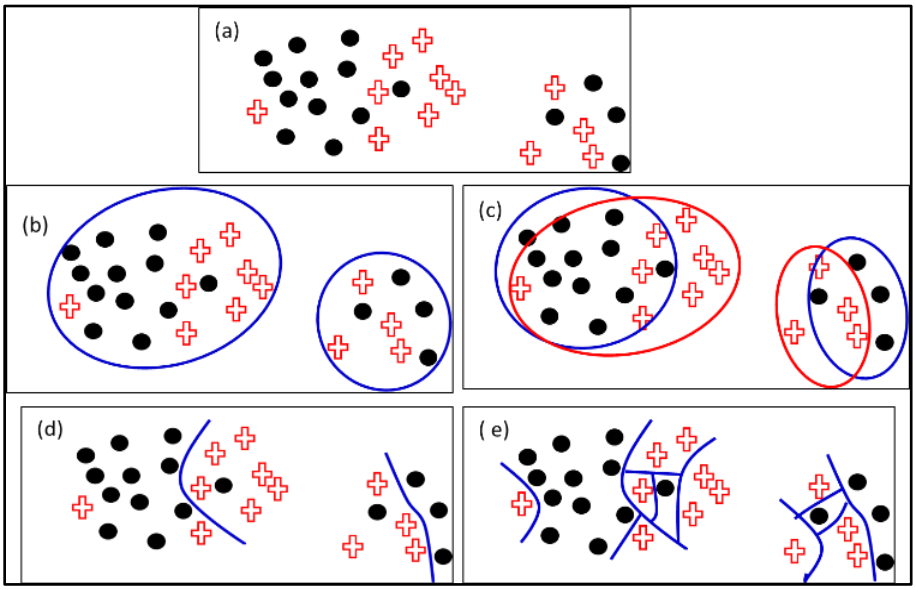


Fig. 5.2: Discovering subclasses for various purity thresholds

A traditional clustering approach may not reveal any subclass as it does not consider class label information in its clustering process (Fig. 5.2.b). Another approach has been proposed in literature [25] where the examples of a given class label are clustered one class at a time (ignoring examples of other classes); however, this approach does not adequate-

ly reveal subclasses in the way we define them, since the examples from other classes are removed from the dataset before the clustering takes place (Fig. 5.2.c). On the other hand for a given taxonomy we are able to identify subclasses for various purity thresholds (Fig. 5.2d and Fig 5.2.e). Fig. 5.2.d shows a hypothetical partitioning of the input dataset into four subclasses with each subclass allowing some degree of impurity whereas Fig 5.2.e shows a clustering of the same dataset where every cluster has 100-percent purity.

Previous work dealing with discovering clusters of high purity in a dataset can be found in [11, 12, 4]. The authors termed their approach *"supervised clustering"* (SC). SC grows clusters around representative examples minimizing both clusters' impurity and total number of clusters. However, SC differs from ST in that SC produces flat clusters and only a single clustering, and requires a user input parameter that determines the granularity of the obtained clustering, whereas ST are hierarchical clusters that represent a set of clusterings and do not require a user-defined input parameter. By generating hierarchical clusters, STs provide the capability of creating background knowledge at different degrees of granularity.

The remainder of this chapter introduces the following results:

(1) Supervised taxonomies; a new form of taxonomy that incorporates both proximity among the examples and background information in the form of class labels.

(2) STAXAC; a hierarchical, supervised clustering algorithm which generates STs with larger sub-trees of higher purities than traditional taxonomy algorithms.

(3) A methodology for comparing and evaluating taxonomies is proposed that uses and generalizes existing approaches and introduces a novel evaluation method which assesses purity at different levels of depth for a given taxonomy.

(4) A meta learning approach, EUREKA, that operates on top of ST and provides the following capabilities:

   a. It automatically identifies and quantitatively assesses the presence of subclasses in a dataset eliminating the need for visual inspection of the tree.

   b. It provides a measure of classification complexity of the dataset which assesses the difficulty of classifying the instances in the dataset at hand.

The rest of this chapter is divided as follows. Section 5.2 provides related background discussion. Section 5.3 discusses supervised taxonomy. Section 5.4 introduces the EUREKA approach and experimental evaluation is presented in section 5.5.

5.2. Related Work

Traditional clustering methods do not use class label information when clustering data. Basu et al. [6] introduce Semi-Supervised Clustering: a clustering method that uses small amount of labeled data as seeds to guide the clustering process of unlabeled data. Cohn, et al. [8] propose a semi-supervised clustering approach that allows a user to iteratively provide feedback to a clustering algorithm. The feedback is incorporated in the form of constraints which guide the clustering algorithm towards a clustering result that the user finds more useful. Eick et al. [11, 12] introduce *Supervised Clustering* which assumes that all examples that are clustered carry class labels and the goal of supervised clustering

is to identify class-uniform clusters that have high probability densities with respect to a single class. Daume et al. [9] introduce a Bayesian approach for supervised clustering. Finley et al. [14] propose a SVM-based approach to supervised clustering. Bagherjeiran et al. [4] introduces supervised similarity assessment. The goal of supervised similarity assessment is to obtain a distance function that separates well cases belonging to different classes. Vilalta et al. [25] proposes a class decomposition method that identifies subclasses before a classifier is applied; however, in this approach subclasses are obtained by clustering the examples of each class separately and not jointly.

5.3. STAXAC — An Algorithm that Creates Supervised Taxonomies

In this section, a new hierarchical clustering algorithm, called STAXAC (Supervised TAXonomy Agglomerative Clustering), is introduced. It creates STs by merging neighboring clusters maximizing purity. Traditional hierarchical agglomerative clustering recursively merges the two closest clusters into a larger cluster until the last two clusters are merged. STAXAC, on the other hand, attempts to maximize purity (minimize impurity growth at each step of the recursive process) by merging clusters in which a majority of the examples has the same class label. It uses distance information as a constraint so that only neighboring clusters are merged. The pseudocode of STAXAC is described in Algorithm 1.

Algorithm 1: *STAXAC*

Input: examples with class labels and their distance matrix D.

Output: Hierarchical clustering

1. Start with a clustering X of one-object clusters.

2. $\forall C^*, C' \in X$; merge-candidate(C*,C') $\Leftrightarrow$ (1-NN_X(C*) = C' or 1-NN_X(C')=C*)

3. WHILE there are merge-candidates (C^*, C') left

BEGIN

a. Merge the pair of merge-candidates (C^*,C') obtaining a

new cluster C=C*$\cup$C' and a new clustering X' for which Purity(X') has the largest value

b. Update merge-candidates:

$\forall C''$ merge-candidate(C'', C) $\Leftrightarrow$ (merge-candidate(C'',C^*) *or* merge-candidate(C'',C'))

c. Extend dendrogram by drawing edges from C' and C* to C

END

4. Return constructed dendrogram

STAXAC identifies/updates merge candidates—clusters that potentially can be merged — and then creates a new cluster by choosing the merge candidate that maximizes the purity objective function. It continues this process until no more clusters can be merged. It starts off with single-object clusters (line 1) and pairs of objects that are 1-nearest-neighbor of each other to form the initial set of merge candidates (line 2)[3]. Then STAXAC merges the best merge-candidate (C*, C'), creating a new cluster C=C*$\cup$C' and computes the merge-candidates of the newly created cluster C as the union of the merge candidates of clusters C* and C' that were merged (lines 3a and 3b). In summary, STAXAC conducts a wider search than traditional agglomerative hierarchical clustering

[3] It should be noted that the initial set of merge candidates is a subset of the actual merge candidates; in general, clusters could be neighboring with more than 2 other clusters. However, determining all clusters which are neighboring is only feasible in 2D-space by computing the Voronoi tessellation for the points in the dataset; unfortunately, the Voronoi tessellation cannot be computed for higher dimensional spaces.

(HAC) algorithms, as it merges neighboring clusters, but it does not necessarily merge the pair of clusters that are closest to each other.

Fig. 5.3 illustrates the effectiveness of STAXAC in finding sub-classes from a given dataset, presenting a hypothetical two-class dataset comprising 3 species with a disease (the white disks) and 3 healthy species (dark disks). The arrows "→" represent the 1-NN relationships among the examples.

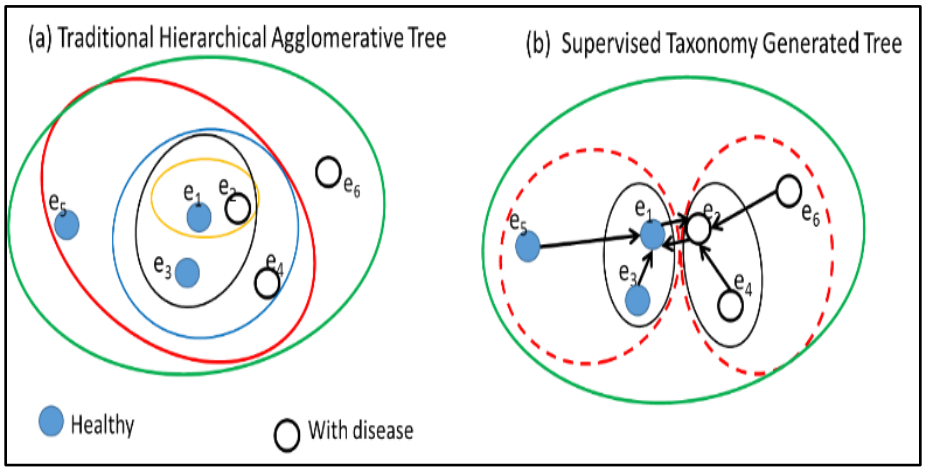


Fig. 5.3: Illustrating partitions obtained by STAXAC vs partitions obtained with traditional hierarchical agglomerative clustering on a hypothetical dataset

In Fig. 5.3, HAC-generated tree (Fig. 5.3.a) and STAXAC-generated tree (Fig. 5.3.b) were induced from same hypothetical two-class dataset. One can visually observe that the optimum division of the dataset into large and high purity partitions is the two sets of 100-percent purities ($\{e_2, e_4, e_6\}$, and $\{e_5, e_1, e_3\}$). The traditional hierarchical agglomerative clustering approach is unable to discover the optimum partition since it merges the two closest clusters.

STAXAC exhibits some interesting properties:

1. In contrast to other supervised clustering algorithms, STAXAC is a hierarchical clustering that maximizes cluster purity.
2. In contrast to other hierarchical clustering algorithms that merge the pair of closest clusters, STAXAC uses a 1-NN graph to determine which clusters are neighboring. Thus STAXAC conducts a wider search, merging clusters that are neighboring and not necessarily the closest two clusters. This ability allows STAXAC to generate larger size, high purity clusters than traditional hierarchical clustering methods.
3. Proximity graphs need only be computed at the beginning of the run which saves time.
4. STAXAC can be generalized by using more powerful proximity graphs, such as Gabriel Graphs, to conduct a wider search.

The above properties can be useful in many areas, such as Bioinformatics, and Meta Learning. For example, in Biology, the ability of STAXAC to generate larger high purity clusters than tradition hierarchical approaches (property 2) can be helpful to identify interesting subclasses of known diseases. In Meta-Learning, property 1 and 2 can be used to create useful background information from datasets by developing algorithms that operate on supervised taxonomies, which will be the subject of the next section.

5.4. Creating Background Knowledge with STs

In this section, we introduce EUREKA, a set of measures and algorithms that operate on top of supervised taxonomies to create background knowledge of a dataset. EUREKA provides two groups of methods. The first group, we term, tree topology evaluation

methods, allows one to evaluate and to compare different taxonomies. The second group consists of computational methods that (1) extract subclasses from STs at various levels of granularity (2) compute the modalities of the classes in the dataset and (3) assess the difficulty of classifying the examples in the dataset. The notation used in the remainder of the chapter is summarized in Table 5.1.

Table 5.1. Notations used in the remainder of the chapter

O	Input dataset
n	Number of examples in O
c	Number of class labels in O
X	A clustering result;|X|= number of clusters in X
C	A cluster; |C| =number of examples in C
T	A taxonomy tree generated from O
T_i	The i_{th} sub-tree of T
θ	Purity threshold; $\theta \in (0,1]$
S_θ	Set of purity thresholds; $S_\theta=\{\theta_1,..,\theta_m\}$ for some m>0
X_θ	A clustering result for purity threshold θ such that each cluster has purity above or equal to θ
X_{all}	Set of all clusterings (for all purity thresholds); for example $X_{all} = \{X_{\theta 1},..,X_{\theta k}\}$ for thresholds $\theta_1,..,\theta_k$

5.4.1. Tree Topology Evaluation Methods

To evaluate and compare different taxonomies, adaptations of two previously proposed methods [10], (1) edited tree complexity, and (2) number of subfamily changes, as well as one new method, (3) depth average purity, are proposed.

Edited Tree Complexity [10]

When computing this measure, first the tree is fully developed then pure leaf-nodes of same class labels are merged. An edited tree complexity is then measured as the total number of nodes in the tree, indicating how compact the smallest possible pure clustering

is. In this research, we generalize this measure by allowing any merging of subtrees as long as the purity remains above a threshold θ. Given two taxonomies the taxonomy with the smaller edited tree complexity is preferred.

Number of Sub-family Changes [10]

We associate to each node a sub-family, which is the class with the largest number of instances, with the root node representing the most frequent class in the dataset. A family change occurs when two nodes with different majority class labels are connected to a branch. The measure evaluates a taxonomy by counting the number of family changes, preferring trees with the fewest family changes. The number of sub-family changes measures the ability of taxonomy to cluster examples of same class label together.

