INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Learning Continuous Functions Using Decision Tree Learning Algorithms

BY

Ahmed Esmat Mahmoud Ibrahim

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

Computer Science

May 2001

King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia

Deanship of Graduate Studies

This Thesis, written by

Ahmed Esmat Mahmoud Ibrahim

under the directions of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Thesis Committee:

Dr. Hussein Al-Muallim (Chairman)

Dr. Sulaiman Al-Bassam (Member)

Dr. Moataz Ahmed (Member)

Kanaan Faisal
Department Chairman

Dean, College of Graduate Studies
Dr. Osama Ahmed Jannadi

5/6/2002
Date

عمادة الدراسات العليا
DEANSHIP OF GRADUATE STUDIES

This work is presented to

my wonderful and always giving parents,

whose continuous support

and prayers led to this achievement,

and to my sister.

Acknowledgments

Acknowledgment is due to King Fahd University of Petroleum and Minerals for support of this thesis work. I would like to acknowledge the department of information and computer science for their support too. I would also acknowledge my colleagues at the systems and operation department at the information technology center of the university for their great support and help.

I wish to express my appreciation to Professor Hussein Al-Muallim my thesis advisor, for his continuous support, guidance and patience. I would also like to thank my thesis committee: Dr. Sulaiman Albassam and Dr. Moataz Ahmed for their support and valuable suggestions.

Contents

LIST OF TABLES ..... **VI**

LIST OF FIGURES ..... **VII**

ABSTRACT ..... **XI**

خلاصة الرسالة ..... **XII**

CHAPTER 1 : INTRODUCTION **1**

CHAPTER 2 : LITERATURE REVIEW **5**

2.1 REGRESSION BY RULE INDUCTION 6

2.2 REGRESSION BY TREE INDUCTION 11

2.2.1 Classification and Regression Trees *12*

2.3 MULTIVARIATE ADAPTIVE REGRESSION SPLINES 19

2.3.1 Piecewise Parametric Fitting Paradigm and Splines *19*

2.3.2 MARS Algorithm *20*

3.4 K-NEAREST NEIGHBORS 25

CHAPTER 3 : DECISION TREE LEARNING AND C4.5 26

3.1 DECISION TREES 26

3.1.1 Decision Tree Construction *29*

3.1.2 Test Selection *30*

2.1.3 Extensions to the Basic Procedure *32*

3.2 C4.5 36

3.2.1 Decision Tree Construction *36*

3.2.2 Pruning *37*

CHAPTER 4 : MULTI-CLASS C4.5 AND MULTI-BINARY TREES C4.5 39

4.1 INTRODUCTION 39

4.2 MULTI-CLASS C4.5 40

4.2.1 Pseudo Code of Multi-Class approach *41*

4.3 MULTI-BINARY TREES C4.5 42

4.3.1 Pseudo Code of Multi-Binary trees approach *43*

CHAPTER 5 : EXPERIMENTS AND RESULTS 46

5.1 EXPERIMENTS 46

5.2 ERROR CALCULATIONS 48

5.3 RESULTS 49

5.3.1 Housing Dataset *49*

5.3.2 Auto-Mpg Dataset *60*

5.3.3 Servo Dataset *62*

5.3.4 Auto Prices Dataset *64*

5.3.5 CPU Dataset *66*

5.3.6 Peptide Dataset *68*

5.4 ANALYSIS 70

CHAPTER 6 : CONCLUSIONS AND FUTURE WORK **72**

REFERENCES **74**

List of Tables

TABLE 2-1: EXAMPLE OF SWAPPING RULE COMPONENTS 9

TABLE 3-1: SAMPLE TRAINING EXAMPLES FOR GOLF PLAYING 28

TABLE 5-1: DATASETS USED FOR THE EXPERIMENTS 47

TABLE 5-2: COMPARISON OF REGRESSION METHODS PUBLISHED BY [UYS00] AND [WEI95] 49

TABLE 5-3: ERROR VALUES OF HOUSING DATASET USING NEW METHODS 50

TABLE 5-4: ERROR VALUES OF AUTO-MPG DATASET USING NEW METHODS 61

TABLE 5-5: ERROR VALUES OF SERVO DATASET USING NEW METHODS 63

TABLE 5-6: ERROR VALUES OF AUTO-PRICES DATASET USING NEW METHODS 65

TABLE 5-7: ERROR VALUES OF CPU DATASET USING NEW METHODS 67

TABLE 5-8: ERROR VALUES OF PEPTIDE USING NEW METHODS 69

List of Figures

FIGURE 2-1: SWAP-1 ALGORITHM 7

FIGURE 2-2: A SOLUTION INDUCED FROM A HEART-DISEASE DATA 8

FIGURE 2-3 COMPOSING PSEUDO-CLASSES (P-CLASS) 10

FIGURE 2-4: EXAMPLE OF REGRESSION TREE 14

FIGURE 2-5: AN EXAMPLE OF TREE CONSTRUCTION PROCESS. FOUR REGIONS ARE DETERMINED BY PREDICTORS X_1 AND X_2 14

FIGURE 2-6: RECURSIVE PARTITIONING ALGORITHM 16

FIGURE 2-7: A BINARY TREE REPRESENTING A RECURSIVE PARTITIONING REGRESSION MODEL WITH THE ASSOCIATED BASIS FUNCTIONS 17

FIGURE 2-8: MARS ALGORITHM 24

FIGURE 2-9: AN EXAMPLE FOR THE REGIONS OF MARS ALGORITHM 24

FIGURE 3-1: DECISION TREE GENERATED FROM LEARNING EXAMPLES IN TABLE 3-1 29

FIGURE 3-2: THE VALUE HIERARCHY FOR A COLOR ATTRIBUTE 33

FIGURE 4-1: MULTI-CLASS C4.5 APPROACH 42

FIGURE 4-2: MULTI-BINARY TREES APPROACH 44

FIGURE 4-3 CHOOSING THE BEST OUTPUT 45

FIGURE 4-4 EXAMPLE OF FIRST SATISFYING POINT (FSP) AND LAST SATISFYING POINT (LSP) 45

FIGURE 5-1: CHUNK SIZE VERSE RELATIVE ERROR FOR THE CROSS VALIDATION WITH MINIMUM RE, FOR THE MULTI-CLASS C4.5 APPROACH USING THE SORTED 10-FOLD CROSS VALIDATION 51

FIGURE 5-2: MINIMUM RELATIVE ERROR FOR EACH CROSS VALIDATION, FOR THE MULTI-CLASS C4.5 APPROACH USING THE SORTED 10-FOLD CROSS VALIDATION . 51

FIGURE 5-3: CHUNK SIZE VERSE RELATIVE ERROR FOR THE CROSS VALIDATION WITH MINIMUM RE, FOR THE MULTI-CLASS C4.5 APPROACH USING THE C4.5 10-FOLD CROSS VALIDATION 52

FIGURE 5-4: MINIMUM RELATIVE ERROR FOR EACH CROSS VALIDATION, FOR THE MULTI-CLASS C4.5 APPROACH USING THE C4.5 10-FOLD CROSS VALIDATION 53

FIGURE 5-5: CHUNK SIZE VERSE RELATIVE ERROR FOR THE CROSS VALIDATION WITH MINIMUM RE, FOR THE MULTI-BINARY TREES C4.5 (FSP) APPROACH USING THE SORTED 10-FOLD CROSS VALIDATION. 54

FIGURE 5-6: MINIMUM RELATIVE ERROR FOR EACH CROSS VALIDATION, FOR THE MULTI-BINARY TREES C4.5 (FSP) APPROACH USING THE SORTED 10-FOLD CROSS VALIDATION 54

FIGURE 5-7: CHUNK SIZE VERSE RELATIVE ERROR FOR THE CROSS VALIDATION WITH MINIMUM RE, FOR THE MULTI-BINARY TREES C4.5 (FSP) APPROACH USING THE C4.5 10-FOLD CROSS VALIDATION. 55

FIGURE 5-8: MINIMUM RELATIVE ERROR FOR EACH CROSS VALIDATION, FOR MULTI-BINARY TREES C4.5 (FSP) APPROACH USING THE C4.5 10-FOLD CROSS VALIDATION 56

FIGURE 5-9: CHUNK SIZE VERSE RELATIVE ERROR FOR THE CROSS VALIDATION WITH MINIMUM RE, FOR THE MULTI-BINARY TREES C4.5 (LSP) APPROACH USING THE SORTED 10-FOLD CROSS VALIDATION. 57

FIGURE 5-10: MINIMUM RELATIVE ERROR FOR EACH CROSS VALIDATION, FOR THE MULTI-BINARY TREES C4.5 (LSP) APPROACH USING THE SORTED 10-FOLD CROSS VALIDATION 57

FIGURE 5-11: CHUNK SIZE VERSE RELATIVE ERROR FOR THE CROSS VALIDATION WITH MINIMUM RE, FOR THE MULTI-BINARY TREES C4.5 (LSP) APPROACH USING THE C4.5 10-FOLD CROSS VALIDATION..... 58

FIGURE 5-12: MINIMUM RELATIVE ERROR FOR EACH CROSS VALIDATION, FOR THE MULTI-BINARY TREES C4.5 (LSP) APPROACH USING THE SORTED 10-FOLD CROSS VALIDATION 59

FIGURE 5-13: RELATIVE ERROR (RE) OF HOUSING DATASET WITH DIFFERENT REGRESSION METHODS 60

FIGURE 5-14: RELATIVE ERROR (RE) OF AUTO-MPG DATASET WITH DIFFERENT REGRESSION METHODS 62

FIGURE 5-15: RELATIVE ERROR (RE) OF SERVO DATASET WITH DIFFERENT REGRESSION METHODS 64

FIGURE 5-16: RELATIVE ERROR (RE) OF AUTO-PRICES DATASET WITH DIFFERENT REGRESSION METHODS 66

FIGURE 5-17: RELATIVE ERROR (RE) OF CPU DATASET WITH DIFFERENT REGRESSION METHODS 68

FIGURE 5-18: RELATIVE ERROR (RE) OF PEPTIDE DATASET WITH DIFFERENT REGRESSION METHODS 70

Abstract

Name: Ahmed Esmat Mahmoud Ibrahim

Title: Learning Continuous Functions Using Decision trees Learning Algorithm

Major Field: Computer Science

Date of Degree: May, 2001

C4.5 as an implementation of a decision tree learning algorithm called ID3. C4.5 accounts for discrete classes and does not consider continuous output. The purpose of this work is to suggest two approaches to modify the C4.5 implementation to account for continuous output. These two approaches are called: Multi-Class C4.5 and Multi-Binary trees C4.5. Multi-Class C4.5 approach groups continuous output values into chunks and averages them. A discrete class replaces the class of the examples in each group. The average of the continuous classes in each group is associated to the discrete class of the group. Multi-Binary trees C4.5 approach divides the examples into chunks and builds as many decision trees as the number of chunks. The end boundary of each chunk becomes the class of the chunk. Each tree is built to indicate that the class of examples is less than or equal to the end boundary. Based on experiments conducted on six well-known domains, these two approaches show an improvement in error rates compared to other regression methods.

Master of Science Degree

King Fahd University of Petroleum and Minerals

Dhahran, Saudi Arabia

May 2001

خـــلاصة الرســـالة

الاسم: أحمد عصمت محمود إبراهيم

العنوان: تعلم دالات متصلة باستخدام خوارزميات شجرات القرار

التخصص: علوم الحاسب الآلي

تاريخ التخرج: مايو 2001 م الموافق صفر 1422 هـ

"C4.5" هو تطبيق عملي لخوارزم "ID3" المستخدم لبناء شجرات القرار. هذا التطبيق يتعامل مع المعلومات ذات الدوال الغير متصلة ولا يتعامل مع المعلومات ذات الدوال المتصلة. هذه الدراسة تعرض أسلوبين يتم بهم تعديل تطبيق "C4.5" كي يتعامل مع المعلومات ذات الدوال المتصلة. هذان الأسلوبان هما: "C4.5" متعدد القرارات و"C4.5" متعدد الشجرات الثنائية. الأسلوب الأول يقسم المعلومات إلى مجموعات منفصلة ويعطي كل مجموعة تصنيف غير متصل هو عبارة عن متوسط قيم القرارات المتصلة الأصلية. الأسلوب الثاني يقسم المعلومات إلى مجموعتين منفصلتين ويعطي المجموعة الأولي تصنيف بقيمة "صفر" والمجموعة الثانية تصنيف بقيمة "واحد". تقوم هذه الطريقة ببناء شجرة قرار ثنائية عند هذه النقطة. بعد ذلك يتم زيادة عدد الأمثلة في المجموعة الأولى و تصغير المجموعة الثانية، وبناء شجرة قرار ثنائية في كل مرة. تقوم بعد ذلك هذه الشجرات بالتصويت لحساب تصنيف مثال معين. في كلا الأسلوبين يتم اختبار حجم تقسيم المجموعات واختيار الحجم الذي ينتج عنه أقل أخطاء التصنيف. دلت نتائج الاختبارات التي أجريت باستخدام ستة من أشهرقواعد البيانات أن كلا الأسلوبين حقق نتائج تفوق الأساليب المعروفة.

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن

الظهران، المملكة العربية السعودية

مايو 2001م الموافق صفر 1422هـ

Chapter 1 :

Introduction

The ability to learn is one of the central features of intelligence. According to Merriam Webster's dictionary, learning means "to gain knowledge or understanding of or skill by study, instruction, or experience". This makes it an important concern for both cognitive psychology and artificial intelligence (AI). The field of machine learning studies the computational processes that underlie learning in both humans and machines. Machine learning include any computer program that improves its performance at some task through experience, more formally [Mit97]:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Machine learning algorithms have proven to be of great practical value in a variety of application domains. They are especially useful in [Mit97],[Alm02]:

- Data mining problems where large databases may contain valuable implicit regularities that can be discovered automatically (e.g., to analyze outcomes of medical treatments from patient databases or to learn general rules for credit worthiness from financial database)
- Poorly understood domains where humans might not have the knowledge needed to develop effective algorithms (e.g., human face recognition from images)
- Domains where the program must dynamically adapt to changing conditions (e.g., controlling manufacturing processes under changing supply stocks or adapting the changing reading interests for individual)

The problem of approximating the values of a continuous variable is described in the statistical literature as regression. Given samples of output (response) variable y and input (predictor) variables $x = \{x_1 \ldots x_n\}$, the regression task is to find a mapping $y = f(x)$. Relative to the space of possibilities, finite samples are far from complete, and a predefined model is needed to concisely map x to y. Accuracy of prediction, i.e. generalization to new cases, is of primary concern. Regression differs from classification in that the output variable y in regression problems is continuous, whereas in classification y is strictly categorical. From this perspective, classification can be thought of as a subcategory of regression. Some machine learning researchers

have emphasized this connection by describing regression as "learning how to classify among continuous classes" [Qui93].

There are five main paradigms in machine learning that handle continuous functions: Standard Regression, Neural Networks, Instance Based Learning, Regression Trees, and Decision Trees [Wan97]. Standard Regression is not very potent way of representing induced functions and imposes a linear relationship on the data. Neural Networks and Instance Based Learning suffer from opacity. Regression Trees approximate a non-linear function by a piecewise constant function. Decision Trees are comprehensible by human experts, directly convertible into production rules, able to state a reason behind decision, and easy and efficient to construct compared to other methods [Alm02].

This thesis describes two approaches to enhance a well-known classification implementation called C4.5 (based on ID3 classification algorithm), to make it accepts class values that are continuous. ID3 is an example of rule-induction learning paradigm. Originally C4.5 deals with discrete classes only. These two methods are called: Multi-Class C4.5 and Multi-Binary Trees C4.5.

Chapter two presents the different regression methods: rule induction, tree induction (e.g. Classification and Regression Trees, CART), and Multivariate Adaptive Regression Splines (MARS). The chapter fully describes each method and compare between them.

Chapter three defines decision trees: how they are built, tested and extended. Finally, the chapter presents the C4.5 package as in implementation of decision tree learner.