Average Purity Per Depth

Average purity per depth is a measure of a tree's performance in clustering examples belonging to different classes into separate clusters. Average purity per depth is calculated by averaging clusters' purities at each depth of the taxonomy tree. For example, if a depth d has only two nodes node1 and node2, and if node1 has purity p(node1)=p1 and node2 has purity p(node2)=p2, then purity of depth d is p(d)= (p1+p2)/2.

5.4.2. Extracting Subclasses from Dataset

Modality measures the number of major peaks in a distribution. Detecting the modalities in a dataset is an important step to data analysis and provides valuable insights and benefits. For example, in data reduction, a dataset may be reduced to a set of representative

examples which are the peaks of the distribution (central point of each subclass). Another motivating example can be found in modeling. For example, given a two-class dataset, if each class is unimodal, one may be prompted to design a binary classifier to classify the examples in the dataset. If on the other hand, the classes are multi-modal, one may resort to a more advanced classification strategy such as an ensemble of binary classifiers. In this section, we provide a method for a class modality assessment that — based on a domain expert's notion of what constitutes a "noteworthy" subclass — determines if specific classes in the dataset are zero-modal, unimodal, or multi-modal.

The problem under consideration consists of identifying in the dataset regions of high density with respect to a single class. To solve this problem we use the following three steps:

Step 1: STAXAC is used to cluster the dataset since its clustering method favors the construction of large sub-trees of high purity (section 5.3 (property 1 and 2)).

Step 2: Extract from the tree generated by STAXAC a clustering X consisting of sub-trees such that the sub-trees' purities are above some minimum purity threshold θ (with $\theta \in (0, 1]$).

Step 3: Delete from X clusters with sizes below a minimum threshold size σ — obtaining a reduced set of clusters, X'. The surviving clusters in X' are referred to as subclasses.

The problem to be solved in step 2 can formally be described as follows.

Given O a set of n labeled examples $x_1,..,x_n$, c the number of classes in O, and T its associated taxonomy tree, the goal is to retrieve from T a clustering X, such that requirements (1) through (5) are met:

(1) $X=\{C_1,..,C_k\}$

(2) $\forall\ C_i \in X$, purity $(C_i) \geq \theta$

(3) $|X|$, the number of clusters k is minimal

(4) $O = \bigcup C_i$

(5) $\forall\ C_i \in X, \forall\ C_j \in X, i \neq j \rightarrow C_i \cap C_j = \emptyset$

The cluster extraction procedure may return one of the below three clustering results (cases):

a) $|X|=c$ with $\theta=1$ (ideal case)

b) $|X|=n$ (worst case)

c) $|X|=k$ (for some $k>0$ and $k<n$)

Result (a) is considered the best case scenario where the number of clusters is equal to the number of classes and $\theta=1$; that is, all the examples of a given class reside in a single cluster. We refer to this case as the *ideal* clustering result named X_{ideal}. Another particular result is the case where the clustering algorithm yields n one-example clusters which are trivially 100-percent pure (result (b)); we consider this solution as the worst case scenario. The ideal case scenario will be used — in section 5.4.3 — to compute the classification complexity of the dataset, a measure of the difficulty of classifying the instances in the dataset at hand.

The remainder of this section is organized as follows. First, we propose an algorithm to solve the clustering extraction problem (step 2) that takes as input a purity threshold θ and extracts a clustering from a taxonomy tree, T, such that requirements 1 through 5 are met. Then we discuss how to compute an efficient set of parameter θ. Finally, we illus-

trate our method with a clustering result obtained using the UCI Iris dataset as input, and we provide an algorithm for subclass extraction.

Given a purity threshold θ, and a taxonomy tree T, a clustering meeting requirement 1 through 5, is extracted from T, by calling a function *ExtractClustering*, whose pseudo-code is listed as algorithm 2.

Algorithm 2: ExtractClustering (T, θ)

1. Inputs: taxonomy tree T; user-defined purity threshold θ
2. Output: clustering X
3.
4. *Function ExtractClustering (T, θ)*
5. IF (T = NULL)
6. RETURN $\varnothing$
7. IF T.purity $\geq \theta$
8. RETURN T
9. ELSE
10. RETURN *ExtractClustering*(T.left, θ) $\cup$ *ExtractClustring*(T.right, θ)
11. *End Function*

Function *ExtractClustering* recursively searches the tree from the root node toward the bottom, for nodes of sub-trees whose purity is above the purity threshold θ and returns the union of the found nodes as its result. The subtrees are not overlapping and a subtree is extracted once. Since the search starts from root node towards the bottom, the largest

possible sub-tree meeting the purity threshold will be returned. Thus requirement 1 through 5 are met and the clustering extraction problem is solved. However, when extracting subtrees from T, many parameter θs return identical clustering; identifying those values helps to avoid repetitious searches.

How to select parameter θ

Suppose that the child-nodes of the root node, T_{left}, and T_{right} have purity θ_{left} and θ_{right} such that $\theta_0 < \theta_{left} < \theta_{right}$ where θ_0 is the root node purity. Then any parameter $\theta \in (\theta_0, \theta_{left}]$ will return the same clustering result which is the clustering X formed by the sub-trees T_{left}, and T_{right}. This is due to condition (2) and (3) which require that the largest sub-trees of T with purity > θ_0 be chosen. The same analysis can be done replacing T with T_{left} (or T_{right}) and leads to the fact that the relevant values to be considered for purity thresholds, are the nodes purities. This is important as it allows a user to better select the input parameter θ. Secondly, knowing that the relevant purity thresholds are the node purities, one may pre-compute all the clusterings in advance — allowing the analysis of all the subclasses for all purity thresholds at once.

Computing an efficient subset of parameter θ

Given a purity threshold θ, algorithm 2 starts the search from the root node down to the leaf-nodes looking for a node purity greater or equal to θ; once such a node is found the sub-tree is extracted. However, while it is generally true that node purities at the bottom of the tree are larger than nodes purities higher up the tree, this is not always the case. A node may have a purity value greater than some of its child-nodes' node purities. When

this is the case, it is unnecessary to consider the child-node purity for sub-tree extraction. Therefore an appropriate set of purity thresholds, S_θ, must be computed from the set of all node purities.

We denote by T_k a sub-tree rooted at node k, $|T_k|$ the size of T_k (number of examples in the leaf-nodes of T_k) and by purity(T_k) the purity of T_k.

Definition 1: Given a tree T and two sub-trees T_p and T_c , T_c is a sub-tree of T_p, noted $T_c \subset T_p$, if all nodes of T_c are also nodes of T_p. Formally, $T_c \subset T_p \Leftrightarrow (T_c \neq T_p)$ AND $(T_p \cap T_c) = T_c$.

Definition 2: Given a node T_c, if none of its parent node $T_{p's}$ has a purity greater than purity(T_c) then purity(T_c) is a purity threshold (purity(T_p) $\in S_\theta$).

Consequently, if $T_c \subset T_p$ and purity(T_p) > purity(T_c) then there is no need to extract T_c separately (because T_p which includes T_c has a higher purity). In such a condition, purity(T_c) is not a purity threshold.

Purity threshold set, S_θ, is computed using algorithm 3.

Algorithm 3: Purity Thresholds Computation

1. Input: Taxonomy tree T
2. Output: S_θ set of purity thresholds

3. $S^*_\theta \leftarrow \emptyset$
4. # Step 1: Assign all node purities to S^*_θ (including duplicated values)
5. FOR each intermediate node p of T
6. $S^*_\theta \leftarrow$ purity(T_p)
7.
8. # Step2: Remove non purity thresholds
9. FOR each intermediate node p of T
10. FOR each sub-tree T_c of T_p
11. IF (purity(T_p) > purity(T_c)) THEN
12. $S^*_\theta \leftarrow S^*_\theta \setminus$ purity(T_c)
13.
14. #Step 3: Remove duplicated values
15. $S_\theta \leftarrow \{\mathbf{1}\}$ # purity of leaf-node are always 1 and included
16. FOR each θ in S^*_θ
17. IF $\theta \notin S_\theta$ THEN
18. $S_\theta \leftarrow S_\theta \cup \{\theta\}$
19.
20. Output S_θ

Algorithm 3 selects a subset of the node purities in the tree as follows: Initially S_θ is filled with all node purities (Step 1) (including duplicated values). In step 2, the algorithm visits each intermediate node (sub-tree T_p) and removes from S_θ the purity values that are not purity thresholds (line 8-12). Finally the duplicated values are removed from the list (line 14-18).

The methods we just introduced have been applied to the UCI Iris dataset, and some of the obtained results have been provided in Fig. 5.4.

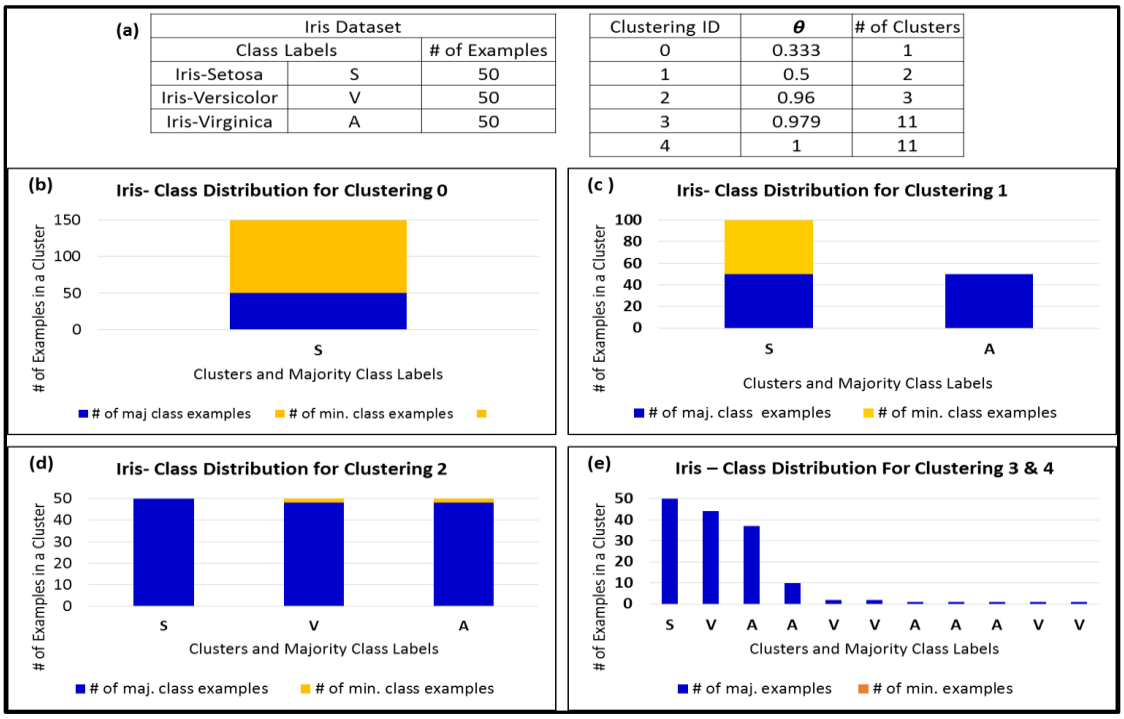


Fig. 5.4: Subclass-extraction result for the UCI Iris dataset

In Fig. 5.4.a, algorithm 3 identifies 5 purity thresholds S_θ = {0.333, 0.50, 0.96, 0.979, 1} corresponding to 5 possible clusterings (Clustering 0, Clustering 1, Clustering 2 , Clustering 3, and Clustering 4). *ExtractClustering* returns the same clustering for purity thresholds θ =0.979 and θ =1; thus clustering 3 and 4 are identical (Fig. 5.4.e). Each rectangle in the histograms is a cluster. The label of the dominant class of the cluster is provided on the horizontal axis. The light color on the rectangles represents the number of minority class examples in the clusters. Clustering 3&4 comprises pure clusters (no contamination), whereas clustering 0 is highly contaminated. Clustering 3&4 with purity thresholds

θ =1 &0.979, has 11 clusters whereas Clustering 0 with θ = 0.333 has only one cluster (any one of the classes A,S, or V could be made majority class since they are 50 examples in each class). The number of clusterings and the number of clusters for a clustering may be large. As purity increases, the number of clusters in a clustering increases as smaller size clusters are formed.

Some evaluation measure may be needed to decide which clusters within the obtained clustering are newsworthy enough to be reported. We use two user-defined parameters θ_{min} (minimum threshold purity value) and σ (minimum number of examples in a cluster) to select the K best clusters. For example a user may find that clustering 2 (in Fig. 5.4.d) suits best his purpose because he may accept a few contamination in favor of improved cluster size; another user may want a clustering in which all clusters are 100-percent pure at a cost of smaller subclass sizes and may select cluster 3. By using the above two simple parameters one can define a measure for the number of meaningful subclasses in the dataset. That is, if the top K clusters contain no clusters dominated by a given class, say A, then A is randomly distributed with no dense areas in the input space; if on the other hand, a number of clusters, say m, are found (for some $0<m \leq K$) then A has m subclasses. For example in Fig. 5.4, if θ_{min} =1 (or $\theta_{min} \in (0.96,1]$) and σ =10, for clustering 3 & 4, the following 4 subclasses will be obtained (disqualifying the smaller clusters as subclasses): Cluster1= (majority class=S, θ=1, size=50), Cluster2= (majority class=V, θ=1, size=44), and Cluster3= (majority class=A, θ=1, size=37), and Cluster 4= (majority class=A, θ=1, size=10), inferring that we have 2 unimodal class (S and V) and one bimodel class (A) in the dataset.