Chapter four presents two approaches to modify the C4.5 implementation to account for continuous output. These two approaches are called: Multi-Class C4.5 and Multi-Binary trees C4.5. Multi-Class C4.5 approach groups continuous output values into chunks and averages them. A discrete class replaces the class of the examples in each group. The average of the continuous classes in each group is associated to the discrete class of the group. Multi-Binary trees C4.5 approach divides the examples into chunks and builds as many decision trees as the number of chunks. The end boundary of each chunk becomes the class of the chunk. Each tree is built to indicate that the class of examples is less than or equal to the end boundary.

Chapter five presents the experiments we have performed on the two suggested methods and their results. The experiments were done on seven datasets taken from University of California at Irvine, UCI machine learning repository and Bilkent University (BU) function approximation repository. The results of the experiments are compared to other regression methods including 5-NN, Rule-based learner, CART, and MARS. The comparison showed a significant improvement in the error rates.

Chapter six, the conclusion to this thesis, includes a summary of the achievements and recommendations suggested for future work.

Chapter 2 :

Literature Review

Predicting values of numeric or continuous attributes is known as regression in statistical literature. This task constitutes a research area for many researchers in this field. Predicting real values is also an important topic for machine learning. Most of the problems that humans learn in real life such as sporting abilities, real-estate price estimation and cancer patient's life expectancy are continuous. Dynamic control is one such a problem, which is researched in machine learning. Learning to catch a ball, moving in a three dimensional space, is an example for this problem, which is studied in robotics. In such applications machine learning algorithms are used to control robot motions, where the response to be predicted by the algorithm is a numeric or real-valued distance measure and direction. As an example for such problems, Salzberg

and Aha [Ahd94] proposed an instance-based learning algorithm for robot control task in order to improve a robot's physical abilities.

In machine learning, most of the research has been done for classification, where the single predicted feature is nominal or discrete. Regression differs from classification in that the output or predicted feature in regression problems is continuous. Even though most of the research in machine learning is concentrated on classification, recently the focus of machine learning community has moved strongly towards regression, since a large number of real-life problems can be modeled as regression problems. Various names are used for this problem in the literature, such as functional prediction, real value prediction, function approximation and continuous output learning.

2.1 Regression by Rule Induction

Inducing rules from a given training set is a well-studied topic in machine learning. Weiss and Indurkhya employed rule induction for regression problem and reported significant results [Wei95]. First, the rule-based classification algorithm [Wei93], Swap-1, that learns decision rules in disjunctive normal form (DNF), is reviewed, and later on its adaptation for regression is described.

The main advantage of inducing rules in DNF is its explanatory capability. It is comparable to decision trees since they can be converted into DNF models as well. The most important difference between them is that the rules are not mutually

exclusive as in decision trees. In decision trees, for each instance, there is exactly one rule, a path from root to a leaf, that is satisfied. Because of this restriction, decision tree models may not produce compact models. However, because of this property of rule-based models, the problem emerges that, for a single instance, two or more classes may be satisfied. The solution found for this problem is to assign priorities or ordering to the rules according to their extraction order. The first rule, according to this ordering that satisfies the query instance, determines the class of a query. The Swap-1 rule induction algorithm [Wei93] and a sample output are shown in Figure 2-1 and Figure 2-2, respectively.

[1] Input: D, a set of training cases
[2] Initialize $R_1 \leftarrow$ empty set, $k \leftarrow 1$, and $C_1 \leftarrow D$
[3] repeat
[4] create a rule B with a randomly chosen attribute as its left-hand side
[5] while (B is not 100-percent predictive) do
[6] make single best swap for any component of B, including deletion of the component, using cases in C_k
[7] If no swap is found, add the single best component to B
[8] endwhile
[9] $P_k \leftarrow$ rule B that is now 100-percent predictive
[10] $E_k \leftarrow$ cases in C that satisfy the single-best-rule P_k
[11] $R_{k+1} \leftarrow R_k \cup \{ P_k \}$
[12] $C_{k+1} \leftarrow C_k - \{ E_k \}$
[13] $k \leftarrow k + 1$
[14] until (C_k is empty)
[15] find rule r in R_k that can be deleted without affecting performance on cases in training set D
[16] while (r can be found)
[17] $R_{k+1} \leftarrow R_k - \{ r \}$
[18] $k \leftarrow k + 1$
[19] endwhile
[20] output R_k and halt.

Figure 2-1: Swap-1 Algorithm

CA > 0:5 And *CP* > 3:5	→	*Class* = 2
THAL>6:5	→	*Class* = 2
[*True*]	→	*Class* = 1

Figure 2-2: A solution induced from a heart-disease data

While constructing a rule, the Swap-1 algorithm searches all the conjunctive components it has already formed, and swaps them with all possible components it will build. This search also includes deletion of some components from the rule. If no improvement is established from these swaps and deletions, then the best component is added to the rule. To find the best component to be added, the predictive value of a component, as the percentage of correct decisions, is evaluated. If the predictive values of them are equal, maximum instance coverage is used as a second criterion. These swappings and additions end when the rule reaches 100-percent prediction accuracy.

Table 2-1 illustrates a sample rule induction. After forming a new rule for the model, all instances that the rule covers are removed from the instance set, and remaining instances are considered for the following steps. When a class is covered, the remaining classes are considered, in turn. This process iterates until instance set becomes empty, that is all instances are covered.

Step	Predictive Value (%)	Rule
1	31	p3
2	36	p6
3	48	p6 & p1
4	49	p4 & p1
5	69	p4& p1 & p2
6	80	p4 & p1 & p2& p5
7	100	p3 & p1 & p2& p5

Table 2-1: Example of swapping rule components

After the formation of the rule set, if removal of any rule does not change the performance on training set, such rules are removed from the model. Furthermore, in order to reach an optimum rule set, an optimization procedure is used [Wei93].

The rule induction algorithms for classification, such as Swap-1, can also be applied to the regression problems. Since these algorithms are designed for the prediction of nominal attributes, by a preprocessing procedure, the numeric attribute in regression to be predicted is transformed to a nominal one.

[1] Input: { y } a set of output values
[2] Initialize n = number of cases, k = number of classes
[3] repeat for each $Class_i$
[4] $Class_i$ = next n/k cases from list of sorted y values
[5] end
[6] repeat for each $Class_i$ (until no change for any class)
[7] repeat for each case j in $Class_i$
[8] 1. Move $Case_{ij}$ to $Class_{i-1}$, compute Err_{new}
[9] If $Err_{new} > Err_{old}$ return $Case_{ij}$ to C_i
[10] 2. Move $Case_{ij}$ to $Class_{i+1}$, compute Err_{new}
[11] If $Err_{new} > Err_{old}$ return $Case_{ij}$ to C_i
[12] next $Case_j$ in $Class_i$
[13] next $Class_i$
[14] repeat for each $Class_i$ (until no change for any class)
[15] If Mean($Class_i$) = Mean($Class_j$) then
[16] Combine $Class_i$ and $Class_j$
[17] end

Figure 2-3 Composing Pseudo-Classes (P-Class)

For this transformation, the P-class algorithm, shown in Figure 2-3, is used in [Wei95]. This transformation is in fact a one-dimensional clustering of training instances on response variable y, in order to form classes. The purpose is to make y values within one class most similar, and across classes most dissimilar. The assignment of these values to classes is done in a way that, the distance between each y_i and its class mean must be minimum.

The P-Class algorithm does the following. First it sorts the y values (line 1), then assigns approximately equal number of contiguous sorted y_i to each class (lines3-5). Finally it moves a y_i to a contiguous class if it reduces the distance of it to the mean of that class (lines 6-13).

The naive way to predict the response for a query instance is to assign the average of responses. The average may be a median or mean of that class. But different approaches also can be considered by applying a parametric or non-parametric model for that specific class. For example the nearest- neighbor approach is used for this purpose, and significant improvements of this combination against the naive approach is reported in [Wei95].

2.2 Regression by Tree Induction

Tree induction algorithms construct the model by successively partitioning the training data set. The task of constructing a tree is accomplished by employing a search to select an attribute to be used for partitioning the data at each node of the tree. The explanation capability of regression trees and their use in order to determine key features from a large feature set are major advantages for these applications. In terms of performance and accuracy, regression tree applications are comparable to other models. Regression trees are also noted to be strong, when there are higher order dependencies among predictors. [Uys00]

The common characteristic to all regression tree methods is that they partition the training set into disjoint regions recursively, where the final partition is determined by the leaf nodes of the regression tree. In order to avoid overfitting and form simpler models, pruning strategies are employed in all regression tree methods.

2.2.1 Classification and Regression Trees

Using trees as regression models are first applied in CART (Classification and Regression Trees) program, developed in a statistical research community [Bre84]. This program induces both regression and classification trees.

The first step starts with the whole training set represented by the root node to construct the tree. A search is done on the features to construct the remaining part of tree recursively. We find the best feature and feature value to split the training set represented by the root node. This splitting forms two leaf nodes that represent two disjoint regions in the training set. In the second step one of these regions is selected for further splitting. This splitting is again done according to a selected feature value. This splitting process continues at a selected leaf node recursively to construct the regression tree.

In CART system, a constant function (a constant response value) is used for the estimation of a query or test instance that falls into the regions represented by leaf nodes. Generally this function is the average of the response values of instances. Each disjoint region has its own estimated value that is assigned to any query instance, located in this region.

In order to construct optimum disjoint regions, an error criterion is employed. The optimum value of this criterion produces a decomposition at any step of tree induction process described above such that the correct region, feature, feature value (splitting surface) and estimates for each region are selected. In order to determine the predicted target values in these regions, averaging methods such as mean and median, are used. As a fitting criterion the variances of the regions are used.

$$Error(Variance) = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

Where n is the number of instances in the region, y_i is the actual output of an instance and $\bar{y}$ is the predicted output,

$$\text{Splitting Error} = \frac{1}{n}\left\{ \sum_{x_i \in X_{left}} (y_i - \bar{y}_{left})^2 + \sum_{x_j \in X_{right}} (y_j - \bar{y}_{right})^2 \right\}$$

After computing splitting error for all possible splits, of a particular predictor, the splitting that maximizes the following criterion is selected.

$$C = Variance - Splitting\ Error$$

The node and the predictor that reaches the maximum criteria C, is selected for splitting. An example regression tree is shown in Figure 2-4. The construction process is illustrated in Figure 2-5

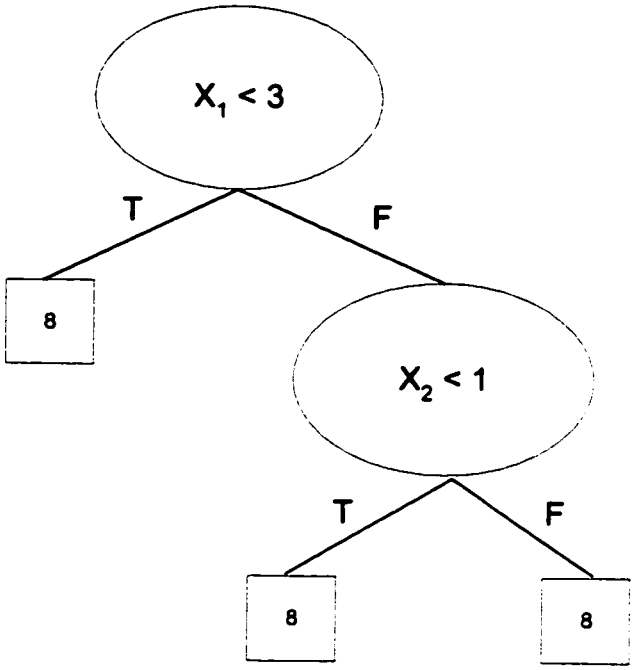


Figure 2-4: Example of Regression Tree

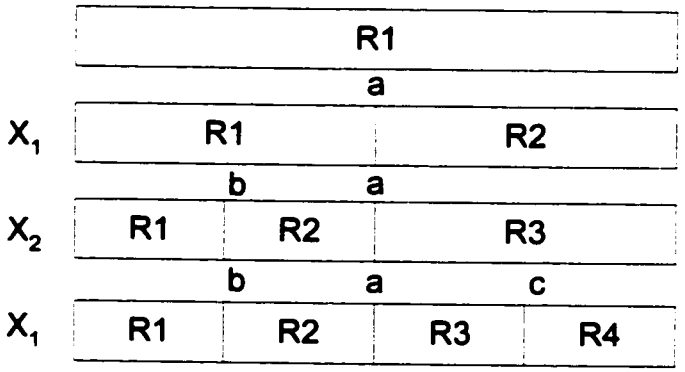


Figure 2-5: An Example of Tree Construction Process. Four regions are determined by predictors x_1 and x_2

Formally, the resulting model can be defined in the following form [Bre84]:

$$\text{If } x \in R_m, \text{ then } f(x) = g_m(x\{a_j\}_1^P)$$

Where $\{R_m\}_1^P$ are disjoint subregions representing p partitions of training set. The functions g are generally in simple parametric form. The most common parametric form is a constant function, which is illustrated with the example given in Figure 2-4:

$$g_m(x \mid a_m) = a_m$$

The constant values of leaves or partitions are generally determined by averaging. More formally, the model can be denoted by using basis functions:

$$\overline{f}(x) = \sum_{m=1}^{M} a_m B_m(x)$$

The basis functions $B_m(x)$ take the form of

$$B_m(x) = I(x \in R_m)$$

where I is an indicator function having the value one if its argument is true and zero otherwise. Let $H[\eta]$ be a step function, indicating a positive argument

$$H[\eta] = \begin{cases} 1 & \text{if } \eta > 0 \\ 0 & \text{otherwise} \end{cases}$$

and let LOF(g) be a procedure that computes the lack of fit of an estimation function g to the data, the recursive partitioning algorithm is given in Figure 2-6.

[1] $B_1(x) \leftarrow 1$
[2] For $M = 2$ to M_{max} do : lof* $\leftarrow \infty$
[3] For $m = 1$ to $M - 1$ do :
[4] For $v = 1$ to n do :
[5] For $t \in \{x_{vj} \mid B_m(x_j) > 0$
[6] $g \leftarrow \sum_{i \neq m} a_i B_i(x) + a_m B_m(x)\, H[+(x_v - t)] + a_M B_m(x) H[-(x_v - t)]$
[7] lof $min_{a1 \ldots aM}\, LOF(g)$
[8] if $lof < lof^*$, then $lof^* \leftarrow lof$; $m^* \leftarrow m$; $v^* \leftarrow v$, $t^* \leftarrow t$ end if
[9] end for
[10] end for
[11] end for
[12] $B_M(x) \leftarrow B_{m\bullet}(x)\, H[-(x_{v\bullet} - t^*)]$
[13] $B_{m\bullet}(x) \leftarrow B_{m\bullet}(x)\, H[+(x_{v\bullet} - t^*)]$
[14] end for

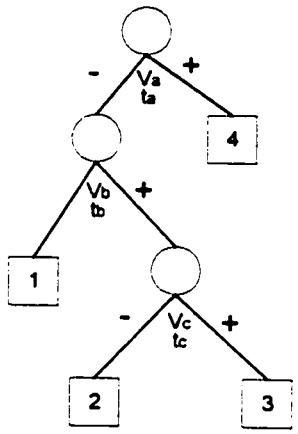
Figure 2-6: Recursive Partitioning Algorithm

The first line of the algorithm assigns the whole training set as the initial region. The first loop iterates the splitting until reaching at maximum number of regions. The next three loop selects the optimum basis function $B_{m\bullet}$ (intuitively the optimum region), predictor $x_{v\bullet}$, and split point t^*. At lines 12 and 13, the selected region for splitting, $B_{m\bullet}$, is replaced with its two partitions. This is done by adding a factor to its product; with $H[-(x_{v\bullet} - t^*)]$ for the negative portion of the region at line 12 by creating a new basis function, and with $H[+(x_{v\bullet} - t^*)]$ for the positive portion of the region at line 13, by modifying or removing the previous basis function. Finally the basis functions formed by the algorithm will take the following form:

$$B_m(x) = \prod_{k=1}^{K_m} H[s_{km} \cdot (x_{v(k,m)} - t_{km})]$$

where the quantity K_m is the number of splits that gave rise to B_m, and the arguments of the step functions contain the parameters associated with each of these splits. The

quantity s_{km} take ± 1 values indicating the right/left portions, $v(k, m)$ label the predictor variables, and t_{km} represent values on the corresponding variables. A possible output of the algorithm is shown in Figure 2-7.



$$B_1 = H[-(x_{va} - t_a)]\ H[-(x_{vb} - t_b)]$$
$$B_2 = H[-(x_{va} - t_a)]\ H[+(x_{vb} - t_b)]\ H[-(x_{vc} - t_c)]$$
$$B_3 = H[-(x_{va} - t_a)]\ H[+(x_{vb} - t_b)]\ H[+(x_{vc} - t_c)]$$
$$B_4 = H[+(x_{va} - t_a)]$$

Figure 2-7: A binary tree representing a recursive partitioning regression model with the associated basis functions

The partition may lead to very small regions with a large tree. This situation may cause over fitting with unreliable estimates. Stopping the process too early also may not produce good results. The solution to this problem is to employ a pruning strategy.