The general approach to class modality discovery is provided in algorithm 4. Before running the algorithm a domain expert has to decide how many instances a cluster needs to have to be newsworthy and how much contamination by other classes is acceptable. Next, clusters are extracted from the taxonomy that meet these characteristics (steps 1-5), and we count the occurrence of classes in the final display, inferring if they are 0-modal, 1-modal, or r-modal.

Algorithm 4: Class Modality Discovery

Inputs:

O; input dataset

σ; a user-defined threshold concerning the minimum number of instances a cluster should have to be considered as newsworthy

θ_{min}; a user-defined purity threshold that specifies how much contamination of instances is tolerable in a cluster

1: Create a ST T from O using STAXAC

2: Extract a clustering X from T by calling ExtractCluster(T, θ_{min})

3: Sort the clusters in $X=\{C_1,\ldots,C_k\}$ by their size obtaining a sequence S

4: Delete clusters from S whose number of instances is less than σ

5: Display the remaining clusters in S in a histogram where each bin displays the number of instances in the respective cluster; label each bin with the name of the majority class of the respective cluster

6: Analyze the composition of the obtained histogram with respect to class labels to determine modalities of particular classes

5.4.3. Classification Complexity

Data complexity has been proposed in [19, 23, 27, 18]. However the proposed complexity measures are with respect to a given classifier (complexity of a classification problem), whereas the complexity proposed in this research are independent of using a particular classifier; it measures the proportion of large size and small size high purity clusters in a dataset.

In the following, we introduce a measure to assess the difficulty of a classification problem. Given a dataset O, of size n with c classes, we denote by X_{100} the clustering of O that algorithm 3 generates using θ=1. The classification complexity of O, called, CC(O), is measured by the dissimilarity between X_{100} and X_{ideal}, which was introduced earlier. CC is computed as the average sum of penalties resulting from using more than c clusters to partition the dataset into clusters of 100% purity. The penalties are computed as the number of examples residing outside the first c largest cluster. How classification complexity is computed is provided as algorithm 5.

Algorithm 5: Classification Complexity

1. Inputs: X_{100}; A clustering of O into 100% purity clusters
2. n; size of input dataset O
3. c; number of classes in O
4. Output: CC; classification complexity of O

5. L← (Sort the clusters in X_{100} from largest size to smallest size)
6. penalty ←0 ; k←0; processed←0;
7. For (k = 1 to | X_{100}|)
8. C_m ← L[k]
9. L ← L\ C_m ;Remove C_m from L
10. processed ← processed + | C_m |
11. IF (k > =c)
12. penalty ← penalty + (n - processed)
13. CC ← penalty /n*(| X_{100}| - c+1)
14. Output CC

Algorithm 5 uses a list (L) to keep track of the clusters not yet processed. The number of examples that have been already processed are stored in variable "processed". The first penalty is computed when k=c (line 11, and 12). The process repeats until the last cluster has been visited. Complexity is computed by dividing the sum of penalties stored in variable "penalty" by n*(|X_{100}| - c +1) (Line 13).

Fig. 5.5 illustrates algorithm 5 for three different X_{100} clusterings. The clustering in Fig. 5.5.a is the best case scenario (X_{100}=X_{ideal}) and the clustering in Fig. 5.5.c is the worst case scenario. In Fig. 5.4.b, X_{100} is a partition which quality is between the best and the worst case scenarios.

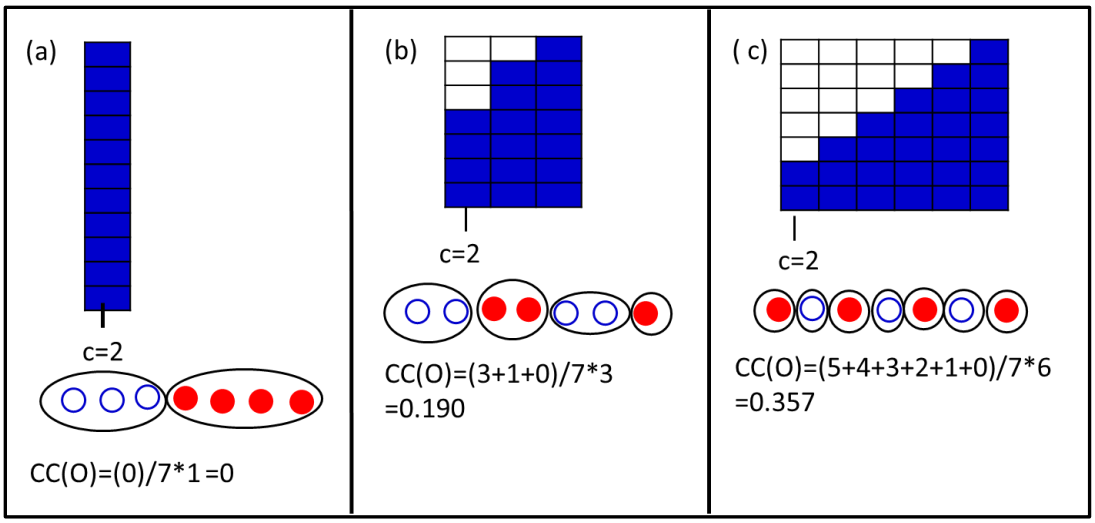


Fig. 5.5: Illustrating classification-complexity computation

Classification complexity is approximated by the area formed by the white cells divided by total area. In general when $X_{100}=X_{ideal}$, CC(O) =0; and when X_{100} is equal to worst case scenario: CC(O) = $\frac{\sum_{i=c}^{n-c} i}{n(n-c+1)} = \frac{(n-c)}{2n} < 1/2$.

5.5. Experimental Evaluation

We evaluate STAXAC against its direct competitor — the widely used taxonomy generating algorithm, Neighbor Joining algorithm (NJ) [21] (a hierarchical agglomerative clustering) — on four biological datasets using the three tree topology evaluation measures we introduced in Section 5.4. The goal of this evaluation is to determine if ST generated trees have larger sub-tree purities than traditional hierarchical agglomerative approaches, such as NJ. In addition, we evaluate the presented subclass extraction methods on eight benchmark datasets to determine the modality of the classes in the datasets. Finally, we use the same benchmark datasets to evaluate the effectiveness of the proposed classifica-

tion complexity measure by comparing classification complexity with accuracy rates of 24 classifiers, obtained from [2].

5.5.1. Supervised Taxonomy Tree Topology Evaluation

Supervised taxonomy tree topology was evaluated using four biological datasets and two artificial datasets.

5.5.1.1. Datasets

We used the datasets summarized in Table 5.2 throughout this experiment.

Table 5.2. Datasets

Dataset Name	***Description***	***Size***	***Number of Class Labels***
E.coli	Niche breadth	82	3
BE	Bacteria ecosystem class: engineered environment	120	4
BV	Bacteria ecosystem class: environmental	311	4
BH	Bacteria ecosystem class: host-associated environment	571	3
Art #1	Artificial dataset #1	100	2
Art #2	Artificial dataset #2	100	2

Distance used for distance matrices was the p-distance for all datasets. P-distance is the proportion (p) of nucleotide sites at which two sequences being compared are different divided by the total number of nucleotides compared.

Real-world datasets

E. coli: This dataset was obtained by measuring the growth of 82 strains of *Escherichia coli* in 10 distinct environments. Strains were then characterized as specialists, intermedi-

ate, or generalists depending on arbitrary divisions of the standard deviation of their growth in the environments.

Ecosystem datasets: Datasets characterize principle ecosystem type of bacteria (engineered environment, BE; environmental, BV; host-associate, BH). Ecosystem type and sequence information were downloaded from the Joint Genome Institute website [15].

Artificial datasets

The artificial datasets were generated using a random sequence generator and have a sequence length of 200. The publicly available software MEGA6 (Molecular Evolutionary Genetics Analysis version 6) [24] was used to compute the distance matrix.

Artificial dataset Art#1: The sequences of this dataset were randomly generated. Two class labels were assigned to the sequences as follows: (1) the two most distant sequences where computed and each was assigned a different class label. (2) The remaining sequences were assigned the same class label as the closest distant sequence identified in (1).

Artificial dataset Art#2: This dataset was generated in the same manner as dataset Art#1 except that 10% of the class labels were randomly assigned.

NJ trees are un-rooted trees. We defined roots of the NJ tree in the order the taxa (clusters) are being joined into their common parent. When two taxa are joined into a common parent, the parent becomes the root node of all taxa associated with its two children (subtree). The last ancestor found is thus the root of the tree.

5.5.1.2. Average Purity Per Depth

Some depths have more clusters than others for different taxonomies; therefore, we report also the percent of nodes present in each depth (for depth d, a balanced binary tree has at most 2^d nodes which is used as a base for computing the percentages). The result is provided in Fig. 5.6 for two real world datasets. In Fig. 5.6 the 0 on the horizontal axis represents the root node (depth 0). On both datasets, the result shows that STAXAC reaches high cluster purity a lot faster than NJ. For example, on the E.Coli dataset at depth 9 the clusters in the node of the STAXAC-generated tree have 100% purity while average purity in the NJ-generated tree is around 90%. The NJ tree reaches average purity of 100% at depth 14.

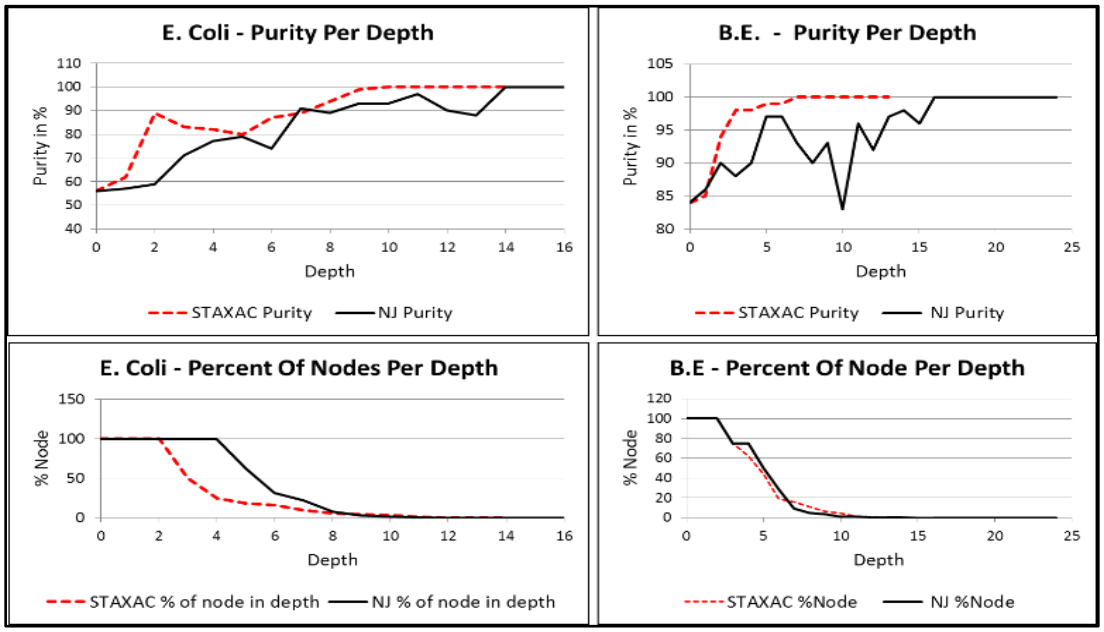


Fig. 5.6: Average purity per depth

5.5.1.3. Tree Complexity

Tree complexity result is shown in Table 5.3 with three pruning purity thresholds (θ=100%, 95%, and 85 %).

Table 5.3. Edited tree size (measured by number of nodes)

Dataset	STAXAC			NJ		
θ =>	100%	95%	85%	100%	95%	85%
E. coli	**55**	**55**	**48**	123	123	123
BE	**25**	**22**	**11**	73	73	71
BV	**111**	**51**	**51**	273	273	269
BH	**19**	**17**	**3**	75	73	65
Art#1	**15**	**15**	**12**	139	139	139
Art#2	**17**	**17**	**17**	153	153	153

The result shows that STAXAC has the best result — the fewest number of nodes — on all datasets for all three pruning thresholds (in bold).

5.5.1.4. Number of Sub-Family Changes

Table 5.4 shows the result of the number of sub-family changes for the datasets used.

Table 5.4. Number of sub-family changes

Data	STAXAC	NJ
E. coli	**18**	35
BE	**7**	21
BV	**26**	56
BH	**4**	8
Art#1	**4**	49
Art#2	**6**	61

STAXAC has the best result on all datasets (in bold). This result is expected since the NJ tree induction approach does not consider class membership information when generating taxonomies.

5.5.2. Subclass Discovery and Classification Complexity

5.5.2.1. Datasets

The datasets used in this section were obtained from UCI [28] website (Table 5.5.).

Table 5.5. Datasets

	Dataset code	No. of classes	# of records	No. input of attributes
Iris	Irs	3	150	4
Congress. Voting	Vot	2	232	16
Bupa Liver Disorder	Bld	2	345	6
Heart Disease	Hea	2	270	7
Boston Housing	Bos	3	506	12
Wisconsin Breast Cancer	Bcw	2	683	9
PIMA Indian Diabetes	Pid	2	768	7
*Silhouette	Veh	4	846	18
Image Segmentation	Seg	7	2310	19

*The Vot dataset original size was 435 examples. Rows with missing attribute values were deleted; leading to 232 examples.

5.5.2.2. Subclass Discovery Results

To test the subclass modality discovery algorithm (Algorithm 4), we used two purities thresholds θ=1, and θ=0.90 and set the minimum number of examples in a cluster to 15. We report on the number of subclasses found, how many examples they contain, clusters' purities and their majority class label — listed on the horizontal axis. The result is shown in Fig. 5.7.

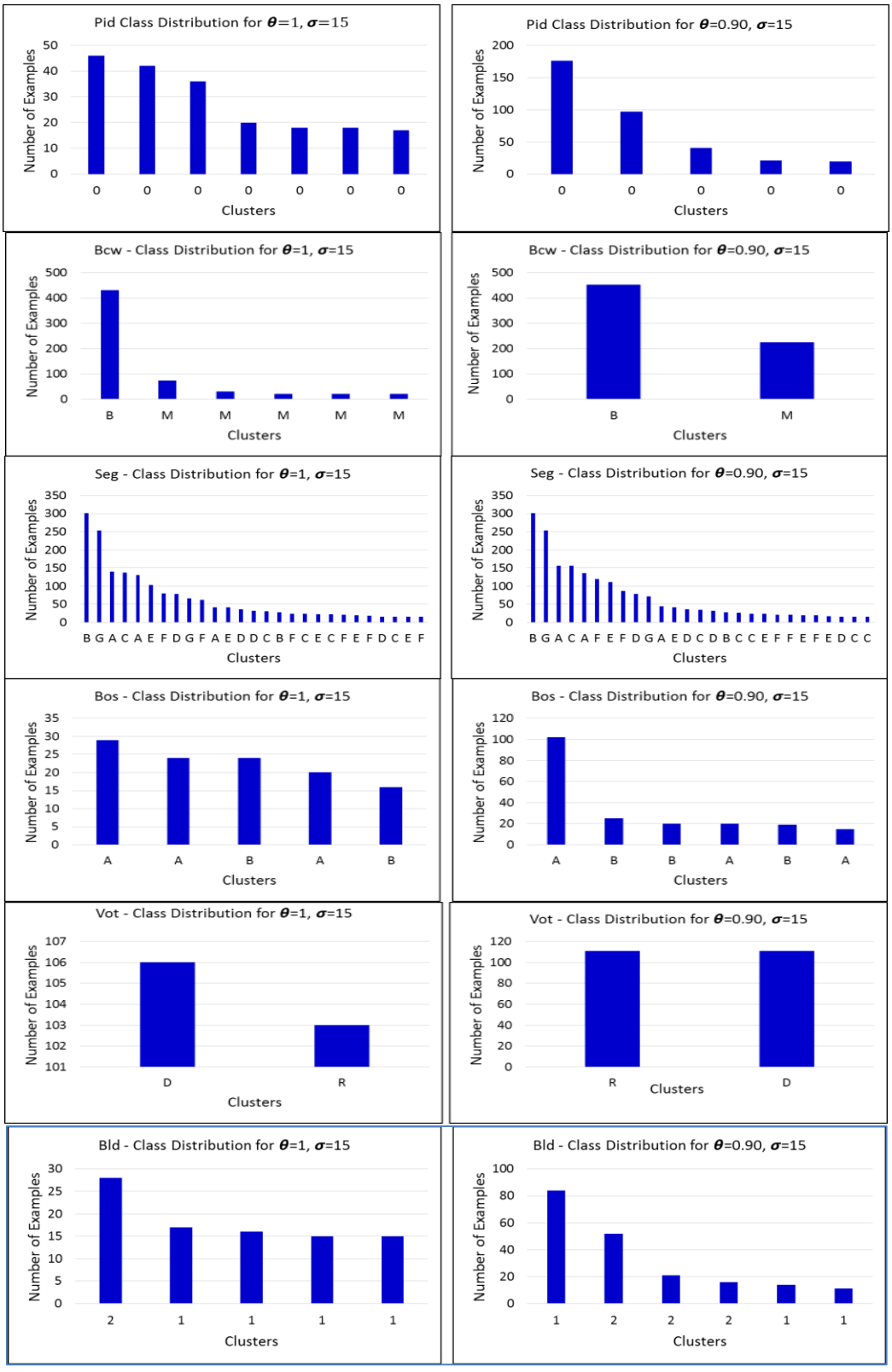

Pid Class Distribution for θ=1, σ=15
Number of Examples
50
40
30
20
10
0
0 0 0 0 0 0 0
Clusters
Pid Class Distribution for θ=0.90, σ=15
Number of Examples
200
150
100
50
0
0 0 0 0 0
Clusters
Bcw - Class Distribution for θ=1, σ=15
Number of Examples
500
400
300
200
100
0
B M M M M M
Clusters
Bcw - Class Distribution for θ=0.90, σ=15
Number of Examples
500
400
300
200
100
0
B M
Clusters
Seg - Class Distribution for θ=1, σ=15
Number of Examples
350
300
250
200
150
100
50
0
B G A C A E F D G F A E D D C B F C E C F E F D C E F
Clusters
Seg - Class Distribution for θ=0.90, σ=15
Number of Examples
350
300
250
200
150
100
50
0
B G A C A F E F D G A E D C D B C C E F F E F E D C C
Clusters
Bos - Class Distribution for θ=1, σ=15
Number of Examples
35
30
25
20
15
10
5
0
A A B A B
Clusters
Bos - Class Distribution for θ=0.90, σ=15
Number of Examples
120
100
80
60
40
20
0
A B B A B A
Clusters
Vot - Class Distribution for θ=1, σ=15
Number of Examples
107
106
105
104
103
102
101
D R
Clusters
Vot - Class Distribution for θ=0.90, σ=15
Number of Examples
120
100
80
60
40
20
0
R D
Clusters
Bld - Class Distribution for θ=1, σ=15
Number of Examples
30
25
20
15
10
5
0
2 1 1 1 1
Clusters
Bld - Class Distribution for θ=0.90, σ=15
Number of Examples
100
80
60
40
20
0
1 2 2 2 1 1
Clusters

Fig. 5.7: Subclass modality

The result shows that in general when purity is high, a lot more subclasses are discovered. For example, the Bcw dataset has only two uni-modal subclasses if the purity threshold 90% is used; however, when the purity threshold is increased to 100% the class M becomes 5-modal; it looks like the decision boundaries between the 5 subclasses are contaminated by a few examples of class B; consequently the subclasses are not merged when the purity threshold is 100%. It also can be observed that datasets Seg, Vot, and Bcw have larger size subclasses than the other datasets. For example when θ=1, Vot dataset which has two attributes, fit 90% of its examples in the two subclasses while the Bld dataset with also 2 attributes was able to fit only 26.38% of its examples in 5 clusters. It also can be seen that all clusters of the Pid dataset are dominated by class 0, and we can conclude from that, that the dataset does not contain any region dominated by the instances of the other classes in the dataset.

5.5.2.3.Classification Complexity

Classification complexity was computed for the 8 benchmark datasets (Table 5.6). We compared our CC results with the accuracy rate of 24 classifiers (Fig. 5.8) (obtained from [2]). The computed classification complexity values are shown encircled in Fig. 5.8 where the y-axis is the average accuracy rate while the x-axis contains the dataset. It can be observed that datasets with low classification complexity have been classified with high accuracy rate by the majority of the classifiers. On the other hand, datasets with high classification complexity values have been classified with low accuracy rate. In addition, in Fig. 5.9.a we show that there is a strong negative correlation of -0.94 between the proposed classification complexity and the average accuracy obtained on the datasets which suggests that our proposed classification complexity adequately provides a measure of the difficulty for learning from a given dataset.

Table 5.6. Twenty four classification approaches [2]

Abbrev.	Algorithm	Abbrev.	Algorithm
QU0	QUEST Versions (Loh,Shih 1997)	IC0	CART versions (Breiman,Friedman,Olsen,Stone 1984)
QU1		IC1	
QL0		0CU	OC1 versions (Murthy, Kassif,Salzberg 1994)
QL1		0CL	
FTU	FACT Versions (Loh,Vanichsetakul 1998)	0CM	
FTL		ST0	S-PLUS versions (Clark,Pregibon 1993)
C4T	C4.5 (Quinlan1993)	ST1	
C4R		IB	IND versions (Buntine 1992)
LMT	LMDT (Brodley,Utgoff 1995)	IBO	
CAL	CAL5 (Müller,Wysotzki 1997)	IM	
T1	T1 single split (Holte 1993)	IM0	

CLMIX CLLDA	CLINE (Fatih,Okan 2008)

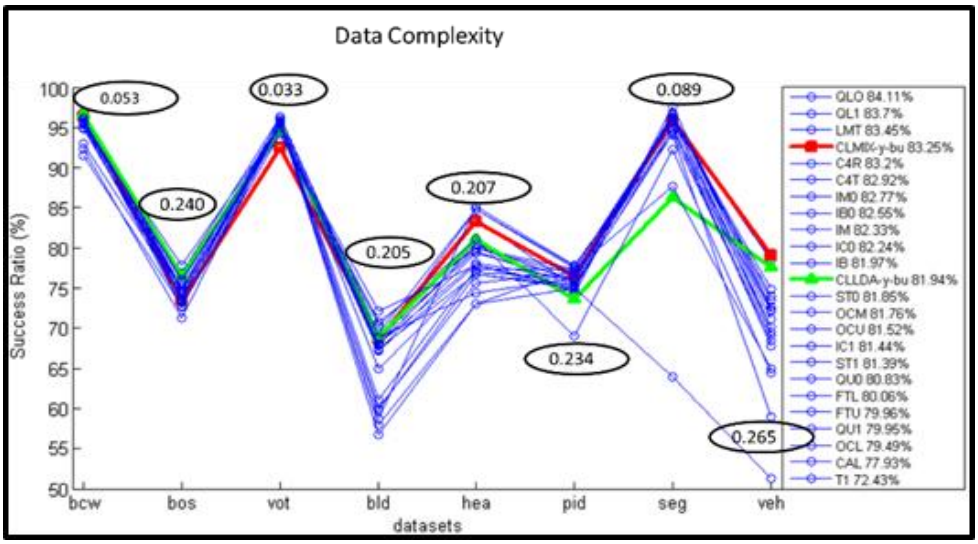


Fig. 5.8: Classification complexity and accuracy rate

In Fig. 5.9.b we generated a clustering X using θ=1 (X_{100}). We computed the average cluster size for each dataset by dividing total number of examples in dataset by number of generated clusters ($|X_{100}|$) and report on the relationship between the average cluster size in the dataset and classification complexity. In Fig. 5.9.b the values of CC have been multiplied by 100 (for better graphical display; clusters were extracted using θ=1). It can be observed that Bcw and Vot datasets which have the lowest complexity values have large average size cluster (above 15 examples per cluster), Seg dataset (with average complexity value) has on average 10 examples per subclass while the remaining datasets with high complexity have below 5 examples per cluster. This suggests that when a dataset has large high purity clusters, its CC is low; meaning the dataset is easy to classify. We also computed the correlation between average cluster size and average accuracy

rate. The high correlation coefficient of -0.97 further confirms that datasets with large high purity clusters are easier to classify than datasets with small high purity clusters.