Pruning the regression tree by removing leaves will leave holes, which is an important problem, since we will not be able to give answer to queries that fall into these regions

or holes. That is why the removal of regions is done pairwise, with the siblings, by merging them into a single (parent) region. [Bre84]

Recursive partitioning regression is an adaptive method. By adaptive it is meant one that dynamically adjusts its strategy to take into account the behavior of particular problem to be solved [Fri91]. For example, recursive partitioning has the ability to exploit low local dimensionality of functions. In local regions, the dependence of the response may be strong on a few of the predictors, and these few variables may be different in different regions. Another property of recursive partitioning regression is that they allow interpretations, especially when a constant estimation is done on the leaves.

On the other hand, it has some drawbacks and limitations. The most important one is that the estimation is discontinuous. The model cannot approximate even simple continuous functions such as linear functions. This problem limits the accuracy of the model. As a consequence of this limitation, one cannot extract from the representation of the model about the structure of the function, e.g. linear or additive, or whether it involves complex interaction among the variables.

2.3 *Multivariate Adaptive Regression Splines*

As stated in the previous section, a fundamental drawback of recursive partitioning regression (CART) is the lack of continuity, which affects the accuracy. Another problem with that method is its inability to provide good approximations to some functions, even to most simple linear ones. Multivariate adaptive regression splines (MARS) addresses these two problems of recursive partitioning regression, in order to increase accuracy [Fri91].

2.3.1 Piecewise Parametric Fitting Paradigm and Splines

There are different paradigms for global parametric modeling to generalize low dimensional data. One of them is piecewise parametric fitting. The basic idea is to approximate a function by several simple parametric functions (usually low order polynomials) each defined over different subregions of training set. The constraint for the formation of polynomial fitting is that, it must be continuous at every point.

The most popular piecewise polynomial fitting procedures are based on splines, where the parametric functions are polynomials of degree q. The procedure is implemented by constructing a set of globally defined basis functions. These functions span the space of qth order spline approximations and fitting the coefficients of the basis function to the data by least squares. The spline basis functions are denoted by,

$$\{(x-t_k)_+^q\}_1^K$$

where $\{t_k\}_1^K$ are set of split (knot) locations. The subscript + indicates a value of zero for negative values of the argument. This is known as truncated power basis in mathematical literature.

2.3.2 MARS Algorithm

The MARS algorithm is the modified recursive partitioning algorithm, given in the previous section, that addresses the problems stated above. The reason that causes recursive partitioning algorithm to be discontinuous, the first problem, is the use of the step function. If the step function were replaced by a continuous function everywhere it appears in that algorithm (lines 6, 12, and 13), it could produce a continuous model. The step function employed in that algorithm can be considered as a special case of a spline basis function, where $q = 0$.

The one-sided truncated power basis functions for representing q*th* order splines are:

$$b_q(x-t) = (x-t)_+^q$$

where t is the knot location, q is the order of the spline and the subscript indicates the positive part of the argument. For $q > 0$, the spline approximation is continuous. A two-sided truncated power basis is of the form:

$$b_q^{\pm}(x-t) = [\pm(x-t)]_+^q$$

The step functions that appear in recursive partitioning algorithm are seen to be two-sided truncated power basis functions for $q = 0$ splines. The solution for discontinuity is solved by employing spline functions, of order $q > 0$, instead of step functions in the algorithm.

The second modification is related with the second problem, the inability of the algorithm to provide good approximations to certain functions. After the first modification, the algorithm tends to involve functions with more than a few variables (higher order interactions). At each split, one such function is removed, and two new functions are produced with one more variable. This causes one level increase in the interaction order. With such complex functions, having high level orders, it becomes difficult to approximate simple functions like linear ones.

The solution for this problem is not to delete the lower order parent after splitting. With this modification, now, all basis functions become eligible for further splitting. The new model involves either high or low order interactions, or both of them.

Another, a third problem emerges after the employment of splines in the algorithm. Since the algorithm allows multiple splits on the same predictor, along a single path of the binary tree, final basis functions may include several factors, involving the same variable in their product. For $q > 0$, higher orders than q may be produced on a single predictor.

After the second modification, not deleting the parents after splits, a restriction on the basis function can be applied to involve distinct predictors. Since we do not remove the parent after splitting, many such splits can be done on the same parent. By employing another split to that parent instead of splitting a child, MARS does not increase the depth or add a new factor to the product.

One remaining problem, which is not solved with the new algorithm, MARS, is the value of q. The general idea is to use $q = 1$.

In summary, the following modifications are done to the recursive partitioning algorithm:

(a) Replacing the step function $H[\pm(x - t)]$ by a truncated power basis function $[\pm(x-t)]_+^q$

(b) not removing the parent basis function Bm* after its split, thereby making it and both its daughters eligible for further splitting;

(c) restricting the product associated with each basis function to factors involving distinct predictor variables.

After using two sided truncated power basis functions, instead of step function, the MARS algorithm, shown in Figure 2-8, now produces multivariate spline basis functions of the following form.

$$B_m^{(q)}(x) = \prod_{k=1}^{K_m} H[s_{km}.(x_{v(k,m)} - t_{km})]_+^q$$

For the pruning of the resulting model, after MARS algorithm, now it is not necessary to employ two at a time deletion strategy used in the previous algorithm. Because the parents are not deleted, there will be no holes left after any deletion. Any pruning algorithm can be employed for the MARS procedure.

In the algorithm above, truncated power basis functions ($q = 1$) are substituted for step functions in lines 6, 12, and 13. The parent basis function is included in the modified model in line 6 and remains in the model through lines 12-14. Basis function products are constrained to contain factors involving distinct variables by the control loop in line 4. Figure 2-9 illustrate the regions after constructing the model. Note that the splitted regions are not deleted from the model as in the CART, and another splitting for the same region can be applied with the same or different predictor.

[1] $B_1(x) \leftarrow 1$; $M = 2$
[2] Loop until $M > M_{max}$: lof* $\leftarrow \infty$
[3] For $m = 1$ to $M - 1$ do :
[4] For $v \notin \{v(k,m) | 1 \le k \le K_m\}$
[5] For $t \in \{x_{vj} \mid B_m(x_j) > 0\}$
[6] $g \leftarrow \sum_{i=1}^{m-1} a_i B_i(x) + a_m B_m(x)\, H[+(x_v - t)]_+ + a_M B_m(x) H[-(x_v - t)]_+$
[7] lof $\leftarrow min_{a1 \ldots aM-1}\ LOF(g)$
[8] if $lof < lof^*$, then $lof^* \leftarrow lof$; $m^* \leftarrow m$; $v^* \leftarrow v$, $t^* \leftarrow t$ end if
[9] end for
[10] end for
[11] end for
[12] $B_M(x)\ B_{m^*}(x)\ H[-(x_{v^*} - t^*)]_+$
[13] $B_{M-1}(x)\ B_{m^*}(x)\ H[+(x_{v^*} - t^*)]_+$
[14] $M \leftarrow M+2$
[15] end loop
[16] end

Figure 2-8: MARS Algorithm

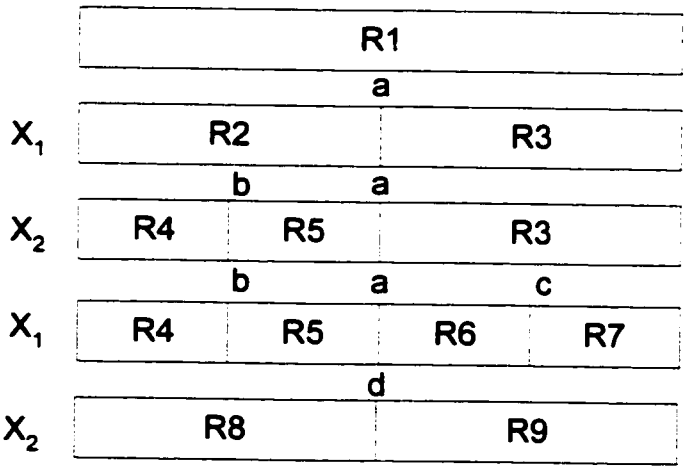


Figure 2-9: An example for the regions of MARS algorithm

3.4 k-Nearest Neighbors

The k-nearest neighbor method is one of the simplest regression methods, relying on table lookup. To classify an unknown case x, the k cases that are closest to the new case are found in a sample database of stored cases. The predicted $y(x)$ of the following equation is the mean of the y values for the k-nearest neighbors. The nearest neighbors are found by a distance metric such as Euclidean distance (usually with some feature normalization). The method is non-parametric and highly non-linear in nature [Wei95]

$$y_{knn}(x) = \frac{1}{K}\sum_{k=1}^{K} y_k \quad \text{for K nearest neighbors of x}$$

A major problem with this approach is how to limit the effect of irrelevant features. While limited forms of feature selection are sometimes employed in a preprocessing stage, the method itself cannot determine which features should be weighted more than others. As a result, the procedure is very sensitive to the distance measure used. In a high-dimensional feature space, k-nearest neighbor methods may perform very poorly. These limitations are precisely those that the partitioning methods address.

Chapter 3 :

Decision Tree Learning and C4.5

3.1 Decision Trees

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree. Learned trees can also be re-represented as sets of If-then rules to improve human readability. A decision tree is used as a classifier for determining an appropriate action for a given case. As an example of a decision tree see Figure 3-1.

Decision tree learning is the task of constructing a decision tree classifier, like the one in Figure 3-1, from a collection of historical cases called *training examples*. The

collection of such examples from which a decision tree is to be constructed is called a training sample. A training example is assumed to be represented as a pair $<X,c>$ where X is a vector of attribute values describing some case, and c is the appropriate class of that case.

A decision tree consists of three components: decision nodes, branches representing outcomes of decision nodes, and leaves. Let A be a set of attributes and C be a set of classes. A Decision tree is a tree structure with the following properties [Qui90]:

- Each non-terminal node, called a decision node, is labeled with a test involving one of the attributes in A. The test has a finite number of disjoint outcomes.
- Each outgoing branch from a non-terminal node corresponds to one of the outcomes of the test at the node.
- Each terminal (leaf) node is labeled with one of the classes belonging to C.

Each leaf in the tree represents a class (classification rule). The conjunction of tests on the branches from the root to that leaf constitutes the preconditions of that rule.

Decision trees classify instances by sorting them down the tree from the root to some leaf node which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from the node corresponds to one of the possible values for this attribute. An instance is

classified by starting at the root node of the tree, testing the attribute specified by starting at the root of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

For example, weather on a particular day could be described by the attributes:

- the *outlook*, whose possible values are *rain*, *overcast*, or *sunny*
- the *temperature*, with continuous values
- the *humidity*, with continuous values
- whether it is *windy* or not, with values *true* or *false*

A decision on whether to play golf or not can be taken depending on the weather conditions. A sample from the training examples might be described as:

Outlook	Temp	Humidity	Windy	Decision
Sunny	75	normal	true	Play (P)
Sunny	80	high	true	Don't Play (N)

Table 3-1: Sample training Examples for golf playing

and a generated decision tree might look like:

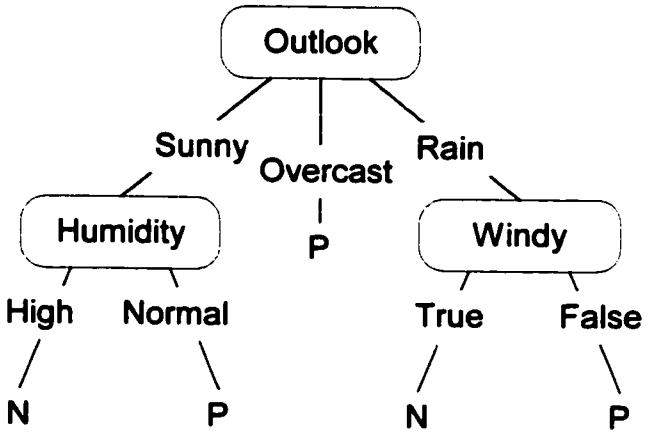


Figure 3-1: Decision tree generated from learning examples in Table 3-1

In order to classify the first case with this tree, we apply the test at the root node first. Since the value of the attribute *outlook* is sunny, the classification process should select the branch labeled *sunny*. Next, we test the value of the attribute *humidity*. As the case has the value *normal* for this attribute, the branch labeled *normal* is selected. Finally the classification process ends with the leaf labeled P (Play), and the label is returned as the class of this case.

Automatic generation of decision trees is possible through the use of ID3 algorithm and implementations like C4.5 by Quinlan J. R. [Qui93]

3.1.1 Decision Tree Construction

Let $S = \{\langle X_1, c_1\rangle, \langle X_2, c_2\rangle, \ldots, \langle X_k, c_k\rangle\}$ be a training sample. Constructing a decision tree from S can be done in a divide-and-conquer fashion as follows: *(The algorithm is called ID3 Algorithm)*

Step 1 If all the examples in S are labeled with the same class, return a leaf labeled with that class.

Step 2 Choose some test t (according to some criterion) that has two or more mutually exclusive outcomes $\{o_1, o_2, ..., o_r\}$.

Step 3 Partition S into disjoint subsets $S_1, S_2, ..., S_r$, such that S_i consists of those examples having outcome o_i for the test t, for $i = 1, 2, ..., r$.

Step 4 Call this tree-construction procedure recursively on each of the subsets S_1, S_2, ... , S_r, and let the decision trees returned by these recursive calls be T_1, $T_2, ..., T_r$.

Step 5 Return a decision tree T with a node labeled t as the root, and the trees T_1, $T_2, ..., T_r$ as subtrees below that node.

3.1.2 Test Selection

For attribute selection during the decision tree generation a criterion called *gain* is used to measure the impurity of information. The information theory says that the information conveyed by a message depends on its probability and can be measured in bits as minus the logarithm to base 2 of the probability. For example, if there are eight equally probable messages, the information conveyed by any one of them is $-\log_2(1/8)$ or 3 bits.

Let S be a set of training cases, $freq(C_i, S)$ be the number of cases in S that belong to class C_i., And $|S|$ denote the number of cases in the set S. If we pick a case from the set S at random and say that it belongs to some class C_j. This message has a probability of

$$\frac{freq(C_j, S)}{|S|}$$

and so the information it conveys is:

$$-\log_2 \frac{freq(C_j, S)}{|S|} \quad \text{bits.}$$

The expected information from this case is calculated by summing over the classes in proportion to their frequencies yielding:

$$info(S) = -\sum_{j=1}^{k} \frac{freq(C_j, S)}{|S|} \log_2 \left(\frac{freq(C_j, S)}{|S|} \right) \quad \text{bits.}$$

This measure gives the average amount of information needed to classify a case in S and is also known as the *entropy* of the set S.

Now let us assume that S has been partitioned based on a test on the attribute X with n outcomes into S_1, S_2, ..., S_n. The expected information needed to identify the class of a case in S can be found as the weighted sum over the subsets as:

$$info_X(S) = -\sum_{i=1}^{n} \frac{|S_i|}{|S|} \times info(S_i)$$

Therefore, the information gain is:

$$Information\ Gain_X(S) = info(S) - info_X(S)$$

The information gain measure indicates the information gained by partitioning S on the test X. To select the most informative test, the computation is repeated for all available tests and the test with the maximum information gain is selected.