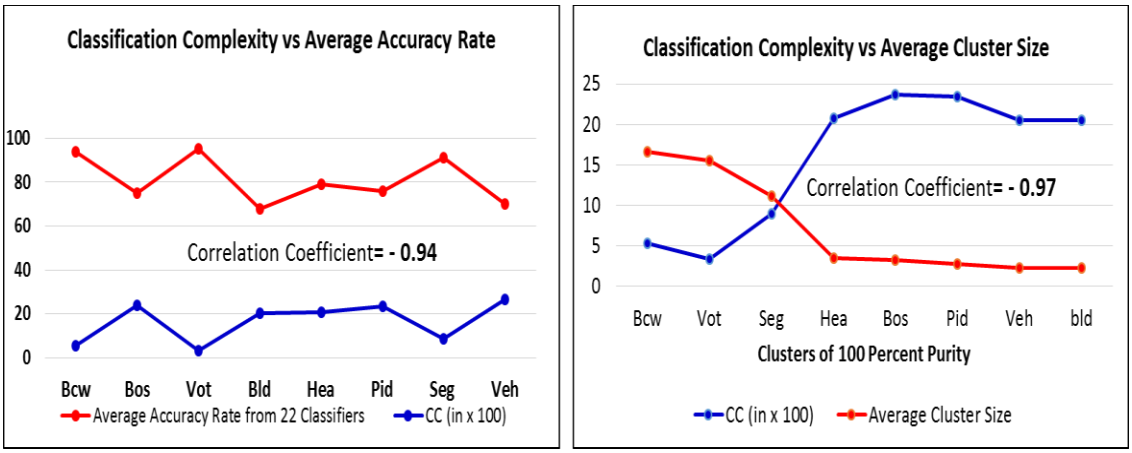
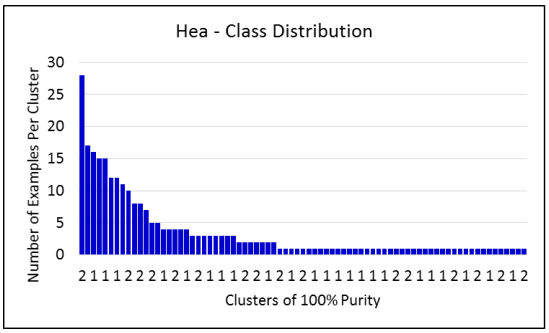
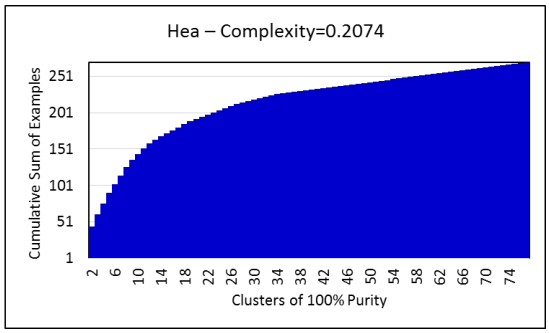
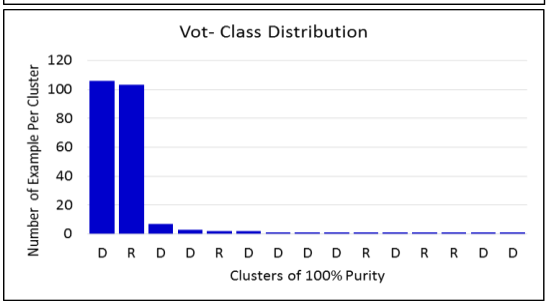


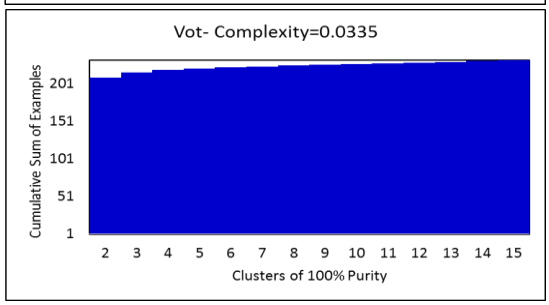
Fig. 5.9: (a) CC and average accuracy rate; (b) Average size of subclasses and classification complexity

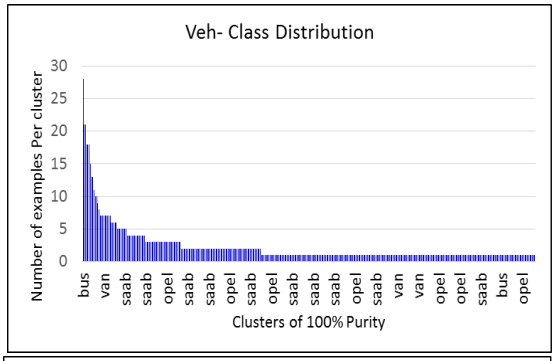
Fig. 5.10 provides additional evidence that when the dataset contains a lot of small size clusters (or no subclass at all) complexity is high. In Fig. 5.10, the clusters are 100-percent pure and sorted from largest size cluster to the smallest. The x-axis represents the clusters and their class labels. The left column of Fig. 5.10 shows the class distribution while the right column shows the complexity measures approximated by the white area of the rectangles. Given a dataset with c classes, the first (c-1) largest clusters are not represented in Fig. 5.10 since CC is computed using the c^{th} largest cluster and the remaining clusters of smaller sizes in the dataset (refer to section 5.4.3). The Bcw, Irs, and Vot datasets that have small white areas in their rectangles (large size subclasses) have low classification complexity whereas the rest of the datasets with larger white areas have higher classification complexity.

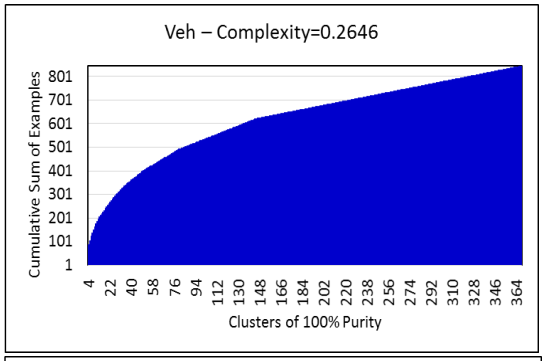

Hea - Class Distribution
Number of Examples Per Cluster
30
25
20
15
10
5
0
2 1 1 1 2 2 2 1 2 1 2 1 2 1 1 1 2 2 1 2 1 2 1 2 1 1 1 1 1 1 2 2 1 1 1 2 1 2 1 2 1 2
Clusters of 100% Purity

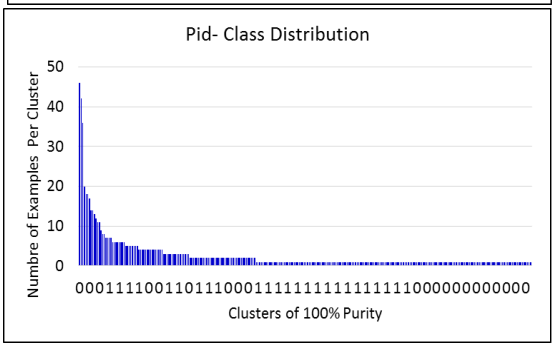

Hea – Complexity=0.2074
Cumulative Sum of Examples
251
201
151
101
51
1
2 6 10 14 18 22 26 30 34 38 42 46 50 54 58 62 66 70 74
Clusters of 100% Purity

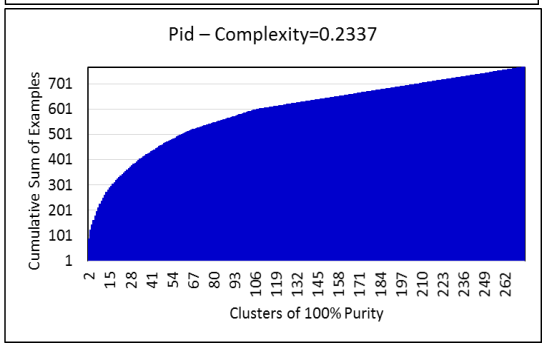

Vot- Class Distribution
Number of Example Per Cluster
120
100
80
60
40
20
0
D R D D R D D D D R D R R D D
Clusters of 100% Purity

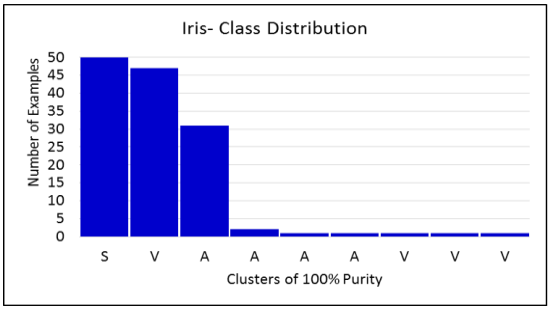

Vot- Complexity=0.0335
Cumulative Sum of Examples
201
151
101
51
1
2 3 4 5 6 7 8 9 10 11 12 13 14 15
Clusters of 100% Purity

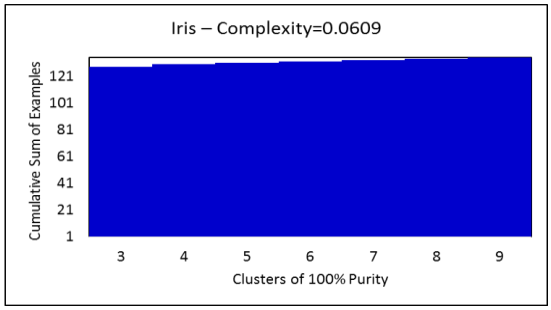

Veh- Class Distribution
Number of examples Per cluster
30
25
20
15
10
5
0
bus van saab saab opel saab saab opel saab opel saab saab opel saab van van opel opel saab bus opel
Clusters of 100% Purity


Veh – Complexity=0.2646
Cumulative Sum of Examples
801
701
601
501
401
301
201
101
1
4 22 40 58 76 94 112 130 148 166 184 202 220 238 256 274 292 310 328 346 364
Clusters of 100% Purity


Pid- Class Distribution
Numbre of Examples Per Cluster
50
40
30
20
10
0
0001111001101110001111111111111111000000000000
Clusters of 100% Purity


Pid – Complexity=0.2337
Cumulative Sum of Examples
701
601
501
401
301
201
101
1
2 15 28 41 54 67 80 93 106 119 132 145 158 171 184 197 210 223 236 249 262
Clusters of 100% Purity


Iris- Class Distribution
Number of Examples
50
45
40
35
30
25
20
15
10
5
0
S V A A A A V V V
Clusters of 100% Purity


Iris – Complexity=0.0609
Cumulative Sum of Examples
121
101
81
61
41
21
1
3 4 5 6 7 8 9
Clusters of 100% Purity

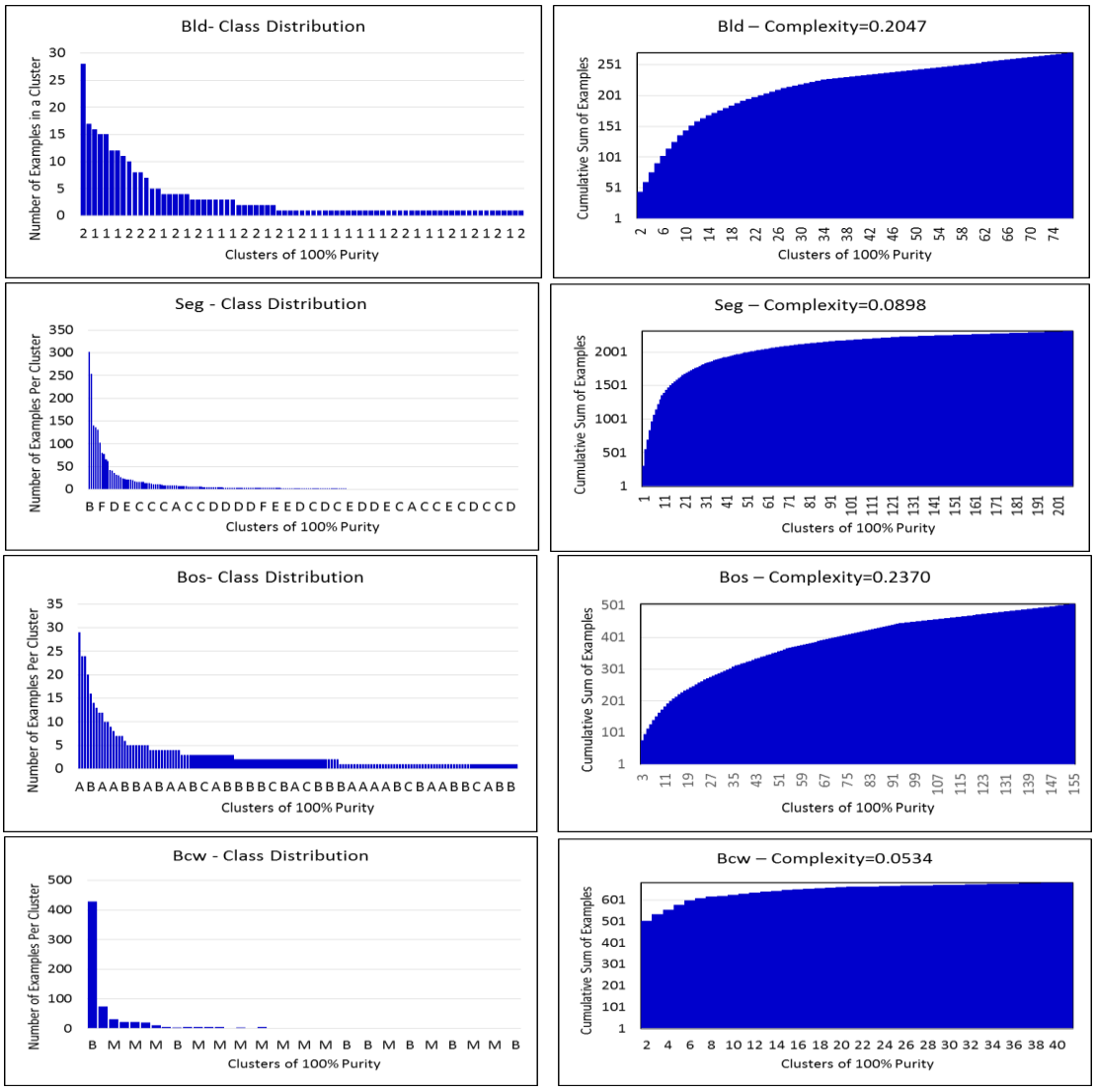


Fig. 5.10. Class distribution and complexity results for 8 benchmark datasets and the Iris dataset

Fig. 5.11 further illustrates our complexity measure with additional artificial datasets. The datasets used are two-class datasets with two attributes.

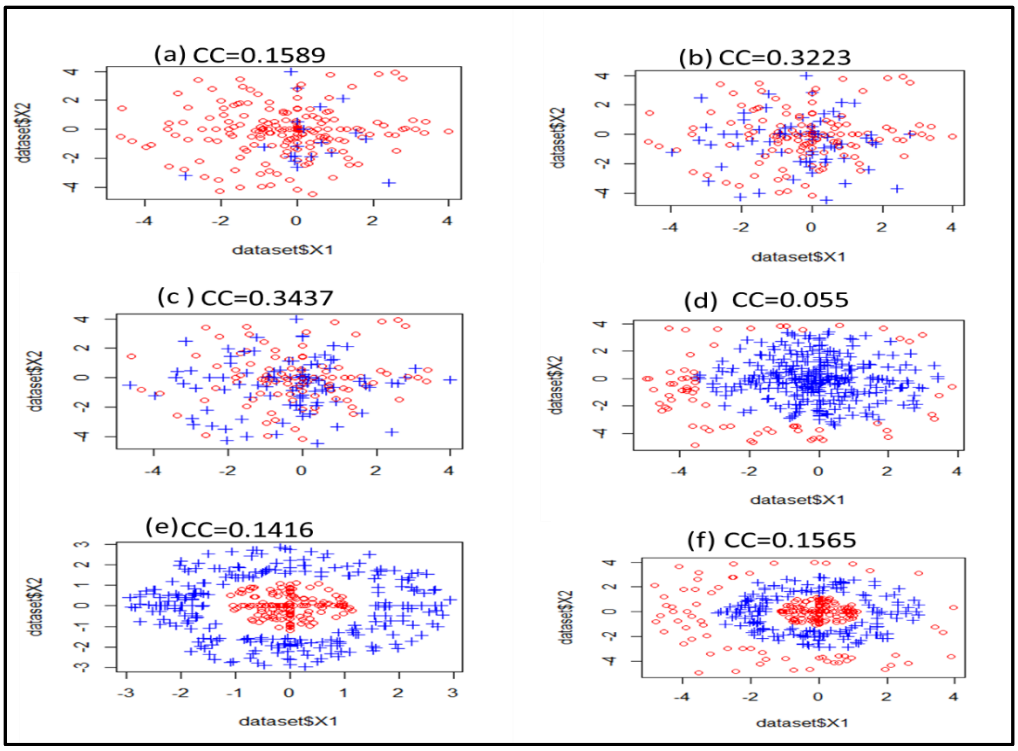


Fig. 5.11. Illustrating classification complexity for some artificial datasets

In general when the majority of the examples are located in few large clusters of high purity, CC is low (Fig. 5.11.a, Fig. 5.11.d, Fig. 5.11.e and Fig. 5.11.f). In contrast, Fig. 5.11.b, and Fig. 5.11.c where there is no large region dominated by a particular class CC is high.

5.6. Conclusion

This chapter introduced a new type of taxonomy called *supervised taxonomy* (ST). Supervised taxonomies are generated considering background information in form of class labels in addition to distance metrics, and are capable of capturing class-uniform regions in a dataset. By analyzing the generated tree structure, a biologist can interpret the clustering result and gain insights into how the biological groupings are related, taking more

a supervised point of view. We also introduced, STAXAC, a novel, supervised, hierarchical clustering algorithm to generate supervised taxonomies; it conducts a "wider" search for the best pair of clusters to merge while maximizing purity, enabling the algorithm to obtain sub-trees of higher purities than traditional agglomerative methods.

We also adapted existing and introduced novel meta-learning methods that operate on top of STs. Measures and algorithms that assess the edited tree complexity, subclass changes, average purities at different tree-depths, classification complexity, and modality of classes have been proposed in this research, demonstrating the merit of STs, particularly as an exploratory data analysis tool. We claim that the latter two contributions are not only novel but useful. We demonstrated in our experimental evaluation that assessing the classification complexity of a ST, provides a good estimate of the difficulty of the classification problem at hand. Moreover, a class modality assessment tool has been provided that — based on a domain expert's notion of what constitutes a "noteworthy" subclass — determines if specific classes in the dataset are zero-modal, unimodal, and multi-modal; shedding light on presence of interesting subclasses in the dataset.

As this research introduces ST for the first time, we believe there are many other useful things that can be developed on top of supervised taxonomies, which is the focus of our future work. Moreover, as taxonomies are very popular in bioinformatics, we plan to assess the usefulness of STs for scientific discovery in bioinformatics in more depth.

References

1. Paul K. Amalaman and Christoph F. Eick.;*"HC-edit: A Hierarchical Clustering Approach To Data Editing"*; Proceding of 22nd International Symposium on Methodologies for Intelligent Systems 2015.

2. M. Fatih Amasyali and Okan Ersoy; *"Cline: A New Decision-Tree Family"*; IEEE Transaction on Neural Networks, Vol. 19, No. 2, February 2008.

3. P Awasthi, RB Zadeh; *"Supervised clustering"*; Advances in Neural Information Processing Systems, 2010, P. 91-99.

4. A. Bagherjeiran and C. F. Eick; *"Distance Function Learning for Supervised Similarity Assessment"*; book chapter in P. Perner (eds.): Case-Based Reasoning in Signals and Images, Springer Verlag, 2008.

5. Maria-florina Balcan , Avrim Blum; *"Clustering with Interactive Feedback"*; In Algorithmic Learning Theory, 316-328,2008.

6. Basu, S., Banerjee, A., Mooney, R. J.; *"Semi-supervised Clustering by Seeding"*; Proceedings of the Nineteenth International Conference on Machine Learning (ICML-2002), pp. 19-26, Australia, July 2002.

7. Basu, S., Bilenko,M., Mooney, R. *"Comparing and Unifying Search-based and Similarity-Based Approaches to Semi-Supervised Clustering"*, in Proc. ICML03 Workshop on The Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining, pp. 42-29, Washington DC, August 2003.

8. Cohn, D., Caruana, R., McCallum, A.; *"Semi-supervised Clustering with User Feedback"*; Technical Report, available at https://ecommons.cornell.edu/handle/1813/5608 (Cornell University).
9. Hal Daume III and Daniel Marcu; *"A Bayesian Model for Supervised Clustering with the Dirichlet Process Prior"*; Journal of Machine Learning Research 6 (2005) 1551–1577.

10. Eduardo Costa, Celine Vens and Hendrik Blockeel; *"Top-Down clustering of protein subfamily identification*"; Proceeding of Evolutionary Bioinformatics 2013:9 185-202.

11. Eick, C.F., Zeidat, N., and Zhenghong, Z; *"Supervised Clustering – Algorithms and Benefits"*; Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence 2004, pp. 774-776.

12. C. F. Eick, B. Vaezian, D. Jiang, and J. Wang; *"Discovery of Interesting Regions in Spatial Datasets Using Supervised Clustering"*; Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD), Berlin, Germany, September 2006.

13. Eick, C. F., and N. Zeidat; *"Using Supervised Clustering to Enhance Classifiers"*; Procedure 15th International Symposium on Methodologies for Intelligent Systems (ISMIS), New York, pp. 248-256, May 2005.

14. Thomas Finley and Thorsten Joachims; *"Supervised Clustering with Support Vector Machines"*; Proceeding ICML '05' Pages 217 – 224.

15. The Joint Genome Institute, https://img.jgi.doe.gov/cgi-bin/w/main.cgi (as of April 2015).

16. Tin Kam Ho and Mitra Basu; *"Complexity Measure of Supervised Classification Problems"*; IEEE transactions on pattern analysis and Machine Intelligence, Vol. 24, No. 3. March 2002.

17. Jiye Liang, Xingwang Zhao, DeyuLi, Fuyuan Cao, Chuangyin Dang; *"Determining the number of clusters using information entropy for mixed data"* ; Journal of Pattern Recognition 45 (2012)2251–2265).

18. Li, M. and Vitanyi, P. (1997); *"An Introduction to Kolmogorov Complexity and Its Applications"*; Springer-Verlag, 2nd edition.

19. Ling Li and Yaser S. Abu-Mostafa; *"Data Complexity in Machine Learning"*; Learning Systems Group, California Institute of Technology.

20. R. Sibson; *"SLINK: an optimally efficient algorithm for the single-link cluster method"*; The Computer Journal (British Computer Society) 16 (1): 30–34.

21. Saitou, N., Nei., M.;*" The neighbor-joining method: a new method for reconstructing phylogenetic trees"*; Mol. Biol. Evol. 4(4) (1987).

22. Tomoya Sakai, Atsushi Imiya; *"Unsupervised cluster Discovery using statistics in scale space"*; Journal Engineering Applications of Artificial Intelligence 22 (2009)92–100.

23. Mitra Basu and Tin Kam Ho; *"Data Complexity in Pattern Recognition"*; Springer ISBN-13: 978-1846281716.

24. Koichiro Tamura, Glen Stecher, Daniel Peterson, Alan Filipski, and Sudhir Kumar (2013) MEGA6: Molecular Evolutionary Genetics Analysis version 6.0. Molecular Biology and Evolution: 30 2725-2729. http://www.megasoftware.net/ (as of October 2014).

25. R. Vilalta, M. Achari, and C. F. Eick; *"Class Decomposition via Clustering: A New Framework for Low-Variance Classifiers"*; Proceedings of the IEEE International Conf. on Data Mining (ICDM), pp. 673-676, 2003.

26. Swee Chuan Tan, Kai Ming Ting, Shyh Wei Teng; *"A general stochastic clustering method for automatic cluster Discovery"*; Journal Pattern Recognition 44 (2011) 2786–2799.

27. Wolpert, D. H. and Macready, W. G. (1999); *"Self-dissimilarity: An empirically observable complexity measure"*; In Bar-Yam, Y., editor, Unifying Themes in Complex Systems, pages 626-643. Perseus Books.

28. UCI repository at http://archive.ics.uci.edu/ml/datasets.html as of 12/04/2012.

Chapter 6

HC-edit: A Hierarchical Clustering Approach To Data Editing[4]

6.1. Introduction

One of the most popular classification techniques is the k-nearest neighbor method (k-NN) which assigns the majority class label in the k nearest neighbor set to a point that needs to be classified [1]. The basic k-NN has the advantage of being easy to implement but also requiring a large memory to store the model — which is the training set. Additionally, the classifier is sensitive to atypical examples whose presence in the training set may lead to poor accuracy and unnecessary storage of examples [7]. Consequently, most k-NN based approaches deal with these two issues using a technique known as "condensing" and "editing" [2]. Condensing aims at reducing a classifier's training time while achieving no degradation in classification accuracy. Editing, on the other hand, seeks to remove noisy examples from the original dataset with the goal of improving classification accuracy.

This chapter focuses on a new editing method, called HC-edit which takes as input an agglomerative clustering —a tree of clusters— to "clean up" the training set. The tree stores in its nodes the purities of the node clusters. The tree can be generated by traditional agglomerative hierarchical clustering algorithm such as the popular UPGMA (Un-

[4] Published in Proceeding of International Symposium on Methodologies for Intelligent Systems (ISMIS); Lyon, France, October, 2015.

weighted Pair Group Method with Arithmetic Mean) which merges the closest pairs first, and the distant examples/clusters last or by a supervised taxonomy algorithm such as STAXAC (Supervised Taxonomy Agglomerative Clustering) which was discussed in chapter 5 (section 5.3). When traditional agglomerative method is used, purity information for the node datasets must be computed and stored after the tree is built. To edit the training set, a minimum purity threshold β has to be selected; next, HC-edit retrieves the clusters whose purities are greater or equal to β such that the union of all the retrieved clusters equals the input dataset and each example appears only in one cluster. Then the minority class examples are removed from the clusters. Finally, the k-NN rule is used to classify unlabeled examples using the edited dataset. The advantage of using a hierarchical clustering is that hierarchical clustering computes all needed clusters in advance, whereas an ordinary clustering algorithm has to be rerun for different purity thresholds. Fig. 6.1 illustrates the steps of HC-edit editing process. Fig. 6.1.a-1 illustrates a hypothetical dataset and selected clusters which purities are above a user-defined threshold β. Fig. 6.1.a-2 shows the resulting clusters after the minority examples have been removed. Different purity thresholds yield different decision boundaries. Fig. 6.1.b-1 illustrates another cluster selection for a different user-defined purity threshold from the same training set and Fig. 6.1.b-2 the corresponding result after editing.

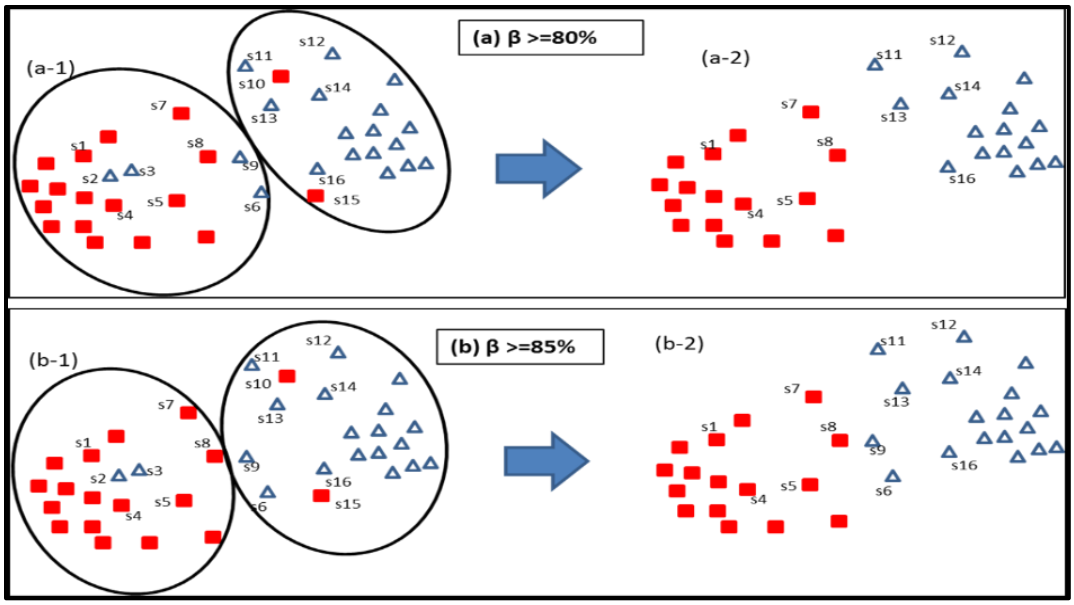


Fig. 6.1. Different purity thresholds yields different decision boundaries

The contributions of this research are the following:

1. The data editing method of previous approaches visits each example and analyzes local information, namely the k-nearest neighbors of the example to decide if the example should be removed or kept. HC-edit, on the other hand, is a more 'regional' approach in that it looks for regions of high purity in the dataset and removes the minority class examples from the regions.
2. The proposed method has the capacity to remove fewer points than widely known editing approaches which can be beneficial to the classifier accuracy rate. Many editing approaches tend to smooth the boundaries between clusters even if the clusters do not overlap. Excessive examples removal, especially, in the boundary regions may lead to a decrease in classifier accuracy since the chances of misclassification increase due to a widened gap between the clusters.