Consider tests defined on attributes that are unique for each case, e.g. a person's name. This test would result a large number of subsets and since the subsets do not have a mixture of examples, their entropy is 0 and the information gain is maximal. This can be fixed by dividing the information gain of a test by the entropy of the test outcomes themselves, which measures the extent of splitting done by the test:

$$split_X(S) = -\sum_{i=1}^{r}\left(\frac{|S_i|}{|S|}\log\frac{|S_i|}{|S|}\right)$$

giving the gain ratio measure:

$$gain\text{-}ratio_X(S) = \frac{gain(S)}{split(S)}$$

Since *Split(S)* is higher for tests on attributes giving large number of subsets, dividing the *gain(S)* by *split(S)* indicates a high penalty on their scores, so they will not be selected. Gain-ratio is used as a criterion for the test selection in the tree construction procedure. [Qui96]

2.1.3 Extensions to the Basic Procedure

3.1.3.1 Handling Various Attribute Types

- Continuous Attributes

  Continuous attributes can be dealt with in two ways. First, training examples are sorted with respect to the current continuous attribute, then a threshold between each consecutive examples can be defined to separate the examples, preferably a mid point. The other alternative is to discretize them, where a

sequence of break points are determined and then the attributes are treated as nominal attributes. [Fay93]

- Set-Valued Attributes

  Set-valued attributes have their values in the form of a set, for example the *color* attribute of a black and white dog is *{black, white}*. A test for the attribute may look like "*black* $\in$ *color?*" and the result is true in this case.

  Cohen [Coh96] introduced a procedure that computes the class frequency in the subsets that result from partitioning the training sample using a test of the form $s \in x$, for every string s that appears in the training examples within the set-values for the attribute x. The class frequencies are used to evaluate the possible tests and return the best.

- Tree Structured Attributes

  Tree structured attributes are discrete attributes having a hierarchy of possible values (is-a hierarchy). As an example, consider an attribute called color with different levels of details like:

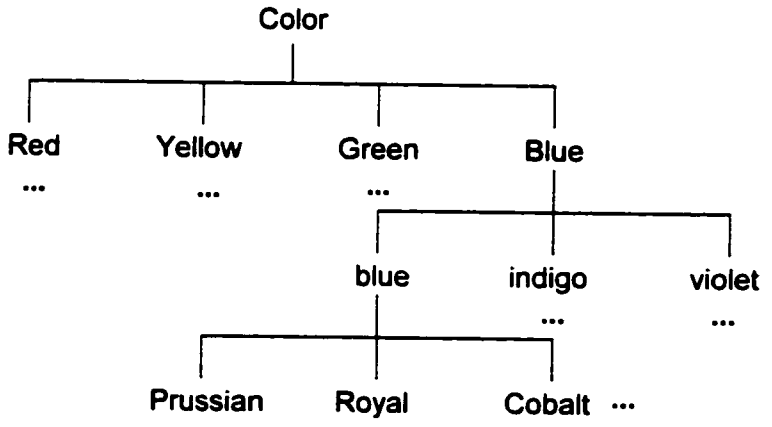


Figure 3-2: The value hierarchy for a color attribute

According to the basic procedure tests are done on the lowest level values. This should treat cases with attribute color of values *Prussian* and *Royal* differentiy, while they should be treated as *Blue* for example. This can be solved by defining a cut in the hierarchy. An efficient algorithm to solve this problem is done by [Alm96]

3.1.3.2 Incorporating More Complex Tests

Tests defined in the basic procedure are done on a single attribute. Although this can improve efficiency, more complex tests can be applied to achieve better performance.

- Linear Combination Tests

  Test nodes can be labeled with a linear combination of attributes instead of one attribute. So, a linear combination like:

  $$w_1x_1 + w_2x_2 + \ldots + w_nx_n > \theta ?$$

  where x_i is a numerical attribute, and w_i, θ are constants. [Bre84]

- Boolean Combination Tests

  For some domains more than one attribute must be tested at once combining these tests using Boolean operators ("and", "or"), for example : (Status = " student" OR age < 23) can be a test. Decision trees utilizing Boolean combination tests can be represented by simple trees (trees with single attribute tests) but at the cost of too many splits. [Bre84]

- Grouping Attribute Values

  In some domains a tests to one attribute can lead to multiple branches that are the same. For example, for a *day* attribute if it is the same to have its value any of "Saturday" through "Friday" then there is no need to have seven branches and these branches can be combined into one branch. Both [Qui93] and [Fay94] introduced algorithms for grouping values.

3.1.3.3 Attributes With Missing Values

In real world applications, some attribute values might be missing. This should be taken care of at the tree building and testing phases.

Breiman et al [Bre84] rank possible tests by ignoring training examples that have unknown values of the attribute in question. Having decided on a test T, called surrogate split, a test on a different attribute which produces a division of the current training objects that most resembles the division produced by the test T. When dividing these examples, the outcomes for examples whose outcome for T is unknown is taken as the corresponding outcome of the surrogate test. When an unseen example is being classified and the outcome of the test of the test in unknown, the outcome of the surrogate test is used. [Qui90]

Another solution to the missing attribute values is to treat the outcome of a test as fuzzy or probabilistic. When a tree is built, the merits of each possible test by

notionally distributing examples with unknown attribute values among the outcomes. Once a test is chosen, examples with missing attribute values of the tested attribute are discarded. When classifying an unseen case with missing attribute value, all outcomes of the test are explored and the results are combined probabilistically, the weights associated with the various outcomes being taken as their relative frequencies in the training set at that point [Qui90]. Another method simply replaces the missing values with the most commonly used values [Cla89] [Qui90].

3.2 C4.5

C4.5 is a package developed by Quinlan J. R. [Qui93] that implements the basic ID3 algorithm in addition to other functions and utilities.

3.2.1 Decision Tree Construction

C4.5 uses a method called **Divide-and-conquer** to construct a decision tree. To illustrate this method, let the set of training examples be called **T** and the classes be denoted $\{C_1, C_2, \ldots, C_k\}$, there are three possibilities for T :

- **T contains one or more cases**: All these cases belong to a single class C_j: the decision tree for T is a leaf identifying class C_j.
- **T contains no cases:** the decision tree is again a leaf, but the class to be associated with the leaf must be determined from information other that T. For example, the leaf might be chosen in accordance with some background knowledge of the domain, such as the overall majority class. C4.5 uses most frequent class at the parent of this node.

- **T contains cases that belong to a mixture of classes:** In this situation, the idea is to refine T into subsets of cases that are, or seem to be heading towards, single-class collections of cases. A test is chosen, based on a single attribute, that has one or more mutually exclusive outcomes $\{O_1, O_2,\ldots,O_n\}$. T is partitioned into subsets $T_1, T_2,\ldots,T_n$, where T_i contains all the cases in T that have outcome O_i of the chosen test. The decision tree for T consists of a decision node identifying the test, and one branch for each possible outcome. The same tree-building mechanism is applied recursively to each subset of training cases, so that the *i*th branch leads to the decision tree constructed from the subset T_i of training cases.

3.2.2 Pruning

Real data are commonly affected by noise arising from misclassification or incorrect measurement or recording of attribute values. This causes divide and conquer algorithm to generate elaborate trees that attempt to model the discrepancies. Such overfitting is usually addressed by pruning the initial tree, identifying subtrees that contribute little to predictive accuracy and replacing each by a leaf [Qui96].

There are two methods in which the recursive partitioning method can be modified to produce simpler tree:

- Stop dividing a set of training cases any more
- Removing retrospectively some of the tree structure

C4.5 follows the second approach. The divide-and-conquer process is given free opportunity and the overfitted tree that it produces is then pruned. Parts of the tree that do not contribute to the classification accuracy on unseen cases are removed, producing something less complex and more comprehensible. The additional computation invested in building parts of the tree that are subsequently discarded can be substantial, but this cost offset against benefits due to more thorough exploration of possible partitions, growing and pruning trees is slower but more reliable [Qui93].

As an effect of pruning the decision tree, more of the training examples are going to be misclassified and the leaves of the pruned tree will not necessarily contain training cases from a single class. On the other hand, it reduces overfitting and produces a smaller tree that has the additional advantage of increased accuracy when classifying unseen cases [Qui90].

Chapter 4 :

Multi-Class C4.5 and Multi-Binary Trees C4.5

4.1 Introduction

Many problems encountered when applying machine learning in practice involve predicting a "class" that takes on a continuous numeric value rather than a discrete category into which an example falls. Classical decision trees learning methods have developed in an environment in which class values and originally attribute values too are discrete. Over the last decade it has become commonplace to extend induction techniques to deal with numerically-valued attributes by choosing a threshold at each node of the tree, or for each conjunct of a rule, and testing the value against that

threshold. However, decision trees learners are not commonly extended to situations where the class value itself is numeric.

There are several learning techniques that do predict numeric values overviewed in chapter 2. These techniques include standard regression, neural nets, instance-based learning and regression trees. All of them have serious weaknesses. Standard regression is not a very potent way of representing induced function because it imposes a linear relationship on the data. Neural nets and instance–based learning are more powerful but suffer from opacity: the model does not reveal anything about the structure of the function that it represents. Regression trees, which are adopted by CART system, approximate a non-linear function by a piecewise constant function.

C4.5 as an implementation of a decision tree learner does not take continuous outputs into account. The purpose of this work is to suggest two approaches to modify the C4.5 implementation to account for continuous outputs. These two approaches are called: Multi-Class C4.5 and Multi-Binary trees C4.5.

4.2 Multi-Class C4.5

This approach groups continuous outputs into chunks and averages them. A discrete class replaces the output of the examples in each group. The average of the continuous outputs in each group is associated to the discrete class of the group.

4.2.1 Pseudo Code of Multi-Class approach

After reading all the examples:

[1] Sort examples according to increasing output value.

[2] Divide examples into chunks of equal size.

[3] Average the output in each chunk into a class and call it "CA".

[4] Save the original output value and call it "Output".

[5] Assign a discrete class to replace the value of the original output and associate the average (CA) of step 3 to the new class.

In the test phase of C4.5, the associated CA of the predicted class of each test example is compared to the class of the test example, and the difference error is reported. Figure 4-1 illustrates how this is done.

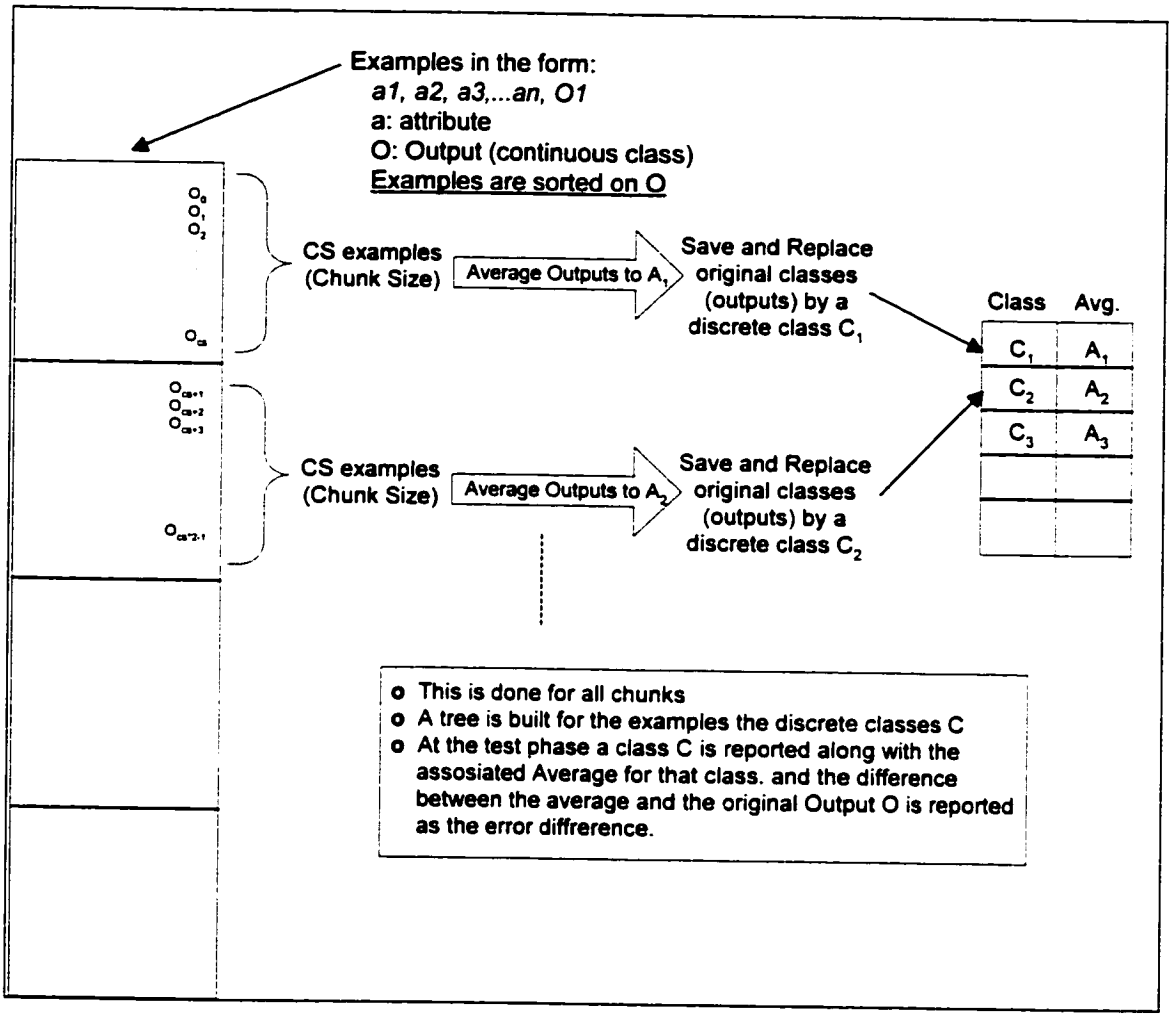


Figure 4-1: Multi-Class C4.5 approach

4.3 *Multi-Binary trees C4.5*

This approach divides the examples into chunks and builds as many decision trees as the number of chunks minus 1. The end boundary of each chunk becomes the class of the chunk. Each tree is built to indicate that the class of examples is less than or equal to the end boundary.

4.3.1 Pseudo Code of Multi-Binary trees approach

After reading all the examples:

[1] Sort examples according to increasing class value.

[2] Divide examples into chunks of equal size.

[3] Save the original class value and call it "Output".

[4] Assign class "0" to each example before the end of a chunk and "1" after that.

[5] Associate the original class of the end of chunk to the whole chunk.

[6] Build a tree for the examples in step 4.

[7] Repeat step 4 and 5 for the number of chunks, to have multiple trees.

In the test phase of C4.5, the output of all these trees are gathered in a string and the best output is chosen. Then the original class of the output is tested against the class of the test example, and the difference error is reported. Figure 4-2, illustrates how this is done.

Regarding choosing the best output, each tree gives a class for a tested example, at each tree the violations are calculated, the tree that gives the lowest violation is chosen. A violation is counted if a "0" is encountered on the left or if a "1" is encountered on the right. The associated Output O is then reported as the class. Figure 4-3 gives an example. When choosing the point that gives the lowest violation, many points can appear to be the lowest, which we called a Satisfying Point. In this work,

the first satisfying point (FSP) and the last satisfying point (LSP) are tested and the results are compared. Figure 4-4 gives an example of FSP and LSP.