To illustrate the second claim, let us consider the dataset in Fig.6.1.a and Fig 6.1.b with border points s7, s8, s9, and s6 where s8 and s9 are nearest neighbor of each other. Fig. 6.1.a-1 and Fig. 6.1.b-1 show non overlapping clusters obtained from HC-edit of same dataset. If k=1, current approaches will remove s8 and s9. On the other hand, HC-edit would discard either s9 or s8, or none but not both. In Fig. 6.1.a-2 s9 is removed but s8 is kept. In Fig. 6.1.b-2 s8 and s9 are kept in the edited dataset.

6.2. Related Work

The motivation for editing the training set resides in the fact that the k-NN classifier achieves higher accuracy when the training set is rid of atypical and mislabeled examples; examples belonging to minority class label in comparison to other examples in the local region [7]. Editing also gets rid of the overlapping region examples.

Given an example x, a training dataset D, the k-neighborhood of x, noted $N_k(x)$, consisting of its k nearest neighbors can formally be defined as

$N_k(x) \subseteq D; | N_k(x) | = k \ (k>0)$

$\forall s \in N_k(x), q \in D \backslash N_k(x) \rightarrow d(s,x) \leq d(q,x) \quad (q \in D \wedge q \notin N_k(x))$

The k-NN rule

Given $N_k(x)$, the k-NN rule for x can be defined as follows:

Assign to x the class label of the majority class label in $N_k(x)$.

This rule gives equal weight to each example's vote.

The weighted k-NN rule

Given $N_k(x)$, we define a weighted k-NN rule as follows:

Assign a weight to each nearest neighbor's vote which is inversely proportional to its distance to x. Sum the vote by class label. The class label with the highest score is assigned to x. Each class score is computed as

$$V(x, c) = \sum_{j=1}^{k} \frac{p(x_j,c)}{(1+d(x,x_j))}$$

where $x_j \in N_k(x)$ and $p(x_j, c) = 1$ if x_j is labeled with c and $P(x_j, c) = 0$ otherwise.

Several editing approaches have been proposed; the most important ones are briefly discussed in the remainder of this section. The pseudo code for each approach is provided in Fig. 6.2.

Wilson Editing

Wilson editing [8] relies on the idea that if an example is erroneously classified using the k-NN rule it has to be eliminated from the training set.

Multi-Edit

Devijver and Kittler [3] proposed the Multi-edit technique. The algorithm repeatedly applies Wilson editing to m random subsets of the original dataset until no more examples are removed.

Supervised Clustering Editing

In supervised clustering editing [4], a supervised clustering algorithm is used to cluster a dataset D. Then D is replaced with a subset consisting of cluster representatives. Supervised clustering [5] deviates from traditional clustering in that it is applied on classified examples with the objective of identifying clusters with high probability density with respect to a single class. However, it does not organize the clusters in hierarchical fashion as HC-edit; it output ordinary clusters maximizing a fitness function. When the parameters to the fitness function change the clustering algorithm needs to regenerate the clusters. With HC-edit, the tree generation is independent of the parameter β —which is only used to select the clusters.

WilsonProb

This method [9] edits the training set based on a probability of an example to belong to a certain class in its neighborhood. The estimated probability is the weighted k-NN rule.

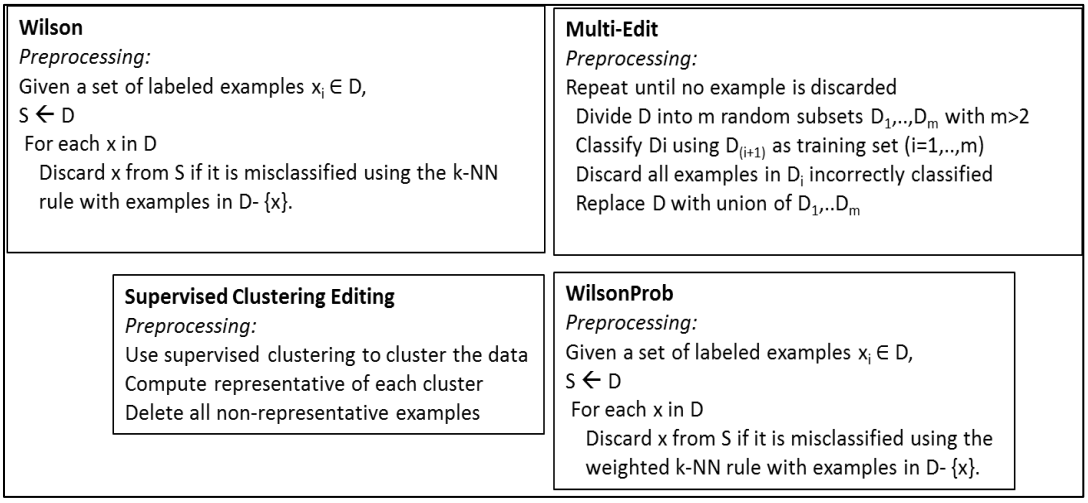


Fig. 6.2. Editing algorithms

After the preprocessing, a new example is classified using k-NN rule.

6.3. The HC-edit Approach

The proposed method uses STAXAC, a supervised taxonomy clustering algorithm which was introduced in chapter 5. A user-defined purity threshold β is then used to retrieve clusters which purities are above the threshold.

The HC-edit pseudo code is as follows:

Preprocessing:

1. Inputs:
2. T: tree generated by hierarchical clustering algorithm such as STAXAC (with
3. purity information in the node) for a dataset O
4. β: cluster purity threshold
5. Output: O_{edited}, a dataset that is a subset of the original dataset O (to be used by the kNN-classifier)
6. Function EDIT_TREE (T)
7. $O_{edited} \leftarrow \varnothing$;
8. *EXTRACT_EXAMPLES*(T);
9. RETURN O_{edited};
10. END
11. *Function EXTRACT_EXAMPLES (T)*
12. BEGIN
13. IF T=NULL EXIT

14. IF *purity(T)* >= β
15. Add majority examples of T to O_{edited}
16. ELSE
17. *EXTRACT_EXAMPLES (T.right)*
18. *EXTRACT_EXAMPLES (T.left)*
19. END

Because the algorithm starts the search from the root, it returns the largest cluster with purity equal or greater than the threshold in the selected branch of the tree. It can be observed that if Wilson editing is applied on the dataset presented in Fig. 6.3, x_3 and x_4 will be removed creating a wider gap between the clusters which may lead to potential misclassification. Secondly, although there are unlimited choices for β, most values return identical cluster sets. The set, β_set, composed of all the node purities, contains potential values for β. Fig. 6.3 illustrates the trees obtained for the dataset depicted in Fig. 6.3 by both clustering approaches (β_set = {60,100} for STAXAC tree and β_set ={60, 66.67, 50, 100} for hierarchical agglomerative clustering (HAC).

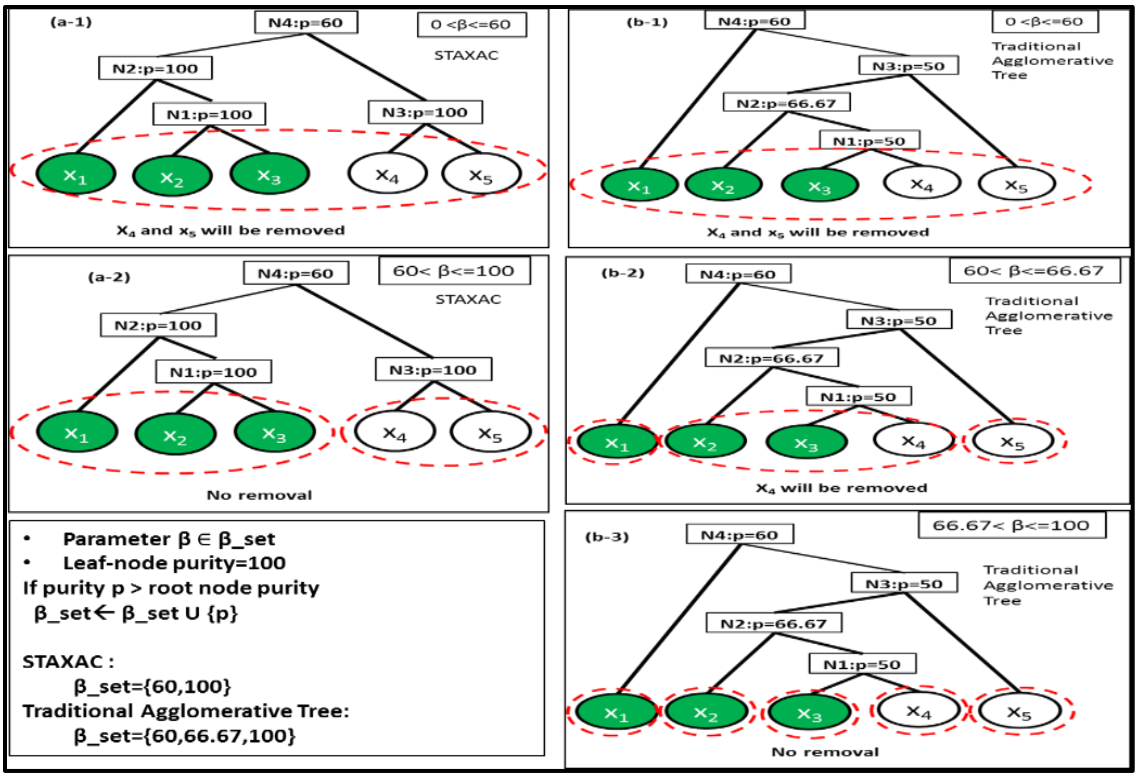


Fig. 6.3. Purity-parameter selection and its impact on the editing result

If user queries the tree with 0< β<=60 both trees return the entire dataset (Fig. 6.3. a.1 and Fig. 6.3.b.1); in that case minority examples x_4 and x_5 will be removed. On the hand, if β > 60, the tree generated by STAXAC returns two 100% purity clusters (Fig. 6.3.a.2). If 60<β<=66.67, the search in the traditional tree returns three clusters (Fig. 6.3.b.2); only x_4 will be removed by editing. If β > 66.67 five singleton clusters are returned; in this case the edited dataset is identical to the original dataset (Fig. 6.3.b.3). Overall, the editing result is influenced by the choice of β. The value 100 is always element of β_set (leaf-nodes have 100% purities). When β=100 the edited dataset is identical to the original dataset (no removal since all clusters are pure). As β decreases, more minority examples are removed.

How to select β?

If β is set to 1, no editing occurs. As β decreases more objects will be removed in the editing process, and the obtained clusters contain less instances. We also observed that using very low values β leads to low accuracy, as clusters get too contaminated by examples belonging to other classes. Moreover, only a finite subset of β values needs to be considered: a subset of the node purities that occur in the supervised taxonomy. Basically, only a finite number of clusterings can be extracted from the supervised taxonomy and for many purity values the extracted supervised clusterings are the same. Therefore, we set a lower bound γ for β (which we chose the root node purity) and we use n-fold cross-validation for purities that occur in the tree in $[\gamma,1)$, and choose the purity value that leads to the highest accuracy for the editing of the dataset for a given k value.

6.4. Experimental Evaluation

HC-edit by default uses as input, trees generated by STAXAC to edit the dataset. We implemented a variant that uses tree generated by the widely used traditional agglomerative hierarchical method UPGMA. To distinguish both we denote them by HC-edit-STAXAC and HC-edit-UPGMA throughout the remaining of the chapter. Additionally, we implemented the Wilson editing, and the WilsonProb algorithms. The performances of the algorithms with respect to accuracy were evaluated on seven real world datasets.

6.4.1. Datasets

We used the datasets summarized in Table 6.1 throughout the experiments.

Table 6.1. Datasets

Dataset Name	***Description***	***Size***	***Number of Class Labels***
E.coli	Niche breadth	82	3
BEE	Bacteria ecosystem class: engineered environment	120	4
BEV	Bacteria ecosystem class: environmental	311	4
AEV	Archaea ecosystem type: environmental	571	3
Bos	Boston Housing	506	3
Bld	Bupa Liver Disease	345	2
Vot	Congress	232	2

Joint Genome Institute Datasets:

Distance used for distance matrices was the patristic distance.