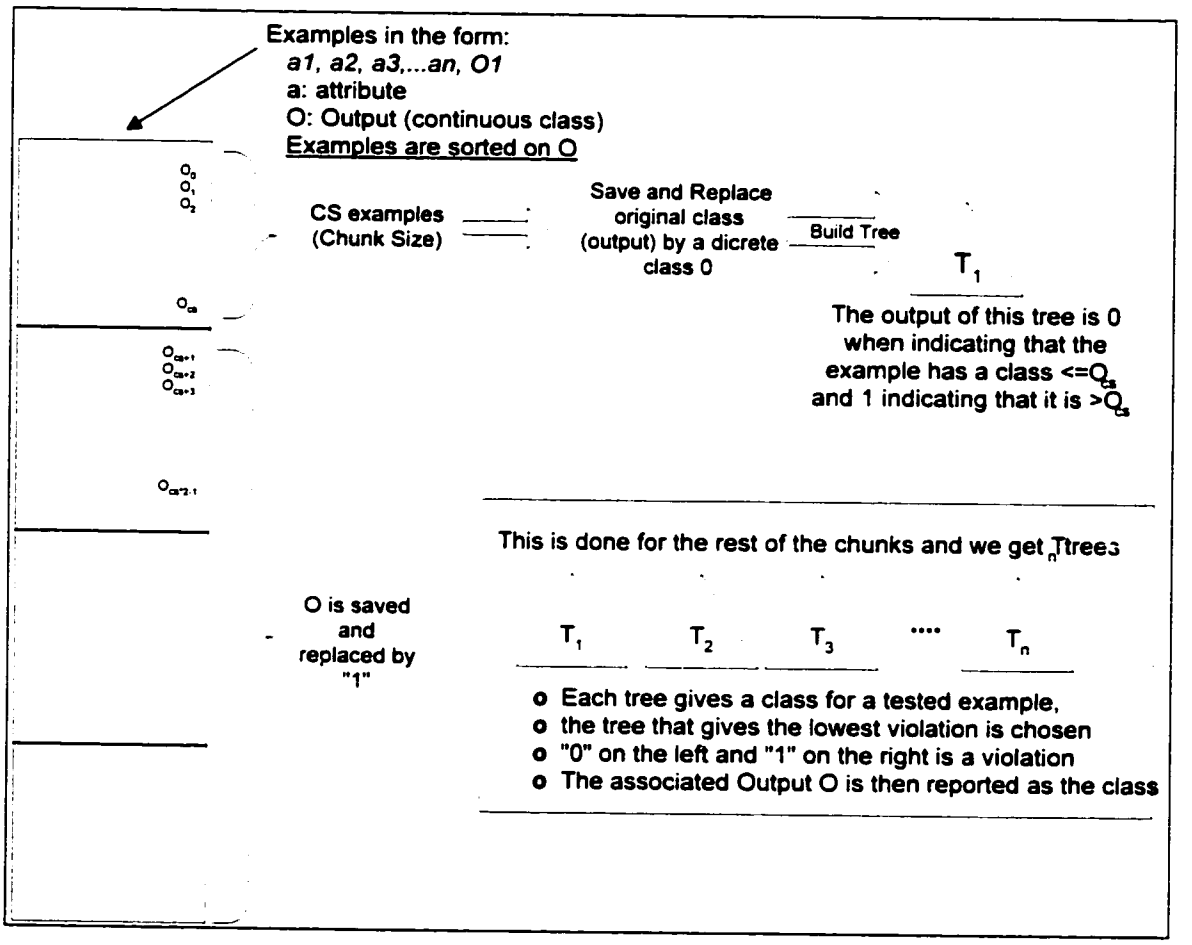


Figure 4-2: Multi-Binary trees approach

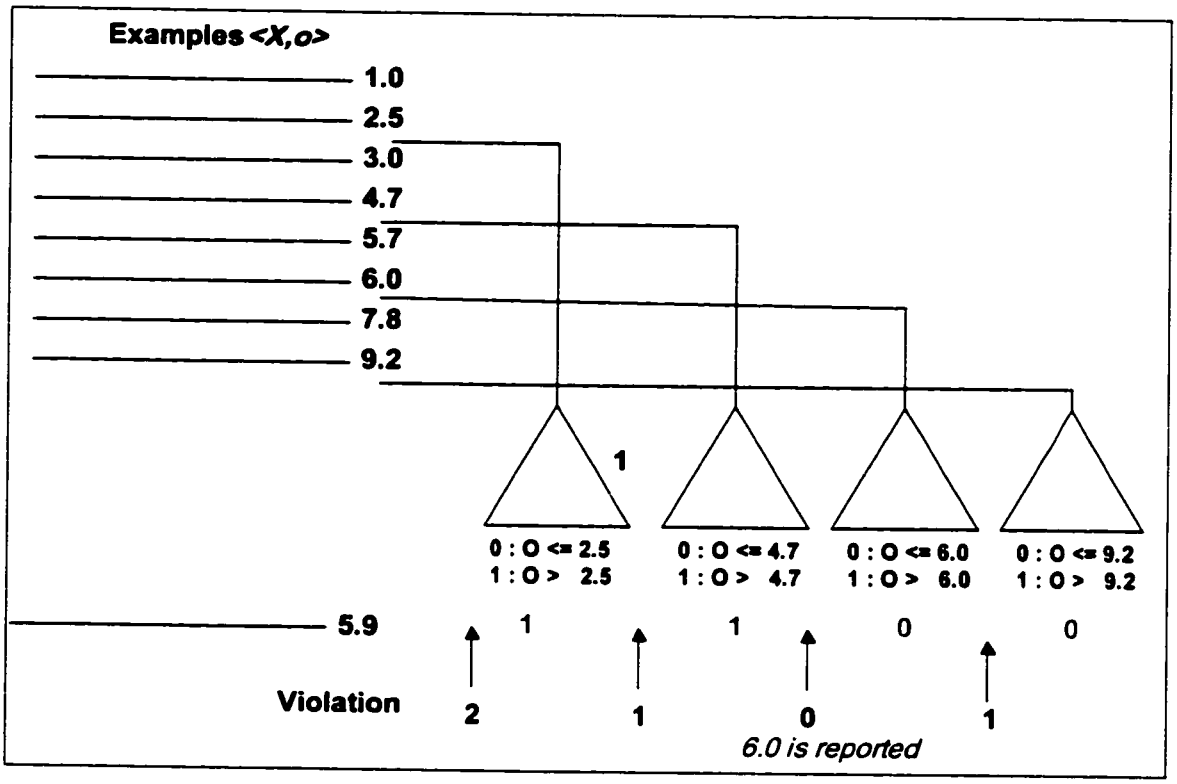


Figure 4-3 Choosing the best output

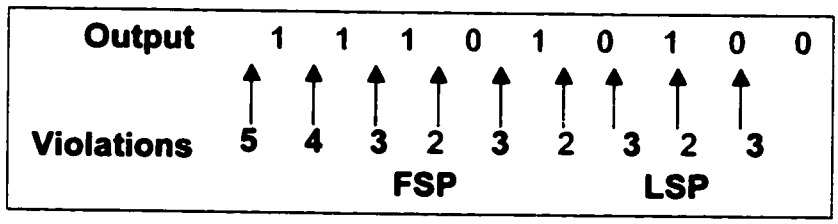


Figure 4-4 Example of First Satisfying Point (FSP) and Last Satisfying Point (LSP)

Chapter 5 :

Experiments and Results

5.1 Experiments

To experiment on these two approaches the C4.5 source code is downloaded into an IBM RS/6000 machine running IBM AIX version 4.0 (IBM's flavor of Unix). Changes are made in order for C4.5 to compile successfully under this operating system. Our proposed approaches are introduced and implemented into two separate versions.

To experiment with the implementations seven datasets with continuous target functions have been downloaded from University of California at Irvine, UCI Machine

Learning Repository, and Bilkent University (BU) Function Approximation Repository. These datasets are listed in Table 5-1.

Dataset	Source	# of examples	Target value range
HOUSING	UCI	506	5 – 50
Auto-mpg	UCI	398	9 – 46.6
Servo	UCI	167	0.13 – 7.10
Auto prices	UCI	205	5118 – 45400
CPU	UCI	209	6 – 1150
Peptide	BU	431	-8.88 – 1.99

Table 5-1: Datasets used for the experiments

For both approaches the experiments are run for chunk sizes "1" to "100". To ensure accuracy, 10-fold cross-validation is employed. Two 10-fold cross validation methods are used. The first is done after sorting the examples according to their output, and then extracting each 10^{th} example for test dataset. This ensures even output distribution on both training and test examples. The second method is the one provided with C4.5 implementation. It first randomly scrambles the examples and then clusters them in 10 portions and selects one for testing.

Experiments on different chunk sizes are done on both pruned and unpruned trees. Also experiments on First Satisfying Point (FSP) and Last Satisfying Point (LSP) of the multi-binary trees approach are gathered for comparison. The results of the

experiments done on the above datasets follow. All the figures plot the chunk size versus the average error value.

5.2 Error Calculations

In order to evaluate the prediction performance of the regression methods mentioned in [Uys00] and [Wei95] and in our work, relative error (RE) is computed by the following formula [Ala00]:

$$RE = \frac{MAD}{\frac{1}{Q}\sum_{i=1}^{Q}\left|t(q_i) - \bar{t}\right|}$$

where Q is the number of query instances, $\bar{t}$ is the median of the target values of training instances and MAD (Mean Absolute Distance) is defined as:

$$MAD = \frac{1}{Q}\sum_{i=1}^{Q}\left|t(q_i) - t(\hat{q}_i)\right|$$

where $t(q_i)$ is the Actual output of an instance and $\hat{t}(q_i)$ is the predicted class.

Previous work of [Uys00] and [Wei95] compares four regression methods, namely: 5-NN (as an example of K-Nearest Neighbor regression), rule based regression, tree based regression using CART, and MARS. Their comparison based on 10-cross-validation, is summarized in Table 5-2.

Dataset	5-nn		Rule		CART		MARS	
	MAD	ER	MAD	ER	MAD	ER	MAD	ER
Housing	2.77	0.42	2.51	0.38	2.74	0.42	2.24	0.34
Auto-Mpg	2.14	0.33	2.17	0.33	2.28	0.35	1.94	0.30
Servo	0.582	0.63	0.235	0.25	0.195	0.21	0.212	0.23
Price	1643	0.40	1335	0.32	1660	0.40	1559	0.38
CPU	29.4	0.38	27.62	0.35	30.5	0.39	27.29	0.35
Peptide	0.95	0.45	0.86	0.40	0.97	0.46	0.98	0.46

Table 5-2: Comparison of regression methods published by [Uys00] and [Wei95]

In the following subsections we experiment with each dataset present error values and compare them with the results in Table 5-2.

5.3 *Results*

Running the experiments as mentioned above on the six datasets yeilded the following results: (in the following figures, MC-C4.5 means Multi-Class C4.5 approach, MB-C4.5 means Multi-Binary Trees FSP approach and MB-C4.5-L means Multi-Binary Trees LSP.) Detailed analysis for the Housing dataset is documented in order to depict how experiments behave. While the overall analyses are given for the other datasets, similar detailed analyses are possible.

5.3.1 Housing Dataset

As mentioned in Table 5-1, Housing dataset has 506 examples with a continuous class value that ranges from 5 to 50. Chunk sizes from 1 to 100 are tested on both

approaches. Running the experiments with Housing dataset gave the results shown in Table 5-3:

Housing		10-fold cross validation (sorted)	10-fold cross validation (C4.5)
Multi-Class C4.5	MAD	1.85	1.89
	RE	0.29	0.291
	CS	21	8
Binary-Trees C4.5 *First Satisfying Point (FSP)*	MAD	1.78	1.82
	RE	0.26	0.29
	CS	2	4
Binary-Trees C4.5 *Last Satisfying Point (LSP)*	MAD	1.76	1.68
	RE	0.27	0.28
	CS	1	13

MAD: Mean Absolute Distance
RE: Relative Error
CS: Chunk Size (best)

Table 5-3: Error values of Housing dataset using new methods

5.3.1.1 Multi-Class C4.5 Analysis

In Table 5-3, the minimum chunk size is 21 for the Multi-Class C4.5 approach using the sorted 10-fold cross validation. Figure 5-1 shows how the relative error varies with the chunk size for the cross validation that gave the minimum relative error. Figure 5-2 shows how the minimum relative error for each cross validation varies for the 10 cross validations.

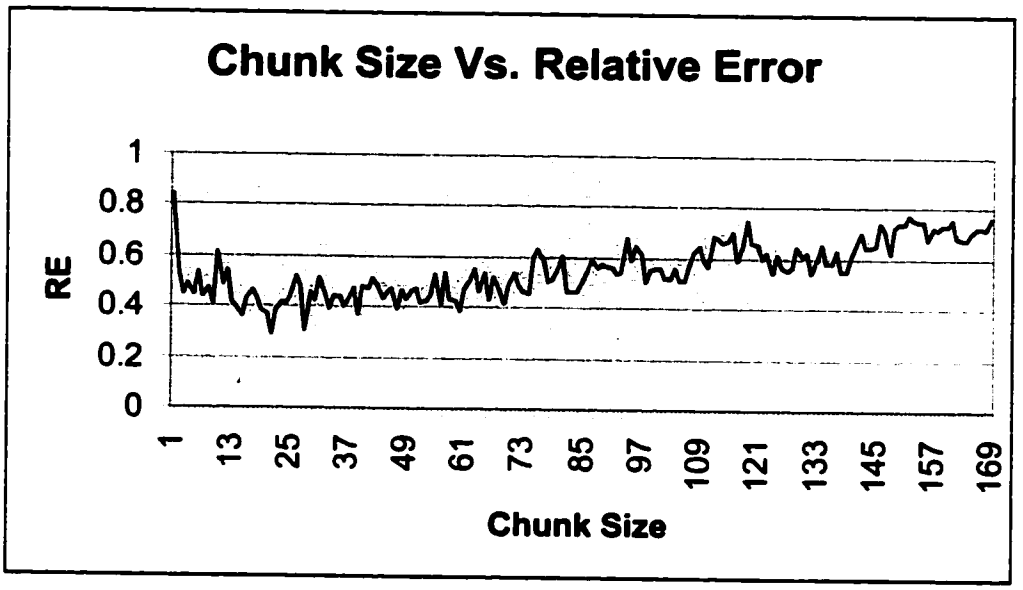


Figure 5-1: Chunk Size verse Relative Error for the cross validation with minimum RE, for the Multi-Class C4.5 approach using the sorted 10-fold cross validation.

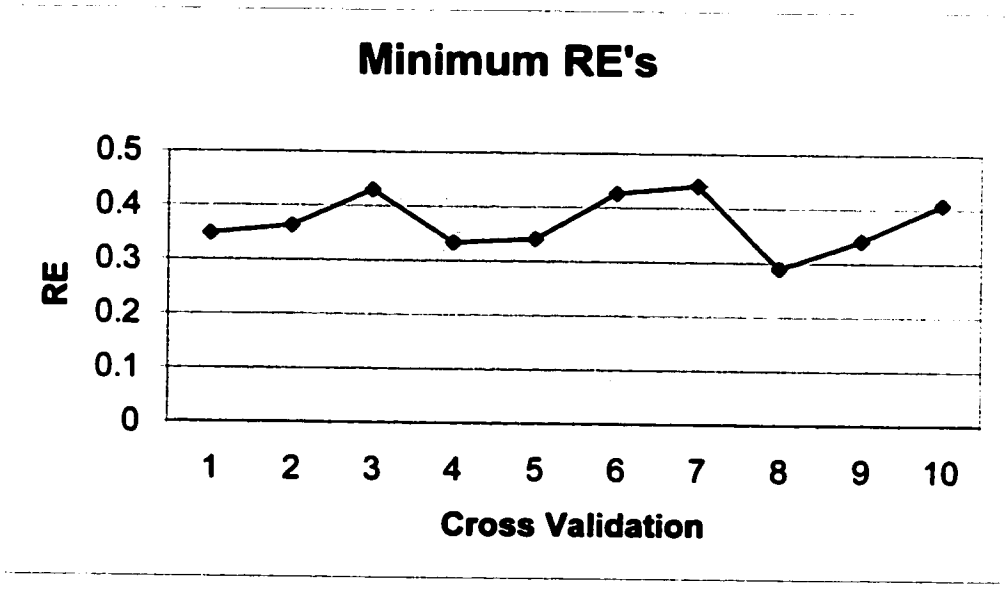


Figure 5-2: Minimum relative error for each cross validation, for the Multi-Class C4.5 approach using the sorted 10-fold cross validation

In Table 5-3, the minimum chunk size is 8 for the Multi-Class C4.5 approach using the C4.5 10-fold cross validation. Figure 5-3 shows how the relative error varies with the chunk size for the chunk cross validation that gave the minimum relative error. Figure 5-4 shows how the minimum relative error for each cross validation varies for the 10 cross validations.

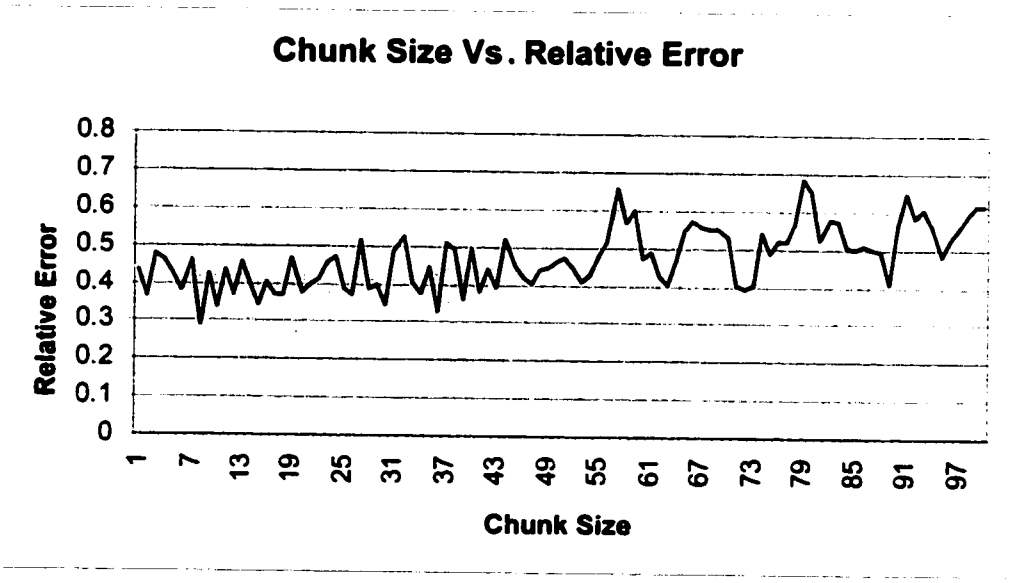


Figure 5-3: Chunk Size verse Relative Error for the cross validation with minimum RE, for the Multi-Class C4.5 approach using the C4.5 10-fold cross validation.

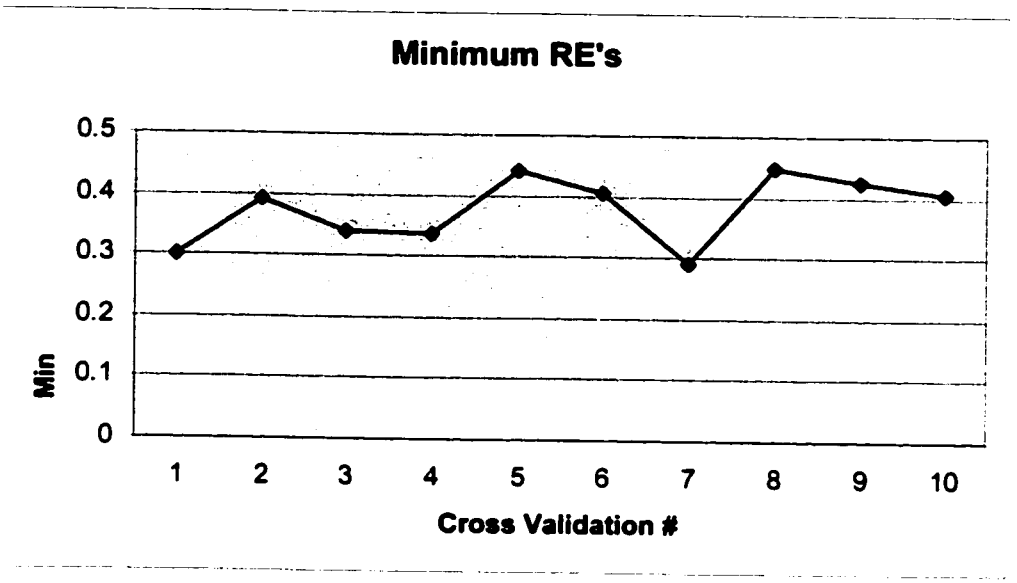


Figure 5-4: Minimum relative error for each cross validation, for the Multi-Class C4.5 approach using the C4.5 10-fold cross validation

5.3.1.2 Multi-Binary Trees C4.5 (FSP) Analysis

In Table 5-3, the minimum chunk size is 2 for the Multi-Binary Trees C4.5 (FSP) approach using the sorted 10-fold cross validation. Figure 5-5 shows how the relative error varies with the chunk size for the cross validation that gave the minimum relative error. Figure 5-6 shows how the minimum relative error for each cross validation varies for the 10 cross validations.

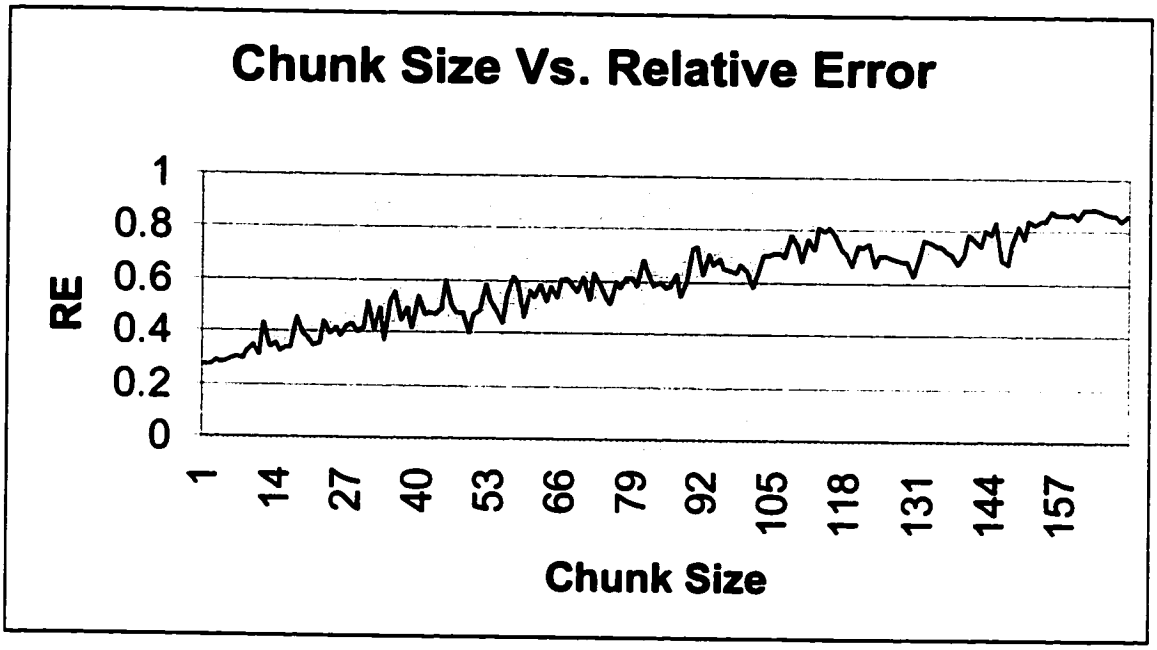


Figure 5-5: Chunk Size verse Relative Error for the cross validation with minimum RE, for the Multi-Binary Trees C4.5 (FSP) approach using the sorted 10-fold cross validation.

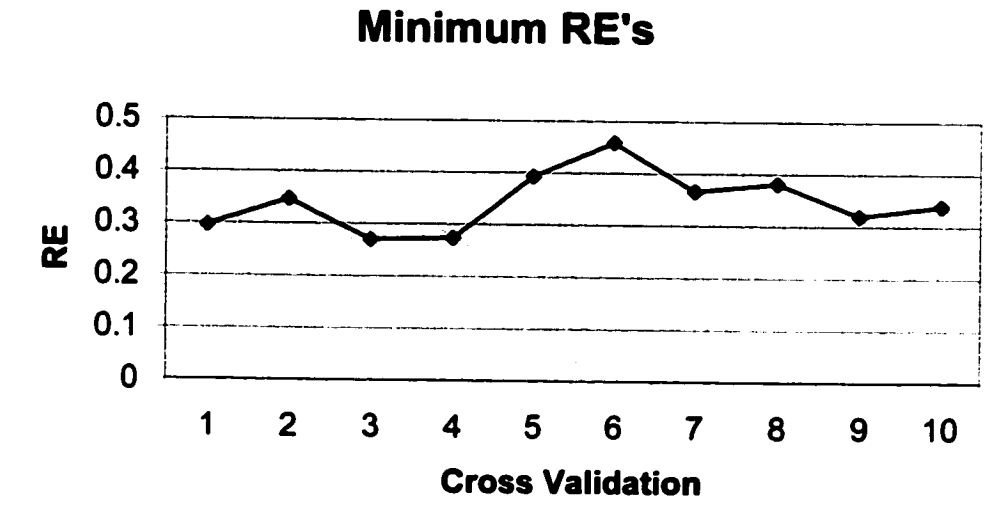


Figure 5-6: Minimum relative error for each cross validation, for the Multi-Binary Trees C4.5 (FSP) approach using the sorted 10-fold cross validation

In Table 5-3, the minimum chunk size is 4 for the Multi-Binary Trees C4.5 (FSP) approach using the C4.5 10-fold cross validation. Figure 5-7 shows how the relative error varies with the chunk size for the chunk cross validation that gave the minimum relative error. Figure 5-8 shows how the minimum relative error for each cross validation varies for the 10 cross validations.

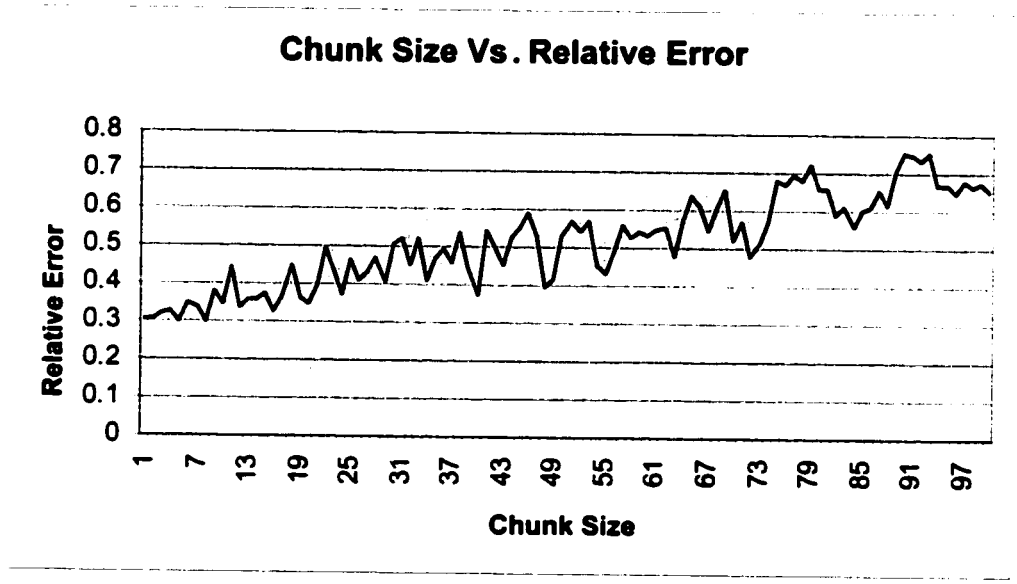


Figure 5-7: Chunk Size verse Relative Error for the cross validation with minimum RE, for the Multi-Binary Trees C4.5 (FSP) approach using the C4.5 10-fold cross validation.

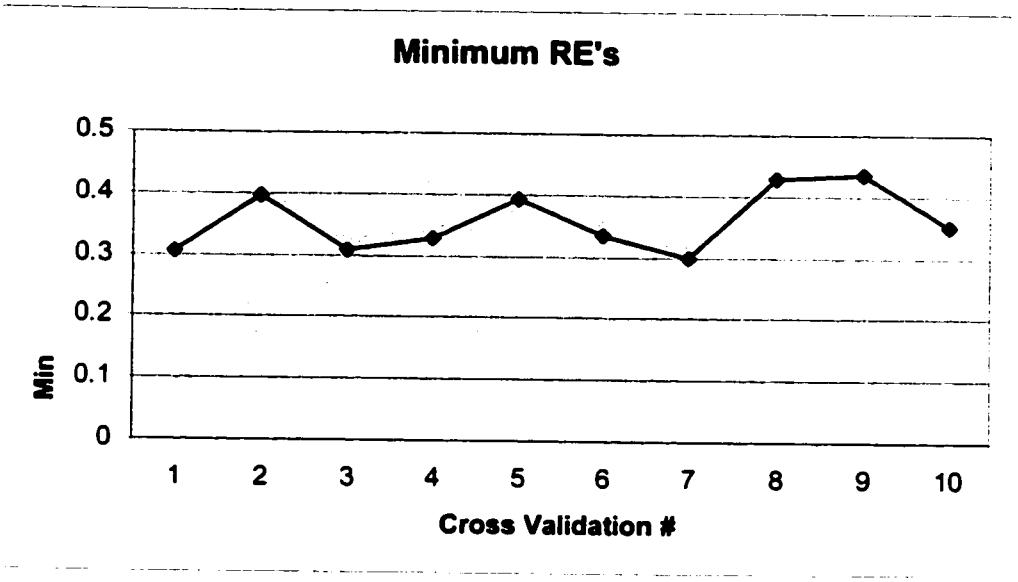


Figure 5-8: Minimum relative error for each cross validation, for Multi-Binary Trees C4.5 (FSP) approach using the C4.5 10-fold cross validation

5.3.1.3 Multi-Binary Trees C4.5 (LSP) Analysis

In Table 5-3, the minimum chunk size is 1 for the Multi-Binary Trees C4.5 (LSP) approach using the sorted 10-fold cross validation. Figure 5-9 shows how the relative error varies with the chunk size for the cross validation that gave the minimum relative error. Figure 5-10 shows how the minimum relative error for each cross validation varies for the 10 cross validations.

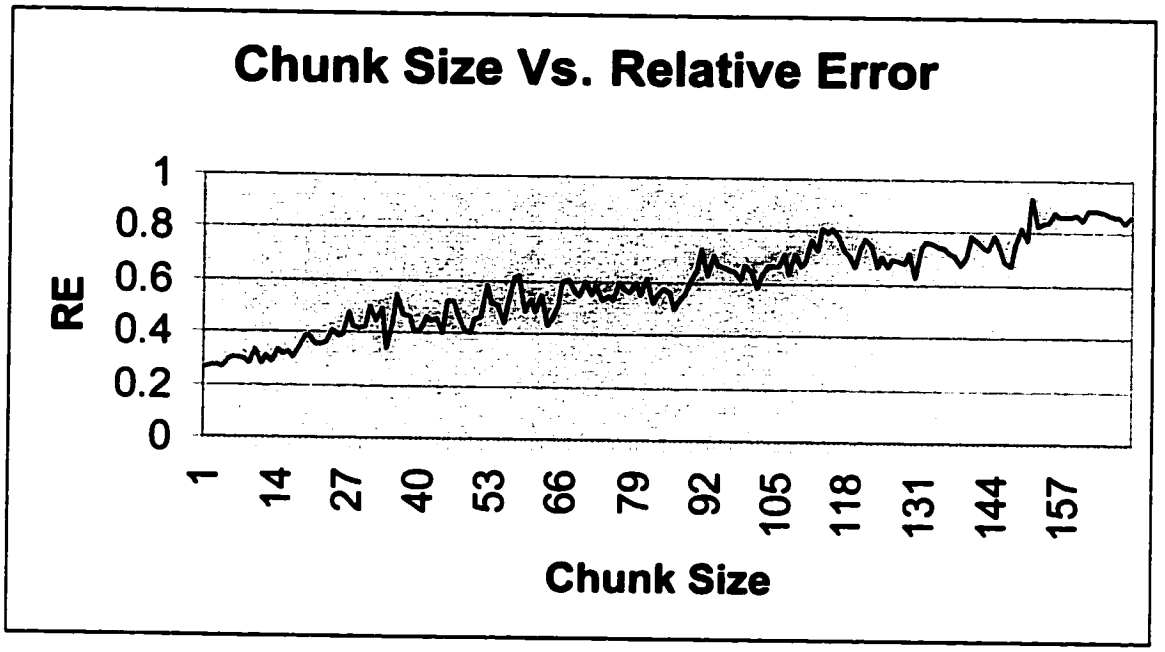


Figure 5-9: **Chunk Size verse Relative Error for the cross validation with minimum RE, for the Multi-Binary Trees C4.5 (LSP) approach using the sorted 10-fold cross validation.**

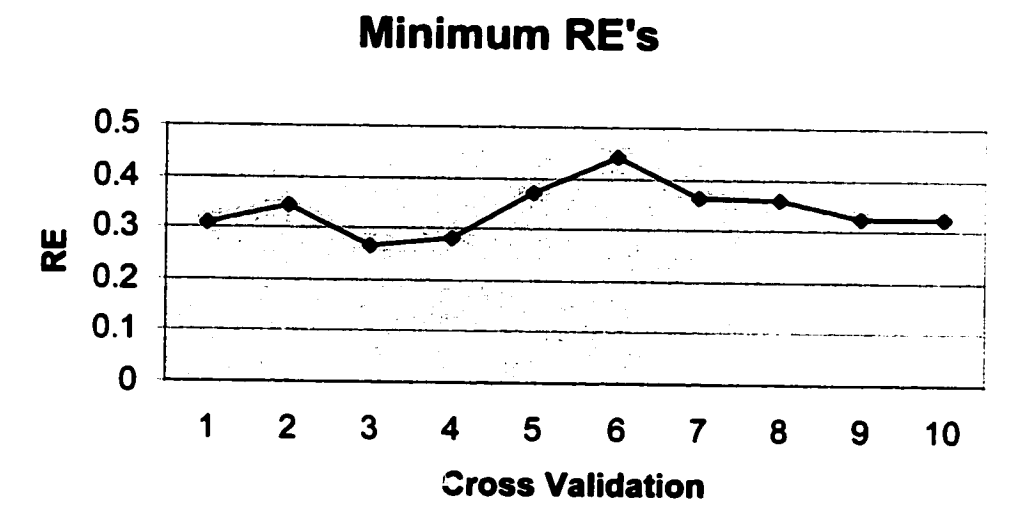


Figure 5-10: **Minimum relative error for each cross validation, for the Multi-Binary Trees C4.5 (LSP) approach using the sorted 10-fold cross validation**

In Table 5-3, the minimum chunk size is 13 for the Multi-Binary Trees C4.5 (LSP) approach using the C4.5 10-fold cross validation. Figure 5-11 shows how the relative error varies with the chunk size for the chunk cross validation that gave the minimum relative error. Figure 5-12 shows how the minimum relative error for each cross validation varies for the 10 cross validations.