E. coli: This dataset was obtained by measuring the growth of 82 strains of *E. coli* in 10 distinct environments. Strains were then characterized as specialists, intermediate, or generalists depending on arbitrary divisions of the standard deviation of their growth in the environments.

Ecosystem datasets: Datasets characterize principle ecosystem type of bacteria (engineered environment, BEE; environmental, BEV) and archaea (environmental, AEV).

Ecosystem type and sequence information were downloaded from the Joint Genome Institute website [10].

UCI Datasets [6]: All datasets were preprocessed into dissimilarity matrices before the experiments. Dissimilarity matrices were generated using z-scores (except for the Vot dataset which has attributes with binary values).

6.4.2. Results

The 10-fold cross-validation method (90% of the original instances have been used as the training set and 10% for test purposes) has been employed to estimate the overall classification accuracy. Fig. 6.4 reports on the experimental results obtained by the different algorithms over the 7 datasets. These results have been averaged over the ten partitions.

Bold figures indicate the best method in terms of classification accuracy for each dataset. The largest compression values are in italic. All k values tried out during classification and training phase are reported (HC-edit uses k for classification only). The result are presented as "x (y)[z]" where x is the accuracy rate, y is the purity parameter β (for HC-edit) and z the compression rate. For HC-edit multiple values of β were run for a given k and best results are reported. Whenever more than one (y)[z] are reported they are separated by commas. With respect to accuracy rate, we observe that the plain k-NN approach has overall best performance on one dataset (Bos). Wilson has best performance on two datasets (BEV, and BEE). WilsonProb wins best performance on one dataset (BEV). HC-edit-STAXAC wins 4 times (Ecoli, AEF,Vot and Bos) and HC-edit-UPGMA wins 2 times (Bld, and Bos).

	E. Coli				
	HC-edit-STAXAC	HC-edit-UPGMA	WilsonProb	Wilson	K-NN
K=1	56.1 (50) [43.20]	47.56 (100)[0.00],(90)[0.00]	57.32 *[49.73]*	57.32 *[49.73]*	51.22
K=3	56.10 (100)[0],(90)[1.06],(80)[8.13]	52.44 (100)[0.00],(90)[0.00]	57.32 [44.53]	57.32 [45.73]	56.10
K=5	58.54 (100)[0.00],(90)[10.67]	53.66 (100)[0.00],(90)[0.00]	51.22 [42.40]	51.22 [41.86]	54.88
K=7	**63.41** (100)[0.00],(90)[10.67]	62.20 (80)[4.00]	52.44 [43.20]	54.88[46.80]	56.1
	BEV				
	HC-edit-STAXAC	HC-edit-UPGMA	WilsonProb	Wilson	K-NN
K=1	81.35 (90)[6.00]	82.64 (80)[14.17]	**83.6** *[19.96]*	**83.6** *[19.96]*	79.1
K=3	81.03 (90)[6.00]	81.99 (80)[14.17]	82.64 [16.92]	81.79 [16.75]	81.67
K=5	81 (100)[0.00]	83.28 (80)[14.17]	81.67[16.50]	79.74 [18.60]	80.39
K=7	81.35 (90)[6.00]	83.28 (80)[14.17]	81.35[16.25]	80.06 [18.60]	80.39
	Vot				
	HC-edit-STAXAC	HC-edit-UPGMA	WilsonProb	Wilson	K-NN
K=1	91.89 (100)[0.00]	90.09 (90)[5.80],(80)*[10.9]*	90.95 *[9.00]*	90.95 *[9.00]*	89.66
K=3	91.38 (100)[0.00]	91.38 (90)[5.80]	90.95 [8.00]	90.95 [8.00]	92.24
K=5	92.67 (100)[0.00],(90)[5.38]	92.24 (100)[0.00],(90)[5.80]	90.95 [8.00]	90.95 [8.00]	91.38
K=7	**93.10** (100)[0.00]	92.67 (100)[0.00]	90.95 [8.00]	90.95 [8.00]	92.67
	Bld				
	HC-edit-STAXAC	HC-edit-UPGMA	WilsonProb	Wilson	K-NN
K=1	61.11 (100)[0.00]	62.09 (80)[7.15]	61.44 *[39.26]*	61.44 *[39.26]*	60.13
K=3	59.80 (100)[0.00]	59.80 (90),(80)[7.15]	59.48 [35.64]	59.48 [35.60]	59.48
K=5	63.07 (90)[3.21]	62.75 (100)[0.00],(90)[0.6]	59.8 [32.00]	59.8 [31.73]	63.4
K=7	64.05 (90)[3.21]	**64.38** (100)[0.00],(90)[0.6]	61.44 [29.48]	61.44 [29.37]	64.05
	AEV				
	HC-edit-STAXAC	HC-edit-UPGMA	WilsonProb	Wilson	K-NN
K=1	**80** (80)[10.50]	78.57 (80)[5.6]	77.14 *[25.61*]	77.14 *[25.61]*	77.14
K=3	78.57 (90)[2.50],(80)[10.50]	78.57 (100)[0.00],(90)[0.7],(80)[5.6]	72.86 [18.25]	74.29 [16.84]	78.57
K=5	**80** (90)[2.5]	77.14 (100)[0.00],(90)[0.7],(80)[5.6]	72.86 [18.60]	71.43 [17.89]	78.57
K=7	78.57 (100)[0.00],(90)[2.5],(80)[10.5]	78.57 (100)[0.00],(90)[0.7],(80)[5.6]	71.43 [18.25]	70 [18.94]	74.29
	BEE				
	HC-edit-STAXAC	HC-edit-UPGMA	WilsonProb	Wilson	*K-NN*
K=1	80.83 (90)[7.31]	84.17 (80)[15.83]	78.33 [17.68]	78.69 [17.68]	78.33
K=3	81.67 (90)[7.31]	84.17 (80)[15.83]	82.5 [19.07]	85 *[20.09]*	82.5
K=5	82.50 (100)[0.00],(90)[7.31]	84.17 (80)[15.83]	82.5 [17.96]	76.15 [17.96]	80.83
K=7	84.17 (90)[7.31]	84.17 (80)[15.83]	84.17 [16.67]	**85.83** [16.57]	83.33
	Bos				
	HC-edit-STAXAC	HC-edit-UPGMA	WilsonProb	Wilson	K-NN
K=1	69.57 (90)[37.28]	72.33 (80)[7.14]	67.00 [23.24]	65.5 [23.24]	67
K=3	70.55 (90)[37.28]	71.94 (80)[7.14]	70.36 [23.68]	70.55 [23.48]	67.39
K=5	70.75 (100)[0.00]	70.36 (90)[1.03]	67.46 [23.61]	66.4 [24.16]	70.55
K=7	**73.12** (100)[0.00]	**73.12** (100)[0.00]	69.37 *[25.83]*	69.57 [24.18]	**73.12**

Fig. 6.4. Accuracy results

Overall, the result suggests a superiority of HC-edit over the traditional k-NN, WilsonProb and Wilson approaches. With respect to compression rate, as expected, HC-edit removes fewer examples than other methods.

6.5. Conclusion

Editing improves the accuracy of the k-NN classifier; however, current editing methods tend to remove too many examples from the training set which does not always lead to optimum accuracy rate. We proposed a new editing algorithm, called HC-edit that identifies regions in the dataset of high purity and removes minority examples from the identified regions. HC-edit takes as input hierarchical clusters augmented with purity information in the nodes; which facilitates clusters retrieval — based on user-defined purity values. Traditional k-nearest neighbor methods used the k parameter for both editing and classification. By allowing two parameters for modeling — purity for editing, and k for the classification —, HC-edit provides greater landscape for a model selection. Experiments over seven datasets have been carried out in order to evaluate the performance of the new editing approach. HC-edit's performance has been compared with that of other traditional techniques. The experiments reveal that the HC-edit has improved accuracy while removing less of examples.

References

1. Belur V. Dasarathy; *"Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques"*; Mc Graw-Hill Comp. Sc. Series, IEEE Comp. Society Press, pp. 217-224, 1991.

2. Devijver, P.A., Kittler, J.; *"Pattern Recognition: A Statistical Approach"*; Prentice Hall, Englewood Cliffs, NJ (1982).

3. Devijver, P. and Kittler, J.; *"On the Edited Nearest Neighbor Rule"*; IEEE 1980 Pattern Recognition.Vol.1, 72-80.

4. Eick, C.F., Zeidat, N., and Vilalta, R.; *"Using Representative-Based Clustering for Nearest Neighbor Dataset Editing"*; ICDM 2004: 375-378.

5. Eick, C.F., Zeidat, N., and Zhenghong, Z.; *"Supervised Clustering – Algorithms and Benefits"*; Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI04) , Boca Raton, Florida, November 2004, pp. 774-776.

6. UCI Repository of Machine Learning http://archive.ics.uci.edu/ml/

7. Sánchez, J.S., Barandela, R., Marqués, A.I., Alejo, R., Badenas, J.; *Analysis of new techniques to obtain quality training sets*; Pattern Recognition. Let. 24 (2003) 1015-1022.

8. Wilson, D.L.; *"Asymptotic Properties of Nearest Neighbor Rules Using Edited Data"*; IEEE Transactions on Systems, Man, and Cybernetics, 2:408-420, 1972.

9. Fernando Vázquez1, J. Salvador Sánchez2, and Filiberto P.; *"A Stochastic Approach to Wilson's Editing Algorithm"*; Proceedings of the Second Iberian conference on Pattern Recognition and Image Analysis - Volume Part II Pages 35-42.

10. http://jgi.doe.gov/ as of May 2015.

Chapter 7

Summary of the Results and Directions for Future Work

In this work, we investigated new methods to induce hierarchical models with the goal of obtaining better predictive models, to facilitate creating background knowledge with respect to an underlining class distribution, to obtain hierarchical groupings of a set of objects based on background knowledge they share, to detect sub-classes within existing class distribution, and to provide methods to evaluate hierarchical groupings. In particular, we investigated recursive splitting methods and proposed two new decision tree approaches (TPRTI, PATHFINDER), and two new hierarchical clustering algorithms (STAXAC, AVALANCHE). AVALANCHE uses a new splitting method to recursively split the input space while STAXAC is the first hierarchical supervised clustering algorithm to be proposed in the literature. The decision tree approaches we developed in this research, TPRTI, and PATHFINDER, decompose the complex task of finding split points (or decision boundaries) in high dimensionality, into a set of simpler tasks of finding split points in 2D. We showed through experiments that the proposed approaches provide improved results, if compared with their immediate competitors. The clustering approaches we introduced are new approaches that open new research directions and answer practical needs. In the following paragraphs, we summarize our key contributions and outline directions for future research.

In chapter 2, we presented TPRTI, a novel linear regression tree approach with a speed comparable to that of the fast but less accurate M5 algorithm and with accuracy comparable to the very accurate, but slow RETIS algorithm. Since accuracy of the TPRTI approach is closely related to the computed turning points locations in the input space, methods that can detect optimum turning points are useful for regression tree induction, in general.

In chapter 3, we presented PATHFINDER, an oblique tree induction approach that computes its decision boundaries in successive 2D spaces instead of the input space. It yields high accuracy rate for the cost of not classifying few examples (reject examples). PATHFINDER reject rate and its accuracy rate can further be improved by investigating new node evaluation methods. Another potential area that may be considered for investigation is to identify reject regions within a given dataset; which may help a domain expert to better understand the dataset at hand.

In chapter 4 AVALANCHE, a new divisive hierarchical approach that uses dissimilarity matrix as its input was introduced. AVALANCHE incorporates in its splitting decision local as well as global information; which provides the algorithm with the capability of generating better clustering results. That capability has not yet been fully explored.

Chapter 5 introduces a new type of taxonomy generating approach called *supervised taxonomy* (ST). Supervised taxonomies are generated considering background information in the form of class labels in addition to distance metrics, and are capable of capturing

class-uniform regions in a dataset. We introduced STAXAC, a novel supervised hierarchical clustering algorithm that generates supervised taxonomies. As this research introduces ST for the first time, we believe that many more supervised taxonomy algorithms such as a top-down hierarchical algorithm that creates STs, can be explored. Measures and algorithms operating on STs that assess the edited tree complexity, subclass changes, average purities at different tree-depths, classification complexity, and modality of classes have been proposed in this research, demonstrating the merit of STs, particularly as an exploratory data analysis tool. We claim that the latter two contributions are not only novel but useful. We demonstrated in our experimental evaluation that assessing the classification complexity of a dataset (CC), provides us with a good estimate of the difficulty of the classification problem at hand. We also provided evidence of the capability of STs to assess the modality of particular classes of a dataset at different sets of granularity.

Finally, there are many other useful things we can develop on top of supervised taxonomies, which is the focus of our future work. One such possibility was explored in chapter 6 were we introduced HC-edit, a new data editing method for the k-NNs classifier that uses as input a STAXAC-generated tree. HC-edit improves the accuracy rate of the k-NN classifier, by removing minority examples for clusters of a STs. Furthermore, as taxonomies are very popular in bioinformatics, we plan to assess the usefulness of STs for scientific discovery in bioinformatics in more depth. As this research has shown that STs can help generate background information about a dataset, we plan to explore the additional potentials of STs for meta-learning as future work.