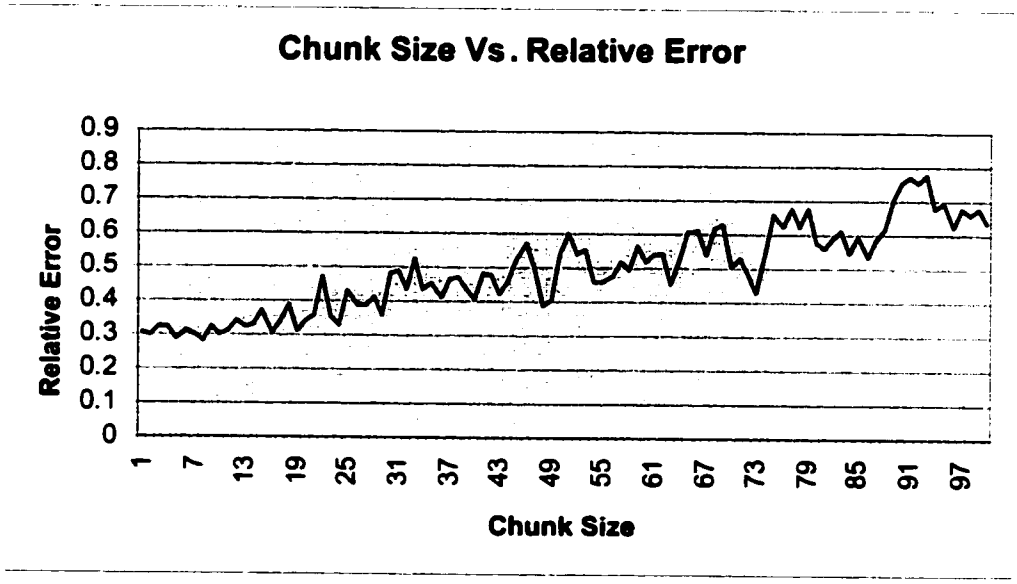


Figure 5-11: Chunk Size verse Relative Error for the cross validation with minimum RE, for the Multi-Binary Trees C4.5 (LSP) approach using the C4.5 10-fold cross validation.

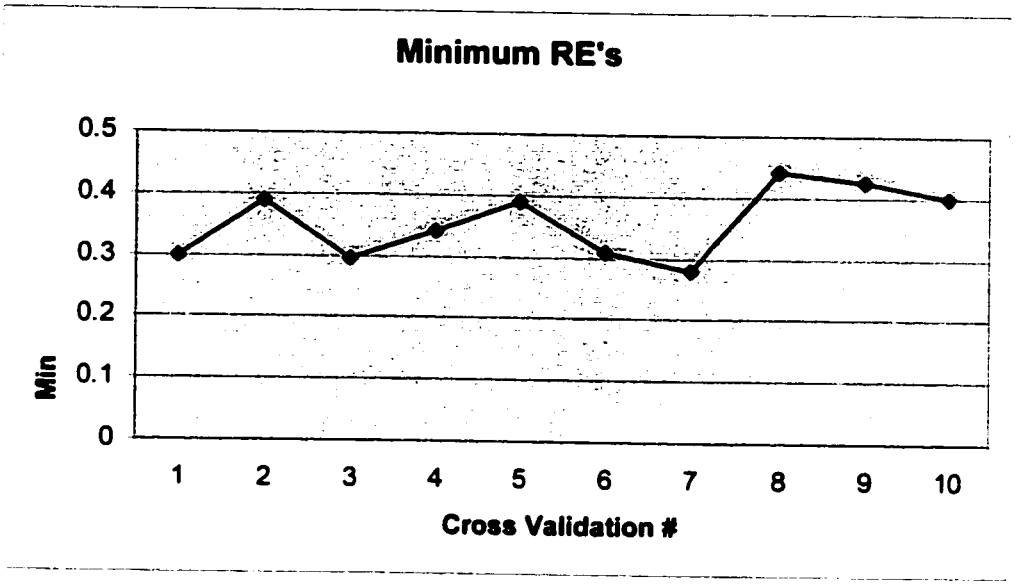


Figure 5-12: Minimum relative error for each cross validation, for the Multi-Binary Trees C4.5 (LSP) approach using the sorted 10-fold cross validation

5.3.1.4 Overall Comparisons

Figure 5-13 plots the relative error (ER) values in comparison with other regression methods. In this figure MC-C4.5 is Multi-Class C4.5, MB-C4.5-F is Multi-Binary tree C4.5 (FSP), and MB-C4.5-L is Multi-Binary tree C4.5 (LSP). The first group of columns plots the relative error for the other regression methods, the second groups of columns plots the relative error for the suggested methods with a sorted 10-cross-validation method, while the third group of columns plots the relative error for the suggested methods with the 10-cross-validation method used by C4.5.

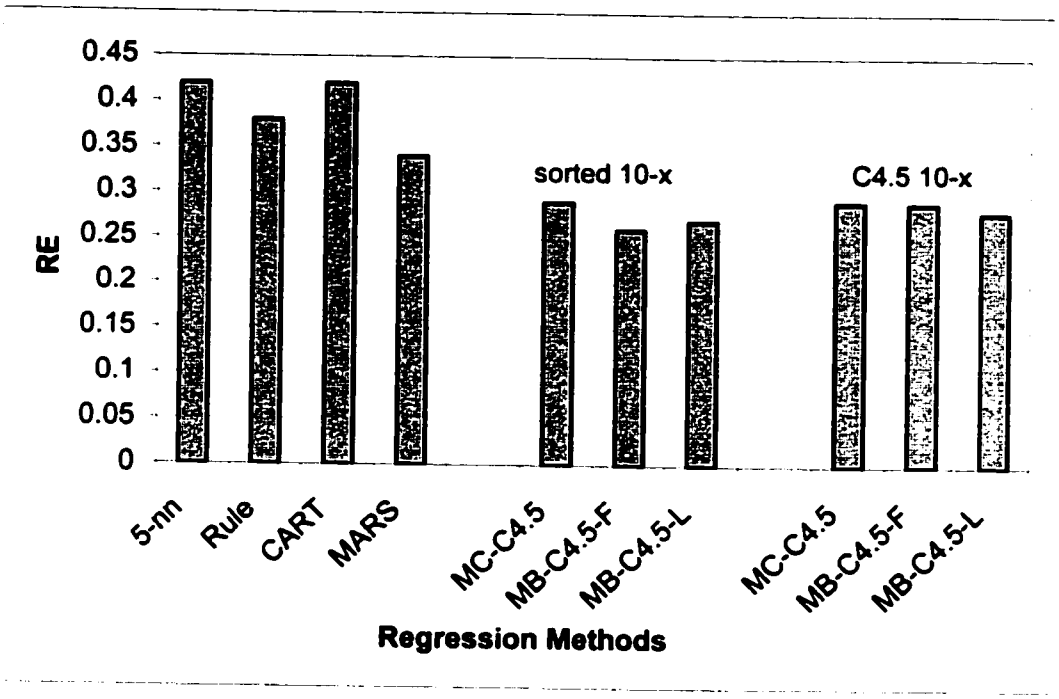


Figure 5-13: Relative Error (RE) of Housing dataset with different regression methods

5.3.2 Auto-Mpg Dataset

As mentioned in Table 5-1, Auto-mpg dataset has 398 examples with a continuous class value that ranges from 9 to 46.6. Chunk sizes from 1 to 100 are tested on both approaches. Running the experiments with Auto-Mpg dataset gave the results shown in Table 5-4.

Auto-MPG		10-fold cross validation (sorted)	10-fold cross validation (C4.5)
Multi-Class C4.5	MAD	1.84	1.597
	RE	0.29	0.245
	CS	23	33
Binary-Trees C4.5 *First Satisfying Point (FSP)*	MAD	1.91	2.01
	RE	0.29	0.3
	CS	13	1
Binary-Trees C4.5 *Last Satisfying Point (LSP)*	MAD	1.86	1.95
	RE	0.29	0.29
	CS	13	1

MAD: Mean Absolute Distance
RE: Relative Error
CS: Chunk Size (best)

Table 5-4: Error values of Auto-MPG dataset using new methods

Figure 5-14 plots the relative error (ER) values in comparison with other regression methods.

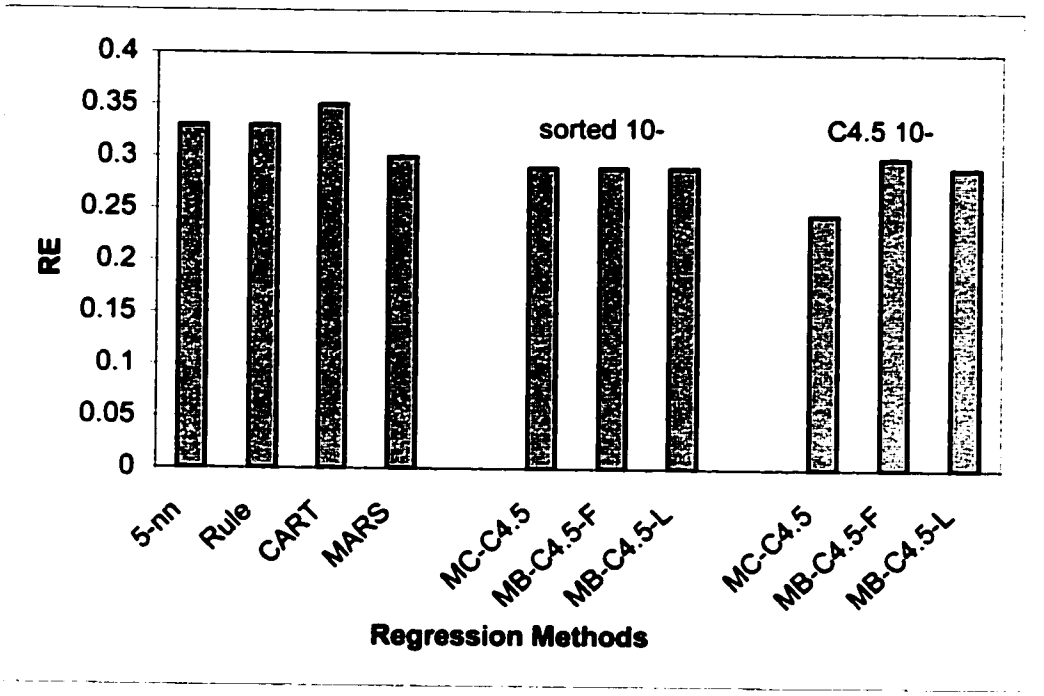


Figure 5-14: Relative Error (RE) of Auto-MPG dataset with different regression methods

5.3.3 Servo Dataset

As mentioned in Table 5-1, Servo dataset has 167 examples with a continuous class value that ranges from 0.13 to 7.10. Chunk sizes from 1 to 100 are tested on both approaches. Running the experiments with Servo dataset gave the results shown in Table 5-5.

Servo		10-fold cross validation (sorted)	10-fold cross validation (C4.5)
Multi-Class C4.5	MAD	0.19	0.139
	RE	0.21	0.18
	CS	11	21
Binary-Trees C4.5 *First Satisfying Point (FSP)*	MAD	0.19	0.16
	RE	0.24	0.24
	CS	6	5
Binary-Trees C4.5 *Last Satisfying Point (LSP)*	MAD	0.19	0.16
	RE	1.24	0.24
	CS	6	5

MAD: Mean Absolute Distance
RE: Relative Error
CS: Chunk Size (best)

Table 5-5: Error values of Servo dataset using new methods

Figure 5-15 plots the relative error (ER) values in comparison with other regression methods.

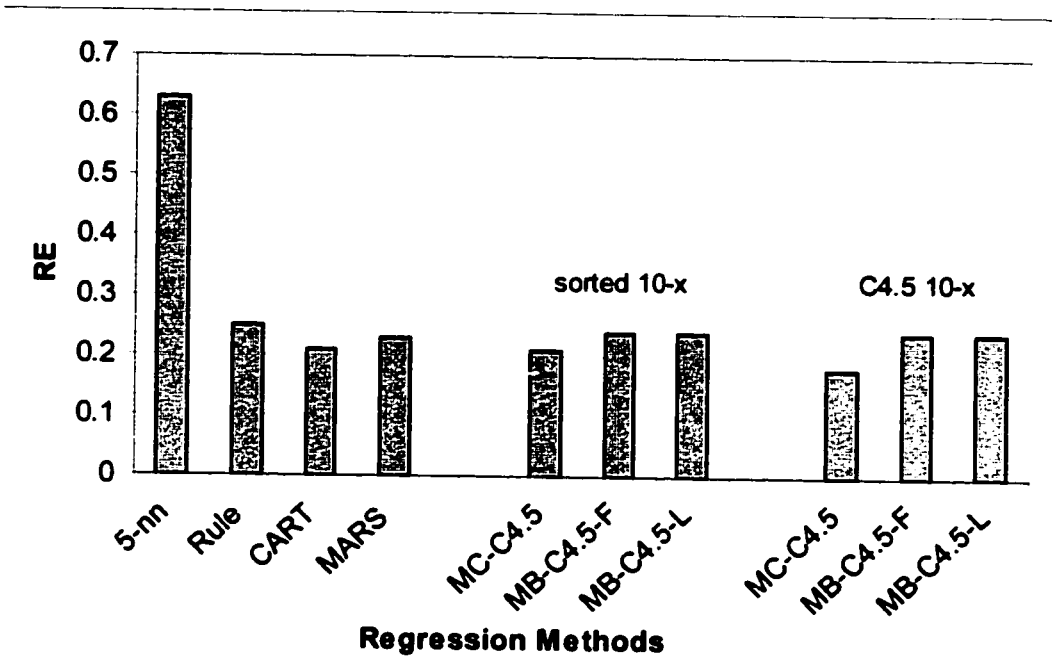


Figure 5-15: Relative Error (RE) of Servo dataset with different regression methods

5.3.4 Auto Prices Dataset

As mentioned in Table 5-1, Auto Prices dataset has 205 examples with a continuous class value that ranges from 5118 to 45400. Chunk sizes from 1 to 100 are tested on both approaches. Running the experiments with Auto-Prices dataset gave the results shown in Table 5-6.

Autos (Price)		10-fold cross validation (sorted)	10-fold cross validation (C4.5)
Multi-Class C4.5	MAD	1015.68	804.44
	RE	0.18	0.169
	CS	15	18
Binary-Trees C4.5 *First Satisfying Point (FSP)*	MAD	1158.5	861.62
	RE	0.226	0.187
	CS	2	9
Binary-Trees C4.5 *Last Satisfying Point (LSP)*	MAD	1091.3	908.23
	RE	0.213	0.19
	CS	5	9

MAD: Mean Absolute Distance
RE: Relative Error
CS: Chunk Size (best)

Table 5-6: Error values of Auto-Prices dataset using new methods

Figure 5-16 plots the relative error (ER) values in comparison with other regression methods.

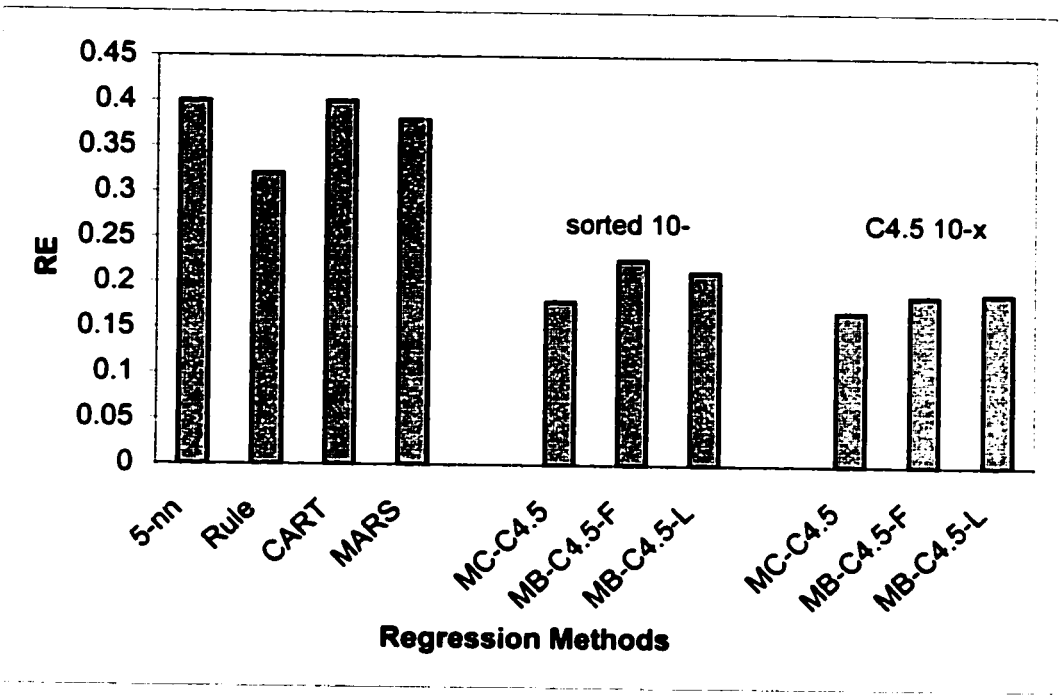


Figure 5-16: Relative Error (RE) of Auto-Prices dataset with different regression methods

5.3.5 CPU Dataset

As mentioned in Table 5-1, CPU dataset has 209 examples with a continuous class value that ranges from 6 to 1150. Chunk sizes from 1 to 100 are tested on both approaches. Running the experiments with CPU dataset gave the results shown in Table 5-7.

CPU		10-fold cross validation (sorted)	10-fold cross validation (C4.5)
Multi-Class C4.5	MAD	16.28	19.96
	RE	0.226	0.21
	CS	11	11
Binary-Trees C4.5 *First Satisfying Point (FSP)*	MAD	14.67	17.19
	RE	0.231	0.30
	CS	4	33
Binary-Trees C4.5 *Last Satisfying Point (LSP)*	MAD	14.29	17.10
	RE	0.225	0.25
	CS	3	33

MAD: Mean Absolute Distance
RE: Relative Error
CS: Chunk Size (best)

Table 5-7: Error values of CPU dataset using new methods

Figure 5-17 plots the relative error (ER) values in comparison with other regression methods.:

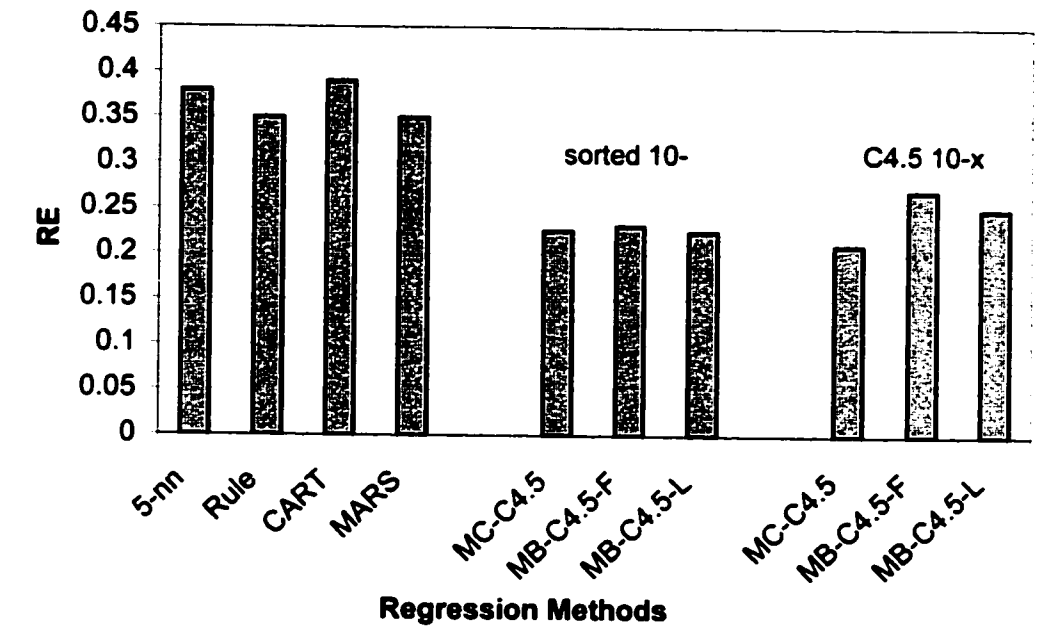


Figure 5-17: Relative Error (RE) of CPU dataset with different regression methods

5.3.6 Peptide Dataset

As mentioned in Table 5-1, Peptide dataset has 431examples with a continuous class value that ranges from –8.88 to 1.99. Chunk sizes from 1 to 100 are tested on both approaches. Running the experiments with Peptide dataset gave the results shown in Table 5-8.

Peptide		10-fold cross validation (sorted)	10-fold cross validation (C4.5)
Multi-Class C4.5	MAD	0.75	0.73
	RE	0.35	0.37
	CS	26	31
Binary-Trees C4.5 *First Satisfying Point (FSP)*	MAD	0.643	0.58
	RE	0.305	0.28
	CS	6	5
Binary-Trees C4.5 *Last Satisfying Point (LSP)*	MAD	0.647	0.58
	RE	0.307	0.28
	CS	6	10

MAD: Mean Absolute Distance
RE: Relative Error
CS: Chunk Size (best)

Table 5-8: Error values of Peptide using new methods

Figure 5-18 plots the relative error (ER) values in comparison with other regression methods.

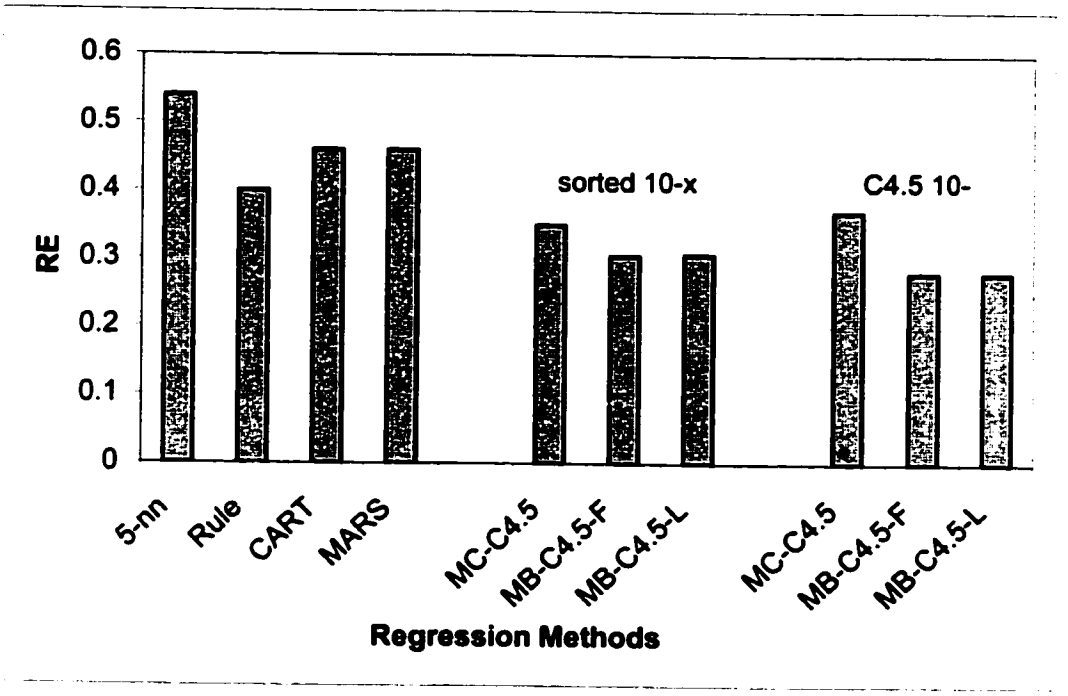


Figure 5-18: Relative Error (RE) of Peptide dataset with different regression methods

5.4 Analysis

Looking at previous figures we find sometimes the Multi-Class C4.5 approach is better and some other times the Multi-Binary trees C4.5 is better depending on the domain. In general, both Multi-Class C4.5 and Multi-Binary trees C4.5 approaches perform better then other regression methods.

In order to find the best chunk size (CS), running the tests on a range of chunk sizes is needed then the RE is to be compared and the minimum is chosen. It is to be noted that a chunk size of 1 means that Multi-Binary trees C4.5 approach will build as many

trees as many examples, on the other hand, in the Multi-Class C4.5 approach each example is assigned a discrete class. This is not acceptable in both cases because in Multi-Binary trees C4.5 huge memory is needed while in Multi-Class C4.5 CPU is consumed much.

The reason that the Multi-Binary Trees C4.5 approach perform better in most of the cases especially at smaller chunk sizes, most of them at chunk size "1", is that at this chunk size a tree is built for each example. That means that the class of the tested example is determined more precisely. If performance of much concern, Multi-Class C4.5 can be chosen since the difference in the relative error is very small.

Chapter 6 :

Conclusions and Future Work

To conclude this work, it can be stated that our two proposed approaches to learning continuous functions, Multi-Class C4.5 and Multi-Binary Trees C4.5, perform better than most of the other regression techniques. Both approaches compete depending on the domain.

Even though the results show significant improvement, there are many extentions that can be thought of. For example in the Multi-Class C4.5 approach, a different clustering technique can be used, chunks or clusters need not be of the same size. Another improvement can be done at assigning a discrete class to each cluster, the

average function is used in this study, but other functions can also be experimented with in order to reduce the error rates.

On the other hand, the Multi-Binary Trees C4.5 can be improved by extending the binary trees to ternary trees, that is the output of each tree can be of the set {0, 1, 2} and count a change from 0 to 2 as a violation instead of from 0 to 1. This could create trees with better confidence, which may reduce the error rates. Also, a different partitioning technique can be used, chunks or clusters need not be of the same size. Another improvement can be done at selecting the point that makes the least violation. As we have seen there can be more than one point, the first satisfying point (FSP) and the last satisfying point (LSP) are tested in this study. The error rates differs in both, so, a technique to select the best of FSP, LSP or the intermediate satisfying point can be devised to reduce the error rate.

References

[Ahd94] Aha, D.W. and Salzberg, S. L., Learning to Catch: Applying Nearest Neighbor Algorithms to Dynamic Control Tasks, *Selecting Models from Data: Artificial Intelligence and Statistics IV.*, New York, NY: Springer-Verlag, 1994.

[Ala00] Güvenir, Altay, H., and Uysal I., *Regression on Feature Projections*, Knowledge-Based Systems, Vol. 13, No. 4, (2000), pp. 207-214.

[Alm02] Almuallim, H., Kaneda, S., and Akiba, Y., *Development and Applications of Decision Trees*, A Book Chapter in "Expert systems : the technology of knowledge management and decision making for the 21st century", Edited by Cornelius T. Leondes, San Diego, Academic Press, 2002.

[Alm96] Almuallim, H., Kaneda, S., and Akiba, Y., An Efficient Algorithm for Finding Optimal Gain-Ratio Multiple-Split Tests on Hierarchical Attributes in Decision Tree Learning, *Proceeding of the 13th National Conference on Artificial Intelligence (AAAI-96)*, August 1996.

[Bre84] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. Classification and Regression Trees. *Wadsworth*, Belmont, California, USA, 1984.

[BUR] Güvenir, H. and Uysal, I., *Bilkent University Function Approximation Repository*, [http://pcguvenir.cs.bilkent.edu.tr/DataSets/], Bilkent University, Department of Computer Engineering, Bilkent Machine Learning Group, Turky.

[Cla89] Clark, P. and Niblett, T., The CN2 Induction Algorithm, *Machine Learning*, 3:261-283, 1989.

[Coh96] Cohen, W., Learning Trees and Rule with Set-Valued Features, *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI96)*, 709-716, August 1996.

[Fay91] Fayyad, U. M., *On the Induction of Decision Trees for Multiple Concept Learning*, PhD thesis, University of Michigan, AnnArrbor, 1991.

[Fay93] Fayyad, U. M. and Irani, K. B. Multi-interval discretization of continuous valued attributes for classification learning, *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI96)*, 1022-1027, August 1996.

[Fay94] Fayyad, U. M. Branching on attribute values in decision tree generation. *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI94)*, 601-606.

[Fri91] Friedman, J. H., *Multivariate Adaptive Regression Splines*, The Annals of Statistics, 19 (1) 1-141, 1991.

[Mit97] Mitchell, Tom M., *Machine Learning*, McGraw-Hill, 1997.

[Qui86] Quinlan, J. R., *Induction of Decision Trees*, Machine Learning, pages 81-106, 1986.

[Qui90] Quinlan, J. R., *Decision Trees and Decisionmaking*, IEEE Transactions on Systems, Man , and Cybernetics, Vol. 20, No. 2, Mar/Apr 1990, pp. 339-346.

[Qui93] J.R. Quinlan, *C4.5: Programs for Machine Learning,* Morgan Kaufman, 1993.

[Qui96] Quinlan, J. R., *Learning Decision Tree Classifiers*, ACM Computing Surveys, Vol. 28, No. 1, March 1996, pp. 71-72.

[UCI] Blake, C.L. & Merz, C.J. *UCI Repository of machine learning databases* [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

[Uys00] Uysal, Ilhan and Güvenir, H. Altay, *An Overview of Regression Techniques for Knowledge Discovery,* The Knowledge Engineering Review, Cambridge University Press, Vol. 14, No. 4, (2000), pp. 319-340

[Wan97] Wang, Y and Witten I., *Inducing model trees for continuous classes*. Proc of Poster Papers, 9th European Conference on Machine Learning, Prague, Czech, April 1997

[Wei93] Weiss, S. and Indurkhya, N., *Optimized Rule Induction*, IEEE Expert, 8(6), 61-69, 1993.

[Wei95] Weiss, S. and Indurkhya, N., *Rule-based Machine Learning Methods for Functional Prediction*, Journal of Artificial Intelligence Research, 3 383-403, 1995.

Vitae

- Ahmed Esmat Mahmoud Ibrahim.
- Born in 1969 at Cairo, Egypt.
- Received Bachelor of Science degree (B.Sc.) in Computer Science in 1993 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
- Started working in the Information Technology industry right away after graduation. Worked for the Information Technology Center at King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, as an Application Developer.
- Received the Master of Science degree (M.S.) in Computer Science in May 2001 from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
- Current e-mail address is: aesmat1@yahoo.com and personal homepage URL is: http://aesmat1.tripod.com